"Planeta Água: a cultura oceânica para enfrentar as mudanças climáticas no meu território"
SECT - ICE
XII Semana de Ciência e Tecnologia SECT ICE
20 a 23 de Outubro de 2025
Realização:
INSTITUTO DE CIÊNCIAS EXATA

Uma análise comparativa do desempenho de filas de prioridade aplicadas ao algoritmo de Dijkstra

Ana Carla Fernandes[1], Lucas Castro[2], Rosiane de Freitas-Rodrigues[3]

Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM) Av. Gen. Rodrigo Octávio, 6200, Coroado I, 69080-900 – Manaus – AM – Brasil.

[1] *ana.fernandes@icomp.ufam.edu.br,* [2] *lucas.castro@icomp.ufam.edu.br,* [3] *rosiane@icomp.ufam.edu.br*

Palavras-Chave: algoritmo de Dijkstra, caminhos mínimos, filas de prioridade.

Introdução

O problema de caminhos mínimos consiste em encontrar o trajeto de menor custo entre uma origem e um ou mais destinos, considerando critérios como distância, tempo ou custo. A modelagem desse problema em grafos — em que vértices representam os pontos e as arestas ponderadas, os caminhos entre eles (Fig. 1) — viabiliza soluções computacionais para diversas áreas, como telecomunicações, navegação autônoma e otimização de rotas logísticas.

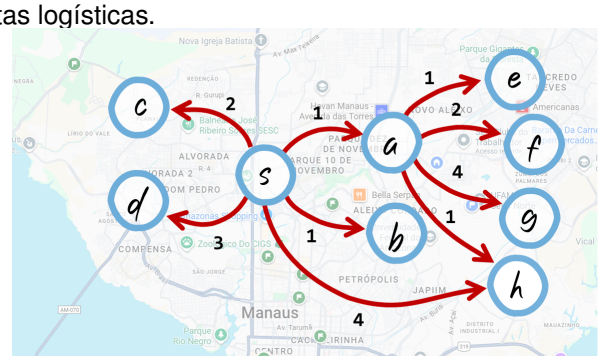

Figura 1: Representação de malha urbana em grafos

O algoritmo de Dijkstra[2] se destaca como uma solução clássica para esse problema, utilizando uma estratégia gulosa para calcular o trajeto ótimo entre um vértice e todos os demais em tempo polinomial. No entanto, seu maior gargalo em desempenho é obter a aresta de menor peso a cada iteração, uma operação custosa que exige uma estrutura de dados auxiliar: a fila de prioridade (*heap*).

Nesse contexto, surgem questões fundamentais para a aplicação prática em sistemas de resposta rápida: o desempenho teórico dos *heaps* descritos na literatura corresponde ao desempenho real? Existem *heaps* cujo desempenho é melhor em casos especiais do problema de caminhos mínimos?

Para responder a essas perguntas, foi realizado um estudo comparativo do desempenho de diferentes *heaps* aplicados ao algoritmo de Dijkstra. Diversas estruturas de dados foram selecionadas, implementadas, e submetidas a testes empíricos utilizando grafos esparsos. Este trabalho é um resultado preliminar de uma pesquisa em andamento do Programa Institucional de Bolsas de Iniciação Científica (PIBIC) da UFAM.

Material e Métodos

O estudo foi conduzido mediante uma revisão bibliográfica para a seleção das estruturas de dados a serem analisadas. Em seguida, procedeu-se a implementação das estruturas e os respectivos testes de desempenho. A análise final comparou os tempos de execução e os desempenhos esperados com base nas complexidades assintóticas. Foram avaliadas as seguintes estruturas:

Implementações de bibliotecas:

- Heap Binário (*priority_queue<>*) e Árvore Rubro-Negra (*set<>*) da biblioteca STL (Standard Template Library) do C++;
- Heap de Fibonacci da biblioteca Boost[1].

Implementações próprias[4]:

- Dial;
- 2-level Bucket Heap;
- 4-level Bucket Heap.

O Heap Binário, a Árvore Rubro-Negra e o Heap de Fibonacci foram selecionados por possuírem implementações de bibliotecas consolidadas. Essas estruturas são baseadas em árvores e mantêm ordenação total ou parcial dos elementos, otimizando as operações de inserção e remoção exigidas pelo algoritmo de Dijkstra. Suas complexidades dependem do número de vértices (n) e arestas (m) (Tabela 1).

Estrutura	Inserir	Extrair menor	Dijkstra
Binário	$O(\log(n))$	$O(\log(n))$	$O((m+n)\log(n))$
Rubro-Negra	$O(\log(n))$	$O(\log(n))$	$O((m+n)\log(n))$
Fibonacci	$O(1)^*$	$O(\log(n))^*$	$O(m+n\log(n))^*$
Dial	$O(1)$	$O(C)$	$O(m+nC)$
2-Level Bucket	$O(1)$	$O(\sqrt{C})$	$O(m+n\sqrt{C})$
K-level Bucket	$O(1)$	$O(k+C^{1/k})$	$O(km+n(k+C^{1/k}))$

*Amortizado

Tabela 1: Complexidade de tempo das filas de prioridade aplicadas ao Dijkstra

Como os grafos da base de dados utilizam pesos inteiros, também foram avaliados o algoritmo de Dial e suas variantes 2 e 4-Level Bucket, otimizadas para esse tipo de grafo. Nessas abordagens, os vértices são distribuídos em k níveis de "baldes" (*buckets*) conforme suas distâncias da origem[2]. Suas complexidades de tempo

dependem do peso máximo (C) das arestas do grafo (Tabela 1).

Os experimentos utilizaram grafos esparsos disponíveis no site do "*9th DIMACS Implementation Challenge*"[3], que representam as malhas rodoviárias da região central do Estados Unidos, e três estados, Baía de São Francisco, Flórida e Califórnia. A seleção dos grafos se baseou nas variedades de quantidades de vértices e arestas, que podem ser conferidos na Tabela 2.

Região	Vértices (n)	Arestas (m)	Peso máximo
usafla	1070376	2712798	214013
usacal	1890815	4657742	215354
usabay	321270	800172	94305
usacr	14081816	34292496	214013

Valor máximo

Tabela 2: Características dos grafos

O ambiente computacional para execução dos testes consistiu em um sistema com processador Intel(R) Core(TM) @ 2.40 GHz, 8,00 GB de RAM e Windows 11. Os códigos foram implementado em C++ e compilados com o G++ 14.2.0.

Resultados e Discussão

A Figura 2 apresenta a média dos tempos de 10 execuções do algoritmo de Dijkstra com cada fila de prioridade. Os *heaps* baseados em *buckets* obtiverem os melhores resultados, com o 2-Level Bucket Heap apresentando o menor tempo médio de execução.

Conforme a Tabela 1, a análise exclusiva da complexidade assintótica gera a expectativa de um desempenho superior para as estruturas baseadas em árvore. Contudo, o melhor desempenho dos *bucket heaps* pode ser atribuído à operação de inserção em $O(1)$ e ao padrão de acesso sequencial à memória, que confere a essas estruturas uma natureza mais eficiente em termos de *cache*. Esse comportamento contrasta com as operações das estruturas em árvore, cujos acessos aleatórios e "saltos" na memória intensificam o *cache miss* e degradam o desempenho.

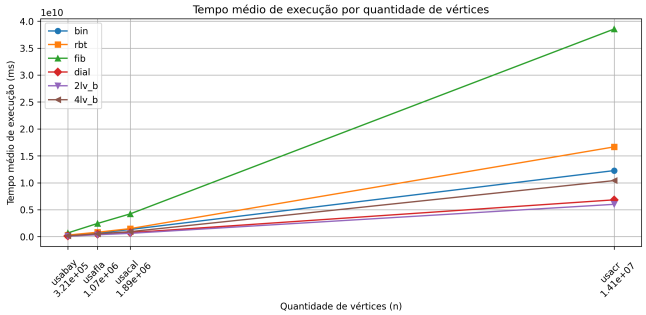

Figura 2: Tempo médio de 10 execuções por quantidade de vértices

Dentre os *bucket heaps*, o 2-Level Bucket demonstrou desempenho superior em comparação ao Dial. Contudo, apesar da complexidade assintótica apontar para ganhos potenciais com a adição de níveis, essa tendência não se manteve no 4-Level Bucket. Isso pode ser atribuído ao custo computacional de operações modulares e à implementação recursiva de acesso aos níveis. Para investigar as vantagens de estruturas multinível, abordagens iterativas serão incorporadas em experimentos futuros.

Apesar da mesma complexidade teórica, o Heap Binário superou a Árvore Rubro-Negra, possivelmente devido à ordenação parcial e uso de vetor ao invés de ponteiros. O Heap de Fibonacci, embora teoricamente superior, apresentou o pior desempenho, resultado atribuído às constantes ocultas e às estruturas auxiliares. Nos próximos experimentos será investigado o impacto da operação *decrease-key*, que mantém constante as quantidades de chaves no *heap*, o que pode melhorar o desempenho.

Conclusões

Os resultados preliminares demonstram que a escolha da estrutura de dados para a fila de prioridade exerce influência direta no desempenho do algoritmo de Dijkstra. Embora árvores apresentem desempenho estável e amplamente difundido nas bibliotecas, os resultados sugerem que abordagens específicas para grafos com pesos inteiros, como os *bucket heaps*, podem alcançar ganhos de eficiência.

Os desempenhos inferiores do Heap de Fibonacci em relação à todas as estruturas e do 4-Level Bucket Heap em relação às estruturas em *buckets* reforçam que a complexidade assintótica não é indicativo absoluto de eficiência prática devido às constantes ocultas e *cache-miss*. No entanto, detalhes da implementação e características das bases de dados podem influenciar os resultados, o que exige mais investigação.

Conclui-se que a seleção da fila de prioridade ideal depende das características do grafo e do contexto de aplicação. Para sistemas que exigem resposta rápida em tempo real e que possuem grafos com pesos inteiros, *bucket heaps* representam alternativas promissoras. Em experimentos futuros serão explorados os desempenhos das estruturas em outras classes de grafos e avaliados o uso de memória e impacto da operação *decrease-key*.

Agradecimentos

Este trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES-PROEX) - Código de Financiamento 001, pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), bem como parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas – FAPEAM – por meio do projeto POS-GRAD 2024/2025.

Referências

[1] Boost C++ Libraries. Boost c++ libraries. `http://www.boost.org/`. 2015.

[2] Costa, J.; Castro, L.; de Freitas, R. Exploring monotone priority queues for dijkstra optimization. *RAIRO-Operations Research*, 59(5):2419–2436. 2025.

[3] Demetrescu, C.; Goldberg, A.; Johnson, D. *9th DIMACS Implementation Challenge: Shortest Paths*. American Mathematical Society. 2006.

[4] Fernandes, A. C. Implementações dos bucket heaps. `https://gist.github.com/anacarlaaf/1fc932ee7d394202e1b5adb6f5f409e0`. Acessado em: 13/10/2025. 2025.