# Teoria dos Grafos - Problema do Caminho Mínimo

**Daniel de Souza Rodrigues - 18.2.8112 - Sistemas de Informação**

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
Rua 36, Número 115 - Bairro Loanda, João Monlevade - CEP: 35931-008

daniel.sr@aluno.ufop.edu.br

***Abstract.*** *In this article we will present a classic problem of the Theory of Graphs that is called: Problem of the minimum path, to better understand this problem I will make a brief introduction on the subject that runs through the world of informatics and I will use 3 classic algorithms being: Dijkstra; Bellman-Ford; Floyd-Warshall; for better understanding and solving the problem of the minimum path.*

***Resumo.*** *Neste artigo será apresentado um problema clássico da Teoria dos Grafos que se chama: Problema do caminho mínimo, para entendermos melhor este problema farei uma breve introdução sobre o assunto que percorre o mundo da computação e utilizarei de 3 algoritmos clássicos sendo eles: Dijkstra; Bellman-Ford; Floyd-Warshall; para melhor compreensão e resolução do problema do caminho mínimo.*

## 1. Introdução

O problema do caminho mínimo consiste basicamente: dado um grafo com pesos nas arestas, obter o caminho de menor custo entre dois vértices U e V, sendo o peso das arestas uma forma de representar a distância e dificuldades diversas que são encontradas entre percursos de U $\longleftrightarrow$ $V$.[Netto and Jurkiewicz 2017]

### 1.1. Entendendo um Grafo

Um grafo G é composto por uma quantidade x de vértices que podem ser ou não interligados, na figura abaixo podemos observar um grafo relativamente simples que possui 5 vértices, o vértice 3 não possui nenhuma conexão com os demais.[Jurkiewicz 2009]

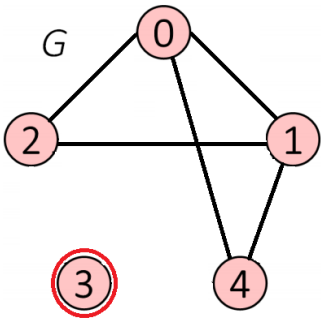



**Figura 1. Representação Visual de um Grafo com 5 vértices**

### 1.2. Entendendo Aresta e Pesos

Arestas são uma forma de determinar se existe ou não uma ligação entre dois ou mais vértices do Grafo.

- Arestas podem ser orientadas ou não orientadas
- Arestas podem possuir ou não um peso x
- Quando o grafo possui arestas não direcionadas entende-se que o vértice U possui uma ligação com o vértice V e vice-versa ($U \longleftrightarrow V$).

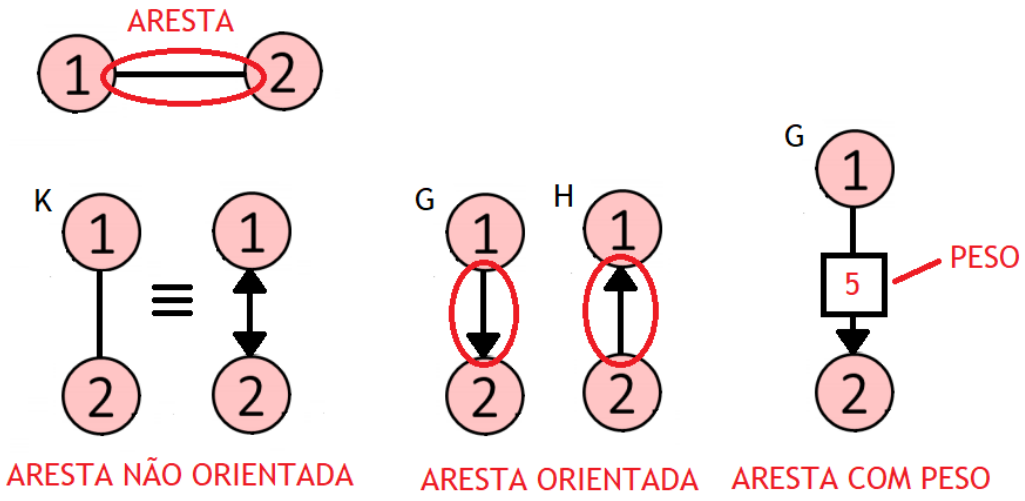



**Figura 2. Arestas, Arestas Orientadas e Não Orientadas, Arestas com Peso**

### 1.3. Entendendo Caminhos e Custo

Um caminho tem como parâmetro uma ORIGEM e um DESTINO.

Caminhos se tornam possíveis em um determinado grafo se existe ou não ligação entre vértices, um caminho pode passar por diversos vértices, sabemos que as arestas podem possuir pesos maiores umas que as outras, sendo assim é possível que exista em um grafo um ou mais caminhos, mas estes caminhos podem possuir um custo muito diferente sendo maior ou menor, exemplo:

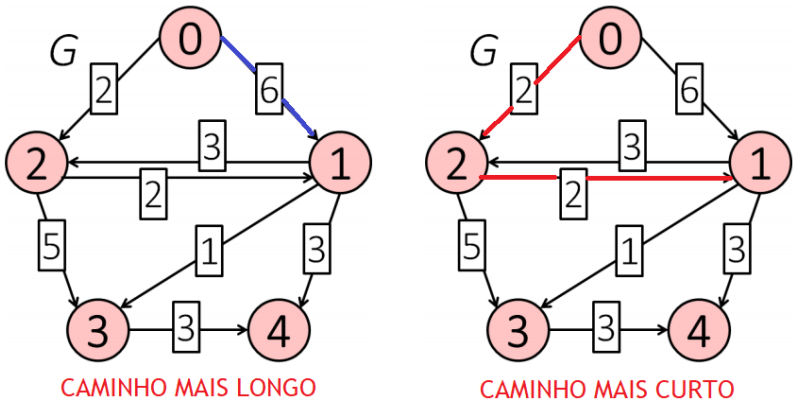



**Figura 3. Exemplo: Maior Caminho e Menor Caminho entre os vértices 0 e 1**

## 2. Soluções clássicas para o Problema do Caminho Mínimo

Como foi explicado na seção anterior (1.3) para um determinado grafo G pode existir diversos caminhos com pesos diferentes e para solucionarmos o problema do caminho mínimo, utilizaremos 3 algoritmos clássicos usados na Teoria dos Grafos e exploraremos o seu funcionamento.

### 2.1. Algoritmo Dijkstra

O algoritmo de Dijkstra recebe como entrada um grafo ponderado G = (V, E, w) e um vértice origem S e computa um vetor das distâncias (dist) de S a todos os demais vértices e um vetor com os predecessores (pred) de cada vértice no caminho mínimo.[Netto and Jurkiewicz 2017]

- A principal vantagem desse algoritmo está na sua eficiência, da ordem O($|V|^2$) (ou até mesmo O($|V|log|V|$) se implementado por uma fila de prioridades).
- A desvantagem desse algoritmo está em não garantir caminhos mínimos corretos para grafos que contenham arestas de peso negativo.

#### 2.1.1. Funcionamento e Pseudocódigo - Dijkstra

A ideia essencial desse algoritmo é buscar, a cada passo, melhores caminhos a partir de um determinado vértice U.

1. Um conjunto Q armazena os vértices a serem processados.
2. O vértice de menor distância atual U é escolhido para ser processado e removido de Q.
3. Novos caminhos são buscados a partir de U, atualizando dist e pred.
4. O algoritmo continua do novo vértice de menor distância em Q e encerra quando Q é vazio.

**Pseudocódigo - Algoritmo Dijkstra**

```
   Entrada: (i) Um grafo G = (V, E, w); (ii) Origem s.
1  para cada v em V faça
2      dist[v] ← ∞;                  ▷ Vetor que armazena a distância de s a cada vértice
3      pred[v] ← null;               ▷ Vetor que armazena o predecessor de cada vértice
4  dist[s] ← 0;
5  Q ← V;                            ▷ Q: lista dos vértices a serem processados
6  enquanto Q ≠ ∅ faça               ▷ Lista Q não for vazia (há vértices a processar)
7      u ← i|min{dist[i], ∀i ∈ Q};   ▷ u: Vértice de menor distância em Q
8      Q ← Q − {u};                  ▷ Remover u de Q
9      para cada v adjacente a u faça
10         se dist[v] > dist[u] + w(u, v) então   ▷ w(u, v): peso da aresta (u, v)
11             dist[v] ← dist[u] + w(u, v);
12             pred[v] ← u;
```

**Figura 4. Pseudocódigo - Algoritmo Dijkstra**

## 2.2. Algoritmo Bellman-Ford

O algoritmo de Bellman-Ford, assim como o de Dijkstra, recebe como entrada um grafo ponderado G = (V, E,w) e um vértice origem S e computa um vetor das distâncias (dist) de S a todos os demais vértices e um vetor com os predecessores (pred) de cada vértice no caminho mínimo.[Netto and Jurkiewicz 2017]

- A principal vantagem desse algoritmo está em garantir caminhos mínimos corretos mesmo em grafos com arestas de peso negativo.
- A desvantagem desse algoritmo está em sua complexidade computacional de pior caso, da ordem de O($|V| \times |E|$), bem maior que a de Dijkstra.

### 2.2.1. Funcionamento e Pseudocódigo - Bellman-Ford

Sua ideia essencial é inspecionar todas as arestas atualizando os caminhos mínimos caso algum caminho menor seja encontrado. As arestas são re-inspecionadas até que nenhum novo caminho seja encontrado.

1. No pior caso as arestas devem ser inspecionadas $|V|$ vezes.
2. É possível detectar ciclos de custo negativo com uma pequena adaptacão a esse algoritmo.
3. Caso as arestas sejam inspecionadas em uma ordem favorável o algoritmo pode encerrar em poucas iteracões.

**Pseudocódigo - Algoritmo Bellman-Ford**

```
Entrada: (i) Um grafo G = (V, E, w); (ii) Origem s.
1  para cada v em V faça
2      dist[v] ← ∞;                    ▷ Vetor que armazena a distância de s a cada vértice
3      pred[v] ← null;                 ▷ Vetor que armazena o predecessor de cada vértice
4  dist[s] ← 0;
5  para i ← 0 até |V| − 1 faça
6      trocou ← False;                 A variável trocou faz parte de uma otimização do algoritmo original
7      para cada (u, v) em E faça
8          se dist[v] > dist[u] + w(u, v) então      ▷ w(u, v): peso da aresta (u, v)
9              dist[v] ← dist[u] + w(u, v);
10             pred[v] ← u;
11             trocou ← True;
12     se trocou = False então         ▷ A execução pode ser encerrada prematuramente
13         break;
```

**Figura 5. Pseudocódigo - Algoritmo Bellman-Ford**

### 2.3. Algoritmo Floyd-Wharshall

O algoritmo de Floyd-Warshall recebe como entrada um grafo G = (V, E,w) e calcula os caminhos mínimos para todos os pares de vértices desse grafo, sendo essa sua principal vantagem.[Netto and Jurkiewicz 2017]

- Esse algoritmo também garante caminhos mínimos corretos para grafos com arestas de peso negativo.
- A desvantagem desse algoritmo reside em sua elevada complexidade computacional, na ordem de O($|V|^3$).
- O algoritmo tem como resultado uma matriz de distâncias $(dist[][])$ entre cada par de vértices e uma matriz de predecessores $(pred[][])$ indicando o predecessor de cada vértice (coluna) j em caminhos a partir de um vértice (linha) i.

#### 2.3.1. Funcionamento e Pseudocódigo - Floyd-Wharsall

A ideia essencial desse algoritmo é comparar todos os possíveis caminhos pelo grafo entre cada par de vértices (i, j). Isso é feito de forma incremental, melhorando a estimativa do caminho mínimo entre dois vértices a cada iteracão.

1. Um índice i representa um vértice origem.
2. Um índice j representa um vértice destino.
3. Um índice k representa um vértice como caminho alternativo de i a j.

**Pseudocódigo - Algoritmo Floyd-Wharsall**

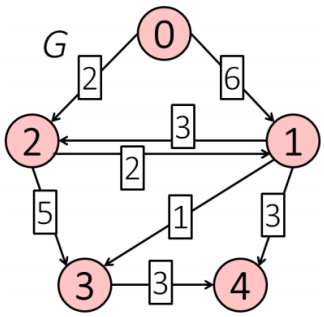
```
Entrada: (i) Um grafo G = (V, E, w).
para i ← 0 até |V| − 1 faça                          ▷ Inicialização das matrizes dist e pred
    para j ← 0 até |V| − 1 faça
        se i = j então
            dist[i][j] ← 0;                  ▷ Matriz da distância entre cada par de vértices (i, j)
        senão se existe aresta (i, j) então
            dist[i][j] ← w(i, j);
            pred[i][j] ← i;   ▷ Matriz do pred. de cada vértice j nos caminhos a partir de i
        senão
            dist[i][j] ← ∞;
            pred[i][j] ← null;
para k ← 0 até |V| − 1 faça                  ▷ Procura caminhos alternativos passando por k
    para i ← 0 até |V| − 1 faça                                        ▷ Caminhos a partir de i
        para j ← 0 até |V| faça                                          ▷ Caminhos até j
            se dist[i][j] > desit[i][k] + dist[k][j] então
                dist[i][j] ← desit[i][k] + dist[k][j];
                pred[i][j] ← pred[k][j];
```

**Figura 6. Pseudocódigo - Algoritmo Floyd-Wharshall**

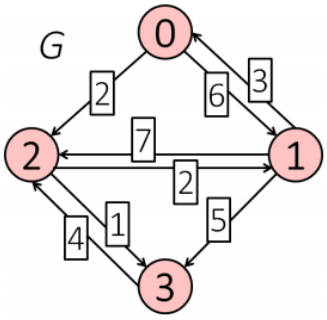
## 3. Testes dos Algoritmos

1. Foi utilizada a linguagem Python para implementação dos algoritmos.
2. Link do Repositório: Problema do caminho mínimo - GitHub Daniel Rodrigues
3. Todos os testes foram executados com Processador: $AMD - Ryzen3 - 2200g$
4. Memorória RAM: $Ballistix$ - $8GB\ DDR4\ 2400mhz$

### 3.1. Validação dos algorimos implementados em Python




[Grafo de Teste 1]




[Grafo de Teste 2]

- O ”Grafo de Teste 1”foi utilizado para validar os resultados do algoritmo Dijkstra e Bellman-ford, caminho = [Origem (0), Destino (4)], resultado:

| DIST | 0 | 4.0 | 2.0 | 5.0 | 7.0 |
|---|---|---|---|---|---|
| PRED | - | 2 | 0 | 1 | 1 |

**Figura 7. Tabela de Resultados - Grafo de Teste 1**

- O ”Grafo de Teste 2”foi utilizado para validar os resultados do algoritmo Floyd-Wharshall , caminho = [Origem (0), Destino (3)], resultado:

MATRIZ DE DIST

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 4.0 | 2.0 | 3.0 |
| 1 | 3.0 | 0 | 5.0 | 5.0 |
| 2 | 5.0 | 2.0 | 0 | 1.0 |
| 4 | 9.0 | 6.0 | 4.0 | 0 |

MATRIZ DE PRED

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | None | 2 | 0 | 2 |
| 1 | 1 | None | 0 | 1 |
| 2 | 1 | 2 | None | 2 |
| 4 | 1 | 2 | 3 | None |

**Figura 8. Matrizes de Resultado - Grafo de Teste 2**

Dado que todos os resultados gerados são válidos e podem ser conferidos manualmente utilizando o pseudocódigo de cada algoritmo, assume-se que a implementação dos algoritmos está correta e que grafos maiores poderão ser processados e os resultados gerados serão verídicos.

### 3.2. Testes com grafos gerados aleatoriamente

Os parâmetros utilizados para criação de cada grafo gerados aleatoriamente estão logo abaixo, o tempo de execução e peso do caminho resultante também é apresentado.

| \|V\| = 50 | \|E\| = 500 | $w^{min}$ = 1 | $w^{max}$ =50 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 29.0 | 0.0009994 |
| BELLMAN-FORD | 29.0 | 0.00 |
| FLOYD-MARSHALL | 29.0 | 0.0247406 |

| \|V\| = 50 | \|E\| = 1500 | $w^{min}$ = 1 | $w^{max}$ =5 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 2.0 | 0.0010015 |
| BELLMAN-FORD | 2.0 | 0.00 |
| FLOYD-MARSHALL | 2.0 | 0.0301573 |

| \|V\| = 100 | \|E\| = 2000 | $w^{min}$ = 1 | $w^{max}$ =50 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 24.0 | 0.00 |
| BELLMAN-FORD | 24.0 | 0.00 |
| FLOYD-MARSHALL | 24.0 | 0.1849517 |

| \|V\| = 100 | \|E\| = 8000 | $w^{min}$ = 1 | $w^{max}$ =5 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 1.0 | 0.00 |
| BELLMAN-FORD | 1.0 | 0.00 |
| FLOYD-MARSHALL | 1.0 | 0.1799685 |

| \|V\| = 500 | \|E\| = 10000 | $w^{min}$ = 1 | $w^{max}$ =50 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 10.0 | 0.0093677 |
| BELLMAN-FORD | 10.0 | 0.0098896 |
| FLOYD-MARSHALL | 10.0 | 23.3083515 |

| \|V\| = 500 | \|E\| = 100000 | $w^{min}$ = 1 | $w^{max}$ =5 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 2.0 | 0.0252795 |
| BELLMAN-FORD | 2.0 | 0.0398917 |
| FLOYD-MARSHALL | 2.0 | 24.3682994 |

| \|V\| = 1000 | \|E\| = 50000 | $w^{min}$ = 1 | $w^{max}$ =50 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 10.0 | 0.0529940 |
| BELLMAN-FORD | 10.0 | 0.0850043 |
| FLOYD-MARSHALL | 10.0 | 202.0384995 |

| \|V\| = 1000 | \|E\| = 500000 | $w^{min}$ = 1 | $w^{max}$ =5 | Origem = 0 | Destino = 16 |
|---|---|---|---|---|---|

| | CUSTO | TEMPO DE EXECUÇÃO EM SEGUNDOS |
|---|---|---|
| DIJKSTRA | 2.0 | 0.1389727 |
| BELLMAN-FORD | 2.0 | 0.2330126 |
| FLOYD-MARSHALL | 2.0 | 206.9118173 |

**Figura 9. Resultado dos Testes com Grafos Aleatórios**

## 4. Conclusão

É indiscutível que todos os algoritmos cumprem perfeitamente com sua função de encontrar o menor caminho possível, mas com base nos testes apresentados podemos retirar algumas conclusões

O algoritmo Dijkstra como podemos avaliar possui um desempenho muito superior em relação ao algoritmo de Floyd-Warshall e chega a ser relativamente mais rápido que o algoritmo de Bellman-Ford, más vale lembrar que o algoritmo Dijkstra não se dá bem com arestas que possuem pesos negativos, o que torna inviável a sua utilização para analisar caminhos em grafos que possuem arestas com caminhos negativos.

O algoritmo Bellman-Ford apesar de possuir um desempenho relativamente menor que o Dijkstra para grafos que possuem apenas arestas com pesos positivos, a sua utilização o torna ideal em casos que precisamos encontrar caminhos mínimos em grafos que possuem arestas com pesos negativos, ele é um algoritmo robusto e que pode ser utilizado em diversas situações.

O algoritmo de Floyd-Warshall como já esperado pelo fato de sua complexidade computacional ser $O(|N|^3)$, é um algoritmo muito custoso e sua utilização em grafos mais densos pode exigir muito mais tempo de processamento como já pode ser observado nos resultados anteriores.

Todos esses fatores devem ser levados em consideração para escolher o algoritmo que melhor condiz com as necessidades da sua aplicação.

## Referências

[Jurkiewicz 2009] Jurkiewicz, S. (2009). Grafos–uma introduçao. *São Paulo: OBMEP*.

[Netto and Jurkiewicz 2017] Netto, P. O. B. and Jurkiewicz, S. (2017). *Grafos: introdução e prática*. Editora Blucher.