Universidade da região de Joinville

Bacharelado em Engenharia de Software

# Trabalho Estruturas de Dados

Matheus Campos

Renan Boettger

JANAINA FONTANA BIFFI DUARTE

## Introdução ao Problema do Caminho Mínimo

O problema do caminho mínimo é um dos problemas fundamentais em teoria dos grafos e algoritmos. Ele envolve encontrar o caminho de menor custo entre dois vértices em um grafo ponderado, onde cada aresta possui um peso que representa o custo de transitar por ela. Este problema tem diversas aplicações práticas, como em sistemas de navegação GPS, onde é necessário determinar a rota mais rápida ou curta entre dois pontos, em redes de computadores para encontrar o caminho mais eficiente para a transmissão de dados, em logística para otimização de rotas de entrega, e em muitas outras áreas onde a eficiência de trajetos é crucial.

Além disso, o problema do caminho mínimo é essencial em engenharia de tráfego, planejamento urbano, robótica, e até mesmo em jogos eletrônicos, onde personagens ou unidades precisam se mover de maneira eficiente em um espaço dado. Em finanças, algoritmos de caminho mínimo podem ser usados para encontrar a rota mais barata de transações financeiras entre diferentes moedas. Na biologia computacional, eles ajudam a modelar e analisar redes metabólicas e de proteínas.

Resolver o problema do caminho mínimo não é apenas uma questão de encontrar uma solução; trata-se de otimizar recursos e tempo, algo crucial em ambientes competitivos e em situações onde a eficiência pode significar a diferença entre o sucesso e o fracasso de um projeto ou empresa. Por isso, diversos algoritmos foram desenvolvidos para abordar esse problema de diferentes maneiras, cada um com suas peculiaridades e aplicações específicas.

## Algoritmo de Bellman-Ford

### *História*

O algoritmo de Bellman-Ford foi proposto por Richard Bellman e Lester R. Ford em 1958. Bellman foi um matemático americano conhecido por suas contribuições à programação dinâmica e ao controle ótimo, enquanto Ford, também americano, trabalhou extensivamente em teoria dos grafos e análise de redes. O algoritmo Bellman-Ford é uma extensão do trabalho de Bellman em programação dinâmica e foi projetado para resolver o problema do caminho mínimo em grafos que podem conter arestas com pesos negativos. A capacidade de lidar com pesos negativos torna o algoritmo de Bellman-Ford particularmente útil em aplicações como a análise de redes econômicas, onde as relações de custo podem ser complexas e não lineares.

### *Complexidade*

A complexidade do algoritmo de Bellman-Ford é $O(V \cdot E)$, onde $V$ é o número de vértices e $E$ é o número de arestas no grafo. Isso significa que o tempo de execução do algoritmo é proporcional ao produto do número de vértices pelo número de arestas, tornando-o menos eficiente que alguns outros algoritmos para grafos densos. No entanto, sua capacidade de detectar ciclos negativos é uma vantagem significativa em muitos cenários práticos.

### *Pseudo-código*

```
funcao BellmanFord(grafo G, inteiro s)
    para cada vertice v em G
        dist[v] <- infinito
        predecessor[v] <- nulo
    fimpara
    dist[s] <- 0

    para i de 1 ate |V| - 1 faca
        para cada aresta (u, v) com peso w em G faca
            se dist[u] + w < dist[v] entao
                dist[v] <- dist[u] + w
                predecessor[v] <- u
            fimse
        fimpara
    fimpara

    para cada aresta (u, v) com peso w em G faca
```

```
        se dist[u] + w < dist[v] entao
            erro "O grafo contem um ciclo de peso negativo"
        fimse
    fimpara

    retorna dist, predecessor
fimfuncao
```

### *Funcionamento*

1. Inicializa a distância de todos os vértices como infinita e a distância da origem como zero.
2. Relaxa todas as arestas repetidamente (|V| - 1 vezes).
3. Verifica se há ciclos de peso negativo, verificando uma vez mais se é possível relaxar alguma aresta.

## Algoritmo de Dijkstra

### *História*

O algoritmo de Dijkstra foi desenvolvido por Edsger W. Dijkstra em 1956. É amplamente utilizado para encontrar o caminho mais curto entre dois pontos em um grafo com arestas de peso não negativo.

### *Complexidade*

A complexidade do algoritmo de Dijkstra, quando implementado com uma fila de prioridade, é $O((V+E)\log V)$ $O((V+E)\ \backslash log\ V)$ O((V+E)logV).

### *Pseudo-código*

```
funcao Dijkstra(grafo G, inteiro s)
    dist[s] <- 0
    para cada vertice v em G - {s}
        dist[v] <- infinito
    fimpara
    Q <- todos os vertices em G

    enquanto Q nao estiver vazia faca
        u <- vertice em Q com a menor distancia
        remover u de Q

        para cada vizinho v de u faca
            alt <- dist[u] + comprimento(u, v)
            se alt < dist[v] entao
                dist[v] <- alt
                predecessor[v] <- u
            fimse
        fimpara
    fimenquanto

    retorna dist, predecessor
fimfuncao
```

### *Funcionamento*

4. Inicializa a distância de todos os vértices como infinita, exceto a origem, que é zero.
5. Utiliza uma fila de prioridade para selecionar o vértice com a menor distância ainda não processada.
6. Relaxa todas as arestas do vértice selecionado.
7. Repete até que todos os vértices sejam processados.

## Algoritmo de Floyd-Warshall

### *História*

O algoritmo de Floyd-Warshall foi proposto por Robert Floyd em 1962, baseado no trabalho de Stephen Warshall. Ele resolve o problema do caminho mínimo para todos os pares de vértices em um grafo.

### *Complexidade*

A complexidade do algoritmo de Floyd-Warshall é $O(V3)O(V^3)$O(V3).

### *Pseudo-código*

```
funcao FloydWarshall(grafo G)
    para cada vertice i em G faca
        para cada vertice j em G faca
            se i = j entao
                dist[i][j] <- 0
            senao
                se existe aresta(i, j) entao
                    dist[i][j] <- peso(i, j)
                senao
```

```
                dist[i][j] <- infinito
            fimse
        fimse
    fimpara
fimpara

para cada vertice k em G faca
    para cada vertice i em G faca
        para cada vertice j em G faca
            se dist[i][j] > dist[i][k] + dist[k][j] entao
                dist[i][j] <- dist[i][k] + dist[k][j]
            fimse
        fimpara
    fimpara
fimpara

retorna dist
fimfuncao
```

### *Funcionamento*

8. Inicializa uma matriz de distâncias com os pesos das arestas.
9. Itera sobre todos os pares de vértices (i, j) para todos os vértices intermediários k.
10. Atualiza a distância entre (i, j) se passar por k resulta em um caminho mais curto.
11. Repete até que todas as combinações de vértices tenham sido consideradas.

## Referências


- Santos, A. (2006). **Algoritmos em Grafos**. Editora da Universidade de São Paulo (USP).
- Ziviani, N. (2011). **Projeto de Algoritmos: Com Implementações em Pascal e C**. Cengage Learning.
- Goldbarg, M. C., & Goldbarg, E. F. (2001). **Algoritmos**. Elsevier.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2012). **Algoritmos: Teoria e Prática**. Elsevier.
- Meidanis, J., & Fonseca, A. J. (2008). **Algoritmos em Grafos: Programação Dinâmica, Backtracking, Branch-and-Bound**. Unicamp.