Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

SIN 343
Desafios de Programação

João Batista Ribeiro

joao42lbatista@gmail.com

Slides baseados no material do prof. Guilherme C. Pena

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Teoria de Grafos

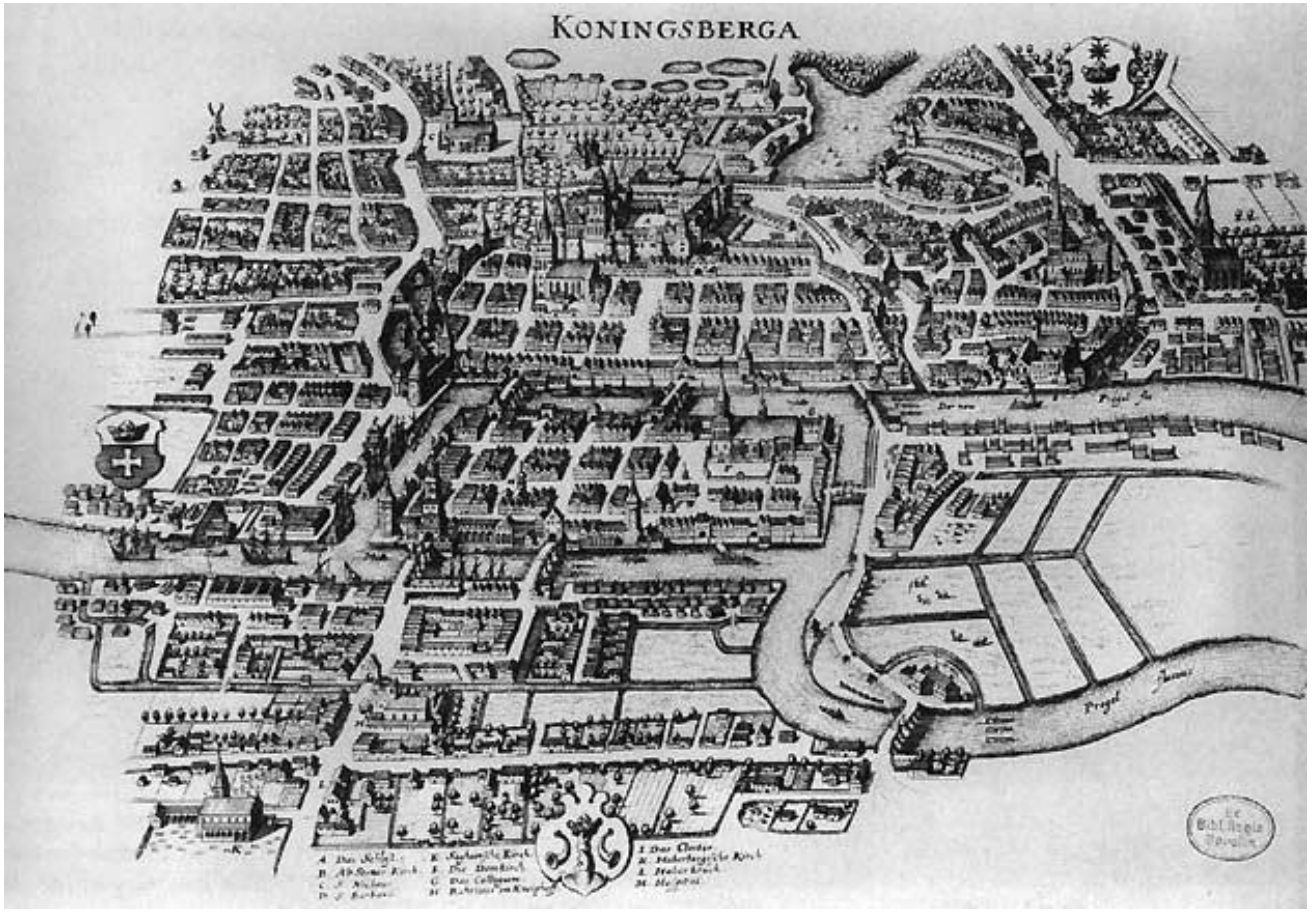
Grafos

Na cidade de Königsberg, na antiga Prússia, havia 7 pontes que cruzavam o rio Pregel, ligando suas margens e duas ilhas da cidade.

Os habitantes se perguntavam se era possível passar por todas elas exatamente uma vez, ou seja, se havia um caminho que não repetia ponte (e incluía todas).

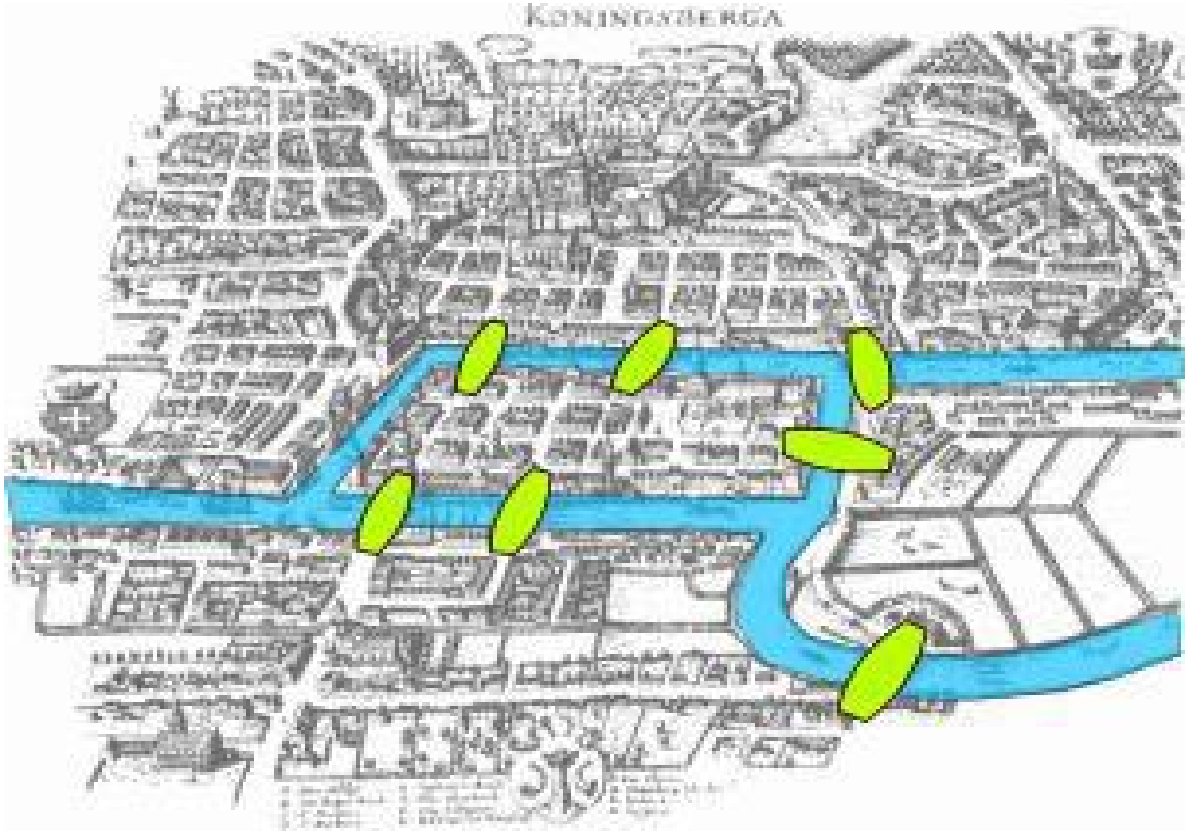
Grafos

Desenho da cidade de Königsberg, naquela época.



Grafos

Destacando as pontes



Em 1736, Euler mostrou que era impossível.

Seu trabalho é considerado o 1º *paper* de *teoria de grafos*.

Grafos

Muitas situações do mundo real podem ser descritas por um diagrama contendo:

- um ***conjunto de pontos***.
- um ***conjunto de linhas*** unindo certos pares desses pontos.

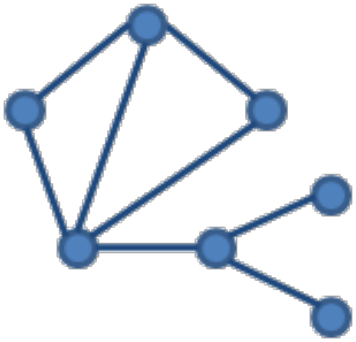
O mais importante é saber se dois pontos são unidos ou não por um linha. A forma como são unidos é, geralmente, irrelevante.

Grafos

Grande representatividade: **Engenharia, Computação, Matemática, Economia** etc.

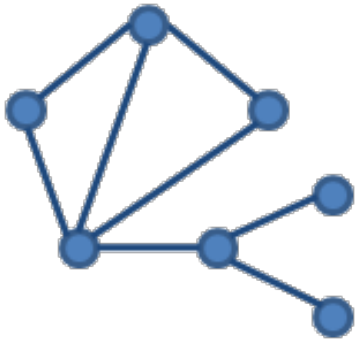
Exemplos de informações representadas por grafos:

- Mapas de rodovias
- Redes de computadores
- Redes de transporte e de comunicação
- Circuitos impressos
- Dentre muitas outras.



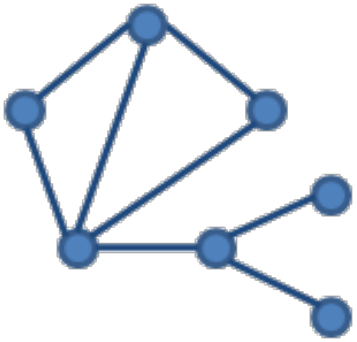
Grafos

Mapa:



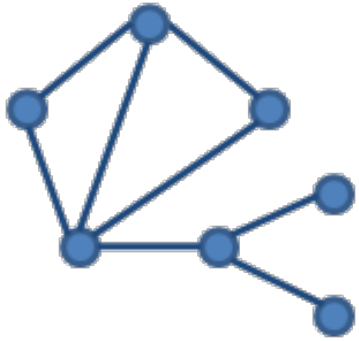
- Cidades
- Estradas

Rede Social:



- Pessoas
- Amizade

Infra-estrutura de rede:



- Computadores
- Cabos

...

Grafos

Um ***grafo*** é um par ordenado ***G = (V , A)***, sendo ***V*** um conjunto de vértices (pontos) e ***A*** *(E - Egde)* um conjunto de arestas (linhas), cada uma conectando dois vértices (u, v) ∈ V. ***G = (V, A) ou G = (V, E).***

O desenho de um grafo meramente descreve a relação entre vértices e arestas, não há uma forma única de desenhar um grafo.

As posições relativas dos pontos e linhas não tem significado.

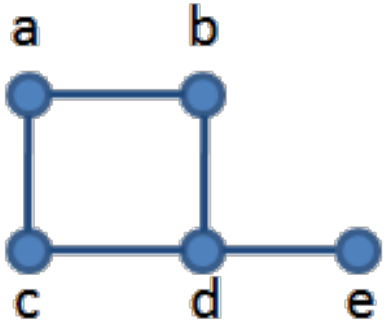
Grafos

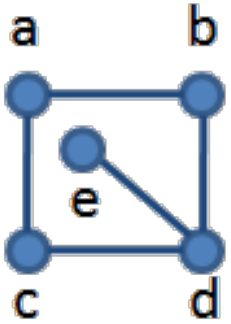
Exemplo de Grafo:

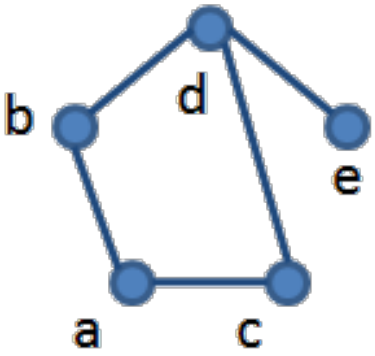
G = (V, A)

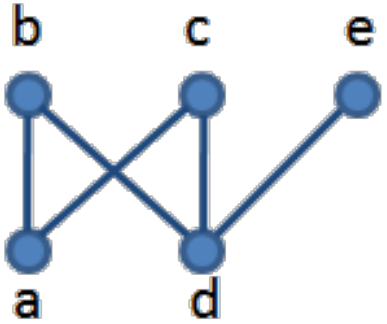
V = {a, b, c, d, e}

A = {{a, b}, {a, c}, {b, d}, {c, d}, {d, e}}





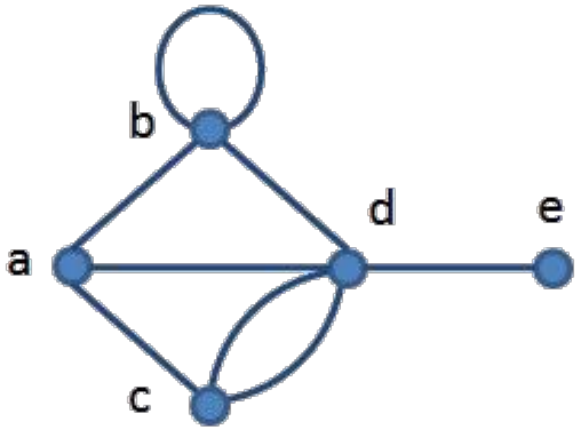




Grafos (Notações)

Notações e Nomenclaturas sobre grafos:

- Os vértices de uma aresta são ditos ***incidentes*** à aresta, e vice-versa.
- Dois vértices incidentes a uma aresta em comum são ditos ***adjacentes*.**
- Uma aresta com vértices incidentes idênticos é chamada ***loop*** ou laço.
- ***Múltiplas arestas*** podem incidir em um mesmo par de vértices.



- a e b são **incidentes** a {a, b}
- a e b são **adjacentes**
- b e c não são **adjacentes**
- {b, b} é um **loop**
- há **múltiplas arestas** {c, d}

Grafos (Notações)

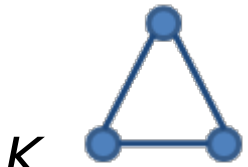
Notações e Nomenclaturas sobre grafos:

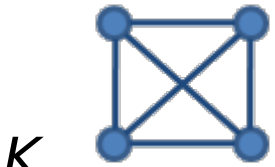
- **Grafo Simples**: Um grafo é simples se não tem loops nem arestas múltiplas .

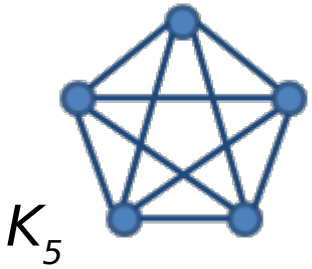
  A teoria de grafos estuda principalmente as propriedades de grafos simples

- **Grafo Completo:** Um grafo completo é um grafo simples em que cada par de vértices é unido por uma aresta.

  Há apenas um grafo completo com *n* vértices. Tal grafo é denotado por K_n.





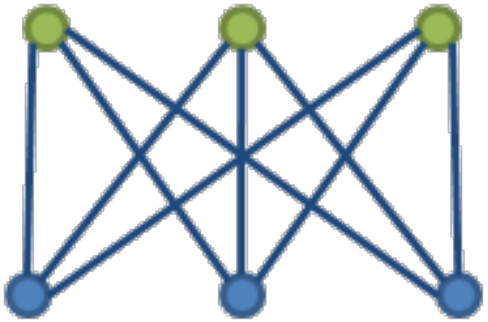


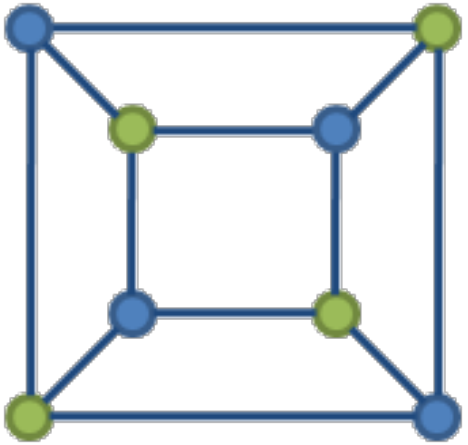
Grafos (Notações)

Notações e Nomenclaturas sobre grafos:

- **Grafo Bipartido**: Um grafo bipartido é um cujo conjunto de vértices pode ser divido em dois subconjuntos X e Y tal que toda aresta seja incidente a vértice em X e a um em Y .

  Ou seja, não há arestas incidentes a vértices do mesmo subconjunto

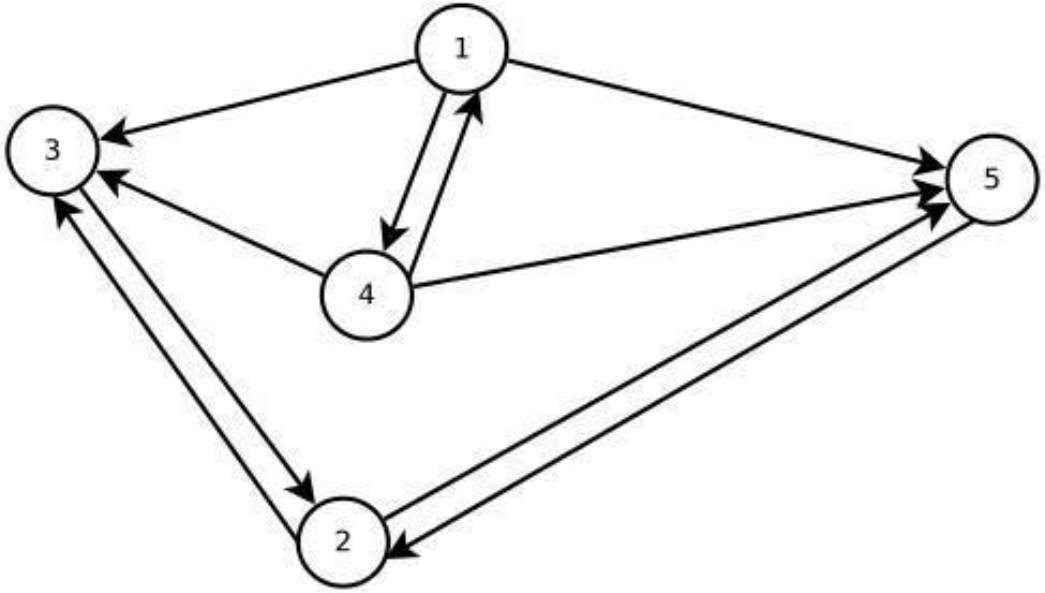




Grafos (Notações)

Notações e Nomenclaturas sobre grafos:

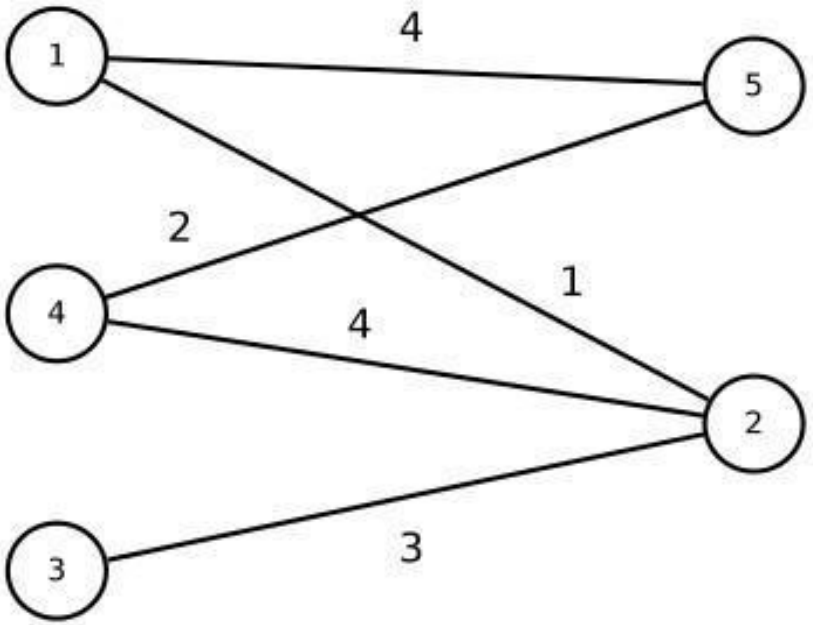
- **Grafo Direcionado**: É um grafo que possui uma aresta especial também chamada de ***arco***, o arco (u, v) indica uma ligação que sai do vértice *u* e entra no vértice *v*. O vértice u é adjacente ao vértice v. **(u, v) ≠ (v , u).**



Grafos (Notações)

Notações e Nomenclaturas sobre grafos:

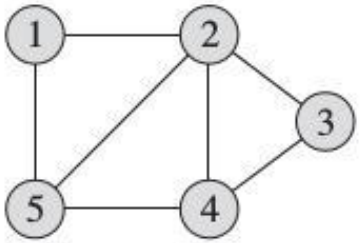
- **Grafo Ponderado**: É um grafo em que as arestas (ou arcos) possuem pesos associados a elas.

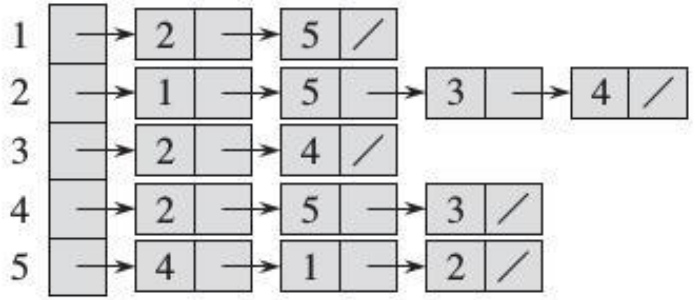


Grafos (Representação)

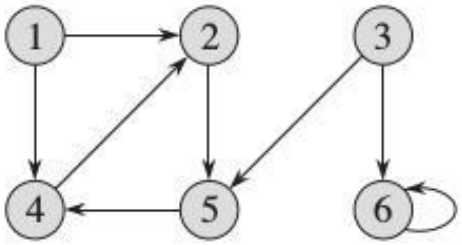
Representação de Grafos:

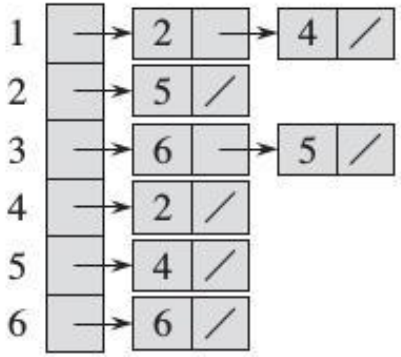
Um grafo G = (V, A) representa-se como uma **lista de adjacência** ou como uma **matriz de adjacência**.





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Grafos (Algoritmos)

Principais Algoritmos:

- Busca em Largura (BFS – Breadth First Search)
- Busca em Profundidade (DFS – Depth First Search)
- Árvore Geradora Mínima (MST - Minimum Spanning Tree)
- Algoritmos de Caminho Mínimo.
- Algoritmos de Fluxo Máximo.

Caminho Mínimo

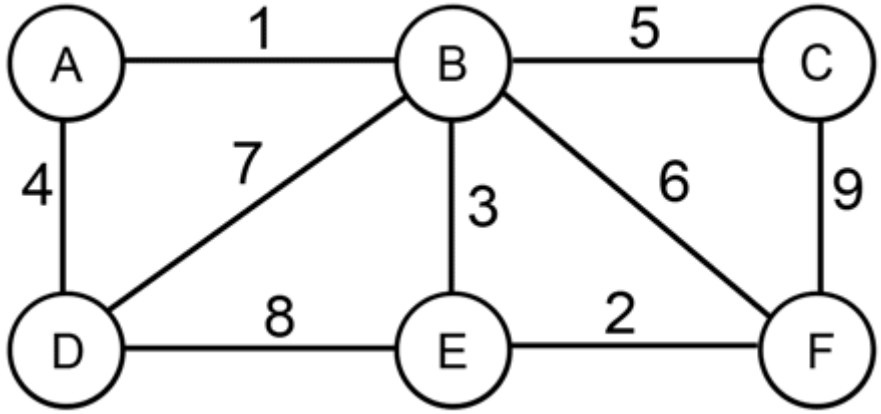
Problema do Caminho Mínimo:

Dado um grafo ponderado e dois vértices ***s*** e ***t*** , descobra um ***caminho-(s, t)*** de menor valor.

- Algoritmo de Dijkstra

- Algoritmo de Floyd-Warshall

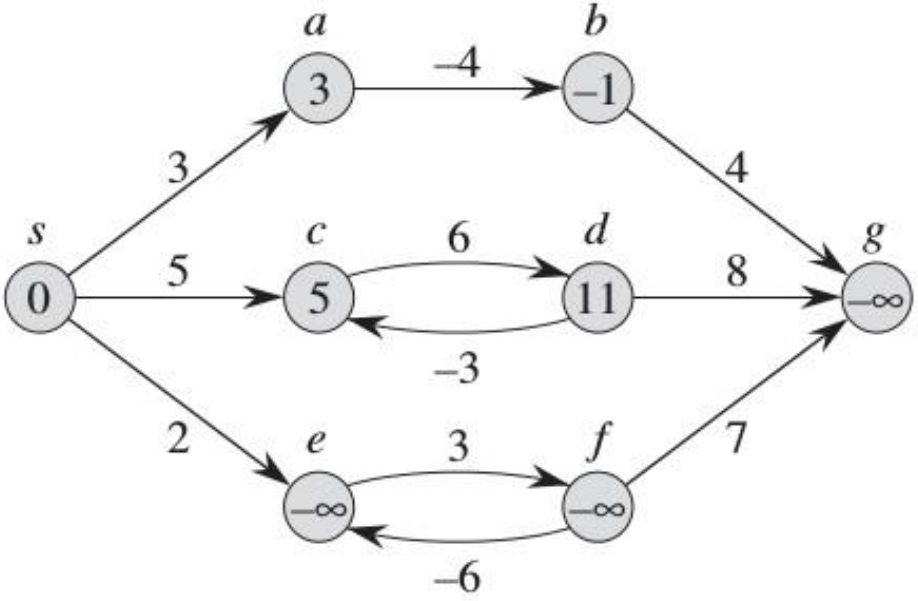
- Algoritmo de Bellman-Ford

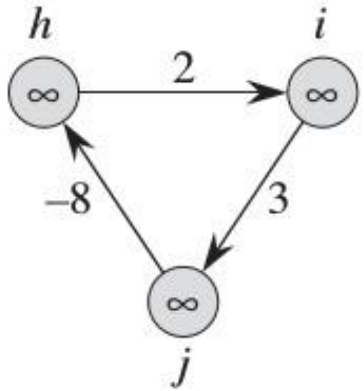


Caminho Mínimo

Problema do Caminho Mínimo:

Arestas e ***ciclos*** de peso negativo: Algumas instâncias podem conter arestas ou ciclos de pesos negativos, e isso pode influenciar muito na descoberta do caminho mínimo.





Caminho Mínimo

Algoritmo de Dijkstra

- descobre o menor caminho de 1 para todos
- não funciona se houver peso negativo

Algoritmo de Floyd-Warshall

- descobre o menor caminho de todos para todos
- funciona inclusive com peso negativo (mas não detecta ciclo negativo)

Algoritmo de Bellman-Ford

- descobre o menor caminho de 1 para todos
- funciona inclusive com peso negativo desde que não alcance ciclo negativo (mas detecta um ciclo negativo)

Algoritmo de Dijkstra

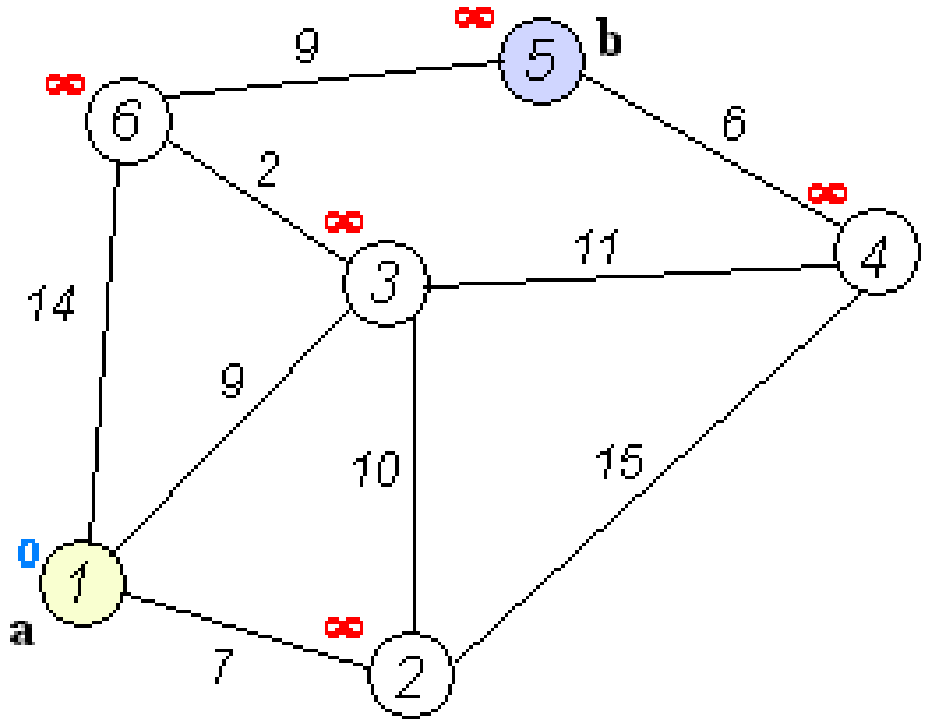
O algoritmo de **Dijkstra** resolve o problema de caminhos mínimos de fonte única em um grafo direcionado ponderado G = (V, A) para o caso no qual todos os pesos de arestas são ***não-negativos***.

```
Dijkstra(Grafo, origem)
  create queue Q

  for all vertex in Grafo:
    dist[v] ← INFINITY
    prev[v] ← NULL
    add v to Q

  dist[origem] ← 0
```

```
while Q is not empty:
  u ← vertex in Q with min dist[u]
  remove u from Q
  for each neighbor v of u:
     alt ← dist[u] + length(u, v)
     if alt < dist[v]:
      dist[v] ← alt
      prev[v] ← u
  return dist[], prev[]
```

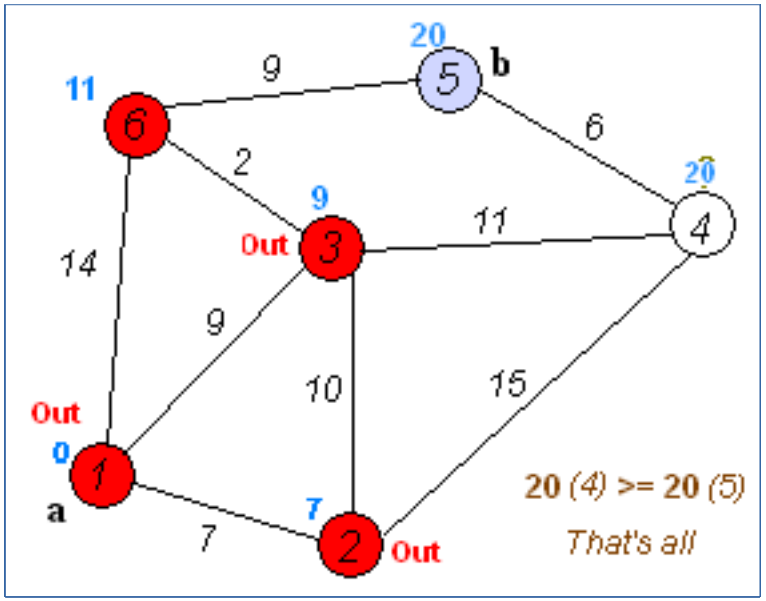
Algoritmo de Dijkstra

```
Dijkstra(Grafo, origem)
  create queue Q

  for all vertex in Grafo:
    dist[v] ← INFINITY
    prev[v] ← NULL
    add v to Q

  dist[origem] ← 0
```



Algoritmo de Dijkstra

```
while Q is not empty:
  u ← vertex in Q with min dist[u]
  remove u from Q
  for each neighbor v of u:
     alt ← dist[u] + length(u, v)
     if alt < dist[v]:
      dist[v] ← alt
      prev[v] ← u
  return dist[], prev[]
```



Algoritmo de Dijkstra

No algoritmo de Dijkstra, se quisermos apenas o caminho da ***origem*** até o ***destino***, basta incluir o seguinte código após a remoção do vértice ***u***:

```
while Q is not empty:
  u ← vertex in Q with min dist[u]
  remove u from Q
```

```
  if u == destino:
     create stack S
     while prev[u] != NULL:
       add u to S
       u ← prev[u]
     add u to S
```

```
  for ...
```

Algoritmo de Floyd-Warshall

O algoritmo de **Floyd-Warshall** resolve o problema do caminho mínimo de todos para todos vértices em um grafo direcionado ponderado G = (V, A) para o caso no qual ***não existam ciclos negativos***. **$\Theta(V^3)$.**

```
Floyd_Warshall(Grafo, dist, next)
  for i from 1 to |V|
    for j from 1 to |V|
      dist[i][j] ← INFINITY
      next[i][j] ← NULL

  ...
```

Algoritmo de Floyd-Warshall

O algoritmo de **Floyd-Warshall** resolve o problema do caminho mínimo de todos para todos vértices em um grafo direcionado ponderado G = (V, A) para o caso no qual ***não existam ciclos negativos***. **$\Theta(V^3)$.**

```
...
for each edge (u,v)
  dist[u][v] ← weight(u,v) //Peso da aresta (u,v)
  next[u][v] ← v
for k from 1 to |V|
  for i from 1 to |V|
    for j from 1 to |V|
      if dist[i][j] > dist[i][k] + dist[k][j] then
        dist[i][j] ← dist[i][k] + dist[k][j]
        next[i][j] ← next[i][k]
```

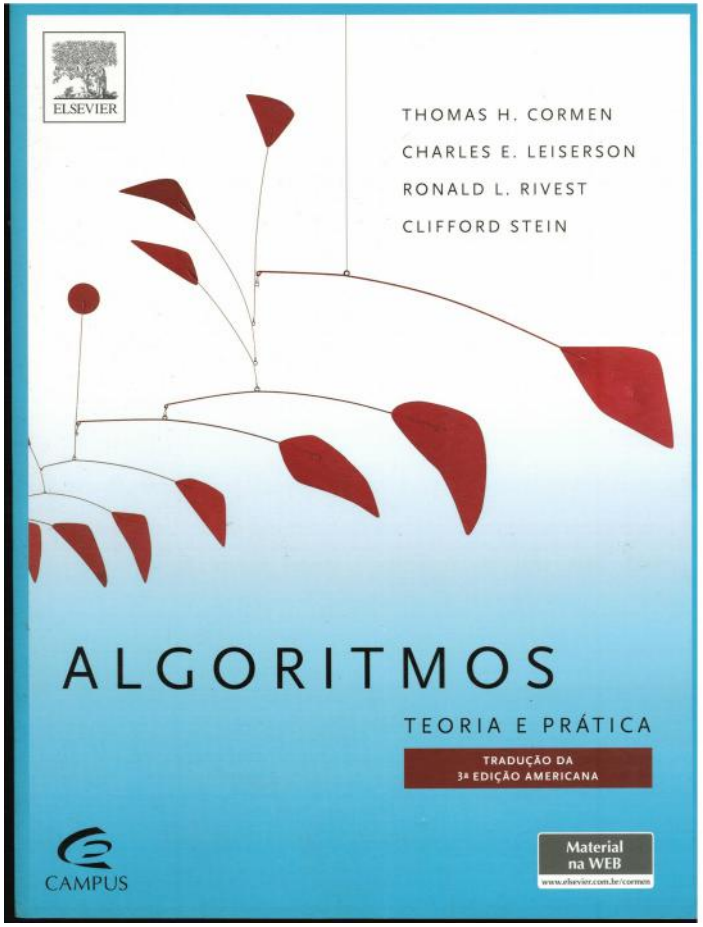
Algoritmo de Floyd-Warshall

No algoritmo de Floyd-Warshall, se quisermos o caminho da ***origem (u)*** até o ***destino (v)***, basta fazer a seguinte função:

```
Caminho_Floyd_Warshall(u, v, next)
  if next[u][v] = null then
    return []
  create queue Q
  add u to Q
  while u ≠ v
    u ← next[u][v]
    add u to Q
  return Q
```

Referência Bibliográfica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. Tradução da 2. ed. Americana. Rio de Janeiro: Campus, 2002.



Referência Bibliográfica

- SKIENA, S.S. REVILLA, M. A. Programming challenges: the programming contest training manual. Springer, 2003.