Trabalho Prático

Projeto e Análise de Algoritmos

Prof. Raquel Mini

Nome: Rithie Natan Carvalhaes Prado

Matricula: 541488

# Introdução

Sabemos que o Algoritmo de Dijkstra oferece uma solução para achar o menor caminho em um grafo, seja orientado ou não, com arestas de peso não negativas. Entretanto, este algoritmo não resolve o mesmo problema com um grafo que contém arestas negativas.

Um dos problemas em grafos com arestas negativas são os ciclos negativos, no qual nos leva a uma situação em podemos entrar em um loop infinito e nunca achar o menor caminho ou achar, porém teremos diminuído comprimento menor caminho. Este problema acontece com algoritmos que não estão preparados para isso, por tanto, é interessante identificar em cada grafo a possibilidade desses loops.

Por tanto, este trabalho consiste em implementar o Algoritmo de Dijkstra para achar o menor caminho em um grafo com arestas positivas e o Algoritmo de Bellman-Ford para determinar se existe ciclos negativos ou a solução existente para achar o menor caminho em um grafo com arestas negativas. Será feito uma análise de complexidade para cada algoritmo e comparar o tempo de execução de ambos.

# Implementação

2.1. Estrutura de dados utilizadas

* Serão utilizadas matrizes para representar cada grafo em cada Algoritmo.

2.2. Algoritmo de Dijkstra

* Grafo representado por matriz dinâmica.
* O algoritmo de Dijsktra recebe um grafo, número de vértices, vértices iniciais e vértice final. A implementação consiste em três vetores auxiliares: verificar se um vértice foi visitado, para determinar o menor caminho, para determinar o caminho mais curto.
* O método main recebe as entradas de um arquivo chamado ‘DijkstraTeste.in’, no qual corresponde a número de vértices e número de arestas na primeira linha. E nas demais linhas as conexões entre vértices e seus respectivos pesos.
* Execução do programa via terminal:
  + g++ -o dijkstra AlgoritmoDeDijkstra.cpp
  + ./dijkstra < DijkstraTeste.in

2.3. Algoritmo de Bellman-Ford

* Grafo representado por uma matriz comum.
* O método BellmanFord recebe por parâmetro o grafo construído, o número de vértices, número de arestas e vértice de início. O algoritmo inicializa um vetor auxiliar que conterá as distâncias dos vértices. Posteriormente será relaxado as arestas e verificamos se existem ciclos negativos. Por último, impresso as distâncias entre os vértices.
* Execução do programa via terminal:
  + g++ -o bellman AlgoritmoDeBellmanFord.cpp
  + ./bellman

2.4. Ambiente Computacional:

* + Processador: Intel Core i3-3220 de 3,30GHz
  + Sistema Operacional: Windows 10
  + Editor de Texto: VS Code
  + Compilador: g++ versão 7.5.0

# Análise de Complexidade

2.1. Algoritmo de Dijkstra - Operações relevantes: alocar matriz, liberar memória e percorrer vértices adjacentes.

* Método AlocarMatriz: Pior e Melhor caso: n^2
* Método FreeMemória: Pior e Melhor caso: n
* Método Dijkstra: O(n^2)
* Método Main: Melhor e Pior caso: n + n
* Soma: n^2 + n + n^2 + n + n = 2n^2 + 3n = O(n^2)

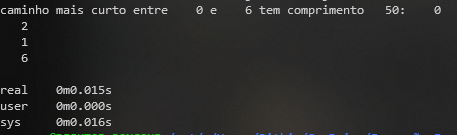
2.2. Algoritmo de Bellman-Ford – Operações relevantes: relaxar as arestas e verificar ciclo negativo.

* Método BellmanFord: Pior e Melhor caso: n^2
* Método main: n^2
* O(n^2)

# Testes

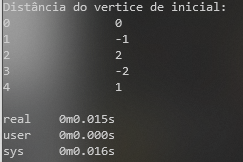
4.1 Testes do Algoritmo de Dijkstra

* Foi criado um arquivo chamado, ‘DijkstraTeste.in’ no qual contém um caso de teste para mostrar a criação de um grafo e a execução do algoritmo.
* Também testamos o tempo de execução de cada algoritmo.



4.2 Testes do Algoritmo de Bellman-Ford

* O teste do Algoritmo de Bellman-Ford foi criado no próprio programa, no metodo main. O teste consiste em criar um grafo qualquer e testar o algoritmo na identificação de ciclos negativos e/ou a distância entre o vértice inicial.



# Conclusão

O algoritmo de Dijkstra e de Bellman-Ford são situacionais dependendo de cada caso. O algoritmo de Bellman-Ford só será utilizado quando temos grafos que contenham pesos negativos e mesmo assim, dependendo da circunstância, os casos de ciclos negativos podem se torna muito recorrentes. Entretanto, sempre que tivermos somente pesos positivos em nosso grafo, devemos sempre utilizar o algoritmo de Dijkstra, pois tem um custo menor de execução e é um algoritmo ótimo em relação a achar o caminho mais curto.

# Bibliografia

* USP - <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dijkstra.html>
* USP - <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bellman-ford.html#concept>
* Wikipédia:
  + <https://pt.wikipedia.org/wiki/Algoritmo_de_Bellman-Ford>
  + https://pt.wikipedia.org/wiki/Algoritmo\_de\_Dijkstra

# Anexos

Os arquivos em anexos são:

* AlgoritmoDeDijkstra.cpp
* AlgoritmoDeBellmanFord.cpp
* DijkstraTeste.in