UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DISCIPLINA LABORATÓRIO DE PROGRAMAÇÃO II - ELC1067

PROBLEMA CAMINHO MÍNIMO - GRAFOS

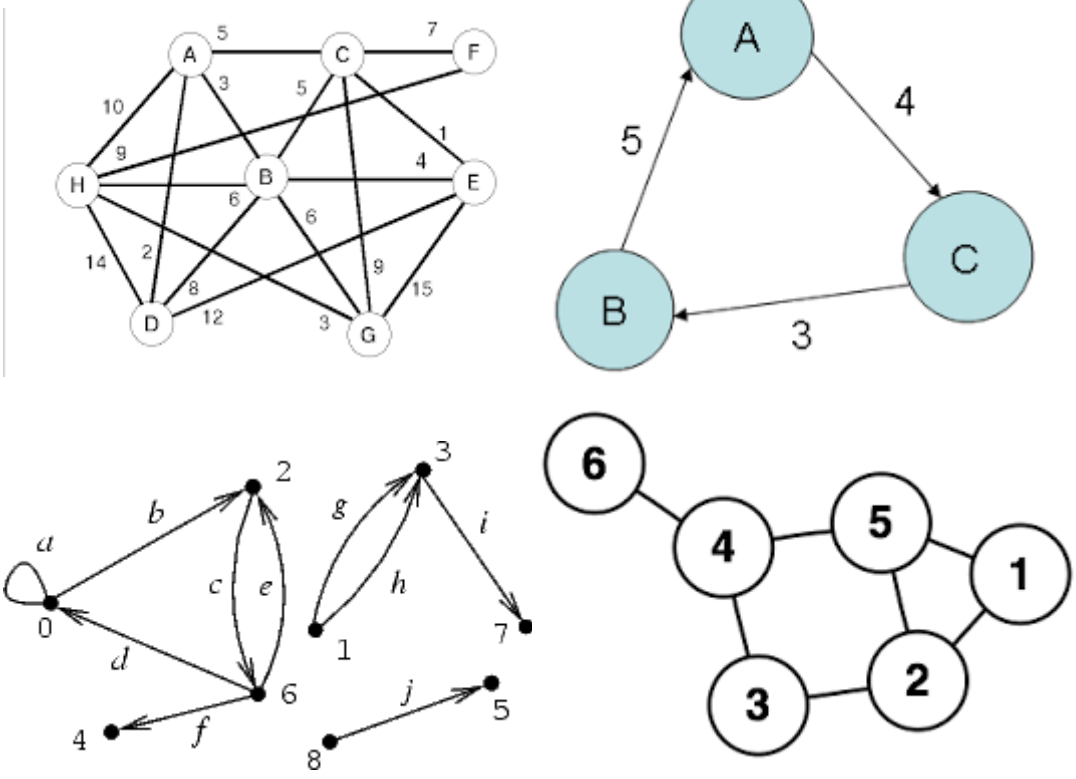
NOME: BRUNO GOTTLIEB
GUSTAVO NOLL
THALISSON FORTE

SANTA MARIA, 03 DE DEZEMBRO DE 2018

O problema do caminho mínimo

O problema do caminho mínimo consiste na obtenção do menor custo possível entre dois vértices em um grafo onde suas arestas possuem pesos.

Abaixo apresento alguns exemplos de grafos:



Há basicamente dois modos de representar grafos em código:

Matriz adjacência (conectado)

Matriz N x N

N = Número de vértices

Custo computacional: N^2

Aresta é representada pelo número 1 na posição i, j

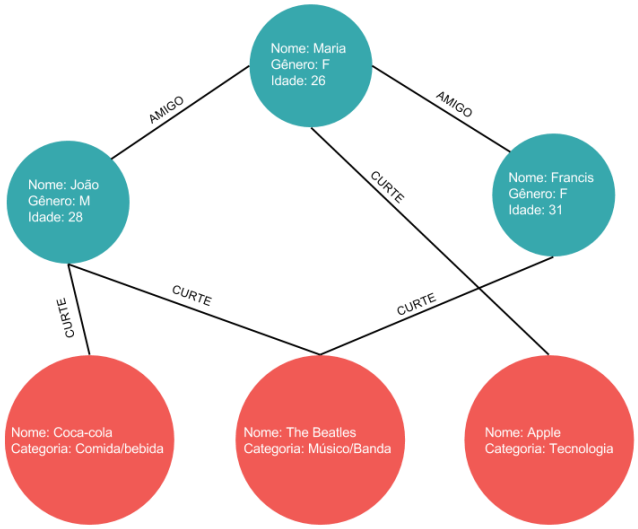
Lista de adjacência (esparso)

Cada posição da lista representa um vértice

Cada vértice aponta para todos os vértices no qual possui conexões

O problema do caminho mínimo é bastante utilizado na área da Ciência da Computação em termos de otimização referentes a buscas operacionais ou até mesmo em Inteligência Artificial.

Exemplos de aplicações que usam esse tipo de busca incluem:
– achar o grau de separação entre duas pessoas em uma rede social;
– achar um trajeto em um mapa rodoviário;
– programar robôs explorar áreas;
– algoritmos de roteamento.



Para solucionar esse problema, utiliza-se alguns algoritmos especializados em caminho mínimo, conhecidos como *algoritmos de busca de caminhos*. Dentre eles, os mais conhecidos são:

- Algoritmo de Dijkstra

- Algoritmo de Bellman-Ford

- Algoritmo A*

- Algoritmo de Floyd-Warshall

- Algoritmo de Johnson

Algoritmo de Dijkstra

Resolve o problema com um vértice-fonte em grafos cujas arestas tenham peso maior ou igual a zero. Sem reduzir o desempenho, este algoritmo é capaz de determinar o caminho mínimo, partindo de um vértice de início *v* para *todos* os outros vértices do grafo.

Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. Ele é bastante simples e com um bom nível de performance. Ele não garante, contudo, a exatidão da solução caso haja a presença de arcos com valores negativos.

Parte-se de uma estimativa inicial para o custo mínimo e vai sucessivamente ajustando esta estimativa. Ele considera que um vértice estará fechado quando já tiver sido obtido um caminho de custo mínimo do vértice tomado como raiz da busca até ele. Caso contrário ele dito estar aberto.

Este algoritmo possui um tempo computacional O([m+n]log n) onde m é o número de arestas e n é o número de vértices. O algoritmo que serve para resolver o mesmo problema em um grafo com pesos negativos é o *algoritmo de Bellman-Ford*, que possui maior tempo de execução que o Dijkstra.

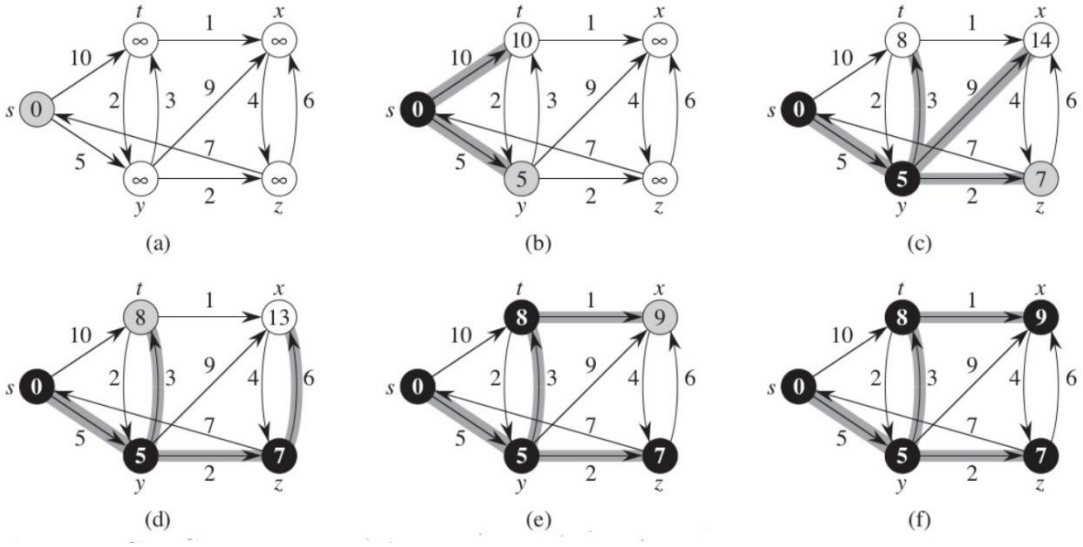
O algoritmo considera um conjunto S de menores caminhos, iniciado com um vértice inicial I. A cada passo do algoritmo busca-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a I e adiciona-o a S e, então, repetindo os passos até que todos os vértices alcançáveis por I estejam em S. Arestas que ligam vértices já pertencentes a S são desconsideradas.

Um exemplo prático do problema que pode ser resolvido pelo algoritmo de Dijkstra é: alguém precisa se deslocar de uma cidade para outra. Para isso, ela dispõe de várias estradas, que passam por diversas cidades. Qual delas oferece uma trajetória de menor caminho?

Algoritmo: Seja G(V,A) um grafo orientado e **s** um vértice de G:

1. Atribua valor zero à estimativa do custo mínimo do vértice **s** (a raiz da busca) e infinito às demais estimativas;
2. Atribua um valor qualquer aos precedentes (o precedente de um vértice **t** é o vértice que precede **t** no caminho de custo mínimo de **s** para **t**);
3. Enquanto houver vértice aberto:
    - seja **k** um vértice ainda aberto cuja estimativa seja a menor dentre todos os vértices abertos;
    - feche o vértice **k**
    - Para todo vértice **j** ainda aberto que seja sucessor de **k** faça:
        - some a estimativa do vértice **k** com o custo do arco que une **k** a **j**;

- caso esta soma seja melhor que a estimativa anterior para o vértice **j**, substitua-a e anote **k** como precedente de **j**.



Algoritmo de Bellman-Ford

Foi proposto pela primeira vez por Alfonso Shimbel em 1995. Resolve o problema para grafos com um vértice-fonte e arestas que podem ter pesos negativos. A sua ideia é de relaxar todas as |E| arestas |V| -1 vezes.

O Algoritmo de Bellman-Ford é um algoritmo de busca de caminho mínimo em um dígrafo ponderado, ou seja, cujas arestas têm peso, inclusive negativo. Ao contrário do algoritmo de Dijkstra, ele não impõe restrição sobre o sinal dos pesos das arestas, portanto ele aceita pesos negativos. Ele computa para um grafo orientado, o menor caminho de um nó de origem até outro nó.

O algoritmo divide-se em 3 etapas:

- Inicialização: Responsável por padronizar as distâncias antes do início da resolução.

- Relaxamento: Calcula o caminho mínimo.

- Verificação de ciclos negativos: Verifica se o cálculo é ou não possível.

Abaixo como sua implementação funciona:

Algoritmo de Bellman-Ford para calcular distâncias mínimas a partir de s para todos os outros nós no grafo G

Bellman-Ford(G, s):
  Para todos os nós v de G **fazer**:
    $v.dist \leftarrow \infty$
  $s.dist \leftarrow 0$
  Para $i \leftarrow 1$ **até** $|V| - 1$ **fazer**:
    Para todas as arestas (u, v) de G **fazer**:
      Se $u.dist + peso(u, v) < v.dist$ **então**
        $v.dist \leftarrow u.dist + peso(u, v)$

As vantagens de usar este algoritmo dão-se pela sua fácil implementação e a possibilidade de lidar com arestas de pesos negativos. As desvantagens são seu tempo de execução, que revela-se maior se comparado com o algoritmo de Dijsktra e a impossibilidade de ciclos negativos.

Algoritmo A*

Algoritmo A* (Lê-se: A-estrela) é um algoritmo para Busca de Caminho. Ele busca o caminho em um grafo de um vértice inicial até um vértice final. Ele é a combinação de aproximações heurísticas como do algoritmo Breadth First Search (Busca em Largura) e da formalidade do Algoritmo de Dijkstra.

O algoritmo foi descrito pela primeira vez em 1968 por Peter Hart, Nils Nilsson, e Bertram Raphael. Na publicação deles, ele foi chamado de algoritmo A; usando este algoritmo com uma heurística apropriada atinge-se um comportamento ótimo, e passou a ser conhecido por A*. Uma das propriedades que A* possui é que ele garante encontrar sempre a melhor solução.

Sua aplicação vai desde aplicativos para encontrar rotas de deslocamento entre localidades a resolução de problemas, como a resolução de um quebra-cabeças. Ele é muito usado em jogos.

O algoritmo A* é um algoritmo de busca em grafos, que se utiliza de uma heurística para auxiliar a ordem em que os nós são processados, com o intuito de diminuir o tempo de processamento.

A função h(x) deve sempre ser otimista, isto é, deve sempre retornar um valor que não seja maior que a distância real. Quanto mais próxima h(x) estiver da real distância, mais rápido será o processamento.

Abaixo uma breve explicação do algoritmo:

Q = conjunto de nós a serem pesquisados;
S = o estado inicial da busca;

1- Inicialize Q com o nó de busca (S) como única entrada;

2- Se Q está vazio, interrompa. Se não, escolha o melhor elemento de Q;

3- Se o estado (n) é um objetivo, retorne n;

4- (De outro modo) Remova n de Q;

5- Encontre os descendentes do estado (n) que não estão em visitados e crie todas as extensões de n para cada descendente;

6- Adicione os caminhos estendidos a Q e vá ao passo 2;

Algoritmo de Floyd-Warshall

O algoritmo de Floyd-Warshall é um algoritmo que resolve o problema de calcular o caminho mais curto entre todos os pares de vértices em um grafo orientado (com direção) e valorado (com peso). O algoritmo é um bom exemplo de programação dinâmica. Pode ter pesos negativos, sem ciclos de pesos negativos;

- Assumindo que os vértices de um grafo orientado G são V = 1,2,3,...,n, considere um subconjunto 1,2,3,...,k;

- Para qualquer par de vértices (i,j) em V, considere todos os caminhos de i a j cujos vértices intermédios pertencem ao subconjunto 1,2,3,...,k, e p como o mais curto de todos eles;

- O algoritmo explora um relacionamento entre o caminho p e os caminhos mais curtos de i a j com todos os vértices intermédios em 1,2,3,...,k-1;

- O relacionamento depende de k ser ou não um vértice intermédio do caminho p.

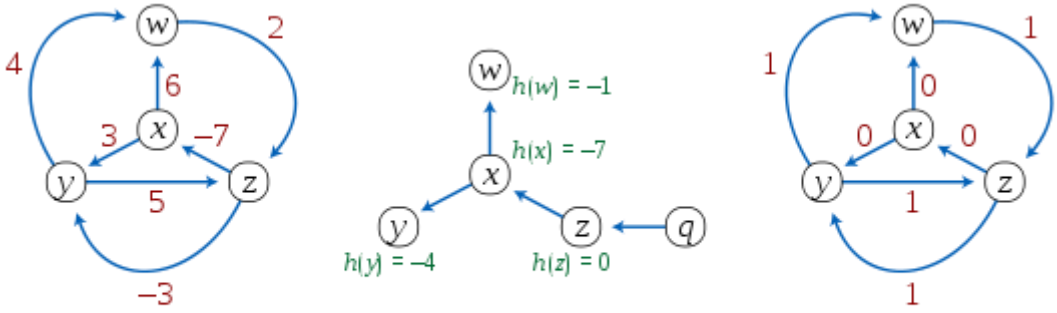
Abaixo uma implementação do algoritmo de Floyd_Warshall:

```
ROTINA fw(Inteiro[1..n,1..n] grafo)
    # Inicialização
    VAR Inteiro[1..n,1..n] dist := grafo
    VAR Inteiro[1..n,1..n] pred
    PARA i DE 1 A n
        PARA j DE 1 A n
            SE dist[i,j] < Infinito ENTÃO
                pred[i,j] := i
    # Laço principal do algoritmo
    PARA k DE 1 A n
        PARA i DE 1 A n
            PARA j DE 1 A n
                SE dist[i,j] > dist[i,k] + dist[k,j] ENTÃO
                    dist[i,j] = dist[i,k] + dist[k,j]
                    pred[i,j] = pred[k,j]
    RETORNE dist
```

Algoritmo de Johnson

Aplicado em grafos que são esparsos, direcionados e valorados, trabalha com base no Algoritmo de Bellman-Ford, para computar uma transformação de um grafo de entrada, que remove todas os pesos negativos, permitindo o uso do algoritmo de Dijkstra no grafo transformado. Recebe esse nome em homenagem a Donald B. Johnson, o primeiro a descrevê-lo, em 1977.

É formado pelos seguintes passos:



original graph with negative edges | shortest path tree found by Bellman-Ford | reweighted graph with no negative edges

1. Primeiro, um novo nó *q* é adicionado ao grafo, conectado com peso zero (0) com cada um dos outros nós.
2. Segundo, é usado o algoritmo de Bellman–Ford, começando a partir do novo nó *q*, para encontrar cada um dos vértices *v*, o de menor peso *h*(*v*) do caminho de *q* para *v*. Se esse passo detectar um ciclo negativo, o algoritmo é terminado.
3. O próxima borda do grafo original é reponderada usando os valores calculados pelo algoritmo Bellman–Ford: uma borda de *u* para *v*, tendo comprimento *w(u,v)*, é dada pelo novo comprimento *w(u,v)* + *h(u)* −*h(v)*.
4. Finalmente, *q* é removido, e o algoritmo de Dijkstra é usado para encontrar o menor caminho para cada um dos nós *s* para todos os outros vértices no grafo reponderado.

Melhores implementações:

Caminho mínimo de fonte única: algoritmo de Dijsktra;

Caminho mínimo de fonte única: algoritmo de Dijsktra;

Caminho mínimo de destino único: inverta a direção das arestas e aplique algoritmo de Dijstra;

Caminho mínimo de fonte única: algoritmo de Dijsktra;

Caminho mínimo de destino único: inverta a direção das arestas e aplique algoritmo de Dijstra;

Caminho mínimo entre quaisquer vértices u e v: algoritmo de Dijsktra;

Caminho mínimo de fonte única: algoritmo de Dijsktra;

Caminho mínimo de destino único: inverta a direção das arestas e aplique algoritmo de Dijstra;

Caminho mínimo entre quaisquer vértices u e v: algoritmo de Dijsktra;

Caminho mínimo em grafos com pesos negativos: algoritmo de Bellman-Ford;

Caminho mínimo de fonte única: algoritmo de Dijsktra;

Caminho mínimo de destino único: inverta a direção das arestas e aplique algoritmo de Dijstra;

Caminho mínimo entre quaisquer vértices u e v: algoritmo de Dijsktra;

Caminho mínimo em grafos com pesos negativos: algoritmo de Bellman-Ford;

Caminho mínimo entre todos os pares de vértices: algoritmo de Floyd-Warshall em tempo O(n 3).

Trabalho Prático

No primeiro código implementado foi utilizado o algoritmo de *Dijkstra* com o uso de matriz de adjacência**.**

Trata-se de um grafo com 10 vértices e 24 arestas, inseridas uma a uma como nas linhas a seguir:

```
insereAresta(gr, 0, 1, 0, 1);
insereAresta(gr, 1, 3, 0, 2);
insereAresta(gr, 3, 7, 0, 3);
insereAresta(gr, 3, 2, 0, 4);
insereAresta(gr, 4, 1, 0, 5);
insereAresta(gr, 4, 5, 0, 4);
insereAresta(gr, 3, 0, 0, 3);
insereAresta(gr, 6, 2, 0, 2);
insereAresta(gr, 1, 7, 0, 1);
insereAresta(gr, 5, 2, 0, 1);
insereAresta(gr, 2, 1, 0, 1);
insereAresta(gr, 1, 5, 0, 2);
insereAresta(gr, 7, 4, 0, 3);
insereAresta(gr, 6, 5, 0, 4);
insereAresta(gr, 7, 2, 0, 5);
insereAresta(gr, 6, 4, 0, 4);
insereAresta(gr, 6, 8, 0, 3);
insereAresta(gr, 2, 8, 0, 2);
insereAresta(gr, 3, 8, 0, 1);
insereAresta(gr, 8, 5, 0, 1);
insereAresta(gr, 9, 4, 0, 2);
insereAresta(gr, 9, 7, 0, 3);
insereAresta(gr, 10, 5, 0, 2);
insereAresta(gr, 10, 9, 0, 2);
```

gr é o ponteiro para a estrutura do grafo, após ele consta o vértice de origem, o vértice de destino, zero representando que não é um dígrafo e, por último, o respectivo peso de cada aresta.

Os resultados podem ser conferidos ao compilar o código. Usando como exemplo o vértice de origem zero, os resultados são os seguintes:

Distancia da origem 0 ate o vertice 0: 0.0

Distancia da origem 0 ate o vertice 1: 1.0

Distancia da origem 0 ate o vertice 2: 2.0

Distancia da origem 0 ate o vertice 3: 3.0

Distancia da origem 0 ate o vertice 4: 5.0

Distancia da origem 0 ate o vertice 5: 3.0

Distancia da origem 0 ate o vertice 6: 4.0

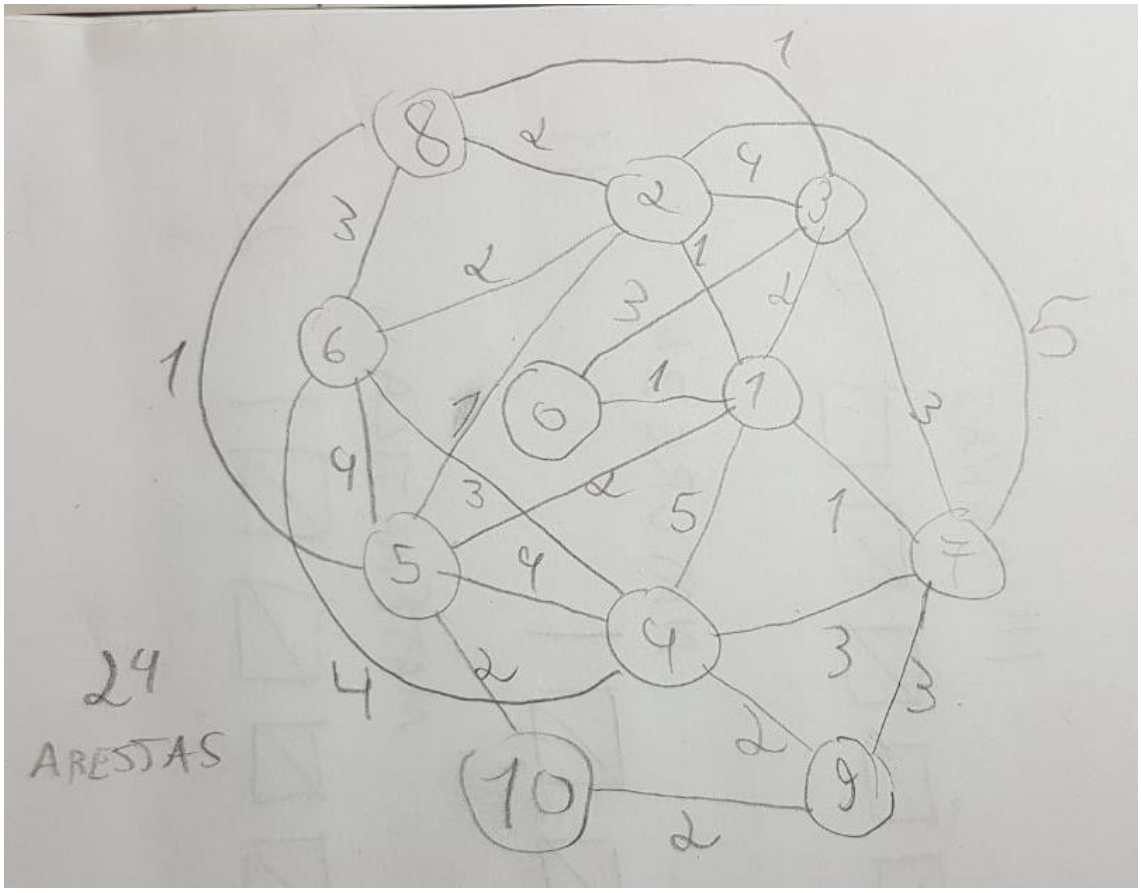
Distancia da origem 0 ate o vertice 7: 2.0

Distancia da origem 0 ate o vertice 8: 4.0

Distancia da origem 0 ate o vertice 9: 5.0

Distancia da origem 0 ate o vertice 10: 5.0

O esboço do determinado grafo utilizado como exemplo encontra-se a seguir:



O segundo algoritmo implementado foi o de *Bellman-Ford* utilizando-se de arestas com pesos negativos.

Trata-se de um grafo com 12 vértices e 21 arestas, inseridas uma a uma como nas linhas a seguir:

```
Inserir_Aresta(aresta, 0, 1, 3, &i);
Inserir_Aresta(aresta, 0, 2, 2, &i);
Inserir_Aresta(aresta, 0, 3, 3, &i);
Inserir_Aresta(aresta, 1, 2, 1, &i);
Inserir_Aresta(aresta, 2, 3, -1, &i);
Inserir_Aresta(aresta, 1, 6, -2, &i);
Inserir_Aresta(aresta, 2, 5, 4, &i);
Inserir_Aresta(aresta, 3, 4, 2, &i);
Inserir_Aresta(aresta, 4, 5, 2, &i);
Inserir_Aresta(aresta, 5, 6, -1, &i);
Inserir_Aresta(aresta, 6, 7, 2, &i);
Inserir_Aresta(aresta, 6, 8, 4, &i);
Inserir_Aresta(aresta, 7, 8, -3, &i);
Inserir_Aresta(aresta, 8, 9, 2, &i);
Inserir_Aresta(aresta, 9, 10, -2, &i);
Inserir_Aresta(aresta, 9, 12, 4, &i);
Inserir_Aresta(aresta, 10, 11, 1, &i);
Inserir_Aresta(aresta, 4, 11, -4, &i);
Inserir_Aresta(aresta, 4, 10, 1, &i);
Inserir_Aresta(aresta, 8, 12, -1, &i);
Inserir_Aresta(aresta, 10, 12, 2, &i);
```

Aresta é o ponteiro para a estrutura do grafo, após ele consta o vértice de origem, o vértice de destino, o respectivo peso de cada aresta e uma variável representando sua posição no vetor de distâncias.

Os resultados podem ser conferidos ao compilar o código. Usando como exemplo o vértice de origem zero, os resultados são os seguintes:

Distancia da origem ate o vertice 0: 0

Distancia da origem ate o vertice 1: 3

Distancia da origem ate o vertice 2: 2

Distancia da origem ate o vertice 3: 1

Distancia da origem ate o vertice 4: 3

Distancia da origem ate o vertice 5: 5

Distancia da origem ate o vertice 6: 1

Distancia da origem ate o vertice 7: 3

Distancia da origem ate o vertice 8: 0

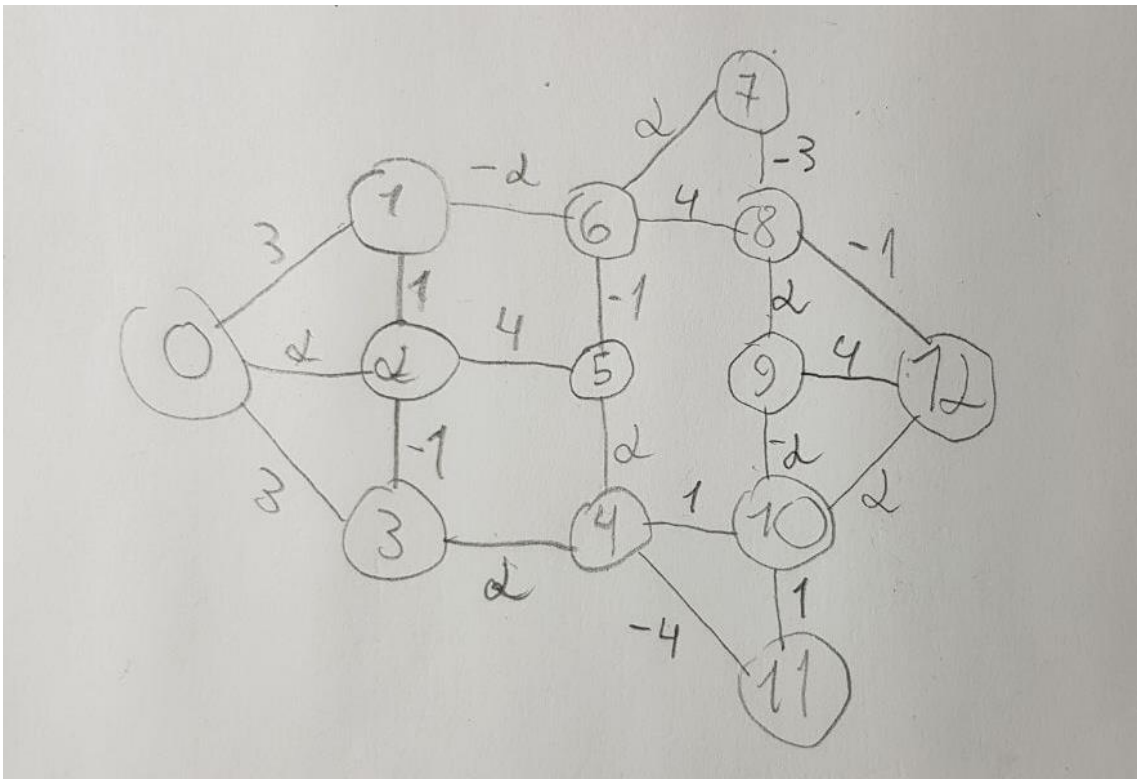
Distancia da origem ate o vertice 9: 2

Distancia da origem ate o vertice 10: 0

Distancia da origem ate o vertice 11: -1

Distancia da origem ate o vertice 12: -1

O esboço do determinado grafo utilizado como exemplo encontra-se a seguir:



Referências

https://pt.wikipedia.org/wiki/Problema_do_caminho_m%C3%ADnimo#cite_note-2

https://programacaodescomplicada.wordpress.com/tag/grafos/

http://miltonborba.org/Algf/Dijkstra.htm

http://www.dcc.fc.up.pt/~pribeiro/aulas/daa1415/slides/8_distancias_06122014.pdf

https://pt.slideshare.net/jackocap/anlise-de-algoritmos-problemas-em-grafos-caminho-mnimo-algoritmo-de-bellmanford

https://pt.wikipedia.org/wiki/Algoritmo_de_Bellman-Ford

http://maratonapuc.wikidot.com/apostilas:a-star

https://pt.wikipedia.org/wiki/Algoritmo_A*

https://www.passeidireto.com/arquivo/5105911/relatorio-do-algoritmo-aestrela?ordem=1

https://pt.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall

http://professor.ufabc.edu.br/~leticia.bueno/classes/teoriagrafos/materiais/floydwarshall.pdf

http://www.wikiwand.com/pt/Algoritmo_de_Johnson

http://professor.pucgoias.edu.br/SiteDocente/admin/arquivosUpload/4490/material/ed2%20-%20johnson.pptx

https://www.youtube.com/watch?v=5y8dch2uHR4&t=

http://www.martinbroadhurst.com/bellman-ford-algorithm-in-c.html

https://www.thecrazyprogrammer.com/2017/06/bellman-ford-algorithm-in-c-and-c.html

https://www.programiz.com/dsa/bellman-ford-algorithm