Trabalho de análise de algoritmos de árvore geradora mínima de Kruskal e Prim - Projeto e Análise de Algoritmos

Clarisse Midori Yoshimura Torres
Maria Carolina Tomain Rodrigues

Novembro, 2024

Instituto de Biociências, Letras e Ciências Exatas, Unesp - Universidade Estadual Paulista

Sumário

Lista de Figuras

1 Introdução

O presente trabalho tem como foco a implementação comparativa dos algoritmos de Kruskel e Prim, de modo que através da mensuração de tempo e visualização gráfica, seja possível avaliar o desempenho relativo de ambas implementações em diferentes volumes de entrada.

Utilizando a linguagem de programação Python, a comparação envolverá grafos densos, nos quais o número de arestas se aproxima do máximo possível, e grafos esparsos, com poucas arestas em relação ao número de vértices, sendo todos os grafos gerados conexos. Com base nos resultados obtidos, será possível identificar qual algoritmo apresenta o melhor desempenho em diferentes condições, fornecendo insights sobre suas eficiências em cenários práticos.

2 Algoritmo de KrusKal

A árvore geradora mínima de Kruskal parte de uma aresta segura para desenvolver a floresta em formação. A partir desse elemento inicial, a cada etapa, o algoritmo verifica entre todas as arestas que conectam duas árvores do grafo aquela que possuí o **peso mínimo** e que não forma um ciclo com as arestas já selecionadas.

O algoritmo possuí uma abordagem **gulosa** e seu tempo de execução é de **O(E logE)**, sendo E o número de arestas a serem ordenadas.

Além disso, para que seja possível para o algoritmo realizar tais operações, a árvore geradora mínima de Kruskal utiliza de estruturas denominadas **Union-Find**.

2.1 Union-Find

A estrutura Union-Find é utilizada para gerenciar conjntos de elementos que não possuem elementos na intersecção. Para tal manutenção de subconjuntos, a estrutura utiliza das operações **Find** e **Union**, responsáveis por determinar para qual subconjunto o elemento deverá er direcionado e por unir dois subconjuntos em um único, respectivamente [**?**].

Devido a natureza da árvore geradora mínima de Kruskal, o algoritmo necessita da estrutura Union-Find para gerenciar os subconjuntos de arestas presentes no gráfico. Dessa forma, essa estrutura é responsável por evitar a formação de ciclos durante a construção da floresta, verificando se os dois vértices que são conectados pela aresta estão no mesmo subconjunto, se não estiverem, a aresta é adicionada à árvore e os vértices são unidos em um único conjunto.

2.2 Evolução do tempo conforme entradas de grafos densos

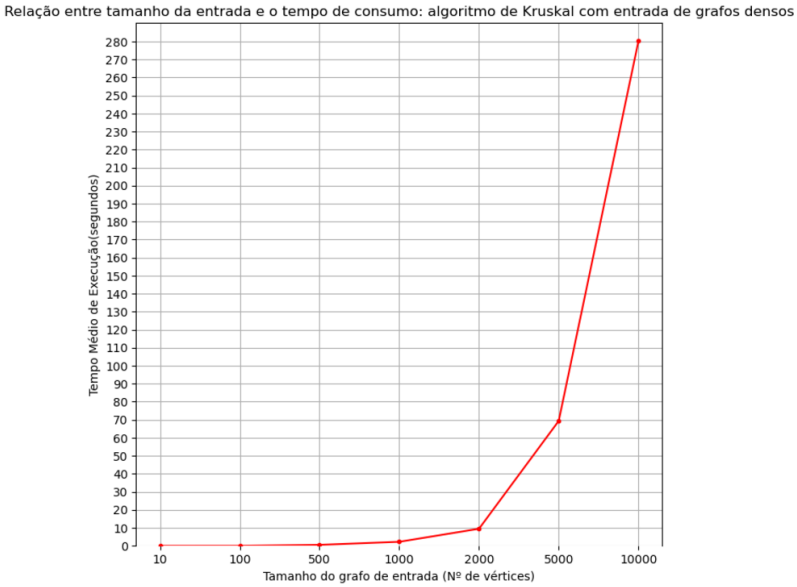

Figura 1: Gráfico de evolução com entradas de grafos densos

2.3 Evolução do tempo conforme entradas de grafos esparsos

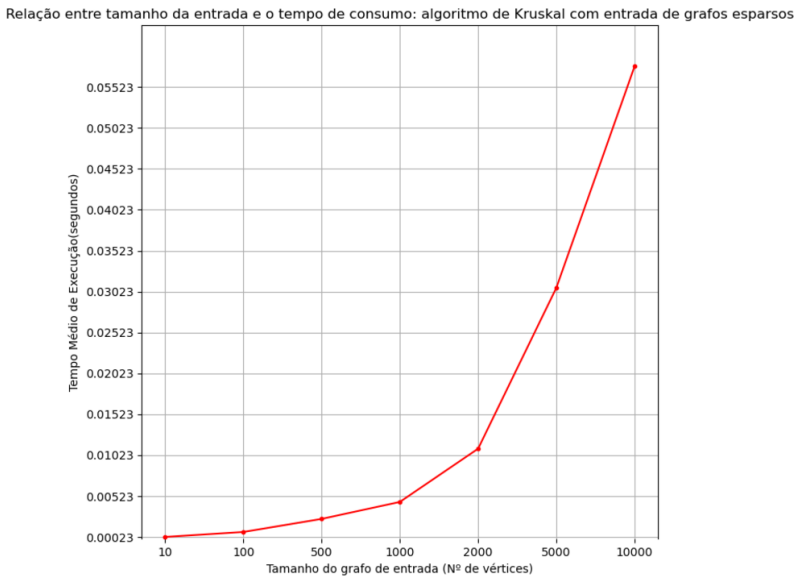

Figura 2: Gráfico de evolução com entradas de grafos esparsos

2.4 Comparativo para os dois tipos de entradas de grafos

Figura 3: Gráfico comparativo da evolução temporal, conforme variação de entrada de grafo

3 Algoritmo de Prim

Diferente da abordagem do algoritmo de Kruskal, o algoritmo de Prim é inicializado a partir de um vértice arbitrário, expandindo a AGM a partir da selecção dinâmica das arestas de um grafo. A cada etapa, o algoritmo seleciona a aresta de **menor peso** que conecta um vértice já incluído na árvore com um vértice ainda não incluso no subconjunto.

Bem como o algoritmo de Kruskal, a árvore geradora mínima de Prim possuí uma abordagem **gulosa** e complexidade de **O(ElogV)**, sendo **E** o numero de arestas e **V** o número de vértices do grafo.

Para realizar tal abordagem, o algoritmo de Prim utiliza de uma **fila de prioridades** para auxiliar na seleção da aresta de menor peso.

3.1 Fila de Prioridade

Uma fila de prioridade pe uma estrutura de dados que permite armazenar dados a partir da associação de chaves. Cada elemento tem uma prioridade de modo que o elemento de maior ou menor prioridade é extraído da fila. Para realizar tal operação, a fila de prioridade consome um tempo de **O(logn)** e utiliza de um **heap máximo ou mínimo** (como é o caso do algoritmo de Prim).

Com isso, a fila é responsável por ordenar as arestas e determinar qual será o próximo vértice a ser adicionado na árvore que contrnha o menor custo (peso) de aresta em um tempo de **O(logn)**.

3.2 Evolução do tempo conforme entradas de grafos densos

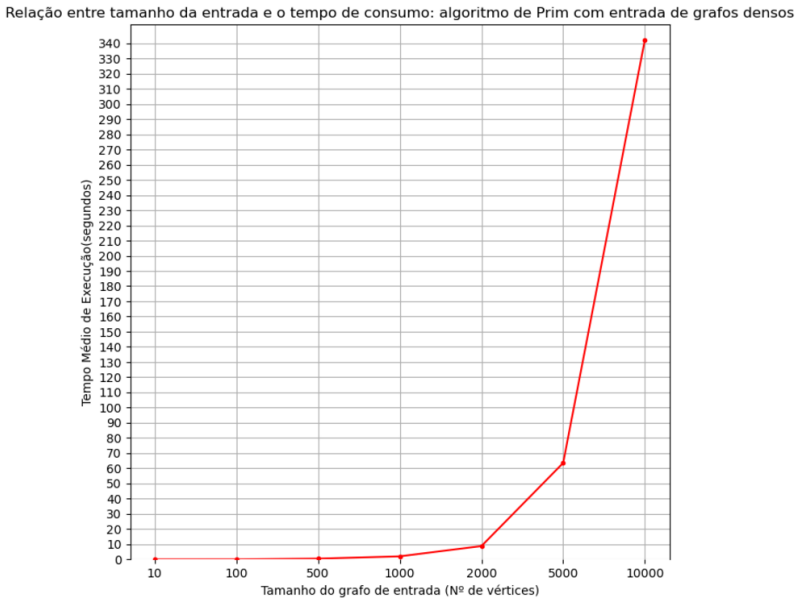

Figura 4: Plotagem do gráfico de evolução com entradas de grafos densos

3.3 Evolução do tempo conforme entradas de grafos esparsos

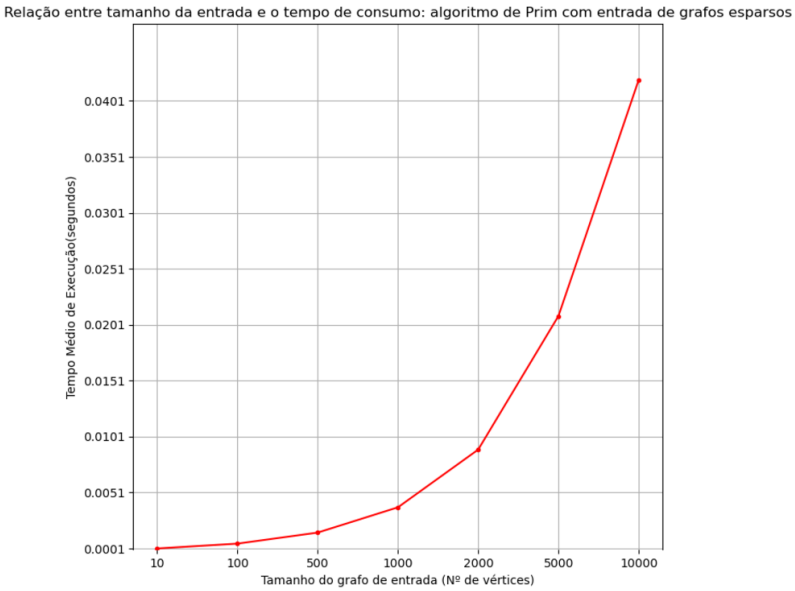

Figura 5: Plotagem do gráfico de evolução com entradas de grafos esparsos

3.4 Comparativo para os dois tipos de entradas de grafos

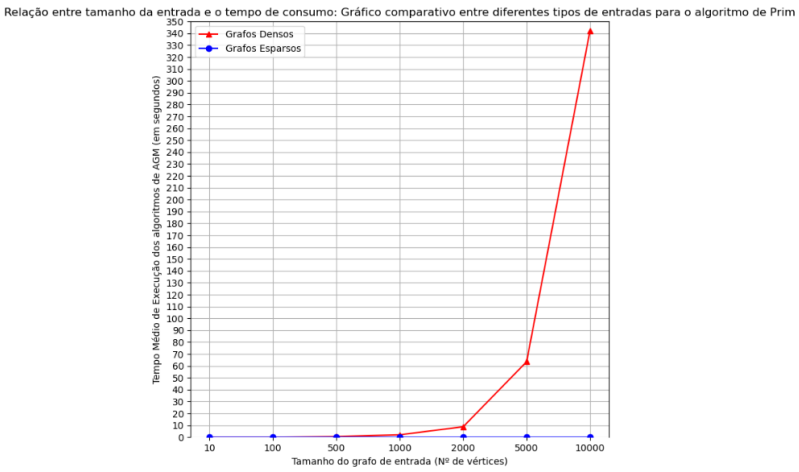

Figura 6: Gráfico comparativo da evolução temporal, conforme variação de entrada de grafo

4 Comparação entre os algoritmos de KrusKal e de Prim

4.1 Comparação entre os desempenhos dos algoritmos com entradas densas

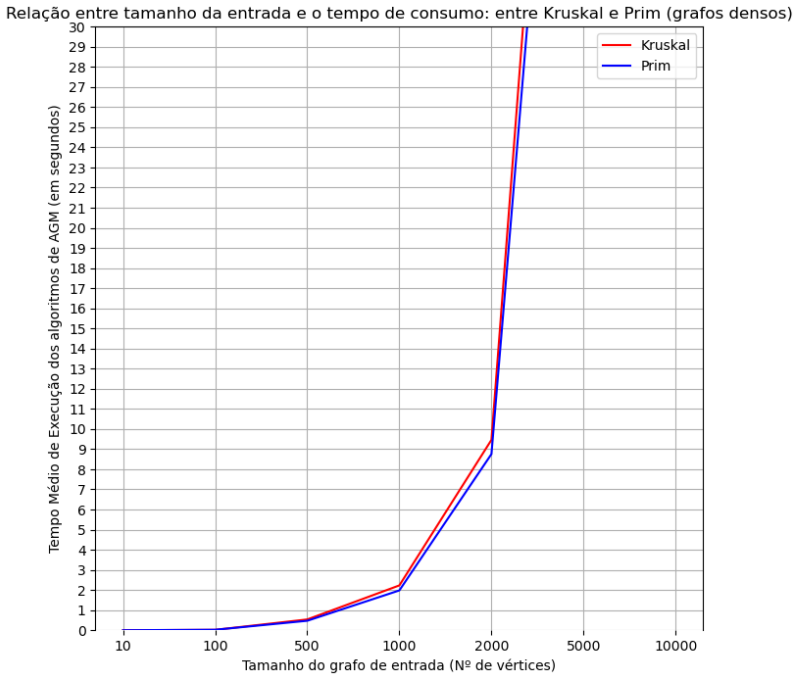

Figura 7: Gráfico de comparação com entradas de grafos densos

4.2 Comparativo para os dois tipos de entradas de grafos

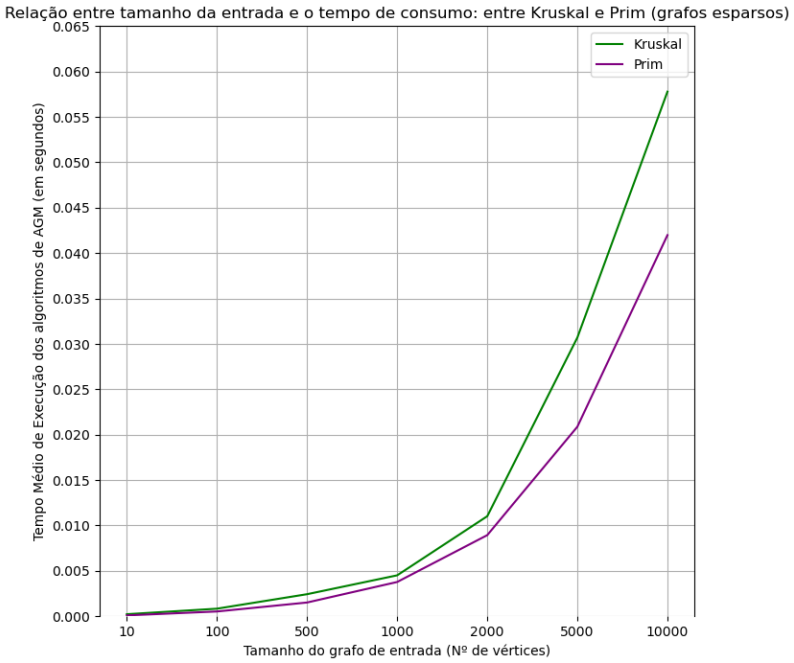

Figura 8: Gráfico de comparação com entradas de grafos esparsos

4.3 Comparativo entre os dois algoritmos para entradas densas e esparsas

Para melhor visualização da evolução temporal e da diferenciação entre os dois algoritmos, foi feito uma plotagem de gráfico unindo todos os dados coletados, de forma a deixar mais evidente a dispersão de ambos e garantir o melhor entendimento da conclusão assumida do projeto.

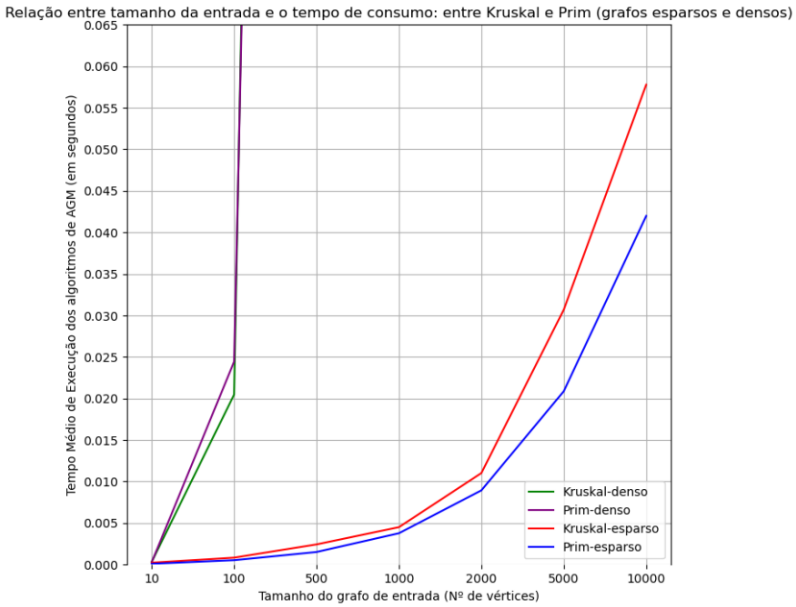

Figura 9: Gráfico de comparação com ambos tipos de entradas

5 Conclusão

Analisando os gráficos e as informações obtidas ao longo do trabalho, fica evidente que o desempenho dos algoritmos de Kruskal e Prim varia de acordo com o tipo de grafo utilizado. Nos grafos densos, ambos os algoritmos apresentam um crescimento exponencial no tempo de execução, com o algoritmo de Prim exibindo uma taxa de crescimento ligeiramente mais rápida devido à sua complexidade O(ElogV), que depende tanto do número de arestas quanto do número de vértices.

Por outro lado, em grafos esparsos, a diferença entre os algoritmos torna-se menos pronunciada, com ambos apresentando um crescimento mais gradual e quase constante à medida que o número de arestas aumenta. Isso é mais evidente quando se compara o comportamento de ambos os algoritmos para grafos esparsos, em que o tempo de execução é significativamente mais baixo em relação aos grafos densos.

Dessa forma, é possível concluir que, para grafos densos, o algoritmo de Prim tende a ser mais afetado pelo aumento das arestas, enquanto Kruskal mantém um desempenho mais estável. Já para grafos esparsos, ambos os algoritmos tendem a ter um desempenho mais equilibrado e eficiente. Esses resultados fornecem insights valiosos sobre a escolha do algoritmo mais adequado dependendo das características do grafo a ser processado.

6 Bibliografia

1. Slides disponibilizados pelo professor.

2. Paulo Feofiloff. "A Estrutura Union-Find", IME. Disponível em: <https://www.ime.usp.br/ pf/analise_de_algoritmos/aulas/union-find.html>. **Acesso em: 20 de Novembro de 2024**.