UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
CIÊNCIAS DA COMPUTAÇÃO

TRABALHO PRÁTICO 1
TÉCNICAS DE BUSCA E ORDENAÇÃO

PEDRO HENRIQUE MENDONÇA - 2023100593
PEDRO SODRÉ MALINI - 2023100263
PEDRO VIEIRA MARQUESINI - 2021102936

VITÓRIA - ES
MARÇO/2025

Introdução

O algoritmo de Dijkstra é um dos mais conhecidos para encontrar o caminho mínimo entre um nó de origem e os demais nós em um grafo ponderado. Desenvolvido pelo cientista da computação Edsger Dijkstra em 1956 e publicado em 1959, ele mantém um conjunto de nós cuja menor distância conhecida a partir do nó de origem já foi determinada. O algoritmo funciona inicializando a distância do nó de origem como 0 e de todos os outros nós como infinito. Em seguida, seleciona o nó com a menor distância atual, marca-o como processado e atualiza as distâncias dos seus vizinhos. Se encontrar um caminho mais curto para um vizinho através do nó processado, a distância é atualizada. Esse processo se repete até que todos os nós tenham sido processados ou o destino seja alcançado. Existem diversas formas para este algoritmo ser implementado, entre elas utilizando Heap Sort

Instruções para compilação e uso

Existem duas implementações para o algoritmo de Dijkstra que foi utilizado para realização do projeto, sendo o primeiro utilizando HeapSort e o segundo Insertion Sort, com isso, se fez necessário a criação de duas *main*, cada uma responsável por chamar a sua implementação. Para rodar o programa basta utilizar o comando **make** no terminal que o programa será compilado e gerará dois arquivos executáveis: **trab1** e **trab2**, com os dois arquivos executáveis gerados, para rodar o programa basta apenas dar o comando de sua preferência:

- *./trab1 <input.txt> <output.txt>* → para rodar utilizando Heap Sort
- *./trab2 <input.txt> <output.txt>* → para rodar utilizando Insertion Sort

Primeira Implementação - Heap

A primeira implementação do algoritmo de Dijkstra neste projeto utiliza uma **fila de prioridade (heap mínimo)** para sempre processar o nó com a menor distância conhecida primeiro, garantindo eficiência. O programa inicializa criando uma fila de prioridade baseada em heap mínimo que vai garantir que o nó com a menor distância seja sempre processado primeiro e o programa também vai alocar memória para para três vetores:

- **`distances[]`**: Guarda a menor distância conhecida de cada nó até a origem. Todos os valores começam com infinito(FLT_MAX), exceto o nó de origem, que recebe 0
- **`visited[]`**: Indica se um nó já foi processado (1 para visitado, `0` para não visitado).

- **`predecessor[]`**: Armazena o nó anterior no caminho mais curto (usado para reconstruir a rota depois). Inicia com -1 para indicar que nenhum caminho foi encontrado ainda.

Após o processo de inicialização feito corretamente, o algoritmo de Dijkstra processa cada nó do grafo até que todos os caminhos mais curtos sejam encontrados. A cada iteração, o nó com a menor distância conhecida é removido da fila de prioridade. Se já foi visitado, é ignorado; caso contrário, é marcado como processado. Em seguida, seus vizinhos são analisados para verificar se há um caminho mais curto passando pelo nó atual. Se um caminho melhor for encontrado, a distância do vizinho é atualizada, seu predecessor é registrado e ele é inserido na fila de prioridade para ser processado depois. Esse processo continua até que todos os nós tenham sido visitados, garantindo a menor distância da origem até cada vértice do grafo.

Após o processamento, os nós são ordenados pela menor distância para organizar a saída, isso permite exibir os caminhos na ordem correta, garantindo que os menores caminhos sejam apresentados primeiro, e vale destacar que uma melhoria para o programa implementado seria o uso de um QuickSort para ordenação dos resultados ao invés de usar o BubbleSort. Após a ordenação correta dos caminhos, a saída é impressa e a memória liberada.

Segunda Implementação - Insertion Sort

Outra abordagem para a implementação do algoritmo é utilizar uma fila de prioridade baseada em um vetor ordenado com Insertion Sort, em vez de uma heap. O Insertion Sort é um método de ordenação que organiza elementos de forma sequencial, inserindo cada novo item em sua posição correta dentro da estrutura. O algoritmo opera de maneira iterativa, iniciando pelo segundo elemento da sequência, considerando que o primeiro já está ordenado. A cada iteração, o elemento atual é comparado com os anteriores e deslocado para a posição apropriada, movendo os valores maiores uma posição à frente para criar espaço para a inserção. Esse processo continua até que todos os elementos tenham sido analisados e corretamente posicionados, garantindo a ordenação da fila de prioridade.

Comparação entre os algoritmos

O algoritmo de Dijkstra pode ser implementado utilizando tanto Heap Sort quanto Insertion Sort para a gestão da fila de prioridade. No entanto, essas abordagens apresentam diferenças substanciais em termos de eficiência computacional, complexidade algorítmica e aplicabilidade prática. A escolha da estrutura de dados apropriada impacta diretamente no desempenho do algoritmo,

especialmente em grafos de grande escala, onde a manipulação eficiente da fila de prioridade se torna um fator determinante para a viabilidade da solução, e como já era de se esperar o Heap Sort teve um desempenho superior ao Insertion Sort nos casos de testes utilizados para verificação do algoritmo, conforme mostra a tabela abaixo:

Comparação em segundos do tempo de execução de cada algoritmo

	Heap Sort	**Insertion Sort**
Caso muito pequeno 1	0.000355 *s*	0.000362 *s*
Caso muito pequeno 2	0.000434 *s*	0.000405 *s*
Caso pequeno 1	0.006564 *s*	0.009401 *s*
Caso pequeno 2	0.010197 *s*	0.009825 *s*
Caso pequeno 3	0.387934 *s*	0.875517 *s*
Caso pequeno 4	0.397326 *s*	0.885983 *s*
Caso médio 1	9.200131 *s*	27.239418 *s*
Caso médio 2	9.196738 *s*	27.093456 *s*
Caso médio 3	9.575089 *s*	26.745782 *s*
Caso médio 4	13.366880 *s*	40.100809 *s*

A partir dessa tabela concluímos que o algoritmo que utilizou a ordenação Heap Sort teve um desempenho melhor, pois o mesmo é superior para grandes conjuntos de dados, pois mantém a complexidade O(n log n) independentemente da ordenação inicial. Já o Insertion Sort foi eficiente apenas para pequenos conjuntos de dados ou listas quase ordenadas, onde sua complexidade pôde chegar a O(n).

O Insertion Sort também teve um desempenho inferior em relação ao uso de memória, utilizando mais memória do que o Heap Sort e isso deve ao fato de que o heap consome menos memória porque mantém a fila de prioridade em um vetor fixo e balanceado, minimizando realocações e cópias desnecessárias. Já o Insertion Sort exige mais memória devido às realocações frequentes e ao deslocamento contínuo de elementos dentro do vetor ordenado.

Conclusão

Com isso concluímos que a utilização de uma heap na implementação do algoritmo de Dijkstra é fundamental para otimizar a seleção do próximo nó a ser processado. Como a estrutura de heap permite a extração eficiente do menor

elemento em tempo O(log n), a fila de prioridade baseada nessa estrutura garante que sempre processamos o nó com a menor distância conhecida de forma rápida. Dessa forma, a escolha dessa estrutura de dados reduziu significativamente o tempo de processamento comparado ao Insertion Sort, sendo assim, ele se torna um algoritmo mais escalável e adequado para as diversas aplicações em que o Dijkstra é utilizado.