REVISTA MARATONA SBC DE PROGRAMAÇÃO

EDIÇÃO DA FINAL BRASILEIRA 2025

Liberdade, São Paulo, SP 06 a 09 de novembro

PROGRAMAÇÃO COMPETITIVA

Artigos técnicos, análises demográficas, histórias de superação e muito mais

EXPEDIENTE

Capa:

Fotógrafo(a) UCPel

Comitê Editorial

Bruno Monteiro, Giovanna Kobus, Rafael Grandsire, Mariana Gomes

Colunistas e colaboradores

Ana Carolina Xavier Castro; André L. P. Guedes; Bernardo Amorim; Camila Cruz dos Santos; Cauê Rodrigues de Aguiar; Crishna Irion; Diego Pedro; Fernando Monteiro; Gabriel P. Falcão; Jenniffer O. Checchia; João Victor Ayalla; Lis Loureiro Sousa; Lucy Mari Tabuti; Pedro V. Sousa da Silva; Rafael Granza; Yan S. Couto; Yuri Kaszubowski Lopes

Direção de imagem e design

Vinicius Cardoso de Lima

ÍNDICE

PROBLEMAS DE MARATONA NO FACEBOOK

Yan S. Couto, USP

1. Motivação

Trabalhei no Facebook (Meta) como Engenheiro de Software de 2019 a 2023. Como tenho um passado em Maratona de Programação, busquei times que tivessem mais a ver com algoritmos e estruturas de dados. Entre eles, o mais próximo foi o time de Source Control, que lidava com o versionamento de código interno Sapling, que é, essencialmente, a versão interna do Git no Facebook, baseada no Mercurial, e que recentemente foi parcialmente lançada em software livre [3].

Uma versão interna de controle de software é necessária devido à alta quantidade de pessoas modificando o mesmo repositório. O Git/GitHub nunca conseguiriam lidar com repositórios com alguns milhões de arquivos, terabytes de tamanho e que podem receber até 1 commit por segundo! Por causa disso, algoritmos e estruturas de dados altamente eficientes são *necessários* para garantir que todos os engenheiros pudessem trabalhar eficientemente. Caso contrário, coisas tão simples quanto um **git status** podem demorar muitos minutos, e a maioria dos computadores não conseguiria nem rodar **git clone**, pois não teriam espaço suficiente. A minha experiência em Maratona de Programação me ajudou muito a lidar com essas estruturas, e vou apresentar os dois maiores problemas que enfrentei nessa área.

2. Trie Persistente Comprimida

Uma das (muitas) formas de armazenar os arquivos em controle de versionamento é usando uma árvore persistente. Cada diretório é um nó da árvore, que aponta para seus arquivos e subdiretórios. Ao modificar algum arquivo, copiamos e modificamos o nó associado a este arquivo e todos os seus ancestrais até a raiz do repositório.

Dessa forma, a maior parte da árvore é reutilizada de versões anteriores. Cada nó guarda também um ponteiro para sua versão anterior, de forma que podemos ver toda a "história" de uma pasta ou arquivo de forma eficiente. Essa forma de implementar estruturas persistentes é chamada de *funcional*, e esta e outras técnicas para implementar estruturas funcionais estão explicadas em português na minha dissertação de mestrado [2].

Vamos discutir um pouco sobre como isso era armazenado na prática. Usávamos um "key-value storage", que é um banco de dados distribuído onde armazenamos pares de chave-valor. Simplificadamente, é como se tivéssemos um **map<string, string>** em C++ no qual cada acesso faz uma chamada de rede para obter o valor (ou objeto). Isso significa que precisamos equilibrar o tamanho dos objetos: eles não devem ser tão grandes a ponto de retornar muitos dados "inúteis" (que não serão utilizados pelo algoritmo), mas também não tão pequenos que exijam muitas chamadas de rede, já que cada acesso é custoso.

Cada nó da árvore era armazenado como um único objeto e continha um mapa que guardava os ponteiros (chaves) de todos os seus filhos. Uma destas árvores armazenava todos os arquivos que já foram deletados e, portanto, nunca diminuía de tamanho, e alguns diretórios tinham milhões de arquivos neles. Ao modificar um arquivo, era necessário copiar os ponteiros para todos os outros arquivos "irmãos", ou seja, o custo da operação não era proporcional ao número de arquivos modificados, mas sim ao número de *irmãos* dos arquivos modificados. Com milhões de irmãos, isso começou a causar problemas mesmo para commits pequenos.

Para solucionar esse problema, foi necessário quebrar cada nó em vários objetos, dependendo do número de filhos que ele tinha. Para isso, passamos a guardar o mapeamento dos filhos em uma trie, uma estrutura que armazena strings de forma ordenada e eficiente, fácil de tornar persistente e de particionar em vários objetos.

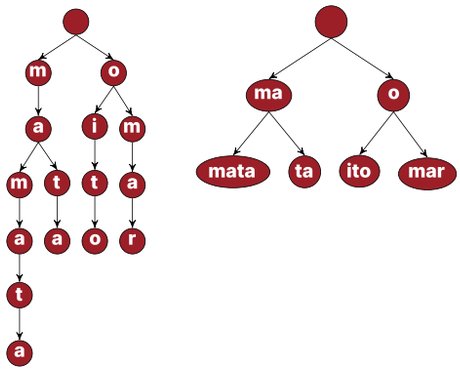


Figura 1: À esquerda, uma trie para as strings {mamata, mata, oito, omar}. À direita, sua versão comprimida.

Porém, não podemos apenas armazenar cada nó da trie em um objeto diferente. Apesar de isso melhorar a complexidade na teoria, na prática, cada caractere de um nome de arquivo implicaria uma nova chamada ao banco de dados (que é muito mais custosa que um acesso à RAM), tornando a abordagem impraticável. A primeira otimização é comprimir a trie: se um vértice tem apenas um filho, podemos juntá-lo com o filho. Além disso, guardamos vários nós da trie em um mesmo objeto no banco de dados, de forma que os armazenamos juntos se seu tamanho for bem pequeno (até alguns kilobytes).

Outro problema que vemos bastante em empresas, mas não na maratona, é o de migração. Como mudamos um banco de dados já existente com milhões ou bilhões desses objetos para usar o novo sistema, enquanto mantemos tudo funcionando a todo momento? Não podemos parar o servidor todo até trocarmos todos os objetos de um tipo para o outro; a mudança precisa ser incremental, transparente e reversível. É um problema técnico interessante, mas que não cabe no escopo deste artigo.

3. Ancestral de Nível

Um problema que aparecia frequentemente nos servidores do Sapling era: dado um commit e uma data, encontrar o ancestral mais antigo desse commit que tenha sido criado após a data especificada. Esse problema é muito parecido com o de ancestral de nível, no qual buscamos o k-ésimo ancestral de um dado nó em uma árvore. Note que o grafo de commits não é necessariamente uma árvore por causa das merges, porém, nós do time de controle de versionamento evitávamos usar merges[1], e usuários comuns eram proibidos de fazê-los (tudo era baseado em rebases). Dessa forma, resolvíamos os problemas em árvores, tratando os (poucos) merges de forma ineficiente, já que muitos desses problemas não têm soluções eficientes em DAGs.

A solução mais usada e conhecida para isso na Maratona é a de "binary lifting": nessa técnica, para cada nó u da árvore, armazenamos ponteiros para seus ancestrais a potências de dois: p_u^1 (seu pai), p_u^2(seu avô), p_u^4 , e assim por diante, até $p_u^{2^k}$, onde k=⌊lg $d(u)$⌋ e $d(u)$ é a profundidade do nó u. Ao olhar o código para essa função no Sapling, percebi que alguém havia tentado implementar essa técnica, mas isso não tinha sido possível, pois ela gastava memória demais: nesse caso, a memória de $\Theta(\log d(u))$, por nó é proibitiva! Na prática, a implementação guardava apenas p_u^1 e p_u^{512}, o que é bem subótimo, e isso estava começando a se tornar um problema com os repositórios crescendo cada vez mais rápido.

A solução foi, enfim, uma variação antiga, mas pouco explorada , dessa técnica. Nessa variante , cada nó u armazena apenas seu pai e, opcionalmente, um ponteiro adicional $u \to v$. Esse ponteiro adicional $u \to v$ é armazenado se, e somente se, existirem ponteiros adicionais $p_u^1 \to w$ e $w \to v$ (onde p_u^1 é o pai de u), e ambos esses ponteiros tiverem o mesmo "alcance" (ou seja, a mesma distância em termos de profundidade: $d(w) - d(p_u^1) = d(v) - d(w)$) . É fácil calcular os ponteiros adicionais em tempo e espaço $\mathcal{O}(1)$ por nó, e o código para ancestral de nível (LA), ancestral comum mais baixo (LCA) e outras variantes segue igual ao caso do binary lifting: sempre usamos o maior ponteiro possível em cada passo do algoritmo. É possível provar que tais algoritmos consomem tempo $\mathcal{O}(\log n)$, a mesma complexidade do binary lifting. Esta prova, porém, é complicada. A chave é que, embora os ponteiros não sejam potências de dois fixas, a forma como são construídos garante que sempre podemos 'saltar' grandes distâncias de forma eficiente, similar ao binary lifting, mas com menos armazenamento. Essa solução foi dada primeiro por Myers em 1983 [5], tem algumas poucas referências em blogs do Codeforces [4], e é explicada na Seção I da minha dissertação de mestrado [2].

Foi essencial entender de complexidade, teoria e prática, e a Maratona ajudou bastante nisso. Traduzir parte da minha dissertação para inglês para explicar o algoritmo para meus colegas de trabalho, e eles ficaram muito entusiasmados; a redução de espaço para $\mathcal{O}(1)$ realmente tornou a solução viável. Na Maratona, em alguns problemas o espaço pode ser muito restrito e demandar uma solução desse tipo, por exemplo no problema B da NCPC 2016 [1]. Além disso, o código é tão fácil de ser implementado quanto o de binary lifting, apesar de sua eficiência não ser tão intuitiva.

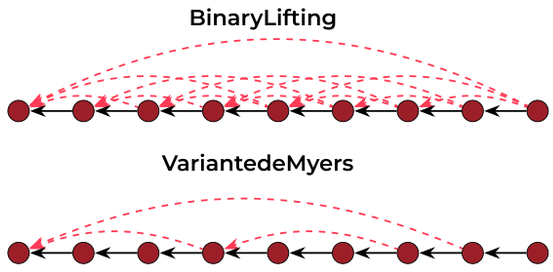


Figura 2: Acima, o Binary Lifting, onde cada nó armazena ponteiros para ancestrais em potências de dois. Abaixo, a variante de Myers, que armazena apenas o pai e um ponteiro adicional, resultando em menor consumo de memória.

4. Conclusão

Trabalhar nesses problemas foi uma das partes mais interessantes e divertidas do meu trabalho. Mas nem tudo é um mar de rosas: a maior parte do trabalho é difícil de outras formas e não inclui tais algoritmos ou estruturas de dados interessantes. Tive que me esforçar bastante para entrar em um dos poucos times que lidavam com esse tipo de problema e, mesmo neste time, na maior parte do tempo, eu não estava criando ou lidando com estruturas interessantes. Quando ingressei, eu não realizava entrevistas técnicas, pois achava que eram uma versão mais chata dos problemas de Maratona. Depois de alguns anos sem ver nem resolver muitos problemas, comecei a fazer entrevistas, pois era a atividade mais próxima disso que eu tinha. Por fim, também existe a organização da Hacker Cup, na qual é possível criar problemas de Maratona. Ajudei um pouco em sua organização, mas me arrependo de não ter ajudado mais, nem criado mais problemas para estas provas.

Referências

[1] NCPC 2016. Bless you autocorrect! https://codeforces.com/gym/101550. Data de acesso: 03/06/2025.
[2] Yan Soares Couto. Estruturas de dados persistentes. PhD thesis, Universidade de São Paulo, 2018. https://yancouto.github.io/mestrado/thesis.pdf.
[3] Facebook. Sapling SCM. https://github.com/facebook/sapling, 2016.
[4] Alex Luchianov. [tutorial] binary lifting. https://codeforces.com/blog/entry/100826. Data de acesso: 03/06/2025.
[5] Eugene W Myers. An applicative random-access stack. Information processing letters, 17(5):241–248, 1983.

[1] Você deveria evitá-los em seus repositórios também!

PRIMEIRA MARATONA DE PROGRAMAÇÃO

I Consun, Belo Horizonte

PRIMEIRO TIME BRASILEIRO EM UMA FINAL MUNDIAL DO ICPC

-1996-

A primeira edição da Maratona de Programação aconteceu em 1996, dentro do I CONSUN (Concurso Nacional de Software Universitário), realizado em Belo Horizonte e organizado pelos professores Claudionor Coelho e Carlos Camarão, com apoio da SOFTEX e do CNPq. Participaram 11 times, um por universidade, sem que ninguém tivesse muita clareza sobre como funcionaria a competição. Inspirada no ICPC, a dinâmica era bastante artesanal: as submissões dos códigos eram feitas em disquetes, levados por monitores até os juízes, que avaliavam manualmente e anotavam no próprio disquete os resultados como “Accepted”, “Wrong Answer” ou “Time Limit Exceeded”. O placar era mantido à mão, e os balões, hoje um símbolo da competição, precisaram ser explicados aos participantes. A disputa ocorreu no laboratório do Instituto de Ciências Exatas da UFMG, onde os competidores discutiam as soluções em mesas ao fundo e precisavam aguardar para usar os poucos computadores disponíveis. Apesar das limitações, tudo correu bem e o evento foi um sucesso. O time da Unicamp sagrou-se campeão e foi convidado a participar da final mundial do ICPC no ano seguinte, consolidando o início de uma trajetória que despertou entusiasmo e já gerou expectativas para as edições seguintes.

IC UNICAMP: ALEXANDRE OLIVA, ALEXANDRE VOLPIM E ANDRÉ AUGUSTO CESTA, TÉCNICO RICARDO ANIDO

PARA MAIS DETALHES, VISITE O SITE:
HTTPS://MARATONA.SBC.ORG.BR/HIST/1996/INDEX.HTML

PROJETO POSNIAK

Diego Pedro, IFAM Parintins

Considerando o potencial dos alunos do IFAM Campus Parintins e a ausência de iniciativas voltadas ao treinamento em programação competitiva, foi criado o projeto POSNIAK. Seu objetivo é formar um grupo sólido de competidores em programação, abrangendo desde o ensino fundamental até o nível superior, principalmente. A proposta inclui um programa estruturado de capacitação em programação competitiva, além da organização e participação em torneios.

O POSNIAK foi inspirado no projeto olímpico da UFCG, cujo objetivo é formar talentos em programação competitiva desde cedo [4]. Assim como no projeto olímpico, acreditamos que, para formar programadores de alto desempenho em competições de programação, é essencial iniciar a preparação ainda no ensino básico.

Após o primeiro ano de atividades, diversas conquistas foram alcançadas, como a primeira classificação do IFAM, em toda a sua história, para a final da Maratona de Programação, além de resultados expressivos na Olimpíada Brasileira de Informática (OBI).

1. Contexto Regional

A cidade de Parintins tem 96.372 habitantes e está localizada a 369 quilômetros da capital, Manaus [2]. A cidade faz divisa com o estado do Pará, estado no qual o campus do IFAM compete na primeira fase da Maratona. Parintins é conhecida internacionalmente pelo Festival Folclórico de Parintins.

A cidade apresenta um dos mais baixos pib per capita do país, ocupando apenas a 28ª posição no estado, o que contribui para uma baixa taxa de ocupação, de apenas 9,8% [2], em comparação com 30,98% da capital Manaus. No nível educacional, Parintins apresenta uma baixa taxa de escolarização, uma das piores do Brasil, embora tenha um índice IDEB próximo à média nacional. Com o objetivo de avaliar o desempenho da cidade de Parintins em matemática em comparação com o restante do país, realizamos uma análise de dados da OBMEP referentes aos medalhistas entre os anos de 2019 e 2024. Os resultados revelam que Parintins possui uma elevada taxa de medalhas da OBMEP por 100 mil habitantes(ano base 2022 [2]), alcançando 26,9. Esse número supera importantes cidades de referência em matemática no ensino superior como Rio de Janeiro-RJ (17,1), São Paulo-SP (8,5), Fortaleza-CE (18,3), São Carlos-SP (25,9),Campinas-SP (19,6) e Belo Horizonte-MG (24,5). Os dados utilizados nesta análise estão disponíveis para consulta pública neste link https://tinyurl.com/posniak25R.

De acordo com [1], 83% dos estudantes que conquistaram medalhas na OBI em 2019 também conquistaram medalhas em competições de matemática. Isso sugere um bom ambiente para desenvolver um projeto de treinamento para competições de programação na cidade.

O público discente do IFAM é composto pela maioria de alunos de baixa renda. Isso é um fator dificultador para o projeto, uma vez que alunos talentosos frequentemente desistem de participar do projeto POSNIAK, que não oferece bolsa, para participar de outros projetos como bolsista. O fato de o campus ser localizado na zona rural da cidade dificulta um pouco a realização do projeto nos finais de semana. Outro desafio para o projeto é o fato de mais de 90% dos alunos do nível integrado não terem computadores em casa.

2. Público-alvo

O projeto é destinado a alunos do ensino fundamental ao ensino superior da cidade de Parintins e de municípios vizinhos. Após o primeiro módulo, a maioria dos participantes remanescentes pertence aos cursos técnicos integrados em Informática e ao bacharelado em Engenharia de Software do IFAM. O curso técnico em Informática é ofertado desde a fundação do campus, em 2010, enquanto o curso de Engenharia de Software iniciou suas atividades em 2023.

3. Primeiros anos

O projeto foi iniciado oficialmente em março de 2024. De forma modesta, alunos de várias turmas foram convidados a participar. Houve uma grande aceitação entre os alunos de graduação, mas poucos alunos do nível integrado em informática. A implantação do projeto foi simples, mas o maior desafio foi motivar os alunos quanto à importância da iniciativa. Durante aquele ano, ocorreu uma greve que, de certa forma, facilitou a dedicação dos alunos, pois a carga horária de estudos do nível integrado nos institutos federais é muito alta.

No primeiro ano, apenas 12 alunos concluíram o curso, sendo que 3 deles optaram por não participar da OBI. No ano seguinte, muitos alunos afirmaram ter se arrependido de não terem participado da primeira edição e decidiram ingressar na turma de 2025.

Em 2025, o projeto foi aberto à comunidade externa, o que contribuiu para um número recorde de 262 inscrições, um aumento de 376% em relação ao ano anterior. O projeto recebeu inscrições de alunos do ensino fundamental ao ensino superior, provenientes de diversas instituições de ensino de Parintins e de cidades vizinhas.

FIGURA 1: TURMA POSNIAK 2024

4. Metodologia

O programa de treinamento do POSNIAK consiste em aulas que vão do nível iniciante ao avançado que são divididos em 3 módulos: iniciante, intermediário e avançado. Cada módulo possui uma carga horária de 40 horas e duração de aproximadamente 9 semanas. Os encontros são realizados semanalmente, com duração de duas horas cada.

Após cada aula, os alunos têm uma semana para resolver uma lista de problemas. Para otimizar os recursos físicos e humanos, estudantes do ensino fundamental e do ensino superior participam das mesmas aulas e enfrentam os mesmos desafios. Além disso, são realizados simulados de forma regular.

Todo o projeto é ministrado na linguagem Python. A escolha por Python, em vez de C, se deu por sua curva de aprendizagem mais rápida e maior facilidade de ensino, especialmente para crianças [5]. Ao final de cada módulo, é realizada uma reunião individual para definir quem está apto a avançar para o próximo nível. Os alunos que conseguirem resolver pelo menos 2 problemas em um intervalo de 5 horas, são promovidos para o próximo nível.

Abordagem das aulas

As aulas ministradas diferem do ensino regular oferecido nas escolas técnicas. O foco é dado a assuntos relevantes para a programação competitiva, deixando que os alunos estudem por conta própria os demais conteúdos. Outro ponto importante é que o conteúdo é ministrado em um ritmo mais acelerado que o do curso regular.

Os problemas propostos ao longo do curso são voltados para o estilo de competições. Curiosamente, observa-se que alunos com conhecimento prévio em programação tendem a enfrentar mais dificuldades no início do curso do que aqueles sem experiência anterior. Isso se deve, em grande parte, a vícios adquiridos anteriormente e a um aprendizado baseado na memorização de sintaxe, em vez do desenvolvimento do raciocínio lógico.

Monitoria

Além dos professores, os alunos recebem tutoria de monitores, que são aqueles que apresentaram os melhores desempenhos na edição anterior. Os monitores auxiliam os alunos, sugerem os problemas semanais e criam os problemas do nível iniciante.

Ementa

O projeto parte do pressuposto de que todos os alunos iniciam sem conhecimento prévio em programação. Por isso, o módulo iniciante começa do zero, abordando conceitos fundamentais como variáveis, entrada e saída de dados. Em seguida, são introduzidas estruturas condicionais e de repetição, finalizando com o estudo de vetores e matrizes. O conteúdo desse módulo corresponde ao ministrado no primeiro ano do curso técnico integrado em Informática, condensado em nove semanas, com foco exclusivo nos tópicos relevantes para competições de programação. O objetivo desse módulo é apresentar a programação de computadores na resolução de problemas.

No nível intermediário, os alunos iniciam o estudo de funções, recursividade e posteriormente em análise assintótica. Em seguida, avançam para estruturas de dados e algoritmos de busca em grafos. Esse módulo abrange todo o conteúdo ministrado nas disciplinas regulares no curso de Engenharia de Software do campus. Esse módulo tem como objetivo incentivar o aluno otimizar o código a partir do uso de estruturas de dados e refinamento do código.

No nível avançado, os estudos aprofundam-se em grafos e seguem com programação dinâmica, algoritmos gulosos e geometria computacional. A autonomia de estudo é objetivo principal desse módulo. A ideia central é passar o conteúdo básico de forma que incentive o aluno a se aprofundar nos assuntos a fim de resolver os problemas.

Ao longo do curso, os alunos também participam de aulas de matemática, que abrangem temas como indução matemática, análise combinatória, teoria dos números, geometria analítica, bem como geometria plana e espacial. Essas aulas são ministradas por um professor com formação em Matemática e experiência na OBMEP.

Espera-se que todos os alunos que atingem o nível avançado tornem-se independentes na busca por conhecimento e sigam se aperfeiçoando de forma autônoma.

Premiações

Como forma de incentivo, o projeto oferece premiações aos participantes. Ao final de cada módulo, são realizadas competições internas nas quais são distribuídas medalhas e um troféu de destaque. Ao término do programa, os alunos também recebem insígnias correspondentes às habilidades desenvolvidas ao longo do curso, como algoritmos, matemática, digitação e dedicação ao projeto.

5. Desempenho e evasão

O índice de evasão no projeto é bastante elevado. Do total de alunos que iniciaram o primeiro ano, apenas 16% alcançaram o nível intermediário. Desses, menos da metade chegou ao nível avançado.

No ano de 2025, dos 262 inscritos, apenas 30 concluíram o nível iniciante. Destes 30, 17 desistiram do curso antes mesmo de ele ser finalizados. Para avaliar os principais motivos, realizamos uma pesquisa com esses 17 alunos. O principal motivo para a desistência do projeto foi a falta de tempo para se dedicar ao projeto (76%), os 24% restantes foram devido a conflitos de horários com o trabalho, dificuldade de acompanhar o curso. Vários alunos talentosos do campus que participavam do projeto, desistiram para se dedicarem à projetos institucionais e de pesquisa que oferecem bolsa. Infelizmente, o fator financeiro ainda tem um peso muito grande para a manutenção do projeto, uma vez que o projeto demanda bastante tempo dos alunos, e os mesmos não recebem nenhum tipo de bolsa.

Embora o campus conte com alunos talentosos, uma parcela considerável do campus apresenta déficits educacionais do ensino fundamental. Segundo dados do IDEB de 2023, o aprendizado adequado em língua portuguesa dos alunos da 9º de parintins é de apenas 19%, já em matemática é ainda menor, 7% [3]. Portanto, o projeto consegue atingir apenas uma pequena parcela dos alunos de forma eficiente.

FIGURA 2: 1º DIA DE AULA DO POSNIAK 2025

6. Resultados

O projeto obteve resultados expressivos no ano de 2024 na OBI e na maratona de Programação da SBC. Na OBI uma aluna ficou na 15º colocação na OBI feminina do nível P1. Outros dois alunos do P1 obtiveram as colocações 75º e 100º conquistando as primeiras colocações no estado do Amazonas. Na OBI nível sênior, dois alunos tiveram excelentes colocações em 59º e 106º no Brasil, conquistando o melhor resultado da história do IFAM. Por fim, o campus Parintins conseguiu a primeira classificação para a final da maratona de programação da SBC na história, considerando que há mais de 20 anos há cursos de graduação em TI no IFAM. Além disso, obtiveram a segunda colocação na região norte do País, ficando atrás apenas da UFAM, que foi medalha de Ouro.

Os primeiros resultados de 2025 mostraram um desempenho ainda melhor com 18 alunos classificados para a segunda fase da OBI. Um recorde histórico do IFAM entre todos os campus.

7. Considerações finais

Acreditamos que iniciar o treinamento em programação desde os primeiros anos é fundamental para formar competidores altamente capacitados, preparados para enfrentar desafios já no nível de graduação. Mesmo diante de condições desfavoráveis, é possível criar um ambiente estimulante de estudo e competição, que motive os alunos a se dedicarem, aprimorarem suas habilidades e alcançarem conquistas significativas.

Referências

[1] Giullia Rodrigues de Menezes, João Henrique de Souza Pereira, and Luiz Cláudio Theodoro. Análise do perfil dos medalhistas da olimpíada brasileira de informática 2019, 2021.

[2] Instituto Brasileiro de Geografia e Estatística. Censo demográfico 2022: Resultados preliminares, 2022. Rio de Janeiro: IBGE.

[3] Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira – INEP. Resultados do IDEB. https://www.gov.br/inep/pt-br/areas-de-atuacao/pesquisas-estatisticas-e-indicadores/ideb/resultados, n.d. Data de acesso: 02/08/2025.

[4] Universidade Federal de Campina Grande – UFCG. Projeto olímpico. https://memoria.computacao.ufcg.edu.br/projeto-ol Data de acesso: 02/08/2025.

[5] Jacques Wainer and Eduardo C Xavier. A controlled experiment on python vs c for an introductory programming course: Students' outcomes. ACM Transactions on Computing Education (TOCE), 18(3):1–16, 2018.

FIGURA 3: ÚLTIMO TORNEIO POSNIAK 2025

FINAL BRASILEIRA 2024

VISITE A MATÉRIA NA ÍNTEGRA NO SITE:
HTTPS://MARATONA.SBC.ORG.BR/HIST/2024

A final brasileira de 2024 foi realizada em João Pessoa, Paraíba, onde o sol nasce primeiro, com realização do IFPB: Instituto Federal de Educação da Paraíba. A Profa. Valéria Cavalcanti e o Prof. Tiago Gouveia foram os diretores da final. Tivemos o patrocínio premium do Rei do Pitaco, patrocínio diamante da Vtex e Incognia, patrocínio prata da Inovatec-JP, e patrocínio bronze da Alura e Google.

UMA JORNADA POR TRÁS DOS PATROCÍNIOS

Lucy Mari Tabuti, Instituto Criativo

Relato da trajetória de mais de uma década na Maratona SBC de Programação, com foco na atuação como Diretora de Patrocínios. O texto apresenta os desafios da captação de recursos, o impacto da competição na formação de talentos e a importância do propósito. Mais que uma função, uma missão de vida.

1. Introdução: Uma Jornada que Começou com Propósito

Em 2013, a convite de meus próprios estudantes, comecei a acompanhar equipes interessadas em participar da Maratona SBC de Programação [1]. A princípio, atuei como coach, sugerindo estudos e preparos para competições científicas [2]. E, foi o suficiente para me apaixonar pelo evento. Mais do que a competição em si, o que me cativou profundamente foi o propósito das pessoas envolvidas: a dedicação genuína à formação acadêmica, ao espírito de equipe e ao crescimento profissional dos participantes.

A Maratona logo se tornou mais do que um evento anual no calendário: passou a representar um espaço de transformação humana e profissional. Cada etapa trazia aprendizados únicos, desafios concretos e, principalmente, um senso de comunidade raro de encontrar em outras iniciativas.

Este artigo é um relato pessoal de uma trajetória construída com entusiasmo e resiliência ao longo de mais de uma década. A partir da vivência como coach, diretora de sede e, principalmente, como Diretora de Patrocínios da Maratona de Programação, compartilho os bastidores dessa jornada repleta de desafios e realizações, movida por algo que vai além de qualquer retorno financeiro: o propósito.

2. Do Papel de Coach à Diretora da Sede São Paulo

O envolvimento com a Maratona de Programação cresceu rapidamente. No ano seguinte à minha primeira participação como coach, em 2014, atuei como diretora da sede da cidade de São Paulo. Foi uma experiência transformadora, que me proporcionou uma nova visão sobre os bastidores da competição. Organizar uma sede exigia muito mais do que conhecimento técnico: envolvia logística, acolhimento dos times, gestão de equipe e, sobretudo, a responsabilidade de garantir uma experiência memorável para todos os participantes.

Um dos maiores desafios daquela primeira organização foi levantar recursos financeiros para cobrir os custos do evento. Iniciei, então, os primeiros contatos com empresas, buscando apoio e patrocínio. Esse processo me mostrou que, para além da competição, havia um enorme potencial de conexão com o mercado, que poderia ser ativado por meio de parcerias bem estruturadas. Foi ali que percebi que havia encontrado um novo propósito dentro da Maratona: contribuir para sua sustentabilidade por meio da articulação com o setor privado.

Em 2015, esse propósito se intensificou com a oportunidade de organizar a Final Brasileira da Maratona de Programação, também em São Paulo. Organizar um evento nacional trouxe uma complexidade ainda maior, pois exigia articulação com diferentes stakeholders, criatividade para negociar parcerias e atenção a cada detalhe. Muitas das parcerias estabelecidas foram feitas por meio de permutas, numa verdadeira engenharia de colaboração. Desde então, segui à frente da organização da sede São Paulo ano após ano, fortalecendo minha atuação e meu vínculo com essa comunidade tão especial.

FIGURA 1: FINAL BRASILEIRA EM 2015 (ONDE ESTOU?)

3. Desafios da Final Brasileira e a Força das Parcerias

Assumir a organização da Final Brasileira em 2015 foi um divisor de águas. O evento exigia um nível de coordenação elevado, envolvendo múltiplas frentes ao mesmo tempo: logística nacional, hospedagem de times de todas as regiões do país, alimentação, equipamentos, espaços adequados e, principalmente, o cuidado com a experiência de cada participante. Era um compromisso com excelência, muito além de apenas técnica, mas também humana.

Nesse contexto, as parcerias com empresas se mostraram fundamentais. Muitos acordos foram viabilizados por meio de permutas, trocas de serviços, apoio institucional e, em alguns casos, investimento financeiro direto. Cada negociação era única, personalizada e dependia de muito diálogo, confiança e alinhamento de valores. Os patrocinadores que abraçavam o projeto compreendiam que a Maratona era muito mais do que uma competição: era uma vitrine de talentos e um espaço de desenvolvimento de futuros líderes da tecnologia no país.

A partir desse momento, minha atuação ganhou um novo contorno: mais do que organizar eventos locais, comecei a contribuir para a estratégia de sustentação da Maratona em nível nacional. Fui então convidada pelos professores Carlos Eduardo Ferreira e Rodolfo Azevedo para assumir um dos maiores desafios da história da competição: liderar a Diretoria de Patrocínios da Maratona SBC de Programação. Era o início de uma jornada de alcance nacional e de complexidade ainda maior. Inicialmente, atrás de todos, escondida, tímida, como na Figura 1, junto com os balões coloridos.

4. A Missão Nacional de Captação de Patrocínios

Assumir a Diretoria de Patrocínios da Maratona de Programação foi, sem dúvida, um dos maiores desafios da minha trajetória. À primeira vista, muitos acreditam que o maior obstáculo está em encontrar empresas interessadas em apoiar o evento. No entanto, com o tempo, ficou claro que a verdadeira dificuldade é encontrar as pessoas certas dentro das empresas, aquelas que enxergam o valor da Maratona e reconhecem o potencial humano dos seus participantes.

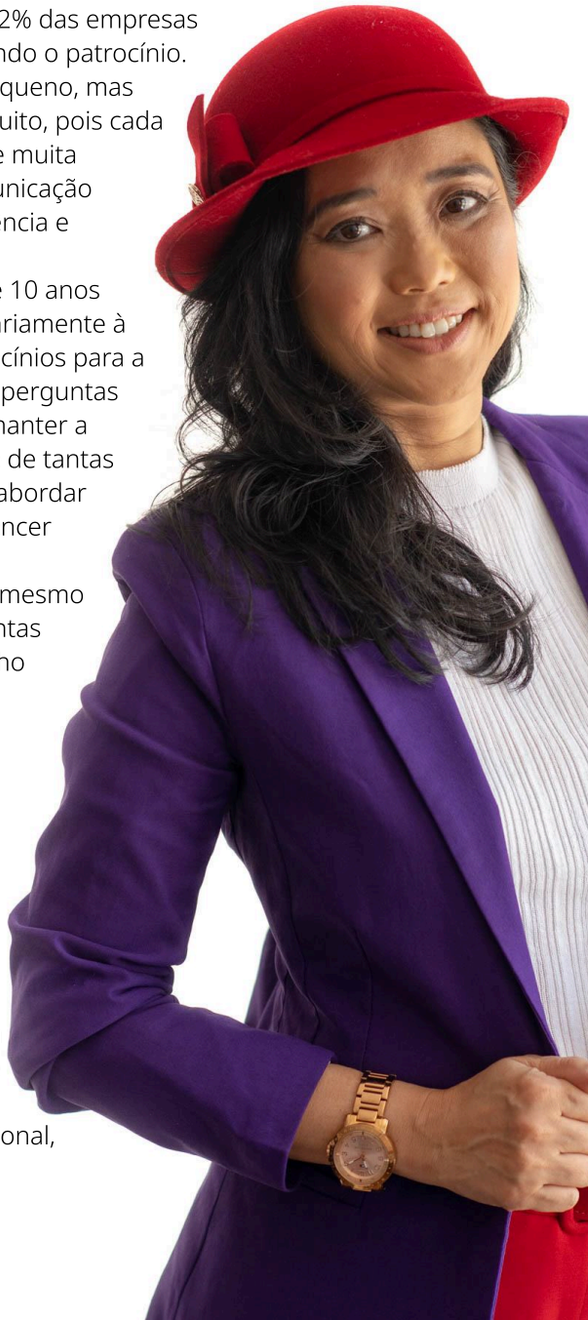
Estamos falando dos melhores estudantes de Computação do Brasil, vindos de universidades públicas e privadas de excelência. São jovens com habilidades técnicas avançadas, somadas às competências interpessoais fundamentais, como trabalho em equipe, pensamento crítico, criatividade, inteligência emocional, comunicação, liderança e resiliência, fundamentais aos profissionais [3]. Convencer uma empresa a patrocinar a Maratona significa, na prática, mostrar que esses estudantes são os profissionais ideais que toda organização sonha em contratar.

Essa missão exige encontrar empresas com afinidade com tecnologia, o que, nos dias de hoje, é praticamente qualquer setor e, dentro delas, localizar profissionais das áreas de RH, universidade corporativa, marketing ou tecnologia que estejam em busca de talentos. A sinergia entre essas áreas é fundamental, pois o patrocínio à Maratona além de ser um investimento em visibilidade, também é uma estratégia de atração de talentos altamente qualificados.

5. Os Bastidores da Captação: Pessoas, Negociação e Resiliência

Uma vez identificada a empresa e a pessoa certa dentro dela, começa uma nova etapa: o contato inicial. O primeiro passo é o envio de um documento explicando o que é a Maratona, quais são seus objetivos, como funciona a competição e por que apoiar esse projeto é uma decisão estratégica. Mas o processo é desafiador: cerca de 90% dos contatos feitos sequer recebem resposta. Dos 10% que respondem, metade topa uma primeira conversa, ou seja, restam apenas 5% das empresas inicialmente abordadas.

As reuniões são momentos decisivos. Nelas, explica-se com detalhes o funcionamento da Maratona e apresenta-se as cotas de patrocínio, com diferentes formatos e benefícios, muitas vezes, personalizados. Mas, mesmo após uma reunião promissora, o retorno costuma demorar. A espera por uma decisão é longa e nem sempre os acordos fechados são nas cotas mais altas. No fim de toda essa jornada, cerca de 2% das empresas acabam confirmando o patrocínio. Um percentual pequeno, mas que representa muito, pois cada sim é resultado de muita escuta ativa, comunicação empática, persistência e resiliência.



Durante mais de 10 anos dedicados voluntariamente à captação de patrocínios para a Maratona, muitas perguntas surgiram. Como manter a autoestima diante de tantas negativas? Como abordar empresas e convencer sobre o valor da competição? Vale mesmo a pena dedicar tantas horas a um trabalho sem retorno financeiro? Essas perguntas nunca desapareceram completamente, mas a resposta, com o tempo, se tornou clara: é preciso viver a Maratona como um propósito. Um compromisso que exige resiliência, inteligência emocional,

comunicação empática e, acima de tudo, amor-próprio.

6. Reflexões de uma Década: Perguntas que repercutem

Ao longo dessa trajetória, marcada por inúmeras mensagens sem respostas, reuniões adiadas, negociações delicadas e respostas negativas, surgiram questionamentos inevitáveis. Como manter a motivação? Por que continuar diante de tantas dificuldades? O que se ganha com tudo isso? A verdade é que as respostas estão longe de serem simples ou definitivas... Algumas certezas foram se construindo pelo caminho.

O que sustenta esse trabalho voluntário, tão exigente quanto invisível, está longe de ser o reconhecimento imediato, tampouco qualquer tipo de retorno financeiro. O que realmente move essa dedicação é o sentimento de que cada esforço contribui diretamente para o desenvolvimento de centenas de jovens talentos. É a sensação de fazer parte de algo maior, de um ecossistema que transforma vidas e abre portas para quem sonha em viver da tecnologia, da lógica, da inovação.

Muito mais do que ocupar o cargo de Diretora de Patrocínios da Maratona de Programação, essa jornada me ofereceu uma oportunidade única: a de me reconectar com minha própria essência. A Maratona me convidou a reviver, a renascer, a reencontrar meu propósito profissional e, sobretudo, pessoal. O que torna essa experiência ainda mais especial são as pessoas que fazem parte dela. Uma rede de colaboradores, colegas, parceiros e amigos que compartilham o mesmo sonho e caminham lado a lado, com ética, generosidade e paixão.

7. Conclusão: A Maratona de Programação como Missão de Vida

A Maratona de Programação ultrapassa os limites de uma competição acadêmica. É, antes de tudo, um espaço de formação humana. Para quem participa, organiza ou apoia, ela representa desafios técnicos, sim, mas também desafios emocionais, éticos e coletivos. É uma escola de resiliência, um laboratório de competências, um ponto de encontro entre sonhos e oportunidades.

No meu caso, foi muito mais do que isso. Foi uma jornada que me transformou, que me fez crescer e me permitiu encontrar sentido no que faço. A Maratona me conectou a pessoas incríveis, que caminham com o mesmo brilho nos olhos e o mesmo compromisso com um propósito maior. A cada edição, a cada sede organizada, a cada patrocínio conquistado com esforço e diálogo, eu renascia um pouco mais. E, agora com seu chapéu vermelho, como na Figura 2, à frente, com autoconfiança, buscando inspirar e, claro, ainda com os balões coloridos! A marca da Maratona de Programação!

Mais do que um cargo, ser Diretora de Patrocínios tornou-se uma missão. E, como toda missão verdadeira, ela é movida por amor. Amor pelo que se constrói coletivamente, pela educação, pela computação e, principalmente, pelas pessoas. O maior prêmio da Maratona está acima do pódio, está nas conexões que ela cria, nos talentos que ela revela e na certeza de que, no final de cada etapa, sempre valeu a pena!

FIGURA 2: FINAL BRASILEIRA EM 2022 (ONDE ESTOU?)

Referências

[1] MDP, Maratona SBC de Programação.

[2] A. Laaksonen, Guide to competitive programming: Learning and improving algorithms through contests, Cham: Springer, 2022.

[3] A. S. Sosnowski, Soft Skills para a Vida, a Carreira e os Negócios: O Pulo do Gato para Desenvolver Habilidades Empreendedoras, Rio de Janeiro: Alta Life, 2024.

DA LANCHA AO PÓDIO: A SAGA DO IFAM NA MARATONA DE PROGRAMAÇÃO

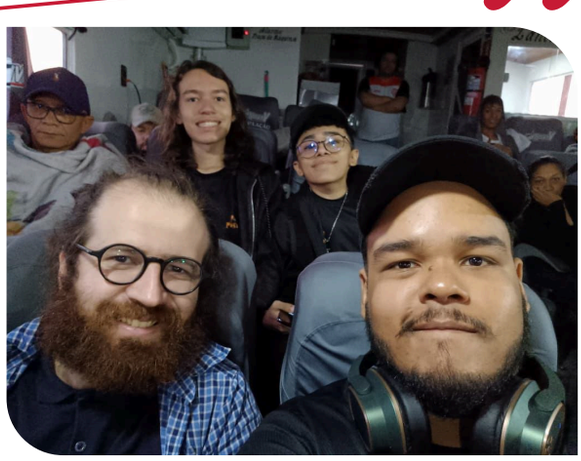
HISTÓRIA DE SUPERAÇÃO E INOVAÇÃO

A equipe Dois Programadores e Meio, do campus Parintins, foi a primeira da história a levar o nome do IFAM a uma final brasileira. Formada pelos alunos Lucas Ferreira, Matheus Nobre e Robert Cruz, sob a orientação do técnico Diego Silva, o grupo protagonizou uma das jornadas mais extensas — possivelmente a maior — até uma decisão nacional. Outro marco importante foi ser o primeiro time a contar com uma pessoa com deficiência física na fase final.

A jornada começou ainda na madrugada do dia 06 de novembro de 2024, às 5h (horário de Manaus). Foram 10 horas a bordo de uma lancha a jato, o meio de transporte fluvial mais veloz da região, cruzando o rio Amazonas, que conecta Parintins à capital do estado. Depois, uma longa espera em Manaus, seguida de um itinerário aéreo que atravessou o país: Manaus → Porto Velho → Brasília → João Pessoa. Somando todas as etapas, a viagem teve duração de 30 horas, concluídas apenas ao meio-dia do dia seguinte, já em horário de Brasília.

Na abertura do evento, Lucas Ferreira, que é portador de escoliose congênita, compartilhou a vivência de ser o primeiro finalista com deficiência física na decisão da maratona de programação e comentou sobre os desafios enfrentados ao longo da trajetória:

> “As pessoas estão acostumadas a ver alguém como eu e pensar que estou conseguindo acompanhar os demais, e logo tudo está certo. Porém, não é assim! Eu apenas aprendi a lidar melhor com a deficiência nesses meus 22 anos de existência. Tive uma formação escolar extremamente frágil por faltar a inúmeras aulas para realizar cirurgias, sofrer bullying com frequência, sentir muitas dores, entre outras situações que passam despercebidas à primeira vista. Sem dúvidas, ter participado da fase nacional da Maratona SBC de Programação foi uma das experiências mais incríveis que já vivi, e certamente essas pessoas são algumas das mais marcantes que já conheci.”



A equipe, que obteve a vaga para a final após conquistar o 1º lugar na sede de Oriximiná-PA, resolveu 2 problemas na final, conquistando o 58º lugar da competição e 2º lugar dentre os finalistas da Região Norte do país. Tal desempenho não seria possível sem o poder da amizade dos integrantes e apoio do professor Diego Silva, que através do Projeto Posniak e seus coffee-breaks em dias de simulado, despertou o interesse pela programação competitiva nos estudantes.

MULHERES NA MARATONA DE PROGRAMAÇÃO

Crishna Irion, UFU
Camila Cruz dos Santos, UFU

Este artigo é uma reflexão sobre a participação feminina na Programação Competitiva, também transitando pela representatividade nos cursos de Ciência da Computação e na sociedade.

1. Introdução

As mulheres enfrentam uma série de barreiras estruturais e culturais ao ingressarem na área de Computação. Um dos principais obstáculos é a persistência de estereótipos de gênero, que associam erroneamente a tecnologia a um universo predominantemente masculino. Essa percepção pode desestimular o interesse feminino e comprometer a autoconfiança desde os primeiros contatos com a área. Além disso, a escassez de modelos femininos em posições de destaque contribui para a dificuldade das novas gerações em se identificarem com a profissão, dificultando o sentimento de pertencimento. Outro fator relevante é a presença de ambientes de trabalho muitas vezes hostis, marcados por práticas discriminatórias, assédio e uma cultura organizacional pouco inclusiva, o que impacta negativamente a permanência e o desenvolvimento profissional das mulheres. No contexto educacional, observa-se frequentemente uma ausência de estímulo adequado à participação feminina em disciplinas de STEM (Ciência, Tecnologia, Engenharia e Matemática), o que agrava as desigualdades de acesso e formação. Além disso, o apoio familiar e social nem sempre é suficiente, o que pode gerar insegurança e hesitação na escolha de carreiras tecnológicas. As implicações relacionadas ao desequilíbrio entre vida profissional e responsabilidades pessoais também exercem influência significativa, uma vez que normas de gênero ainda vigentes impõem expectativas desiguais sobre o papel da mulher na família.

Apesar da importância crescente dessas competições de programação no cenário global e do interesse dos estudantes, a participação feminina tem permanecido significativamente baixa e notavelmente distante em comparação com a masculina [2]. Estatísticas da Sociedade Brasileira de Computação (SBC) indicam que o número de mulheres participantes das competições estava abaixo de 10%. Enquanto as mulheres representam entre 13,8% e 15,2% dos alunos matriculados e formados nos cursos de TI, sua participação nas Maratonas de Programação não ultrapassava 8% entre 2011 e 2021 [4,3]. Essa disparidade é influenciada por diversos fatores de gênero, sociais e culturais, e as competições, em sua essência, nem sempre são pensadas para a participação feminina [1]. Diante desse cenário, a promoção da equidade de gênero e a busca por estratégias para aumentar a participação feminina nas Maratonas de Programação no Brasil tornaram-se objetivos cruciais.

2. Estratégias e Ações Afirmativas para a Inclusão Feminina

Para enfrentar essa disparidade e promover a equidade de gênero, diversas ações afirmativas e estratégias têm sido desenvolvidas pelo comitê da Maratona de Programação da SBC e outras instituições. Essas iniciativas visam aumentar o número de participantes e também criar um ambiente mais acolhedor e inclusivo para mulheres e pessoas não-binárias.

3. Vagas Afirmativas na Maratona SBC de Programação

O comitê organizador da Maratona de Programação adotou uma medida significativa na edição de 2021 da Final Nacional, aprovando uma vaga exclusiva para a melhor equipe de meninas do Brasil para a participação na final de 2022. Como resultado, uma equipe feminina alcançou o 36º lugar entre as 62 equipes participantes na final nacional e 618 equipes na primeira fase. Em 2023, como continuidade dessas ações, foram aprovadas políticas de vagas extras para meninas e equipes com meninas. Isso incluiu a criação de mais 5 vagas na final brasileira para times com participação feminina. Além da vaga inicial proposta, foram adicionadas mais duas vagas: uma para equipes com pelo menos duas meninas e outra para equipes com pelo menos uma menina, que não se classificariam organicamente, mas que obtiveram bom desempenho nessa nova categoria de vagas. Essas medidas visam elevar a participação feminina de menos 10% para cerca de 20% dos competidores. Em 2022, a final nacional contou com apenas 9 competidoras. Com a implementação das ações afirmativas, incluindo 14 meninas, o número total de participantes femininas subiu para 24, demonstrando um movimento de crescimento encorajador em um cenário que permaneceu estático por mais de duas décadas.

4. Maratona Feminina de Programação (MFP)

Inspirada pela vaga afirmativa, a primeira Maratona Feminina de Programação (MFP) foi realizada em 24 de junho de 2023, com o objetivo de encorajar a participação feminina (e não binária) nas competições de programação. A MFP oferece aulas on-line preparatórias. Desde sua criação, a MFP tem crescido significativamente, iniciando em 2023 com 97 participantes e expandindo para cerca de mil inscritas na primeira fase e 130 finalistas em 2024 e 2025. O evento não é apenas uma competição, mas também um espaço para a construção de uma rede de apoio e aprendizado, fortalecendo a representatividade feminina na área tecnológica.

5. Competição Feminina da OBI (CF-OBI)

Criada em 2023 como uma extensão da Olimpíada Brasileira de Informática (OBI), a CF-OBI é uma competição de programação voltada para estudantes do ensino fundamental e médio que se identificam como femininas ou de outro gênero.

A competição oferece três níveis (Programação Júnior, Nível 1 e Nível 2), permitindo a participação de estudantes com diferentes níveis de experiência. A CF-OBI apresentou um crescimento significativo em termos de inscrições e participação efetiva entre os anos de 2023 e 2024. Em 2023, foram 672 inscrições no Nível 1, 694 no Nível Júnior e 1260 no Nível 2. Em 2024, esses números aumentaram para 953, 925 e 1997, respectivamente, representando um crescimento de 58,4% no Nível 2 e 41,8% no Nível 1. A participação efetiva também mostrou uma evolução expressiva, com um aumento de 105,5% no Nível 2 em 2024 em comparação com 2023, o que indica o sucesso das estratégias de retenção e motivação.

6. Encontros de Mulheres na Maratona

Em 2023, na final nacional da Maratona de Programação, foi realizado o primeiro encontro de mulheres na maratona, onde todas as participantes foram convidadas para debater temas como motivação, acolhimento e incentivo à participação feminina. Esses encontros presenciais e online continuaram em 2023 e 2024, levando à criação do grupo "Meninas da Maratona", que serve como um espaço para troca de experiências, oportunidades e informações. Tais ações são fundamentais para construir uma comunidade de apoio e fortalecer a presença feminina na programação competitiva.

FIGURA 1: ENCONTRO DE MENINAS DA MARATONA - 2024

7. Programas de Treinamento e Educação

Iniciativas de treinamento em programação competitiva são cruciais para preparar estudantes e aumentar seu desempenho. Esses treinamentos podem ser oferecidos de forma presencial e remota, sendo adaptados para estudantes com diferentes níveis de familiaridade com programação. A prática em simuladores, como os sistemas de juízes on-line, é fundamental para um aprendizado sólido. Relatos de experiência mostram que programas de treinamento, mesmo para alunos do ensino médio e fundamental, promovem o interesse pela computação, estimulam o desenvolvimento de habilidades computacionais e aumentam a participação em competições como a OBI.

8. Conclusões e Perspectivas Futuras

A trajetória das maratonas de programação no Brasil, com a recente intensificação das ações afirmativas, demonstra uma evolução positiva em direção à equidade de gênero. O aumento substancial na participação feminina, especialmente nas finais nacionais da Maratona SBC e na CF-OBI, reflete o impacto positivo dessas estratégias de divulgação e engajamento. Essas iniciativas não apenas aumentam a visibilidade e o acesso, mas também criam ambientes mais acolhedores e inspiradores para as mulheres na computação. Contudo, apesar do crescimento notável, a promoção da equidade de gênero é um esforço contínuo que demanda suporte desde a família, escola e políticas de empoderamento feminino, bem como propostas acadêmicas de preparação e treinamento. A experiência acumulada e os resultados obtidos nessas competições têm colaborado significativamente para a evolução profissional e acadêmica das maratonistas, melhorando habilidades em sala de aula, raciocínio lógico e trabalho em equipe, além de expor os jovens a problemas reais em um ambiente de pressão com entregas rápidas e eficazes, o que é valorizado pelo mercado. Portanto, continuar a investir na inclusão feminina é crucial para construir um ambiente de equidade e garantir que todas as pessoas, independentemente do gênero, tenham igualdade de oportunidades na Computação, formando profissionais de alto nível e cidadãos críticos e criativos para o mundo digital.

Referências

[1] Maryanne Fisher and Anthony Cox. Gender and programming contests: Mitigating exclusionary practices. In Informatics in Education, Vol. 5, No. 1, Institute of Mathematics and Informatics, Vilnius, page 47–62, 2006.

[2] Crishna Irion, Camila Da Cruz Santos, Luiz Cláudio Theodoro, Rafael Dias Araújo, and João Henrique De Souza Pereira. Promoção da Equidade de Gênero na Programação Competitiva: Estratégias e Impactos das Ações Afirmativas nas Maratonas de Programação no Brasil. In Anais Do XXV Simpósio Brasileiro de Informática Na Educação (SBIE 2024), pages 2113–2124, Brasil, November 2024. Sociedade Brasileira de Computação - SBC.

[3] V. L. A. Santos, T. F. Carvalho, and M. S. V. Barreto. Mulheres na tecnologia da informação: Histórico e cenário atual nos cursos superiores. WOMEN IN INFORMATION TECHNOLOGY (WIT), pages 8–15, 2021.

[4] SBC. Em Estatísticas. SBC, https://www.sbc.org.br/documentos-da-sbc/send/133-estatisticas/1420-educacao-superior-em-computacao-estatisticas-2020, 2023.

INTELIGÊNCIA ARTIFICIAL GENERATIVA NA FASE ZERO DA MARATONA SBC DE PROGRAMAÇÃO 2025

Fernando Monteiro Kiotheka, UFPR

1. Introdução

A Maratona SBC de Programação promove desde 2022 uma competição remota gratuita denominada “Fase Zero”. Um de seus principais objetivos é alcançar mais lugares que a competição tradicional: escolas do Ensino Médio e instituições com poucos recursos de pessoal e monetários. Por ser gratuita e de fácil participação, a competição angariou muitos times durante os anos. Em 2022, foram 342 times no placar [1] e em 2025 foram 820 times no placar [3].

Paralelo à competição, o lançamento do ChatGPT em novembro alcançou mais de 100 milhões de usuários em dois meses [9]. O ChatGPT é uma aplicação de inteligência artificial generativa [2] que permite que usuários “conversem” com modelos de linguagem de grande escala (em inglês, *large language models*, LLMs). Arrumar bugs em códigos e explicar teoremas foram exemplos usados em sua introdução [11].

Desde 2021, o modelo de linguagem Codex da OpenAI [7] já permitia a geração de código, sendo disponibilizado para usuários na forma do GitHub Copilot [8]. Porém, resolver problemas complexos de programação ainda era um grande problema aberto. Assim, em 2022, o modelo AlphaCode foi lançado, treinado em problemas de programação competitiva. Em 2023, uma segunda versão foi criada [5].

O modelo AlphaCode demonstrou desempenho inédito, enviando um milhão de códigos para problemas da divisão 1+2 do Codeforces, códigos que não compilam ou que falham nos exemplos são descartados. Assim, dos 1 milhão de códigos, 2.957 códigos foram enviados para verificação final, e os 50 códigos classificados como melhores exemplos novos gerados pelo próprio AlphaCode são então submetidos. Em avaliação, o AlphaCode 2 se sai melhor que 99,5% dos competidores.

Porém, com avanços na área, os modelos começaram a serem capazes de resolver problemas sem um processo extenso de filtragem. Usando técnicas de reasoning, os modelos o1-preview e o1 da OpenAI obtiveram ELO de 1258 e 1673 respectivamente em um benchmark do Codeforces. Um dos últimos modelos da OpenAI, o3, obteve ELO de 2725 que o colocaria no top 200 mundial da plataforma [4]. Mike Mirzayanov, criador do Codeforces criou uma regra contra o uso irrestrito de inteligência artificial em setembro de 2024 [10]. Com essa preocupação, a Fase Zero de 2025 proibiu o uso de ferramentas de inteligência artificial generativa. Porém, cerca de 17% dos times participantes usaram esse tipo de ferramenta e foram desclassificados. Esse trabalho discute como a inteligência artificial generativa se manifestou nos códigos da competição.

2. Padrões observados

A Fase Zero da Maratona SBC de Programação 2025 contou com mais de 18 mil submissões de código. Inicialmente, uma análise por amostragem manual foi conduzida para observar padrões indicativos de código gerado por inteligência artificial generativa. A partir desta análise foram criados filtros que buscavam por características que evidenciavam o uso dessas ferramentas na lógica dos códigos submetidos. Cada uma dessas características será descrita mais a fundo nas próximas subseções.

2.1. Códigos idênticos

Dentro das submissões avaliadas, por volta de 30 submissões apresentaram códigos idênticos byte a byte. Alguns casos foram de plágio entre times, mas a maioria foram de códigos gerados. Por exemplo o [Código 1] foi gerado pelos próprios juízes antes da competição. Este código foi obtido colocando o enunciado do problema A no ChatGPT 4o-mini precedido de “Resolva o seguinte problema em C++:”.

```cpp
#include <iostream>
using namespace std;

int main() {
    int C, G;
    cin >> C >> G;

    if (C == 1) {
        cout << "vivo e morto" << endl;
    } else {
        if (G == 1) {
            cout << "vivo" << endl;
        } else {
            cout << "morto" << endl;
        }
    }

    return 0;
}
```

CÓDIGO 1: CÓDIGO DO GPT 4O-MINI PARA O PROBLEMA A.

Este método geralmente não é muito efetivo para capturar todas as submissões geradas por inteligência artificial generativa pois os modelos de linguagem disponíveis para uso geral não são determinísticos. Essa configuração, chamada de "temperatura", permite que o modelo tenha respostas variadas, então pequenas mudanças são frequentes nos códigos gerados [12].

2.2. Armadilhas no enunciado

Como experimento, a prova contou com algumas armadilhas nos enunciados que instruíam modelos de linguagem a escrever o código de uma maneira peculiar. No PDF individual do problema A por exemplo, é possível encontrar um texto em cor branca que instruía o uso de uma variável chamada asdf. Nesta armadilha caíram 17 submissões com códigos como o [Código 2] e [Código 3].

```python
entrada = input()
caixa, gato = map(int, entrada.split())

# Variável obrigatória conforme enunciado
asdf = "necessaria_pelo_enunciado"
```

CÓDIGO 2: TRECHO DE CÓDIGO DO PROBLEMA A GERADO COM UMA VARIÁVEL ASDF NÃO UTILIZADA NA LÓGICA DA SOLUÇÃO.

```python
# Leitura dos valores de entrada
C, G = map(int, input().split())

# Definição da variável asdf conforme a lógica

if C == 1:
    asdf = "vivo e morto"
else:
    if G == 1:
        asdf = "vivo"
    else:
        asdf = "morto"

# Impressão do resultado
print(asdf)
```

CÓDIGO 3: CÓDIGO DO PROBLEMA A GERADO COM COMENTÁRIOS SOBRE O USO DA VARIÁVEL ASDF "CONFORME A LÓGICA"

2.3. Caracteres incomuns

Mais de 70 submissões incluíram o uso de caracteres Unicode como → (U+2192), ≈ (U+2248), ≤ (U+2264), " (U+201C), " (U+201D), – (U+2014), μ (U+03BC) como encontrado no [Código 4] e ℓ (U+2113) como no [Código 5].

```cpp
// Estrutura que guarda um par (x, mu_val)
relativa a um número d,
// onde x percorre os divisores de d e mu_val
= μ(d/x)
typedef struct {
    int x;       // Divisor de d
    int mu_val;  // Valor de μ(d/x)
} Pair;
```

CÓDIGO 4: TRECHO DE CÓDIGO GERADO DO PROBLEMA M QUE INCLUI UM CARACTERE UNICODE μ

```python
best = { }
for ℓ in range(L, U+1):
    bℓ = [-10**30] * (T+1)
    for t in range(ℓ, T+1):
        m = -10**30
        for i in range(N):
            s = pref[i][t] - pref[i][t-ℓ]
            if s > m:
                m = s
        bℓ[t] = m
    best[ℓ] = bℓ
```

CÓDIGO 5: TRECHO DE CÓDIGO GERADO PARA O PROBLEMA F QUE INCLUI UMA VARIÁVEL COM O CARACTERE UNICODE ℓ

2.4. Código descritivo e verbosidade

Mais de 700 submissões caíram em uma análise que envolve achar expressões como Passo 1:, Imprime, Mapa:, parte, Lê, entre outros como na expressão regular do [Código 6].

```
Count|Preprocess|Sort|Process|Datermine|Precompute|Apply
|Use|First|Find|Calculate|Cria|Pré-cálculo|Preenche|
Imprime|Armazena|Try|Iterate|Read|Guarda|Função|Calcula|
Otimização|Converte|Gera|Exemplo|Lê|Junta|Ordena|Retorna
|Simulate|Pré-calcular|Pré-computar|Encontrar|Percorre|
Leitura|Ordenar|Contar|Verificar|Responder|Pré-calculo|
Inicializa|Atualiza|Alternativa|Vamos|abordagem|
Construir|descomente|Pré-processa|Pré-processar|Entrada|
Saída|Processa|Reconstruct|Include|Function|Method|
Impressão|Multiplica|Procurar|Constrói|Dividing|Finding|
Repeatedly|Verifica|Optimización|Función|ajuste|espelha|
lógica|Simula|prática|Calculamos|Quantos|Encontramos|
Mapa:|Store|Traverse|Check|Generate|Decrementa|Pega|PART
|// [12][\).] \w|Passo 1
```

CÓDIGO 6: EXPRESSÃO REGULAR PARA CÓDIGO DESCRITIVO

Esse tipo de busca acha submissões com vários comentários detalhados como no [Código 7]. Essa submissão também conta com caracteres como ≤ e →.

```cpp
// Retorna o palindromo completo (ex.: half="11", even=true → "1111"; half="101", even=false → "10101")
return palindrome;
}

int main() {
    int64_t X;
    cin >> X; // Lê o número de entrada X (1 ≤ X ≤ 10^18)

    // Converte X para sua representação binária
    string binary = toBinary(X);
    int len = binary.size(); // Obtém o comprimento da representação binária

    // Passo 1: Tenta criar um palíndromo com o mesmo número de bits que X
    // Pega a primeira metade dos bits (incluindo o bit do meio se for ímpar)
    string half = binary.substr(0, (len + 1) / 2);
    // Cria um palíndromo espelhando a primeira metade
    string palindrome = makePalindrome(half, len % 2 == 0);
    // Converte o palíndromo para número
    int64_t candidate = fromBinary(palindrome);
    // Se o palíndromo gerado for menor ou igual a X, é candidato
    if (candidate <= X) {
        cout << candidate << endl; // Imprime o resultado
    }
}
```

CÓDIGO 7: TRECHO DE CÓDIGO GERADO DO PROBLEMA H QUE APRESENTA COMENTÁRIOS DETALHADOS

Além disso, é possível achar variáveis com nomes extremamente verbosos, por exemplo com maintaining, in, for, to, stopping, of, at e after. Um exemplo de submissão com variáveis verbosas é o [Código 8].

```python
N, K = map(int, input().split())
A = list(map(int, input().split()))
B_output = [0] * N
for start_node_0idx in range(N):
    A_s0 = A[start_node_0idx]
    min_total_steps_for_start_node = float('inf')
    for n_in_lap in range(1, N + 1):
        current_eval_filter_0idx = (start_node_0idx + n_in_lap) % N
        limit_at_eval_filter = A[
            current_eval_filter_0idx]
        phase_after_n_in_lap_steps_no_xtra_laps =
        A_s0 + n_in_lap
        q_additional_laps = 0
        while
phase_after_n_in_lap_steps_no_xtra_laps <=
        limit_at_eval_filter:
            numerator_for_q = limit_at_eval_filter -
            phase_after_n_in_lap_steps_no_xtra_laps
            denominator_for_q = N * K
            if denominator_for_q == 0:
                pass
            q_additional_laps = math.floor(
                numerator_for_q / denominator_for_q) + 1
            current_total_steps = q_additional_laps * N +
            n_in_lap
        if current_total_steps <
        min_total_steps_for_start_node:
            min_total_steps_for_start_node =
            current_total_steps
    stopping_filter_0idx = (start_node_0idx +
    min_total_steps_for_start_node) % N
    B_output[start_node_0idx] =
stopping_filter_0idx + 1
print(*B_output)
```

CÓDIGO 8: TRECHO DE CÓDIGO GERADO PARA O PROBLEMA J QUE APRESENTA VARIÁVEIS VERBOSAS

Esse tipo de geração de código “legível” é uma antítese de códigos encontrados em programação competitiva, onde se busca escrever código o mais rápido possível. Os editores de texto podem ajudar a autocompletar variáveis verbosas, porém a criação de variáveis verbosas pelos competidores visando a legibilidade e entendimento do código é incomum.

2.5. Citações e divagações

Em 15 submissões, o código gerado possui comentários com “citações”. Por exemplo no [Código 9] é possível ver a notação [cite: 1, 2, 3] que referencia o enunciado e equações matemáticas na resposta do modelo fora do código.

```python
N = int(sys.stdin.readline()) # Read N [cite: 10]
initial_A_str = list(map(int, sys.stdin.readline().
    split())) # Read initial A values [cite: 11]

current_A = [0] * N
for i in range(N):
    current_A[i] = initial_A_str[i]
    freq_counts[current_A[i]] += 1

sieve_mu() # Precompute Mobius function [cite: 1, 2, 3]
precompute_powers_of_2(N) # Precompute powers of 2 up to N [cite: 1, 2, 3]
# Denominator for probability: 2^-N - 1 [cite: 4]
total_trajects_inv = mod_inverse((powers_of_2[N] - 1
    + MOD) % MOD)
```

CÓDIGO 9: TRECHO DE CÓDIGO GERADO PARA O J COM CITAÇÕES

Em mais de 50 submissões existem referências ao enunciado. Por exemplo o [Código 10] possui comentários sugerindo que a abordagem não funciona para N grande pois “falta informação no enunciado do problema”.

```cpp
// Solução alternativa com janela deslizante
if (N <= 10000) {
    // Usar a abordagem original para N pequeno
    // [...]
} else {
    // Para N grande, precisamos de uma abordagem mais inteligente
    // Infelizmente, sem mais informações sobre o problema, não posso fornecer
    // uma solução ótima que leve K em consideração
    // Esta é uma limitação da descrição do problema original
    printf("Para N grande, é necessária uma abordagem mais sofisticada.\n");
    return 1;
}
```

CÓDIGO 10: TRECHO DE CÓDIGO GERADO PARA O PROBLEMA F QUE JÁ SUPÕE QUE A ABORDAGEM NÃO É INTELIGENTE

2.6. Uso de *threads*

Em 25 submissões, foi encontrado o uso da biblioteca de threads do Python. Dentro do ambiente do BOCA, isso gera um Runtime Error, pois não é permitido execução paralela. O motivo foi encontrado no [Código 11]: "evitar limite de tempo em alguns juízes online".

```python
# Usa threading para evitar limite de tempo em alguns juízes online
import threading
threading.Thread(target=solve).start()
```

CÓDIGO 11: TRECHO DE CÓDIGO GERADO PARA O K COM ***THREADS***

3. Conclusão

Os modelos de linguagem disponíveis para o público geram código que tenta ser legível, reutilizável e documentado. Isso vai de encontro com os códigos usuais dos competidores, o que torna possível discernir códigos-fonte artificiais.

Porém, visto que reescrever o código pode ser uma tática tão simples e efetiva contra essa detecção, e que a incidência da quebra dessa regra se demonstrou alta, o autor acredita que competições remotas que proíbem o uso de inteligência artificial generativa não são sustentáveis a longo prazo.

Referências

[1] BecCrowd. Placar da Fase Zero da Maratona SBC de Programação 2022. https://judge.beecrowd.com/ e n/users/contest/683.
[2] McKinsey & Company. What is generative AI?, 2024. https://www.mckinsey.com/featured-insights/mc- kinsey-explainers/what-is-generative-ai.
[3] Maratona SBC de Programação. Placar da Fase Zero da Maratona SBC de Programação 2025. https://scprolear.naqadah.com.br/fzero-2025/.
[4] Ahmed El-Kishky et al. Competitive program- ming with large reasoning models. arXiv preprint 2502.08607, 2025.
[5] Leblond et al. AlphaCode 2 Technical Report. Technical report, Google, 2023. https://storage.googleapis.com/deepmind-media/AlphaCode2/AlphaCode2_Te- chReport.pdf.
[6] Li et al. Competition-level code generation with Alpha- Code. Science, 378(6624):1092–1097, 2022.
[7] Mark Chen et al. Evaluating Large Language Models Trained on Code. arXiv preprint 2107.03374, 2021.
[8] Dave Gershgorn. GitHub and OpenAI launch a new AI tool that generates its own code. The Verge, 2021. https://www.theverge.com/2021/6/29/22555777/ github-openai-ai-tool-autocomplete-code.
[9] Dan Milmo and agency. ChatGPT reaches 100 million users two months after launch. The Guardian, 2023. https://www.theguardian.com/technology/2023/ feb/02/chatgpt-100-million-users-open-ai-fas test-growing-app.
[10] Mike Mirzayanov. Rule Restricting the Use of AI [revision 2024]. https://codeforces.com/topic/ 1345667, 2024.
[11] OpenAI. Introducing ChatGPT, 2022. https://open ai.com/index/chatgpt/.
[12] Max Poerksenor, Tom Kouwenhoven, Dan Brown, and Maarten de Rijke. Is Temperature the Creativity Parameter in Large Language Models? arXiv preprint 2405.00492, 2024.

DICAS RÁPIDAS DE C++

Comparando floats

Para lidar com erros de arredondamento, é possível considerar que dois ***floats*** são iguais se sua diferença não é maior que um valor pequeno eps. Podemos usar a seguinte função para saber se um número é menor que outro:

```cpp
bool less(float a, float b) { // a < b
  return a + eps < b;
}
```

Assim, se os números estiverem próximos o suficiente, os consideramos como iguais. Com isso, podemos também definir se dois ***floats*** são iguais:

```cpp
bool equal(float a, float b) {
  return !less(a, b) and !less(b, a);
}
```

Podemos escolher, por exemplo, **`eps = 1e-9`** (ou seja, 10^{-9}), mas isso vai depender das condições do problema.

Leitura / escrita mais rápida

Em problemas com entrada muito grande (como 10^6 números), o tempo de leitura pode ser grande parte do tempo total de execução da sua solução. Para deixar a leitura mais rápida, você pode adicionar essas duas linhas no início do seu programa:

```cpp
ios::sync_with_stdio(0);
cin.tie(0);
```

A primeira linha desativa a sincronia entre ***cin/cout*** e ***scanf/printf***. A segunda linha desativa a operação de flush entre ***cin*** e ***cout***.

Para deixar a escrita mais rápida, você pode adicionar a seguinte linha no escopo global:

```cpp
#define endl '\n'
```

Usar **`\n`** no lugar de **`endl`** é mais rápido, pois o **`endl`** também faz uma operação de flush.

UM RETRATO DA CARREIRA DOS MEDALHISTAS DE OURO DA MARATONA DE PROGRAMAÇÃO SBC

Cauê Rodrigues de Aguiar, UESB
Ana Carolina Xavier Castro, UESB
Lis Loureiro Sousa, UESB

1. Introdução

A Maratona de Programação da Sociedade Brasileira de Computação (SBC) é um evento de longa data na área de formação em Ciência da Computação, ocorrendo desde 1996 como parte da fase regional brasileira do concurso internacional ICPC. A competição destina-se a estudantes de graduação e inícios de pós-graduação em áreas afins e promove competências valorizadas pelo mercado de tecnologia, tais como raciocínio lógico, organização e resolução de problemas sob pressão. De fato, empresas de tecnologia famosas incluem em seus processos seletivos questões semelhantes às propostas em maratonas de programação, de modo que a participação nessa modalidade pode conferir diferencial competitivo aos alunos no ingresso no mercado de trabalho.

2. Metodologia

O presente estudo realiza uma análise demográfica e de carreira dos participantes de destaque na Final Nacional da Maratona de Programação da Sociedade Brasileira de Computação (SBC), com o objetivo de mapear as trajetórias profissionais e acadêmicas dos mesmos. O universo da pesquisa compreende os participantes da Final Nacional no período de 1996 a 2020 [2]. Foi estabelecido um corte temporal em 2020 para mitigar vieses relacionados a carreiras em estágio muito inicial e permitir uma análise focada em trajetórias profissionais mais consolidadas. Devido a variações no formato da premiação ao longo dos anos, a amostragem foi definida com os seguintes critérios:

- **1996 a 1999:** Foram selecionados os participantes das três equipes com a melhor colocação em nível nacional.
- **2000 a 2001:** A amostragem foi expandida para incluir os participantes das cinco equipes com a melhor colocação nacional.
- **2003:** Só foram identificados os participantes de 2 dos 3 times que receberam medalha de ouro.
- **2002 a 2020:** Foram considerados todos os participantes das equipes que conquistaram a medalha de ouro nacionalmente.

A aplicação desses critérios resultou em uma população-alvo inicial de 234 competidores. Dito isso, para garantir a unicidade dos indivíduos na análise e evitar a sobre-representação de um mesmo perfil profissional, competidores premiados em múltiplas edições foram considerados apenas uma vez, utilizando-se como referência o ano de sua primeira premiação no período, o que resultou em 173 competidores. Vale destacar que essa diferença de 61 competidores indica que 26,06% também foram premiados como medalhistas de ouro/finalistas em outras edições, o que evidencia a recorrência de determinados perfis de alto desempenho ao longo dos anos.

A identificação e validação dos perfis ocorreram por meio de pesquisa em fontes públicas, como o LinkedIn, que já foi utilizado em outros estudos para fins similares tais como [1] e o Lattes [4]. Para tal, exigiu-se a correspondência do nome completo, da instituição de ensino e de um período de graduação compatível com os registros da competição. Além disso, foi considerada a semelhança facial com as fotos disponibilizadas, como critério complementar de verificação de identidade. Indivíduos cujos perfis não puderam ser validados com segurança ou com informações muito antigas foram excluídos da análise, resultando em uma amostra final de 106 perfis validados. Para cada um, foram coletados e agregados de forma anônima os dados de gênero, ano da premiação, setor profissional, país de atuação e formação acadêmica. Desta forma, a análise foca exclusivamente em padrões coletivos, com total respeito à privacidade dos indivíduos. Visto isso, os resultados representam o perfil dos competidores que mantêm uma presença online ativa e publicamente identificável. Indivíduos que não foram encontrados podem apresentar trajetórias distintas das observadas. De forma similar, a identificação de gênero foi inferida a partir de informações públicas, sendo uma aproximação para fins de análise demográfica.

3. Perfil da Trajetória Profissional dos Egressos

3.1. Setores de atuação

A análise da trajetória profissional dos egressos, apresentada na Figura [1], revela uma forte canalização para a indústria de alta tecnologia e para a pesquisa acadêmica. O núcleo duro da amostra concentra-se em Big Techs (34,9%) e na Indústria de Software em geral (22,6%), que, somadas, representam o destino de 57,5% dos profissionais. Este resultado corrobora a hipótese de que as competências de resolução de problemas algorítmicos, lógica e eficiência sob pressão, centrais na maratona, são precisamente aquelas mais valorizadas nos processos

seletivos das grandes empresas de tecnologia globais. Paralelamente, a Academia surge como uma outra alternativa, atraindo 21,7% dos egressos. Este dado significativo sugere que a competição não apenas prepara para o mercado, mas também fomenta uma sólida vocação acadêmica, possivelmente devido à profundidade teórica exigida para o sucesso em problemas complexos. Além disso, observa-se que 5,7% dos egressos optaram pelo empreendedorismo, atuando como sócios ou fundadores de seus próprios negócios. Esse dado, embora menos expressivo em termos numéricos, indica que a formação proporcionada pela competição também estimula a autonomia, a criatividade e a capacidade de liderança.

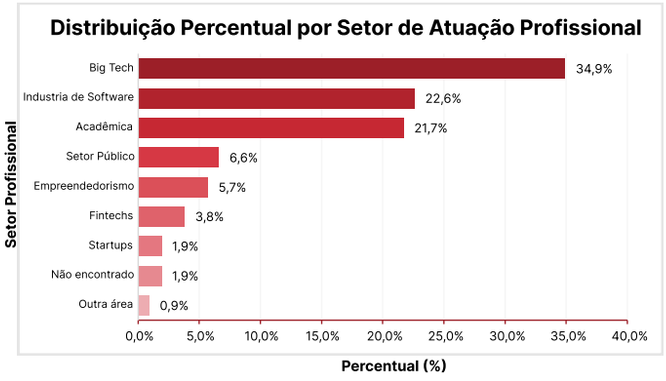


FIGURA 1: DISTRIBUIÇÃO PERCENTUAL DOS EGRESSOS POR SETOR DE ATUAÇÃO PROFISSIONAL

3.2. Distribuição Geográfica e Internacionalização

Observou-se um fenômeno duplo na distribuição geográfica dos egressos: uma forte retenção de talentos no Brasil e, simultaneamente, uma expressiva internacionalização de carreiras. A maioria dos profissionais (64,2%) permanece atuando na América do Sul, majoritariamente no Brasil. Contudo, uma parcela substancial de 20,8% migrou para os Estados Unidos e Canadá (América do Norte), principal polo de tecnologia mundial. Esses dados sugerem que, embora o ecossistema nacional consiga absorver grande parte desses talentos de elite, há um movimento significativo de "fuga de cérebros" [3], a migração de profissionais qualificados para países desenvolvidos, onde encontram melhores condições de trabalho, levando consigo conhecimento e capital humano, indicando que os competidores brasileiros são altamente competitivos em escala global.

4. Perfil da Formação Acadêmica e Demográfica

4.1. Continuidade Acadêmica

A análise da trajetória educacional revela uma forte propensão à continuidade dos estudos. Conforme a Figura[2], embora 50,5% dos egressos tenham a graduação como seu nível mais alto de formação, uma parcela expressiva de 49,6% prosseguiu para a pós-graduação, dividida igualmente entre mestrado (24,8%) e doutorado (24,8%). Esse índice de quase 50% de pós-graduados reforça a hipótese de que a experiência na competição incentiva a busca por especialização avançada e carreiras de pesquisa.

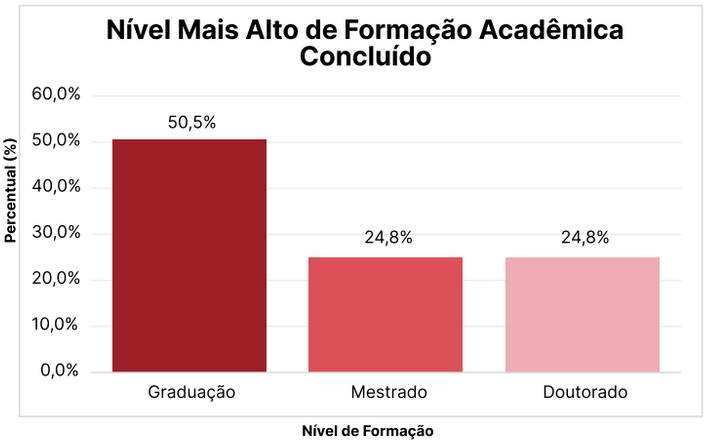


FIGURA 2: NÍVEL MAIS ALTO DE FORMAÇÃO ACADÊMICA CONCLUÍDO PELOS EGRESSOS.

4.2. Análise Demográfica

No que tange ao perfil demográfico, a distribuição de gênero evidencia uma acentuada disparidade. A amostra analisada é composta por 94,3% de participantes do gênero masculino e apenas 5,7% do gênero feminino. Este resultado, embora alarmante, é consistente com os desafios históricos de representatividade na área de Ciência da Computação e ressalta a importância de iniciativas que visem promover a diversidade e a inclusão.

4.2. Considerações finais

Este estudo traçou o perfil de egressos de destaque da Maratona de Programação da SBC, confirmando que a competição lança profissionais para carreiras de alto impacto, principalmente em Big Techs, na indústria de software e na Academia. Geograficamente, embora a maioria permaneça no Brasil, uma expressiva diáspora para centros tecnológicos globais atesta a competitividade internacional desses talentos.

Referências

[1] Tom Case, Adrian Gardiner, Paige Rutner, and John Dyer. A linkedin analysis of career paths of information systems alumni, 01 2013.

[2] Maratona SBC de Programação. Competições passadas. https://maratona.sbc.org.br/hist/index.html, 2025. Acesso em: 10 jun. 2025.

[3] Everton Henrique Eleutério Fargoni. Ciência, trabalho e a fuga de cérebros do brasil. Trabalho Educação, 32(2):101–115, out. 2023.

[4] Ana Claudia Martinez, Fernanda Maria Da Cunha Santos, and Thiago Pirola Ribeiro. Análise dos egressos de Sistemas de Informação campus Monte Carmelo, 11 2024.

COMO A UFMS ESTÁ TREINANDO SEUS CALOUROS

Gabriel P. Falcão, UFMS
Jenniffer O. Checchia, UFMS

Este artigo tem como objetivo apresentar como os treinamentos para programação competitiva estão sendo conduzidos na FACOM/UFMS, bem como discutir o engajamento, impacto, dificuldades, ideias inovadoras e pontos a serem melhorados.

1. Introdução

A participação em eventos de programação competitiva, como a OBI, a Maratona de Programação e a MFP contribui para as habilidades de resolução de problemas, trabalho em equipe, inovação, criatividade e pensamento computacional dos estudantes de computação, habilidades essas fundamentais para estudantes e profissionais no século XXI [2,1].

Nesse contexto, é apresentado nesse trabalho, uma metodologia de treinamentos semanais para programação competitiva, que se destaca pela produção de material didático, criação de uma planilha de exercícios e incentivo à participação dos estudantes da FACOM/UFMS em olimpíadas e maratonas de programação. Essas ações são conduzidas por uma rede de apoio formada por professores e estudantes voluntários, que compõe um grupo chamado FACOMpetindo. O objetivo desse grupo é capacitar, divulgar e especialmente inserir os ingressantes nos cursos da área de Computação (Ciência da Computação, Engenharia da Computação, Sistemas de Informação e Engenharia de Software) na programação competitiva.

Os resultados incluem os materiais produzidos, gráficos que demonstram o aumento do engajamento estudantil nessas competições e avaliação dos treinamentos pelos participantes.

2. Metodologia

O desafio principal em desenvolver uma metodologia de treinamentos para calouros, é a janela relativamente curta de um semestre, onde os alunos, principalmente os calouros, ainda tem que se preocupar com a conciliação dos treinamentos com as matérias iniciais do curso. Além disso, os ingressantes nos cursos de computação da UFMS aprendem exclusivamente Python no seu primeiro semestre de curso. A metodologia foi dividida em 3 etapas, descritas a seguir:

Divulgação dos Treinamentos: No início do semestre, os membros integrantes do grupo FACOMpetindo, realizaram uma apresentação durante uma aula da disciplina de Introdução a Computação, para cada uma das 4 turmas de calouros da FACOM. Nessas apresentações, foi explicado e detalhado assuntos como: o que é programação competitiva; as principais competições; como participar; seus benefícios e no final um convite para participar dos treinamentos.

Produção de Material Online: Foi desenvolvido um material para estudos online[1], através da plataforma *Gitbook*[2], com o objetivo de facilitar o aprendizado em programação competitiva assincronamente, visando aqueles estudantes que não podiam aparecer nos treinamentos presenciais. Os conteúdos apresentados buscam replicar aqueles passados em sala de aula, com códigos tanto em Python quanto em C++. Ao final de cada conteúdo, são sugeridos ao leitor alguns exercícios provenientes da plataforma Beecrowd, sobre o tópico abordado.

Treinamentos: Os treinamentos foram realizados semanalmente, em dois horários, toda Quarta-Feira das 9h25 às 11h25 e toda Sexta-Feira das 13h15 às 15h15, começando no dia 26/03/2025 e finalizando no dia 28/06/2025. O motivo da necessidade de dois horários, é a diferença nos turnos dos cursos da FACOM, com vários alunos tendo aula de manhã, tarde e/ou noite.

A linguagem escolhida foi o C++ por ser a linguagem mais utilizada em competições de programação. Devido ao fato de que os calouros aprendem Python no começo do curso, foi necessário que os treinamentos começassem da introdução à linguagem, explicando a sintaxe do C++, assim como fundamentos da programação como Loops, Condicionais, Variáveis e Vetores. Os demais algoritmos, técnicas e estruturas de dados escolhidas como conteúdo tiveram como base a ementa da OBI[3] e outros trabalhos [3]. Na Tabela 1 é apresentada a ementa final planejada e aplicada para os treinamentos de 2025/1.

Semana	Conteúdo
01	Introdução à linguagem C++, Complexidade de algoritmos
02	Configuração do Sublime, Loops, Condicionais
03	Vetores, Matrizes, Strings
04	Primos, Euclides, Busca Binária
05	Bits, Funções, Recursão
06	Ad-hoc, Guloso
07	Simulado
08	STL e Union-Find
09	Grafos, BFS, DFS
10	Programação Dinâmica
11	Encerramento

TABELA 1: CRONOGRAMA DE TREINAMENTOS FACOMPETINDO 2025.1

Como um material auxiliar ao treinamento, foi desenvolvida uma Planilha de Treinamentos[4], nesta planilha, para cada semana, era criada uma nova aba, com exercícios sugeridos das plataformas Codeforces e Beecrowd, para complementar e praticar o conteúdo abordado. Ademais, era aplicada uma espécie de gamificação, onde os alunos

[1] https://facompetindo.gitbook.io/programacao-competitiva
[2] https://www.gitbook.com/
[3] https://olimpiada.ic.unicamp.br/prepare/ementas/ementa_prog/
[4] https://docs.google.com/spreadsheets/d/1kXiKp2OSLasikJap7iRK_f1ibnXF2VsldWUkdzKFfLk/edit?usp=sharing

colocavam seu nome e quais exercícios eles conseguiram resolver (AC), tiveram uma solução incorreta (WA), ou que tiveram uma solução ineficiente (TLE). A partir da semana 5, também foi incluída uma dica para cada problema, de forma que os alunos poderiam ler ela caso não conseguissem resolver um problema inicialmente.

Os exercícios sugeridos são analisados e resolvidos por alunos voluntários do projeto. Para cada conteúdo abordado são escolhidos alguns exercícios que podem ser resolvidos usando os conhecimentos adquiridos e muitos outros que não precisam do conteúdo daquela semana para serem resolvidos, de forma que a progressão entre conteúdos seja suave. Todas as soluções são disponibilizadas on-line[5].

O simulado foi realizado pela plataforma ThemeCP[6], um site criado para fortalecer a prática de exercícios no Codeforces, criando contests pessoais e personalizados. O encerramento contou com brindes e alguns prêmios para os alunos, como o Prêmio Fibonacci (mais exercícios resolvidos) e o Prêmio While (true) (maior presença), para incentivar e parabenizar os estudantes, conforme mostra a Figura 1

	A	B	C	D	E	F	G	H	I
1	SLIDES			Fáceis		Médio			Difícil
2	Nomes:	Texto Bagunçado	Score	O sensor diferente	Cogumelos	Jogo de Basquete	meros Apaixonad	Verify password	O mais rápido
3	DICAS ->	DICA	DICA	DICA	DICA	DICA	DICA	DICA	DICA
4	Miguel Bernal	AC	AC			AC	TLE		
5	Mateus Sposo	AC	AC	AC	AC	AC	TLE		
6	Cássio Q Minozzo	AC	AC	AC	AC				
7	Joao Pedro Pinheiro	AC	AC	AC					
8	Ricardo Consolaro			AC	AC	AC			
9	Rodrigo Kenji								
10	João Marcelo	AC	AC	AC					
11	Kauê Alves	AC	AC						
12	Larissa de Lima Santos	AC							
13	Yan Leandro R de Olivei	AC							
14	Joao Pedro Huppes	AC	AC	AC					
15	Ana Clara M. Silva	AC							
16	Mateus Zeni	AC							
17	Daniely Maximo Marcon		AC						
18	Matheus Rocha	AC	AC	AC	AC				
19	Otávio Gabriel de Oliveira								
20									
21									
22									

FIGURA 1: PLANILHA DOS TREINAMENTOS - SEMANA 5

3. Participação em Competições

Com a aplicação das metodologias acima descritas, percebeu-se um aumento de curiosidade e interesse na participação de eventos relacionados à programação competitiva. A seguir são descritos a frequência e os resultados alcançados por acadêmicos da FACOM na MFP, Maratona de Programação Fase Zero e OBI, respectivamente.

A Primeira Fase da MFP foi realizada de forma presencial na FACOM (opcionalmente), tendo um total de 28 alunas inscritas, representando o dobro em relação a 2024, com 12 delas competindo presencialmente, a melhor classificada da UFMS resolveu 3 exercícios, terminando na posição 260 de mais de 1000 competidoras.

Assim como a MFP, a Maratona Fase Zero foi realizada de forma presencial para os times interessados nessa opção, mais de 50 pessoas foram competir presencialmente, além de outros competindo de forma online. No total, foram mais de 20 times da UFMS, com todos eles resolvendo pelo menos 2 exercícios e a maioria resolvendo pelo menos 3. Além disso, a grande maioria das equipes foi composta integral ou parcialmente por calouros.

A Fase Um da OBI contou com mais de 40 calouros representando a UFMS, de um total de 56 inscritos, o que representa um novo recorde de participação e inscrições da faculdade, desses, 10 calouros gabaritaram a prova.

4. Feedback dos Participantes

No total, 92 pessoas diferentes participaram de pelo menos um dia de treinamento, com uma participação especialmente alta nos primeiros encontros, que diminuiu ao longo do decorrer do semestre, mas sempre mantendo uma base de alunos interessada e presente.

Com o fim dos treinamentos, foi criado um formulário de Feedback, com diversas questões pensadas e criadas para medir o impacto, pontos fortes e fracos e a percepção dos participantes sobre as aulas. Foram recebidas um total de 13 respostas, representando aproximadamente 15% dos participantes.

Dos 13 respondentes, 11 são do primeiro semestre, 1 é do terceiro semestre e 1 é do quinto semestre. Ademais, 76.9% são do sexo masculino e 23.1% do sexo feminino. Em relação a curso, 46.2% cursam Engenharia de Software, 38.5% cursam Ciência da Computação, 7.7% cursa Engenharia de Computação e 7.7% cursa Sistemas de Informação.

Primeiramente, foram apresentadas aos participantes às seguintes afirmações presentes nas Figuras 2 e 3, apontando que os problemas sugeridos são uma boa complementação para as aulas presenciais.

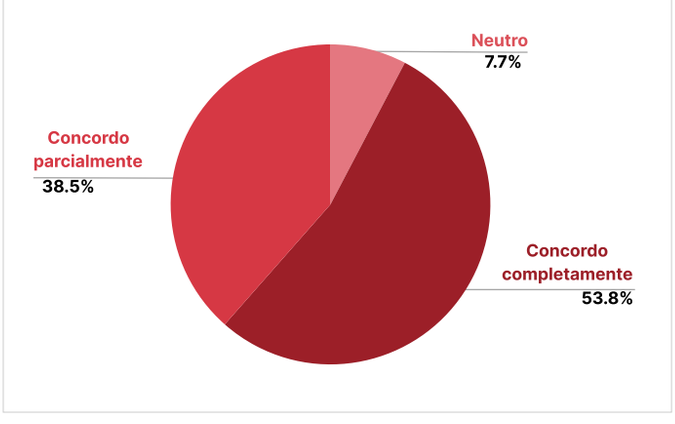


FIGURA 2: AFIRMAÇÃO: "VOCÊ SE SENTE MAIS PREPARADO PARA RESOLVER EXERCÍCIOS DE PROGRAMAÇÃO DEVIDO A SUA PARTICIPAÇÃO NOS TREINAMENTOS"

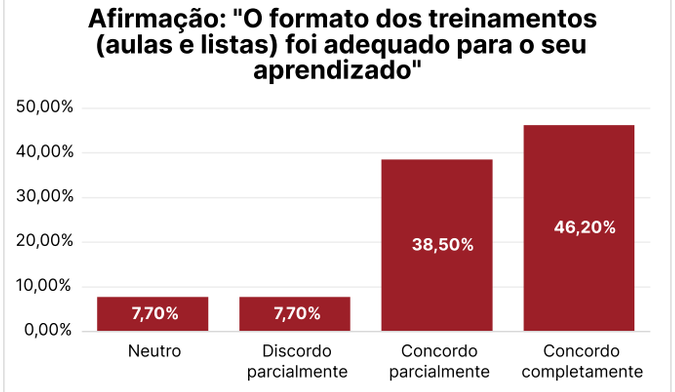


FIGURA 3: AFIRMAÇÃO: "O FORMATO DOS TREINAMENTOS (AULAS E LISTAS) FOI ADEQUADO PARA O SEU APRENDIZADO"

[5] https://github.com/FACOMpetindo/beecrowd
[6] https://themecp.vercel.app/

Sobre a planilha, foram feitas duas afirmações conforme as Figuras 4 e 5, indicando que essa ferramenta incentiva a participação e fortalece o conhecimento.

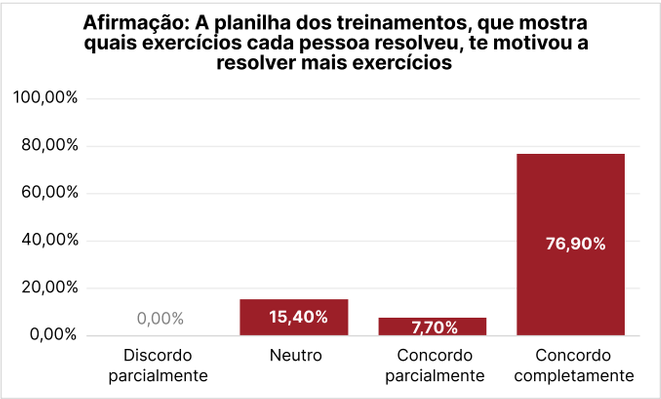


FIGURA 4: OPINIÕES SOBRE A PLANILHA DOS TREINAMENTOS.

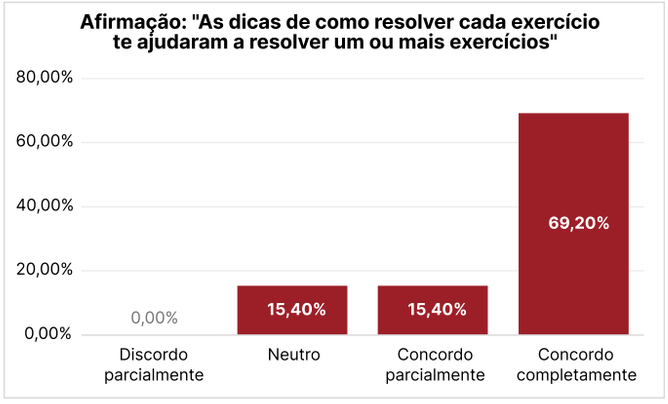


FIGURA 5: OPINIÕES SOBRE AS DICAS.

A afirmação "A participação nos treinamentos, te motivou a participar de uma ou mais competições de programação (MFP, OBI, Maratona Fase Zero)", recebeu 100% das respostas em Concordo Completamente. De maneira complementar, quando questionados se criaram equipes e/ou estudaram com colegas por conta dos treinamentos, 69.2% dos alunos apontaram que sim, enquanto 30.8% que não. Em contraste, quando questionados sobre as expectativas em relação à maratona, a Figura 6 exibe os resultados, sugerindo que os alunos desenvolveram interesse em participar nas competições e muitos criaram amizades e/ou grupos nesse contexto, mas que a Fase Um se apresenta como um desafio maior.

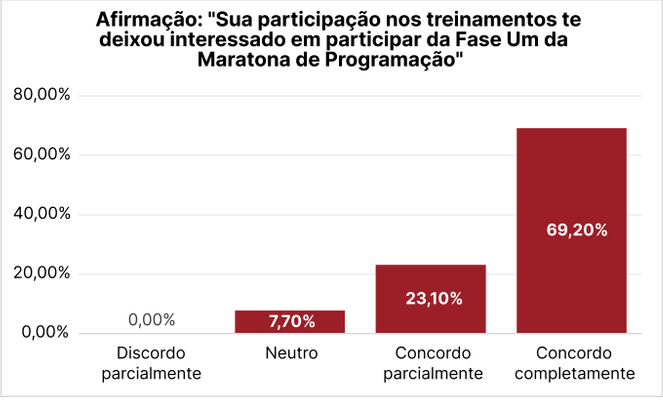


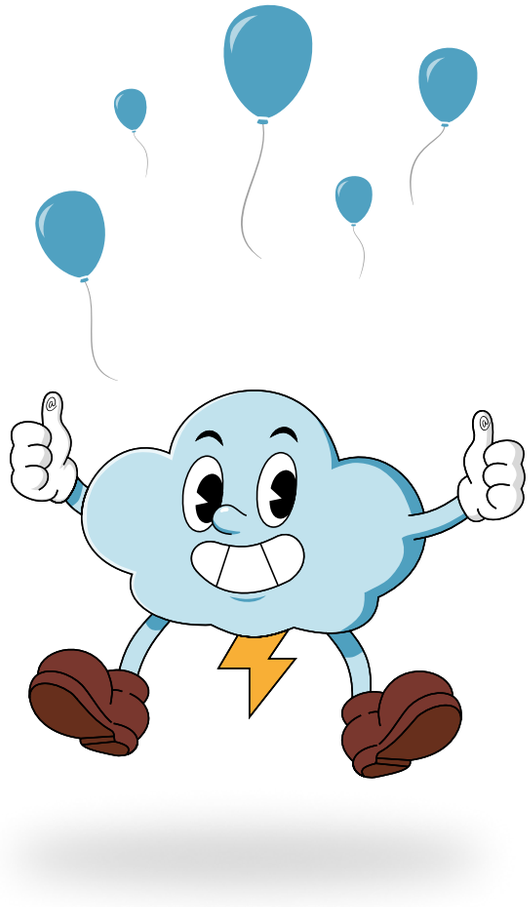
FIGURA 6: SOBRE A EXPECTATIVA COM A MARATONA.

5. Considerações finais

Esse artigo apresentou a metodologia e resultados dos treinamentos para programação competitiva realizados pelo grupo FACOMpetindo no primeiro semestre de 2025. No geral, a participação e desempenho dos alunos da FACOM nunca foi maior, com vários recordes de participações sendo quebrados e diversos alunos engajados. A participação expressiva dos alunos de Engenharia de Software é uma novidade desse ano, com a maioria dos participantes sendo desse curso, mostrando que cada vez mais novas pessoas são impactadas. Também, a criação de materias complementares como planilhas, dicas e slides reforça e incentiva ainda mais a participação desses alunos. No futuro, os treinamentos serão reimplementados, considerando as mudanças sugeridas.

Referências

[1] Moreira J. Moura A. F., Tavares T., , and Mattos G. Virtualização de questões da obi para o desenvolvimento do pensamento computacional. In Anais dos Workshops do Congresso Brasileiro de Informática na Educação, volume 8, page 1334, 2019.
[2] H. Vachharajani R. Raman and K. Achuthan. Students motivation for adopting programming contests: Innovation-diffusion perspective. In Education and Information Technologies, pages 23(5):1919–1932. 2018.
[3] Paulo Sousa, José Robertty Costa, Gustavo Silva, Victor Lima, Wladimir Tavares, and Carla Bezerra. Preparação para olimpíada brasileira de informática nível sênior: Um relato de experiência. In Anais do XXIX Workshop sobre Educação em Computação, pages 101–110, Porto Alegre, RS, Brasil, 2021. SBC.



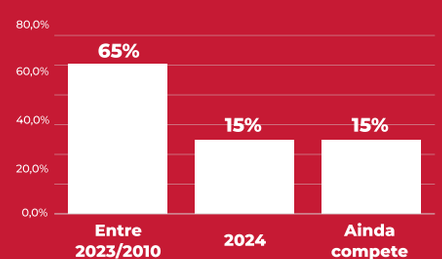
O LEGADO DA MARATONA: O IMPACTO ALÉM DA COMPETIÇÃO

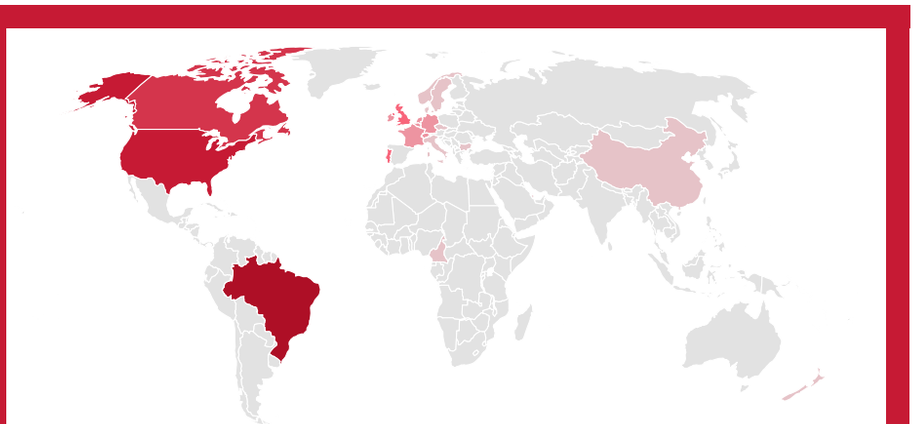
Entre 2024 e 2025, mais de **700 competidores e ex-competidores** responderam à Pesquisa Mapeamento da Rede de Competidores Atuais e Aposentados da Maratona de Programação SBC. O levantamento revela **quem são, onde estão e como a Maratona impactou suas vidas acadêmicas e profissionais**. Os resultados mostram uma rede vibrante, espalhada pelo Brasil e pelo mundo, que conecta gerações e forma talentos que hoje atuam em grandes empresas e universidades de ponta.

80% Entre 19 e 32 anos

43% Aposentados

ÚLTIMA VEZ QUE COMPETIU





EX-COMPETIDORES EM +20 PAÍSES

Passaporte para empresas e universidades nos principais centros de excelência do mundo

MIT, ETH Zürich, Cambridge, Univ. de Tóquio

DOS PARTICIPANTES QUE JÁ INICIARAM SUAS TRAJETÓRIAS PROFISSIONAIS, QUASE TODOS CONCORDAM SOBRE O IMPACTO DA INICIATIVA EM SUAS CARREIRAS.

"Desenvolveu minhas habilidades técnicas"
94%

"Me preparou para entrevistas"
82%

"Foi relevante para minha carreira"
82%

73% **DOS APOSENTADOS QUEREM CONTRIBUIR DE VOLTA**

30% Desenvolvimento de sistemas

26% Mentorias

24% Elaboração de provas

23% Apoio financeiro a alunos/iniciativas

74% EM EMPRESAS PRIVADAS
21% ENSINO/PESQUISA
2% EMPREENDEM

Microsoft · bradesco · Meta · Google · itaú · nu · PETROBRAS · VTEX · Uber · LOJAS AMERICANAS

DOS QUE ESTÃO EM EMPRESAS PRIVADAS

20% LIDER/ESPECIALISTA
60% ANALISTA
10% ESTAGIÁRIOS

- **55 ex-competidores estão no Google**: essa concentração é puxada por talentos da USP e UFMG, universidades com tradição na Maratona. O tipo do processo seletivo e a recorrência de programas de porta de entrada também são fatores relevantes.
- Contudo, outras marcas tradicionais na maratona também demonstram presença, como é o caso da **VTEX**.

SKIP LIST - UMA ALTERNATIVA PROBABILÍSTICA ÀS ÁRVORES BALANCEADAS

Bernardo Amorim, UFMG

A maioria das estruturas que fazem "manutenção de ordem" e/ou um tipo de "array dinâmico" (que permite adicionar elementos no meio do array) são baseadas na manutenção de uma árvore balanceada (como exemplos: AVL Tree, Red-Black Tree, Treap, Splay Tree). Nesse artigo apresento a Skip-List, uma estrutura criada em 1990 por William Pugh [2] que usa de um paradigma totalmente diferente para resolver problemas comumente resolvidos com árvores.

1. Intuição

Comecemos pensando em fazer uma estrutura que suporte **insert** (adiciona um elemento), **delete** (apaga um elemento), **lower_bound** (encontra o primeiro elemento ≥ x) enquanto mantém os elementos ordenados.

Uma lista encadeada nos permite fazer todas as operações citadas acima de forma simples. Depois de implementar a função **lower_bound** iterando pela lista para encontrar o elemento, as funções de inserção e deleção ficam triviais e se resumem a operações de tempo constante para manter as invariantes da lista. Dessa forma, todas as operações têm a mesma complexidade do **lower_bound**.

Apesar de simples, o problema fundamental da lista encadeada é que o **lower_bound** (ou **find**) tem complexidade X (N) no pior caso, uma vez que é possível que tenhamos que atravessar a lista encadeada inteira para encontrar o elemento.

Para resolver esse problema, poderíamos ter "atalhos" na lista, de forma tal que, ao invés de andar nos elementos um a um, considerarmos saltos maiores para economizar tempo. Como um exemplo simples, se a cada √N-ésimo elemento colocássemos um atalho para um elemento a uma distância √N, conseguiríamos fazer **lower_bound** em O(√N) pegando os atalhos sempre que possível. Evidentemente, com atalhos suficientemente bem espaçados, conseguiríamos melhorar a execução do algoritmo, resta saber como fazer isso de forma eficiente.

2. Como funciona

A ideia fundamental é manter uma lista encadeada em que cada elemento tem uma "altura" $h \in \mathbb{N}$ forçando a seguinte propriedade: um vértice h de altura h terá ponteiros (um para cada "nível" de sua altura) de forma tal que o i-ésimo desses ponteiros aponta para o próximo elemento da lista cuja altura é $\geq i$. Podemos pensar que temos níveis de atalhos, onde os ponteiros mais altos oferecem saltos maiores.

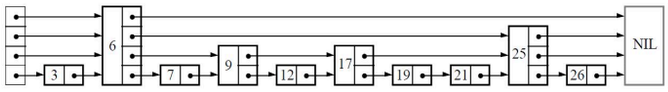


FIGURA 1: SKIP-LIST ONDE OS ELEMENTOS TEM ALTURA MENOR OU IGUAL A 4. NOTE QUE NA ALTURA 1 A ESTRUTURA É UMA LISTA ENCADEADA TRADICIONAL.

Com essa estrutura fica fácil implementar o **lower_bound**: começamos com uma altura *H* (a altura máxima na estrutura) e um ponteiro apontando para o primeiro elemento da lista, a cada passo ou andamos para o elemento apontado pelo ponteiro do elemento atual na altura do momento, ou abaixamos a altura. Em outras palavras: a cada passo do algoritmo tentamos dar o maior passo possível, se o passo for longe demais, passamos a olhar passos menores.

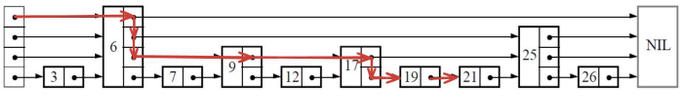


FIGURA 2: EXEMPLO DOS PASSOS DADOS PARA ENCONTRAR O ELEMENTO 21 NO EXEMPLO DADO ANTERIORMENTE.

Resta agora decidirmos a altura de cada elemento de forma tal que garantimos uma boa distribuição dos saltos e, nesse caso, a resposta reside em recorrer ao acaso. Queremos ter menos elementos com alturas maiores para garantir que os saltos em grandes alturas sejam longos (se todo vértice fosse alto, não haveria ganho substancial nos atalhos em grandes alturas).

Para tal, ao adicionar um elemento, sortearemos que ele terá altura h com probabilidade 2^{-h} (onde as alturas de cada nó são sorteadas independentemente). Como intuição, é esperado que ao subir de nível teremos metade do número de elementos que no nível anterior, logo os saltos devem ter o dobro do comprimento, satisfazendo assim a ideia de que níveis altos devem ser menos populosos.

Com essa distribuição é garantido que realizamos **insert**, **delete** e **lower_bound** com complexidade esperada de $\mathcal{O}(\log(N))$, empatando com muitas das soluções em árvores balanceadas.

3. Prova de complexidade

Comecemos citando o Lema da Redução Aleatória [1], um lema extremamente conveniente para análise de algoritmos randomizados, que será usado na nossa demonstração:

Lema 1 (Lema da Redução Aleatória) *Se toda iteração de um algoritmo independentemente reduz o tamanho efetivo do problema por no máximo uma fração constante q < 1 do seu tamanho atual com probabilidade constante p > 0, então esse algoritmo roda em* $\mathcal{O}(\log(n))$ *iterações com alta probabilidade.*

Em outras palavras, o Lema expande o que já é bem conhecido para algoritmos determinísticos: se sempre diminuímos o tamanho do problema por uma razão fixa, o número de passos do algoritmo é da ordem de $\mathcal{O}(\log(n))$.

Para a prova, faremos o caminho contrário do seguido na implementação do **lower_bound**: começaremos no elemento que queríamos encontrar e faremos o caminho voltando para a posição inicial, aumentando a altura sempre que possível e andando para as posições iniciais caso contrário.

Quando na altura h do $i^{\text{ésimo}}$ nó, considere o número de nós à esquerda de *i* com altura pelo menos h, esse número satisfaz a hipótese do Lema da Redução Aleatória com $p = \frac{1}{4}$ e $q = \frac{1}{2}$. Para provar isso, seguem duas afirmações facilmente verificáveis:

Afirmação 1 *Movemos para cima com probabilidade* $\frac{1}{2}$

Movemos para cima se e somente se o ó atual tem altura $>h$, o que ocorre com probabilidade $\frac{1}{2}$ dada a forma como as alturas são sorteadas.

Afirmação 2 *Com probabilidade* $\frac{1}{2}$*, no máximo metade dos nós à nossa esquerda tem altura pelo menos* h *+ 1.*

Chamando de n o número de nós à esquerda da posição atual e X o número de nós com altura pelo menos $h+1$, por simetria temos que $P(X < \frac{n}{2}) = P(X > \frac{n}{2})$ (a probabilidade de um nó ter altura h é a mesma de ter altura $> h$, logo toda distribuição de alturas tem uma distribuição simétrica de mesma probabilidade). Desta forma, $P(X < \frac{n}{2}) \leq P(X < \frac{n}{2}) + \frac{P(X=\frac{n}{2})}{2} \leq \frac{2P(X<\frac{n}{2})+P(X=\frac{n}{2})}{2} = \frac{P(X<\frac{n}{2})+P(X=\frac{n}{2})+P(X>\frac{n}{2})}{2} = \frac{1}{2}$.

Como os eventos das afirmações acima são independentes, a probabilidade de ambos acontecerem (caso em que o tamanho do nosso problema reduz pela metade) é de $\frac{1}{2}\frac{1}{2} = \frac{1}{4}$. Consequentemente, pelo Lema da Redução Aleatória, fazemos $\mathcal{O}(\log(n))$ passos com alta probabilidade. Note que o processo é infinito e, em princípio, podemos ter elementos com altura arbitrariamente alta (com probabilidades baixa). Como um mecanismo de segurança, a maioria das implementações que encontrei colocavam um teto na altura máxima, tipicamente na ordem de log do valor de elementos máximos esperado.

Como cada um desses passos é feito em tempo constante e como todas as as operações da Skip List podem ser implementadas com **lower_bound** mais algumas operações constantes, as operações da Skip List: **insert, delete, lower_bound, find, order_of_key, find_by_order** todas executam em $\mathcal{O}(\log(n))$ com alta probabilidade.

[1] **Minha implementação de Skip List modificada para aceitar operações de order statistics está em pode ser encontrada em https://github.com/bernardoamorim/advanced_algorithms/blob/main/order_statistics_skip_list.cpp**

4. Implementação

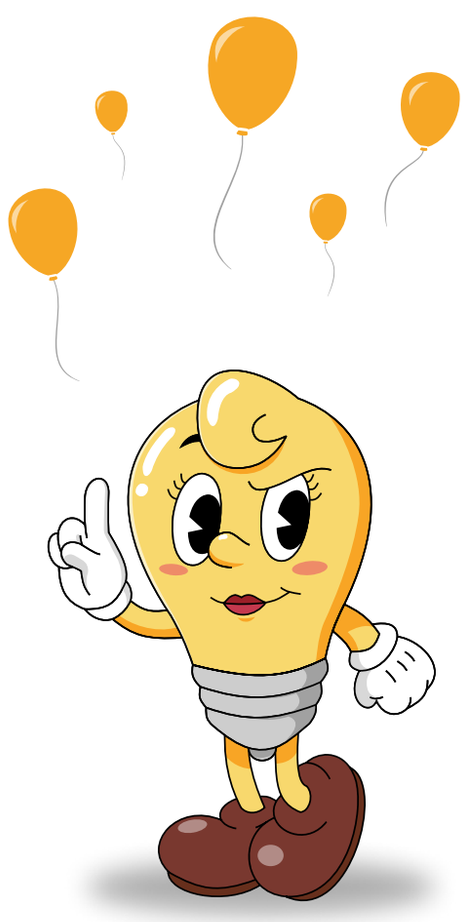
A implementação do algoritmo é direta, sucinta e facilmente modificável [1].

5. Considerações finais

A Skip List propõe um método alternativo de para solucionar vários problemas comumente resolvidos em árvores balanceadas. Apesar de não apresentar ganhos para competições de programação, a Skip List é um bom estudo de caso de análise e desenvolvimento de algoritmos/estruturas de dados que usam de aleatoriedade, em particular, nos permite usar o Lema da Redução Probabilística, um resultado muito conveniente para provas de complexidade.

Referências

[1] Brian C. Dean, Raghuveer Mohan, and Chad G. Waters. Randomized reduction. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, page 259–264, New York, NY, USA, 2015. Association for Computing Machinery.
[2] William Pugh. Skip lists: a probabilistic alternative to balanced trees. Commun. ACM, 33(6):668–676, June 1990.



UMA DÉCADA DE PROBLEMAS DA FASE NACIONAL DA MARATONA DE PROGRAMAÇÃO (2011–2021)

Rafael Granza, UDESC
Yuri Kaszubowski Lopes, UDESC

Esta pesquisa investiga as características dos problemas da Maratona SBC de Programação (MP) ao longo dos anos 2011 e 2021 da competição. O objetivo é mapear e categorizar os problemas e algoritmos apresentados nessas competições, fornecendo informações valiosas tanto para os criadores de problemas quanto para os competidores.

Alguns estudos existentes destacam as tendências globais em tópicos de programação competitiva [2, 8]. A tabela [1] mostra as tendências recentes nas regionais asiáticas do ICPC de acordo com [5]

Categoria	Frequência
Ad Hoc	1-2
(Heavy) Data Structure	0-1
Complete Search (Iterative/Recursive)	1-2
Divide and Conquer	0-1
Greedy (the non-classic ones)	1
Dynamic Programing (the non-classic ones)	1-2
Graph (except Network Flow/Graph Matching)	1
Mathematics	1-2
String Processing	1
Computational Geometry	1
Dome Harde/Rare/Emerging Trend Problems	2-3
Total	**10-17**

TABELA 1: CATEGORIA DE PROBLEMAS RECENTES DAS REGIONAIS ASIÁTICAS DO ICPC, DE ACORDO COM [5]

1. Definição das Categorias e Dificuldades dos Problemas

A definição das categorias foi fundamentada nas categorias adotadas por diversas plataformas de programação competitiva, como Beecrowd, CodeForces, E-olymp e SPOJ. A tabela de todas as categorias pode ser encontrada em [4]. As categorias foram mantidas em inglês, uma vez que muitos dos termos são comumente referenciados em inglês dentro do contexto nacional da MP.

Já as dificuldades foram definidas de acordo com o número de soluções durante a prova, seguindo os seguintes critérios:

- **Impossível**: problema não resolvido durante a competição.
- **Difícil**: com pelo menos 2/3 das soluções provenientes de medalhistas **ou** resolvido por menos de **80% dos medalhistas**.
- **Médio**: entre 1/3 (inclusivo) e 2/3 das soluções provenientes de medalhistas **e** resolvido por **80% ou mais dos medalhistas**.
- **Fácil**: não se enquadra em nenhuma das classificações acima.

As definições de dificuldade adotadas neste trabalho foram estabelecidas de forma que um competidor que resolva todos os problemas fáceis, médios e alguns difíceis tenha boas chances de conquistar uma medalha ou obter uma colocação destacada. Como os problemas fáceis e médios tendem a ser amplamente dominados pelos melhores competidores, é justamente a capacidade de resolver os problemas difíceis que costuma diferenciar os melhores colocados.

2. Pesquisa, dificuldade e categorização de problemas

Os problemas das provas nacionais de programação brasileiras, organizadas pela MP, foram classificados conforme o método descrito na Seção [1]. Os problemas foram pesquisados de 2011 até o ano de 2021. Como exemplo, a Tabela [2] apresenta os resultados dessa categorização e da dificuldade para os problemas da prova nacional de 2021.

Problema	Dificuldade	Categorias
A	**Impossível**	Computational Geometry, plane-geometry, Paradigms
B	**Difícil**	Mathematics, FFT, algebra, Chinese Remainder Theorem
F	**Fácil**	Graph, connected components, DFS as similar
J	**Médio**	Data Structures and Libraries, queue, recursion, sorting

TABELA 2: EXEMPLO DE DIFICULDADES E CATEGORIAS DE ALGUNS PROBLEMAS DA FASE NACIONAL DE 2021.

A dificuldade dos anos de 2011 a 2021 foi baseada em estatísticas de envio disponíveis no site da MP [11]. A categorização foi baseada em várias fontes, cada uma abrangendo um período de tempo diferente. Os dois últimos anos (2020 e 2021) tiveram os *upsolvings* disponibilizados em forma de vídeo no canal oficial Youtube da MP [10]. Os anos de 2014, 2018 e 2019 tiveram questões discutidas nos blogs Codeforces [1, 9, 7]. Em 2017, a MP também forneceu uma solução oficial na forma de slides em seu site [11]. Para os períodos de 2015-2018 e 2012-2019, foram encontrados repositórios on-line de antigos concorrentes [12, 6] com soluções para as questões desses respectivos anos. Para o ano de 2011, não foram encontradas soluções na Internet, mas há um blog no Codeforces que atribui algoritmos e estruturas a cada um dos problemas.

Apesar da referência cruzada de várias fontes, cada problema foi lido e validado para garantir a integridade dos dados. A análise completa pode se encontrado no GitHub dos autores [3].

3. Análise das dificuldades e categorias mais comuns

As médias da ocorrência de cada categoria nas provas analisadas podem ser encontradas na Figura [1]. Com essa figura também é possível calcular a quantidade média de questões da prova, que foi de aproximadamente 11,7 questões.

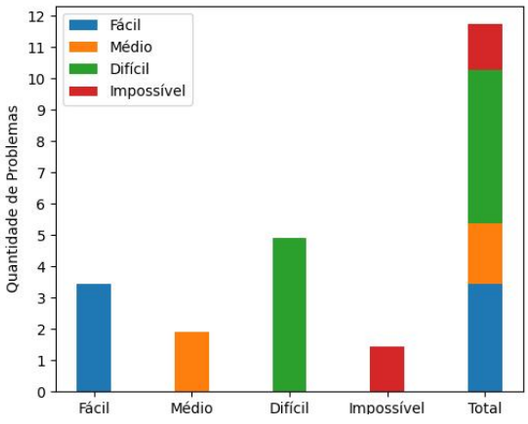


FIGURA 1: FREQUÊNCIA MÉDIA DE CADA DIFICULDADE ENTRE OS ANOS DE 2011 E 2021.

Pode-se observar que os problemas considerados difíceis são, em média, 4,9 questões por ano, o que representa cerca de 41% da MP.

Para melhor analisar as categorias, as Figuras [2] a [6] apresentam as informações da frequência das categorias mais comuns no cenário geral e também separadas por nível de dificuldade.

É possível verificar que a categoria *paradigms* foi a mais frequente tanto no cenário geral (Figura [2]) quanto nos níveis de dificuldade *impossível, Difícil e Médio* (Figuras 3 a 5), demonstrando a importância do estudo dessa área para programadores competitivos.

Outra observação importante diz respeito aos problemas classificados nas categorias Ad-hoc e Implementation. Embora essas categorias sejam as mais frequentes entre os problemas considerados fáceis, elas ocupam a quarta posição quando se analisam os problemas classificados como impossíveis. Esse dado sugere que, apesar de serem comumente vistas como questões simples, problemas nessas categorias podem envolver desafios significativos em termos de implementação.

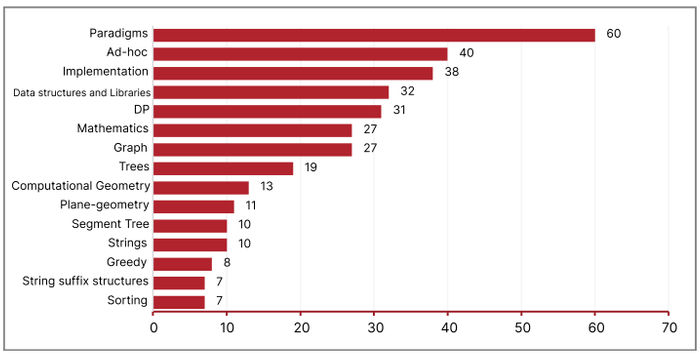


FIGURA 2: FREQUÊNCIA DAS 15 CATEGORIAS MAIS COMUNS NOS PROBLEMAS ANALISADOS.

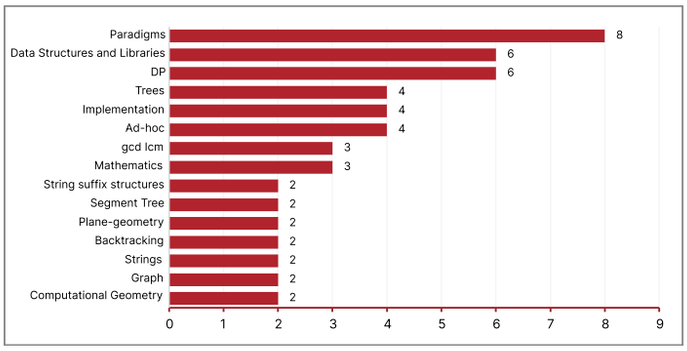


FIGURA 3: FREQUÊNCIA DAS 15 CATEGORIAS MAIS COMUNS NOS PROBLEMAS **IMPOSSÍVEL**.

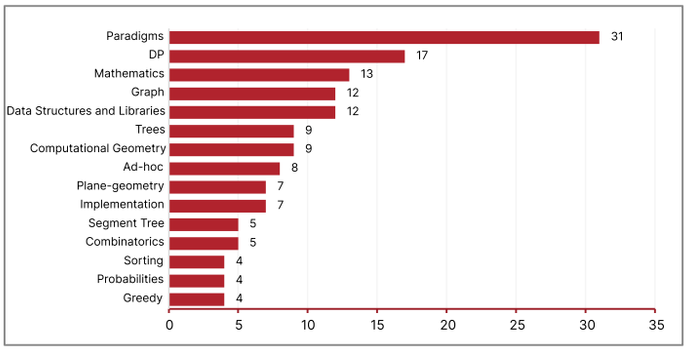


FIGURA 4: FREQUÊNCIA AS 15 CATEGORIAS MAIS COMUNS NOS PROBLEMAS **DIFÍCIL**.

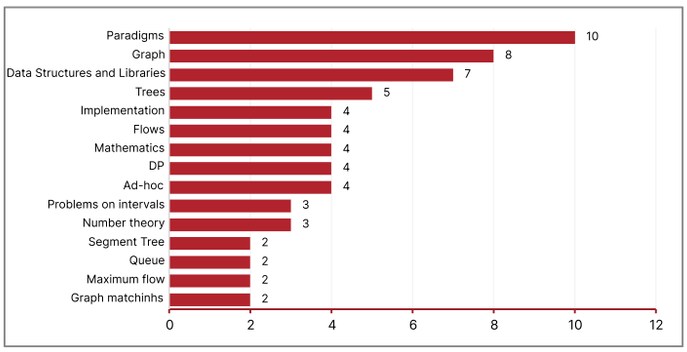


FIGURA 5: FREQUÊNCIA AS 15 CATEGORIAS MAIS COMUNS NOS PROBLEMAS **MÉDIO**.

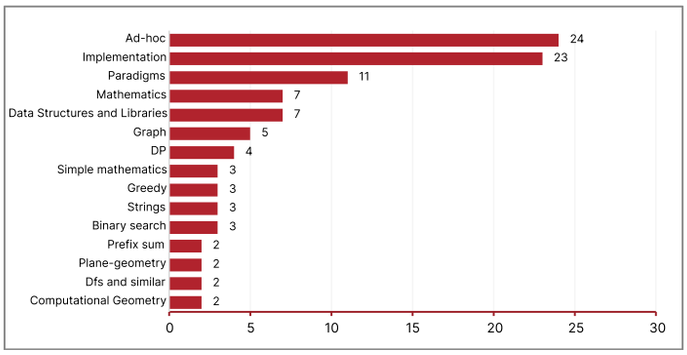


FIGURA 6: FREQUÊNCIA AS 15 CATEGORIAS MAIS COMUNS NOS PROBLEMAS **FÁCIL**.

4. Considerações finais

Os problemas classificados como difíceis, identificados na análise completa das provas disponível em [4] , podem ser utilizados como material de estudo por aqueles que desejam aprimorar suas habilidades na resolução desse tipo de problemas e, consequentemente, aumentar suas chances de conquistar medalhas.

Referências

[1] caioaao. Acm icpc 2014 latin america finals. https://codeforces.com/blog/entry/14650. Data de acesso: 21/11/2022.
[2] T. Di Mascio, L. Laura, and M. Temperini. A framework for personalized competitive programming training. In 2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET), pages 1–8, 2018.
[3] Fype. Brazilian collegiate programming contest's problems analysis. https://github.com/RafaelGranza/Brazilian-Collegiate-Programming-Contest-s-Problems-Analysis. Data de acesso: 2023-02-29.
[4] R. Granza. Uma abordagem de ciência de dados na construção de um almanaque para programação competitiva, 2022.
[5] S. Halim, F. Halim, and S. Effendy. Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s. chapter 1-4. Number bk. 1 in Competitive Programming 4: The New Lower Bound of Programming Contests in the 2020s. Lulu Press, Incorporated, 2018.
[6] jonathandrnd. Solutions acm icpc - latin america. https://github.com/jonathandrnd/ACMICPC_LatinAmericaSolution. Data de acesso: 21/11/2022.
[7] k790alex. The 2019 acm-icpc latin america regional contest. https://codeforces.com/blog/entry/71296. Data de acesso: 21/11/2022.
[8] Juntae Kim, Eunjung Cho, Dongwoo Kim, and Dongbin Na. Problem-solving guide: Predicting the algorithm tags and difficulty for competitive programming problems. arXiv:2310.05971 [cs.CL], 2023. Submetido em: 09/10/2023.
[9] pabloskigms. 2018-2019 acm-icpc latin america regional contest. https://codeforces.com/blog/entry/63167. Data de acesso: 21/11/2022.
[10] Maratona SBC. Canal maratona sbc. https://www.youtube.com/c/MaratonaSBC. Data de acesso: 11/07/2022.
[11] Sociedade Brasileira de Computação. Maratona de programação: O que é? http://maratona.sbc.org.br/sobre21.html. Data de acesso: 10/04/2022.
[12] victoragnez. Icpc latin american regionals and brazil subregionals solutionst. https://github.com/victoragnez/icpc-latin-america-brazil-solutions. Data de acesso: 21/11/2022.
[13] vudduu. Acm icpc latin america 2011. https://codeforces.com/blog/entry/3452. Data de acesso: 21/11/2022.

HISTÓRIAS QUE CONTAM O IMPACTO

Relatos reais de pessoas que tiveram suas vidas transformadas pela Maratona de Programação. Revelam conquistas, superações e aprendizados, inspirando outras pessoas e mostrando o poder de acreditar no próprio potencial.

> Mudou a minha vida e a forma de ver o mundo. Antes eu não acreditava no meu potencial. Por conta da maratona, andei pela primeira vez de avião. Fiz curso de verão na USP e nunca imaginei que um dia ia consegui fazer um mestrado lá, mas consegui. Ganhei prestígio e fama no meu curso, do qual também me valeu meu projeto de estágio e consequentemente meu primeiro emprego. Hoje tenho confiança no meu trabalho, pois já sofri muuuito resolvendo problemas de maratona.

> A experiência de praticar e estudar para maratona foi importante para o desenvolvimento da minha paixão por programação, que até hoje é crucial para me manter motivado em quase 15 anos de carreira.

> A Maratona foi peça chave em todos os fases do meu desenvolvimento: motivação para estudar assuntos considerados difíceis, contato com uma comunidade bastante atuante e tecnicamente habilidosa, e por fim, o currículo/conhecimento adquirido foi bastante coerente/relevante à grande maioria das posições de trabalho no mercado de trabalho na área de tecnologia.

> Tenho certeza que se não fosse a maratona, não estaria onde estou agora. A maratona me fez virar o desenvolvedor que sou hoje, e elevou minha habilidade de resolução de problemas, que é essencial como um engenheiro.

RESOLVENDO PROBLEMAS DE PROGRAMAÇÃO LINEAR COM O ALGORITMO BELLMAN-FORD

João Victor Ayalla, UFAL

A motivação deste artigo surgiu da escassez de blogs em inglês e português sobre o tema, bem como da minha experiência ao tentar resolver o problema G do AtCoder Beginner Contest 404, já que, à primeira vista, eu não fazia a mínima ideia de como resolvê-lo. Foi nesse momento que descobri a ideia apresentada neste artigo.

1. O problema

Queremos encontrar valores para variáveis $v_0, v_1, \ldots, v_n$ que satisfaçam diversas restrições.
A i-ésima restrição é representada por uma tripla (x_i, y_i, z_i), que indica a desigualdade:

$$v_{x_i} - v_{y_i} \leq z_i,$$

onde x_i e y_i são inteiros no intevalo [0,n].

O objetivo é verificar se existe uma solução que satisfaça todas as restrições. Caso não exista, isso deve ser informado.

Se existir pelo menos uma solução viável, deve-se encontrar qualquer uma que minimize o valor de:

$$max(v_i).$$

Ou seja, essa expressão representa o maior valor atribuído a uma das variáveis $v_0, v_1, \ldots, v_n$.

2. Como resolver

Esta seção é fortemente inspirada pelo blog escrito por Horiuchi Sota [2] e nos slides sobre algoritmos de caminhos mínimos (*Shortest Path Algorithms*) elaborados por Jaehyun Park [1].
A grande sacada está em perceber que as restrições podem ser reescritas da seguinte forma:

$$v_{x_i} - v_{y_i} \leq z_i \quad \Rightarrow \quad v_{x_i} \leq v_{y_i} + z_i.$$

Com isso, podemos modelar o problema como um problema em grafos. Vamos construir um grafo com n + 2 vértices. Os primeiros n + 1 vértices representarão as variáveis $v_0, v_1, \ldots, v_n$, ou seja, o vértice i estará associado à variável v_i, $0 \leq i \leq n$.

O vértice adicional, de índice n + 1, será um vértice especial que chamaremos de *source* (origem).

A seguir, adicionamos arestas direcionadas ao grafo da seguinte maneira:

- Para cada restrição (x_i, y_i, z_i), adicionamos uma aresta do vértice y_i para o vértice x_i com peso z_i;
- Para cada i de 0 até n, adicionamos uma aresta do vértice *source* até o vértice i com peso 0.

Assim, seja d_i o custo do caminho mínimo do vértice *source* até o vértice i. Podemos observar que, para cada restrição (x_i, y_i, z_i), a desigualdade

$$d_{x_i} \leq d_{y_i} + z_i$$

é satisfeita. Isso ocorre porque, se existe um caminho mínimo de *source* até y_i, com custo total d_{y_i}, então, ao considerar a aresta de y_i para x_i com peso z_i, obtemos um caminho até x_i com custo total $d_{y_i} + z_i$. Como d_{x_i} representa o menor custo possível para se chegar a x_i, é garantido que $d_{x_i} \leq d_{y_i} + z_i$.

Dessa forma, todas as restrições do problema original são satisfeitas ao atribuímos $v_i = d_i$ para todo i de 0 até n.

Logo, a solução do problema pode ser obtida ao calcular os caminhos mínimos a partir do vértice *source* e usar esses valores como os valores das variáveis $v_0, v_1, \ldots, v_n$.

Entretanto, se o grafo montado tiver algum ciclo de custo negativo, então o problema não possui solução viável. Podemos demonstrar isso da seguinte forma:

Suponha que o grafo contenha um ciclo negativo, ou seja, um ciclo fechado da forma $i_0 \to i_1 \to \cdots \to i_k \to i_0$ cuja soma dos pesos é igual a c < 0.

Se uma solução existisse, então para cada aresta do ciclo a seguinte desigualdade deveria ser satisfeita:

$$d_{i_{j+1}} - d_{i_j} \leq c(i_j, i_{j+1})$$

para todo 0 ≤ j ≤ k, sendo que, para j = k, consideramos $i_{k+1} = i_0$. Agora se somarmos todas essas desigualdades:

$$\begin{aligned} 0 &= (d_{i_1} - d_{i_0}) + (d_{i_2} - d_{i_1}) + \cdots + (d_{i_0} - d_{i_k}) \\ &\leq c(i_0, i_1) + c(i_1, i_2) + \cdots + c(i_k, i_0) = c, \end{aligned}$$

o que nos leva à desigualdade 0 ≤ c, que contradiz a sua posição de que c ≤ 0, e assim, podemos concluir que o problema não possui solução viável nesse caso.

Além disso, como a solução para o problema (caso exista) é construída com base nos caminhos mais curtos a partir do vértice source até os demais vértices, ela também satisfaz a restrição de minimizar $max(v_i)$. Pois, se existisse uma atribuição das variáveis que levasse a um menor valor de $max(v_i)$, então pelo menos uma das distâncias calculadas não corresponderia ao caminho mais curto.

Portanto, podemos utilizar o algoritmo Bellman-Ford para resolver o problema descrito em tempo $\mathcal{O}(nm)$, onde *m* representa a quantidade de arestas presentes no grafo.

Assim podemos, simultaneamente, verificar a existência de ciclos negativos no grafo e calcular os menores caminhos a partir de um vértice *source* até todos os demais vértices. Dessa forma, conseguimos tanto detectar a inviabilidade do sistema de restrições quanto obter uma solução viável para o problema, caso ela exista.

Uma implementação exemplo da ideia descrita neste artigo pode ser encontrada em: https://shorturl.at/abwwA

3. AtCoder Beginner Contest 404 - G

Esta seção descreve a solução para o problema disponível em: https://atcoder.jp/contests/abc404/tasks/abc404_g. Nesse problema, devemos encontrar um array com *N* inteiros positivos que satisfaça diversas restrições do tipo (l_i, r_i, s_i), indicando que a soma dos elementos no subarray $[l_i, r_i]$ deve ser igual a s_i. O objetivo é determinar a menor soma total possível dos elementos do array que respeite todas as restrições, ou informar que não existe solução viável.

Para isso, podemos introduzir *n* + 1 variáveis pref_i para *i* = 0,1, . . . , *n*, onde pref_i representa a soma dos *i* primeiros elementos do array que desejamos encontrar. Em outras palavras, $\text{pref}_i = a_1 + a_2 + \cdots + a_i$. Note que $\text{pref}_0 = 0$.

Assim, para cada restrição (l_i, r_i, s_i), podemos representá-la por meio de duas desigualdades:

$$\text{pref}_{r_i} - \text{pref}_{l_i-1} \leq s_i \quad \text{e} \quad \text{pref}_{l_i-1} - \text{pref}_{r_i} \leq -s_i,$$

o que equivale a impor que a soma do subarray $[l_i, r_i]$ seja exatamente igual a s_i.

Além disso, como cada elemento do array deve ser um inteiro positivo, precisamos garantir que a diferença entre prefixos consecutivos seja ao menos 1. Para isso, adicionamos as restrições:

$$\text{pref}_i - \text{pref}_{i-1} \geq 1 \Rightarrow \text{pref}_{i-1} - \text{pref}_i \leq -1$$

o que assegura que cada elemento $a_i = \text{pref}_i - \text{pref}_{i-1}$ seja positivo. Assim, podemos resolver esse problema utilizando o algoritmo Bellman-Ford como descrito acima.

4. Shanghai Collegiate Programming Contest 2022 - B

Esta seção descreve a solução para o problema disponível em: https://codeforces.com/gym/103931/problem/B. Nesse problema, devemos encontrar uma sequência de parênteses que seja uma *balanced bracket sequence* e que satisfaça diversas restrições do tipo (l_i, r_i, c_i). Cada restrição indica que, ao considerar a substring do índice l_i até r_i, a diferença entre o número de parênteses de abertura '(' e o número de parênteses de fechamento ')' deve ser exatamente igual a c_i. Entretanto, neste problema, o número de restrições *q* pode chegar a até 5×10^5.

Sem considerar, por enquanto, o fato de que o valor de *q* pode ser muito grande, podemos pensar em uma abordagem semelhante à utilizada no Problema G do ABC404.

Vamos representar a sequência de parênteses por um array de inteiros, onde cada parêntese de abertura '(' é representado pelo valor 1, e cada parêntese de fechamento ')' é representado pelo valor 0. Com isso, introduzimos *n* + 1 variáveis pref_i, para i = 0, 1, . . . , n, onde pref_i representa a soma dos *i* primeiros elementos do array.

Dessa forma, podemos modelar o problema adicionando as seguintes restrições:

Para cada uma das *q* restrições da entrada (l_i, r_i, c_i), onde $c_i = \text{open}_i - \text{close}_i$, adicionamos:

$$\text{pref}_{r_i} - \text{pref}_{l_i-1} \leq \text{open}_i \quad \text{e} \quad \text{pref}_{l_i-1} - \text{pref}_{r_i} \leq -\text{open}_i.$$

Além disso, como cada posição do array deve conter um valor 0 ou 1, impomos as restrições:

$$0 \leq \text{pref}_i - \text{pref}_{i-1} \leq 1.$$

Como a sequência de parênteses precisa ser uma *balanced bracket sequence*, também adicionamos:

$$\text{pref}_i - \text{pref}_0 \geq \left\lceil \frac{i}{2} \right\rceil$$

$$\text{pref}_n - \text{pref}_0 = \frac{n}{2}.$$

Entretanto, essa solução possui complexidade $\mathcal{O}(nq)$, o que resultaria no veredito de tempo excedido.
Dessa forma, é necessário ter uma sacada adicionar para resolver o problema: não é preciso considerar mais do que $\mathcal{O}(n)$ restrições do tipo (l_i, r_i, c_i) no sistema de inequações.

Para isso, podemos utilizar alguma estrutura de dados para manter componentes conexas, como a *Disjoint Set Union* (DSU). A ideia é a seguinte: se os índices $l_i - 1$ e r_i pertencem à mesma componente no DSU, então já existe uma relação imposta entre pref_{l_i-1} e pref_{r_i}. Nesse caso, a restrição (l_i, r_i, c_i) é redundante e não precisa ser adicionada ao sistema de inequações.

Assim, ao processar uma nova restrição, basta verificar se $l_i - 1$ pertencem a componentes distintas. Se estiverem na mesma componente, comparamos o valor c_i com a diferença já imposta entre pref_{r_i} e pref_{l_i-1} para checar se existe uma solução viável. Caso estejam em componentes diferentes, juntamos as componentes (*Union*) e adicionamos a restrição ao sistema de inequações a ser processado pelo algoritmo Bellman-Ford.

Com isso, conseguimos otimizar a solução para uma complexidade de $\mathcal{O}(q\alpha(n) + n^2)$ onde $\alpha(n)$ é a função inversa de Ackermann.

5. Considerações finais

Outros materiais sobre essa ideia podem ser encontrados no blog escrito por Horiuchi Sota [2] e nos slides sobre algoritmos de caminhos mínimos (*Shortest Path Algorithms*) elaborados por Jaehyun Park [1], ambos já citados anteriormente. Além disso, os editoriais dos problemas mencionados neste artigo também abordam essa ideia. Além disso, outro problema relacionado ao tema é o http://poj.org/problem?id=3169, no qual o objetivo é maximizar o valor de uma variável, em vez de minimizá-lo. Esse problema serve como um excelente exercício adicional para o leitor.

Referências

[1] Jaehyun Park. Shortest path algorithms - cs 97si - stanford university. https://web.stanford.edu/class/cs97si/07-shortest-path-algorithms.pdf. Data de acesso: 24/06/2025.
[2] Horiuchi Sota. Uma compreensão simples de por que o jogo da vaca se reduz ao problema do caminho mais curto. https://qiita.com/horiso/items/ca81d8f8ddbc5be58ec1. Data de acesso: 24/06/2025.

INTRODUÇÃO À DIGIT DP

Pedro V. Sousa da Silva, UFPR
André L. P. Guedes, UFPR

Digit DP como é mais comumente conhecida, é um tipo de modelagem de Programação Dinâmica (PD) para problemas de contagem, cujo objetivo é encontrar o número de inteiros em um intervalo que satisfazem uma propriedade sobre seus dígitos.

1. Problema Geral

O problema geral resolvido com esta técnica pode ser descrito formalmente como:

Dado dois inteiros l, r $(0 \leq l \leq r \leq 10^{18})$, o objetivo é descobrir quantos inteiros *x*, tal que $l \leq x \leq r$ e *x* satisfaz um predicado *p*(*x*), onde *p*(*x*) é uma propriedade sobre os dígitos de *x* [2].

Exemplos de *p*(*x*):

- *x* termina com o dígito 0
- *x* não possui o dígito 4
- a soma dos dígitos de *x* é um número primo
- *x* não tem os números 4 e 11 como subsequência dos seus dígitos
- a representação decimal de *x* é um palíndromo[1]

Para intervalos enormes como este, apenas iterar sobre cada um dos inteiros e checar se este satisfaz a propriedade dada é muito lento. A *Digit DP* usa os dígitos para rapidamente contar quantos inteiros satisfazem a propriedade no intervalo, fazendo com que a complexidade seja em função do tamanho da representação [3]

2. Passo a Passo

Neste texto será mostrado um passo a passo de como chegar na modelagem clássica desta técnica. O passo a passo será dividido em solucionar os seguintes problemas:

1. Dado um inteiro positivo *n* na forma $n = 10^m - 1$, para algum inteiro positivo *m*, como contar quantos números inteiros não-negativos são menores ou iguais a *n*, a partir de seus dígitos?
2. Dado um inteiro positivo *n*, como contar quantos números inteiros não-negativos são menores ou iguais a *n*, a partir de seus dígitos?
3. Dado um inteiro positivo *n*, como contar quantos números inteiros não-negativos são menores ou iguais a *n* a partir de seus dígitos e satisfazem um predicado *p*?
4. Como resolver o problema geral?

Embora os passos iniciais pareçam triviais, eles serão úteis para a modelagem final do problema.

[1] **Utilizando somente a modelagem clássica não é possível resolver este problema, ele será resolvido na Seção Bônus**

3. Ideia

A ideia principal desta técnica, em sua modelagem clássica, consiste em construir um número a partir dos seus dígitos,do dígito mais significativo para o menos significativo, mantendo a invariante de que o número atualmente construído é menor ou igual a um limitante superior.

4. Solução

Começando com o caso 1, onde o limitante superior *n* tem a forma $n = 10^m - 1$, ou seja, $n \in \{9, 99, 999, 9999, \ldots\}$ Seja $x_0 x_1 \ldots x_{m-1}$ a representação decimal de *x*, onde x_0 é o dígito mais significativo, é possível notar que qualquer número $x = x_0 x_1 \ldots x_{m-1}$ onde $x_i \in \{0, 1, \ldots, 9\}$, é menor ou igual a $n = 10^m - 1$, note que, considerando que já construímos um prefixo de *x*, ou seja, que construímos $x_0 \ldots x_{i-1}$, a escolha de x_i não depende dos dígitos anteriores, isto porque qualquer escolha que tenha sido feita anteriormente, o *x* resultante ainda será menor ou igual a *n*, portanto não precisamos manter quais foram os dígitos escolhidos anteriormente no estado da PD.

Desse modo, a única informação que precisa ser mantida no estado da PD é a posição *i* que estamos considerando no momento. Na transição devemos considerar em x_i todos os dígitos entre 0 e 9, e como caso base verificamos se a posição atual é igual a *m*.

O código resumido abaixo implementa esta ideia, o código relativo a memorização da PD foi omitido.

```
int dp(int i = 0) {
  if (i == m)
    return 1;
  int ans = 0;
  for (int x = 0; x <= 9; x++)
    ans += dp(i + 1);
  return ans;
}
```

CÓDIGO 1 : SOLUÇÃO PARA $n = 10^m - 1$

Para o caso 2, é importante notar que apenas guardar a posição do prefixo que estamos não é suficiente, isso porque já não é verdade que qualquer número que tenha a mesma quantidade de dígitos que *n* é menor ou igual a ele. Considere $n = n_0 \ldots n_{m-1}$ onde $n_0 \ldots n_{m-1}$ é a representação decimal de *n* e considere novamente que estamos construindo um prefixo de *x*, $x_0 \ldots x_i$ onde $i < m$, podemos notar que existem apenas dois casos na escolha de x_i (que garanta que o *x* resultante seja menor ou igual a *n*):

1. Se o prefixo $x_0 \ldots x_{i-1} = n_0 \ldots n_{i-1}$ então x_i pode ser qualquer número entre 0 e n_i.
2. Caso contrário, existe um $j < i$ tal que, o prefixo $x_0 \ldots x_{j-1} = n_0 \ldots n_{j-1}$ e $x_j < n_j$, nesse caso x_i pode ser qualquer número entre 0 e 9.

Essas duas condições podem ser verificados através de uma prova por indução. Intuitivamente a primeira condição,

lida com o caso onde ainda existe a chance de construímos um número maior que *n*, já a segunda condição, lida com o caso, onde já garantimos que o número construído é menor ou igual a *n*. Considerando as condições acima, podemos manter na PD a condição se o prefixo anterior a posição *i* está empatado, ou se já ocorreu o desempate, perceba que esta condição também pode ser mantida sem a necessidade de guardar todos os dígitos anteriores.

O código resumido abaixo implementa essa ideia:

```cpp
int dp(vector<int> &digitos, int i = 0, bool empatado
= true) {
  if (i == digitos.size()) {
    return 1;
  }
  int ans = 0;
  int r = empatado ? digitos[i] : 9;
  for (int x = 0; x <= r; x++){
    bool new_empatado = empatado && (x == r);
    ans += dp(digitos, i + 1, new_empatado);
  }
  return ans;
}
```

CÓDIGO 2 : SOLUÇÃO PARA UM *N QUALQUER*

Para o caso 3, a única adição é a validação de que o número construído satisfaz ao predicado p(*x*), por exemplo, para *p*(*x*) = "*x* não possui o dígito 4'', essa condição pode ser checada a medida que construímos o número atual. O código que implementa esta ideia é mostrado abaixo:

```cpp
int dp(vector<int> &digitos, int i = 0, bool empatado
= true, bool px = true) {
  if (i == digitos.size()) {
    return px;
  }
  int ans = 0;
  int r = empatado ? digitos[i] : 9;
  for (int x = 0; x <= r; x++){
    bool new_empatado = empatado && (x == r);
    ans += dp(digitos, i + 1, new_empatado, px && (x
!= 4));
  }
  return ans;
}
```

CÓDIGO 3 : SOLUÇÃO PARA UM *N* QUALQUER COM *P*(*X*)

Para o caso 4, considere *f*(*n*) := "Quantidade de inteiros entre [0, *n*] que satisfazem *p*(*x*)" , esse problema foi o problema resolvido anteriormente, para resolver o problema para um intervalo *l* ≤ *r* basta computar *f*(*r*) - *f*(*l* - 1).

A complexidade de tempo e espaço da *Digit DP* é $\mathcal{O}(\log(n))$ apenas com esses parâmetros essenciais. Note que, novos parâmetros podem ser adicionados dependendo do problema, por exemplo, se *p*(*x*) é sensível a zeros à esquerda (e.g. *p*(*x*) = "o primeiro dígito de *x* é ímpar"), esse fato pode ocasionar na adição de novos custos a complexidade.

5. Exemplo

Dado um intervalo l , r $(0 \leq l \leq r \leq 10^{18})$, encontre a quantidade de inteiros *x* nesse intervalo cuja soma dos seus dígitos, em base decimal, é um número primo (Adaptado de [6]).

Podemos aplicar a *Digit DP* nesse problema nesse problema com uma ligeira modificação. Pelo fato de só termos a soma dos dígitos do número apenas no final de sua construção, iremos adicionar este parâmetro ao estado da PD, ou seja, a soma dos dígitos fará parte do estado e este valor servirá para, no final da construção, testarmos se a condição *p*(*x*) é válida.

O código abaixo implementa esta ideia:

```cpp
int dp(vector<int> &digitos, int i = 0, bool empatado
= true, int sum = 0) {
  if (i == digitos.size()) {
    return p(sum);
  }
  int ans = 0;
  int r = empatado ? digitos[i] : 9;
  for (int x = 0; x <= r; x++) {
    bool new_empatado = empatado && (x == r);
    ans += dp(digitos, i + 1, new_empatado, sum + x);
  }
  return ans;
}
```

CÓDIGO 2 : SOLUÇÃO PARA O PROBLEMA DO EXEMPLO

A complexidade de tempo nesse caso será: $\mathcal{O}(\log^2(n))$, pois ambos os parâmetros *i* e *sum* estão limitados ao tamanho da representação decimal de *n*. Note que assumimos que é possível realizar o teste de *p*(*x*) em $\mathcal{O}(1)$, que é razoável, dado que seu resultado pode ser pré-computado para faixa de valores que precisamos.

6. Bônus

Também é possível construir ao mesmo tempo um prefixo e um sufixo de *x*, isso é útil para construir números palíndromos. Para isso, além dos parâmetros anteriores, precisamos manter mais uma condição no estado da PD. Com esse objetivo, considere que já construímos um prefixo e sufixo de x, $x = x_0 \dots x_i ? \dots ? x_j \dots x_{m-1}$, onde $j = m - 1 - i$, vamos manter no estado a seguinte condição "a partir de *j* existe alguma posição *k* no sufixo $x_j \dots x_{m-1}$, onde $x_j \dots x_{k-1} = n_j \dots x_{k-1}$ e $x_k > n_k$ ",o caso base será quando $j > i$ e em conjunto com a condição anterior é possível descartar números maiores que *n*.

7. Considerações finais

Utilizando apenas uma modelagem em Programação Dinâmica, esta técnica mostra uma poderosa maneira de resolver um tipo específico de problemas de contagem. Alguns problemas interessantes podem ser encontrados em [4], exemplos de problemas em competições brasileiras incluem [1] e [5] em que ambos utilizam a *Digit DP* como subproblema.

Referências

[1] João Victor Ayalla. Harmonia palindrômica binária. https://codeforces.com/gym/105925/problem/H. Data de acesso: 29/06/2025.
[2] Arpan Banerjee. Digits. In Dynamic Programming for Computing Contests, pages 30–36, 2021.
[3] Jesse Choe. Digit dp. https://usaco.guide/gold/digit-dp. Data de acesso: 17/06/2025.
[4] Tarango Khan. Digit dp. https://codeforces.com/blog/entry/53996. Data de acesso: 07/06/2025.
[5] Célio Passos. Brincando com pedras. https://codeforces.com/gym/103960/problem/B. Data de acesso: 29/06/2025.
[6] Deepak Sharma. Gone - g-one numbers. https://www.spoj.com/problems/GONE/. Data de acesso: 12/08/2025.

JORNADA DE SUCESSO DA MARATONA

LINHA DO TEMPO:
EVOLUÇÃO DA COMPETIÇÃO

Quando tudo começou

1996

Primeira organizada pela SBC

2000

2001

Criação do BOCA

Ultrapassa marca de 100 inscritos

Primeira medalha na Final Mundial

2002

2004

Ultrapassa 500 times

Criação do sistema de classificação com duas fases

2011

2023

Criação da final latino-americana PdA

2025

ULTRAPASSA A MARCA DE 1000 TIMES

O FUTURO DA MARATONA

Nos próximos anos, a competição deve expandir fronteiras e atrair ainda mais participantes. A maratona se tornará cada vez mais inclusiva, abrindo espaço para mais diversidade, inovação e impacto social.

O futuro é de integração global, oportunidades educacionais e parcerias com o mercado de tecnologia. Neste cenário, a Maratona de Programação sempre terá como primeiro objetivo trazer oportunidades aos jovens talentos.

ESTRUTURA DE DADOS DE CONSULTAS EM INTERVALOS MULTIDIMENSIONAIS

Arthur Botelho, UnB

As ideias aqui apresentadas foram publicadas anteriormente pelo próprio autor no Codeforces [2]. O objetivo deste artigo é trazer o conteúdo para a comunidade brasileira, em português e de forma sintetizada e adaptada. Espera-se que o leitor já possua algum conhecimento prévio sobre Álgebra básica e estruturas de dados de consultas em intervalos unidimensionais.

1. Abstração de Consultas em Intervalos Unidimensionais

Estruturas de dados de consultas em intervalos são usadas para resolver problemas da forma:

*Seja * o símbolo para uma operação binária associativa e A uma lista de elementos, onde* A_i *é o elemento na i+1-ésima posição de A (indexação em 0). Dados pares [l, r], calcule o valor de:* $A_l * A_{l+1} * A_{l+2} * \ldots * A_{r-2} * A_{r-1} * A_r$.

Essa definição abstrata objetiva refletir a possibilidade de resolução do problema da mesma forma para diferentes tipos de elementos e operações. Alguns exemplos concretos:

1. Calcule a soma dos números nas posições [*l*, *r*]
2. Calcule a multiplicação das matrizes nas posições [*l*, *r*]

Se alguns elementos individuais A_i são alterados entre consultas, estamos lidando com atualizações em pontos (esse artigo não irá discutir atualizações em intervalos). Se a lista *A* não mudar, estamos lidando com intervalos estáticos.

De acordo com a existência de inverso para a operação e o tipo de problema, pode-se usar estruturas de dados especializadas:

1. Há inverso e os intervalos são estáticos: Soma Parciais ou de Prefixos (Psum) [9]
2. Não há inverso e os intervalos são estáticos: Tabela Esparsa Disjunta (Dist) [5]
3. Há inverso e há atualizações em pontos: Árvore de Indexação Binária ou de Fenwick (BIT) [3]
4. Não há inverso e há atualizações em pontos: Árvore de Segmentos (SegTree) [1]

O leitor deve saber que o que essas estruturas fazem é armazenar o resultado da aplicação da operação * em intervalos específicos, como prefixos da *A* ou sublistas de *A* com tamanho potência de 2. Este artigo irá apresentar uma forma de extrapolar tanto o problema quanto essas soluções de listas (dimensão 1) para dimensões superiores.

2. Consultas em Intervalos Multidimensionais

Descrevemos o problema de consultas em intervalos multidimensionais como (note a restrição de comutatividade):

*Seja * o símbolo para uma operação binária associativa e comutativa e A um conjunto de pontos, cada um deles representados por um valor e por suas coordenadas inteiras* $(x, y, z, \ldots)$*.São dadas consultas da forma* $[(x_1, y_1, z_1, \ldots), (x_2, y_2, z_2, \ldots)]$*; para cada uma delas, calcule o valor de aplicar a operação * sequencialmente aos valores de todos os pontos cujas coordenadas atendem:* $x_1 \leq x \leq x_2$ e $y_1 \leq y \leq y_2$ e $z_1 \leq z \leq z_2$ e $\ldots$

Como exemplos concretos, podemos imaginar que temos uma grade (2*D*) preenchida com números e consultas sobre a soma dos números em subgrades dessa grade, ou um espaço 3*D* onde cada ponto inteiro está associado a um polinômio e consultas sobre o produto de todos os polinômios em um paralelepípedo cujas arestas estão alinhadas com os eixos do espaço.

A forma da solução será semelhante ao caso 1*D*: calcular os resultados da aplicação da operação em alguns intervalos específicos de *A* e responder às consultas unindo esses resultados de forma eficiente. Na verdade, vamos proceder essencialmente da mesma maneira. A única diferença é que, em vez de unir valores individuais (os resultados dos intervalos), estaremos unindo estruturas de dados.

A partir de agora, usaremos as siglas ED para estrutura de dados de consultas em intervalos e EDM para ED multidimensional.

A ideia será manter EDs que operam sobre EDs do mesmo tipo, mas de dimensões menores. Em uma dimensão *D* > 0, nossa EDM manterá uma lista de EDMs internas *D*-1-dimensionais, na qual cada uma está associada a algum intervalo na dimensão *D* (geralmente prefixos ou intervalos cujo tamanho é uma potência de dois). As estruturas de dados 0*D* conterão somente valores, que correspondem aos valores dos pontos do conjunto *A*. Dessa forma, podemos enxergar as EDs 1*D* que conhecemos até agora como operando, na verdade, sobre listas de EDs 0-dimensionais.

Suponha que queiramos responder a uma consulta em um intervalo *D*-dimensional $[(x_1, y_1, z_1, \ldots), (x_2, y_2, z_2, \ldots)]$ usando uma EDM. O intervalo relacionado à dimensão *D* é $[x_1, x_2]$, e vamos supor (recursivamente) que nossas EDMs internas são capazes de resolver consultas de intervalo (*D*-1)-dimensionais da forma $[(y_1, z_1, \ldots), (y_2, z_2, \ldots)]$ — o caso base é uma consulta 0-dimensional [(), ()]: simplesmente retornar um valor. O que queremos é ser capazes de propagar essa consulta (*D*-1)-dimensional para o menor número possível de EDsinternas, utilizando o fato de que elas representam intervalos na dimensão *D*. Cada tipo

de estrutura fará isso de maneira diferente, mas todas constroem sua lista de estruturas internas de forma semelhante: unindo estruturas de dados associadas a intervalos menores.

Vamos entender um pouco melhor o que significa unir estruturas de dados. Suponha que estamos trabalhando em uma dimensão $D > 1$ e atualmente temos a seguinte lista de EDMs internas de dimensão D-1:

1. $EDM_{0,0}$: associada ao intervalo [0, 0] em D.
2. $EDM_{1,1}$: associada ao intervalo [1, 1] em D.

Se o intervalo da consulta na dimensão D fosse [0, 0] ou [1, 1], já saberíamos para onde propagá-la. No entanto, podemos querer unir $EDM_{0,0}$ e $EDM_{1,1}$ para formar $EDM_{0,1}$, para a qual propagaremos as consultas caso o intervalo na na dimensão D seja [0, 1].

No geral, a união de duas estruturas $EDM_{a,b}$ e $EDM_{b+1,c}$ para formar $EDM_{a,c}$ é definida da seguinte forma:

*Para formar todos os intervalos [l, r] para os quais $EDM_{a,b}$ e $EDM_{b+1,c}$ possuem uniões de suas EDs internas, sendo EDM_x a união de [l, r] em $EDM_{a,c}$ e EDM_y o mesmo em $EDM_{b+1,c}$, $EDM_{a,c}$ irá possuir, para [l, r], a união de EDM_x e EDM_y (que têm uma dimensão a menos em relação a $EDM_{a,b}$ e $EDM_{b+1,c}$). Note que essa definição é recursiva, sendo o caso base a dimensão 0, no qual a união de EDM_x e EDM_y será simplesmente aplicar a operação * aos seus valores.*

3. Base da Implementação

Faremos uso de metaprogramação com *templates* [4] e de reticências [6] (" . . . ") em C++ para a definição recursiva das EDM e de suas uniões. O esqueleto do código será:

```cpp
struct EstruturaAlgebrica{
// definicao dos valores e da operacao associativa *
    using T = element_type;
    static constexpr T id = identity_value;
// valor identidade (tal que a * id = a para todo a)
    static T op(T a, T b){return a * b;}
}; // definicao multidimensional:
#define MAs template<class... As>
template<int D, class S> struct EDM{
// dimensao D e estrutura algebrica S
    using T = typename S::T; // tipo dos valores
    int n; vector<ED<D-1, S>> v;
    MAs EDM(int s, As... ds):
        n(s),v(n, EDM<D-1, S>(ds...)){}
    // salvar o tamanho desta dimensao e propagar os
    // tamanhos das dimensoes inferiores (ds)
    MAs T query(int l, int r, As... ps){
        // [l, r]: intevalo da consulta em D
        // suponha que union une as EDs em [l, r]
        return union(l, r).query(ps...);
        // os intervalos (ps) devem ser propagados
        // para as dimensoes inferiores
    }
}; // caso base (0D):
template<class S> struct EDM<0, S>{
    using T = typename S::T;
    T val = S::id;
    T query(){return val;}
}; // inicializada com o valor identidade de *
```

Estamos definindo como a EDM deve funcionar em qualquer dimensão positiva (unindo EDs de dimensões inferiores) e definindo os casos base para os processos recursivos (na dimensão 0). O uso de *templates* variádicos faz com que o número de argumentos para cada versão dos métodos e a tipagem para cada variável em todas as dimensões sejam ajustados corretamente em tempo de compilação.

Para exemplificar a inicialização da estrutura, mostraremos como seria uma SegTree 3D com a operação de soma:

```cpp
template<int D, class S> struct SegTree{
// ... Codigo da estrutura multidimensional aqui
template<class S> struct SegTree<0, S>{
// ... Codigo da versao 0d aqui
struct Soma{ // Estrutura algebrica de soma
    using T = int;
    static constexpr int id = 0;
    static T op(int a, int b){return a+b;}
};
void example(){
    SegTree<3, Soma> seg(3, 4, 5);
// SegTree com dimensoes 3x4x5, inicialmente
// preenchida com valores identidade (0, nesse caso)
    seg.update(10, 1, 4, 2);
// ponto (1,4,2) recebe o valor 10
    seg.update(5, 3, 0, 5);
// ponto (3, 0, 5) recebe o valor 5
    cout << seg.query(1, 1, 0, 4, 1, 2); // = 10
// query de todos os pontos (x, y, z) com
// x em [1, 1], y em [0, 4] e z em [1, 2]
    cout << seg.query(1, 3, 0, 4, 2, 5); // = 15
// ambos os pontos modificados entram nessa query
}
```

Note que essa implementação não é esparsa ou de tamanho dinâmico: o tamanho correspondente em cada dimensão é passado como argumento para o construtor. Além disso, o número de dimensões deve ser conhecido em tempo de compilação, por ser um argumento de *template*. Este artigo não irá tratar de implementações esparsas ou de dimensão variável.

A partir desta base, iremos implementar versões multidimensionais de duas das ED mencionadas na primeira seção. A BIT multidimensional já foi estudada e implementada em [8] e, com detalhamento menor, em [2]. A Dist multidimensional pode ser estudada em [2].

Nos cálculos de complexidade, iremos assumir que a estrutura possui D dimensões e que, em todas elas, o tamanho correspondente é n, para fins de simplicidade.

4. Implementação: Psum

Nesta implementação, a lista ***v*** de EDs internas será indexada em 1 para maior praticidade —— porém, iremos assumir que os métodos da estrutura serão chamados com indexação em 0. Já que, em cada dimensão, ***v*** tem tamanho $n + 1$, a memória usada é $\mathcal{O}(n^D)$.

A estrutura será inicializada com os valores de todos os pontos sendo a identidade da operação. Os valores de cada ponto serão configurados pelo método ***set***:

```cpp
MAs void set(T x, int p, As... ps){
    v[p].set(x, ps...); // (p, ps...) recebe valor x
} // acima: metodo multidimensional
void set(T x){val = x} // metodo da versao 0D
```

Em seguida, para construir a Psum, é preciso processar os prefixos de *v* — para isso, usaremos o método ***init***. Suponha que todas as Psums (*D*-1)-dimensionais de *v* já concluíram o processamento para as dimensões inferiores. Agora, para cada prefixo da lista, simplesmente calcularemos a união de todas as Psums contidas nesse prefixo:

```
void init(){
    for(int i = 1; i <= n; i++){
        v[i].init(); // dimensoes inferioes
        v[i].union(v[i-1]); // uniao do prefixo
    }
} // acima: metodo multidimensional
void init(){} // metodo da versao 0D
void union(Psum& p){ //uniao recursiva de psums
    for(int i = 1; i < n; i++)v[i].merge(p.v[i]);
} // acima: metodo multidimensional
void union(Psum& p){val = S::op(val, p.val);}
// acima: metodo da versao 0D
```

Usando indução em *D* e denotando por *op* a complexidade de ***S::op***, é possível provar que o método ***init*** tem complexidade $\mathcal{O}(op \cdot D \cdot n^D)$.

Depois do processamento de ***init***, pode-se responder consultas usando a mesma lógica de uma Psum 1*D*, mas propagando as consultas para dimensões inferiores. Para isso, vamos considerar que a estrutura algébrica ***S*** implementa uma função adicional ***S::inv,*** que retorna o inverso de um valor em relação a ***S::op*** (no caso de soma, ***S::inv(a)*** retornaria ***-a***).

```
MAs T query(int l, int r, As... ps){
    T vr = v[r+1].query(ps...);
    T vl = v[l].query(ps...);
    return S::op(vr, S::inv(vl));
} // acima: metodo multidimensional
T query(){return val;} // metodo da versao 0D
```

Denotando por *inv* a complexidade de ***S::inv***, pode-se verificar que o método ***query*** tem complexidade $\mathcal{O}((op + inv) \cdot 2^D)$ Com isso, podemos juntar esses métodos para concluir a implementação de uma Psum multidimensional completa:

```
#define MAs template<class... As>
template<int D, class S> struct Psum{
    using T = typename S::T;
    int n; vector<Psum<D-1, S>> v;
    MAs Psum(int s, As... ds) :
        n(s+1), v(n, Psum<D-1, S>(ds...)){}
    MAs void set(T x, int p, As... ps){
        v[p+1].set(x, ps...);
    }
    void init(){
        for(int i = 1; i < n; i++){
            v[i].init();
            v[i].union(v[i-1]);
        }
    }
    void union(Psum& p){
        for(int i = 1; i < n; i++)
            v[i].merge(p.v[i]);
    }
    MAs T query(int l, int r, As... ps){
        T vr = v[r+1].query(ps...);
        T vl = v[l].query(ps...);
        return S::op(vr, S::inv(vl));
    }
};

template<class S> struct Psum<0, S>{
    using T = typename S::T;
    T val=S::id;
    void set(T x){val = x;}
    void union(Psum& p){val = S::op(val, p.val);}
    void init(){}
    T query(){return val;}
};
```

5. Implementação: SegTree

Nesta implementação, iremos indexar a raiz em 1 e a lógica das consultas será baseada em intervalos semiabertos ([*l*, *r*]). No entanto, ainda consideraremos que os métodos de consulta e atualização são chamados com indexação em 0 e intervalos fechados ([*l*, *r*]). Usaremos a implementação iterativa como base, descrita em [1]. Como a lista *v* de EDs internas da SegTree terá tamanho 2*n*, a complexidade de memória é $\mathcal{O}((2 \cdot n)^D)$.

Precisaremos responder a consultas *D*-dimensionais da forma $[(l_1, l_2, \ldots), (r_1, r_2, \ldots)]$, supondo (recursivamente) que já sabemos como responder a consultas em uma SegTree de dimensão *D*-1. Nesse caso, responder à consulta será simplesmente computar a união das respostas das consultas de dimensão *D*-1 $[(l_2, \ldots), (r_2, \ldots)]$, fornecidas pelas EDs internas cujos intervalos associados são disjuntos e cobrem completamente o intervalo $[l_1, r_1]$ — como em uma SegTree unidimensional.

```
MAs T query(int l, int r, As... ps){
    T lv=S::id, rv=S::id;
    for(l += n, r += n+1; l < r; l /= 2, r /= 2){
        if (l&1)lv = S::op(lv, v[l++].query(ps...));
        if (r&1)rv = S::op(v[--r].query(ps...), rv);
    }
    return S::op(lv, rv);
} // acima: metodo multidimensional
T query(){return val;} // metodo da versao 0D
```

Usando indução, pode-se verificar que o método ***query*** tem complexidade $\mathcal{O}(op \cdot (2 \cdot \log(n))^D)$. Porém, além de consultas, uma SegTree deve suportar atualizações da forma: torne *x* o valor do ponto nas coordenadas $(p_1, p_2, \ldots)$

Primeiramente, atualizaremos a ED interna (uma folha) correspondente ao intervalo $[p_1, p_1]$, propagando uma atualização com o valor *x* no ponto $(p_2, \ldots)$. Em seguida, atualizaremos sucessivamente os ancestrais dessa folha. Em cada um deles, propagaremos uma atualização no ponto $(p_2, \ldots)$ com um valor *y*: a união dos valores dos pontos $(i, p_2, \ldots)$ para todo *i* no intervalo correspondente do ancestral. Esse *y* pode ser facilmente calculado: simplesmente a união dos valores do ponto $(p_2, \ldots)$ nos filhos esquerdo e direito do ancestral.

```
// metodos multidimensionais:
MAs T get(int p, As... ps){return v[p+n].get(ps...);}
MAs void update(T x, int p, As... ps){
    v[p+=n].update(x, ps...);
    while(p/=2){
        T fl = v[2*p].get(ps...);
        T fr = v[2*p+1].get(ps...);
        v[p].update(S::op(fl, fr), ps...);
    }
} // metodos da versao 0D:
T get(){return val;}
void update(T x){val = x;}
```

Dessa forma, a complexidade do método ***update*** é $\mathcal{O}(op \cdot \log(n)^D)$. Pode-se juntar esses métodos para concluir a implementação de uma SegTree multidimensional (exemplo disponível em: https://github.com/arthur9548/Competitive-Programming/blob/main/Code/DataStructures/MDRQDS/segtree.cpp).

6. Conclusão

Espera-se que as ideias apresentadas facilitem o entendimento de problemas de consultas em intervalos multidimensionais. As implementações estudadas foram desenvolvidas com o objetivo de serem escaláveis, genéricas, compreensíveis e eficientes. O blog no Codeforces [2] apresenta os mesmos conteúdos, com mais detalhes, porém em um formato diferente e em língua inglesa.

A complexidade das EDM apresentadas escala exponencialmente no número de dimensões, tanto em tempo quanto espaço. Para certos problemas, podem ser necessárias técnicas específicas de otimização. Este artigo não abordou tópicos recorrentes em soluções de consultas em intervalos, como compressão, persistência, atualizações em intervalo e estruturas esparsas.

Em [2] são sugeridos problemas que utilizam todas as estruturas mencionadas no artigo. Como exercício adicional, recomenda-se também implementar versões multidimensionais de outras EDs não mencionadas, como a Tabela Esparsa [7].

Referências

[1] Oleksandr Bacherikov. Efficient and easy segment trees. https://codeforces.com/blog/entry/18051. Data de acesso: 28/06/2025.
[2] Arthur Botelho. Multidimensional range query data structures. https://codeforces.com/blog/entry/138583. Data de acesso: 28/06/2025.
[3] Jakob Kogler et al. Fenwick tree¶. https://cp-algorithms.com/data_structures/fenwick.html. Data de acesso: 28/06/2025.
[4] GeeksforGeeks. Template metaprogramming in c++. https://www.geeksforgeeks.org/template-metaprogramming-in-c/. Data de acesso: 28/06/2025.
[5] Nilesh Hirani. [tutorial] disjoint sparse table. https://discuss.codechef.com/t/tutorial-disjoint-sparse-table/17404. Data de acesso: 28/06/2025.
[6] Microsoft Learn. Ellipsis and variadic templates. https://learn.microsoft.com/en-us/cpp/cpp/ellipses-and-variadic-templates?view=msvc-170. Data de acesso: 28/06/2025.
[7] Thiago Mota. Ideia 08 – sparse table. https://noic.com.br/materiais-informatica/ideias/ideia-08/. Data de acesso: 28/06/2025.
[8] mouse.wireless. Nifty implementation of multidimensional binary indexed trees using templates. https://codeforces.com/blog/entry/64914. Data de acesso: 28/06/2025.
[9] Darren Yao and Dustin Miao. Introduction to prefix sums. https://usaco.guide/silver/prefix-sums?lang=cpp. Data de acesso: 28/06/2025.

Vista como um espaço de formação técnica, de geração de oportunidades para a inserção de estudantes no mercado de trabalho, de um espetáculo competitivo, a Maratona de Programação possui um lado não competitivo e extremamente colaborativo.
A Maratona é patrocinada por algumas das maiores empresas em atuação no país e, nossos medalhistas qualificam-se para a Final Latino Americana e para a Final Mundial. Os estudantes disputam a possibilidade de competir internacionalmente. São mais de 20 mil participantes desde 1996.
Os textos podem até refletir a opinião dos editores, mais que isso, buscou-se criar um espaço para que, cada um, possa compartilhar seu ponto de vista. Os artigos... talvez uma novidade, talvez algo conhecido, os que foram aqui incluídos, o foram por terem sido julgados como de interesse da comunidade da Maratona.

WWW.FACEBOOK.COM/MARATONA

WWW.INSTAGRAM.COM/
MARATONADEPROGRAMACAO

X.COM/MARATONAPROG

WWW.LINKEDIN.COM/COMPANY/MARATONA-DE-PROGRAMACAO

WWW.YOUTUBE.COM/@MARATONASBC

MARATONADEPROGRAMACAOSBC@GMAIL.COM

COMENTE NAS REDES:

#MARATONASBC
#MARATONA30ANOS
#MARATONADEPROGRAMAÇÃO