UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Algoritmo de Dijkstra
Caminho mínimo de origem única

Bruno Santos de Lima

brunoslima4@gmail.com

Estrutura de dados
Docente: Profa. Dra. Roberta Spolon
Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Índice

Caminho Mínimo

Caminho Mínimo

- Um motorista deseja sair de São Paulo e ir ao Rio de Janeiro
  - Qual é o caminho mais curto entre essas duas cidades?

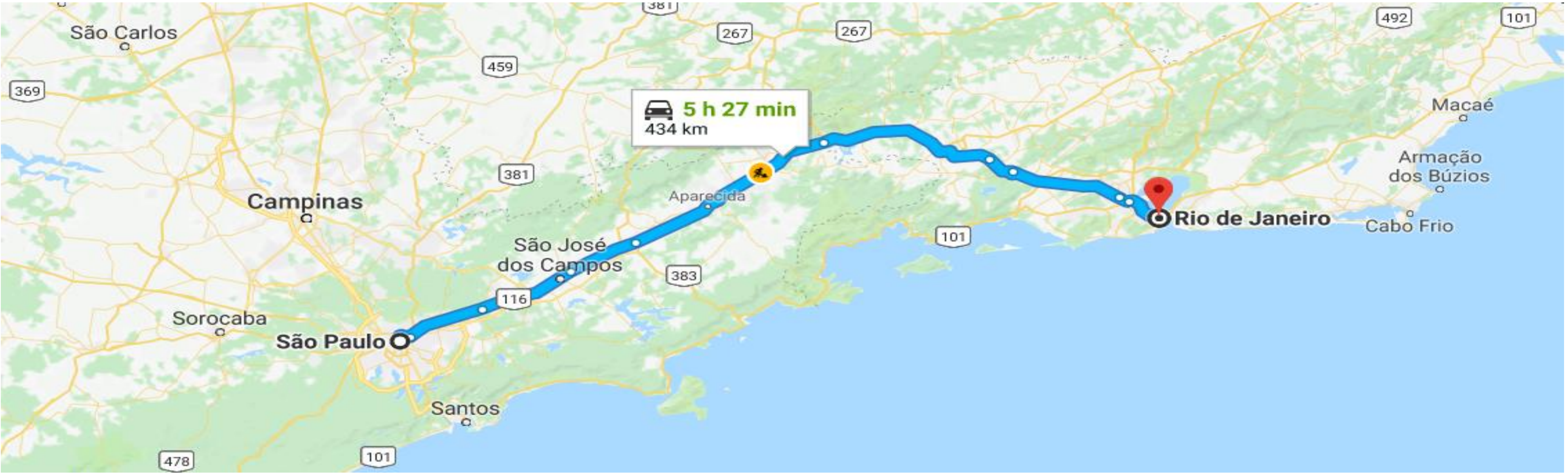


Figura 1: Caminho de São Paulo até o Rio de Janeiro – Fonte: Google Maps

Caminho Mínimo

- Como saber se o caminho anterior é o mais curto?
  - Afinal, existem vários caminhos distintos para esse trajeto!

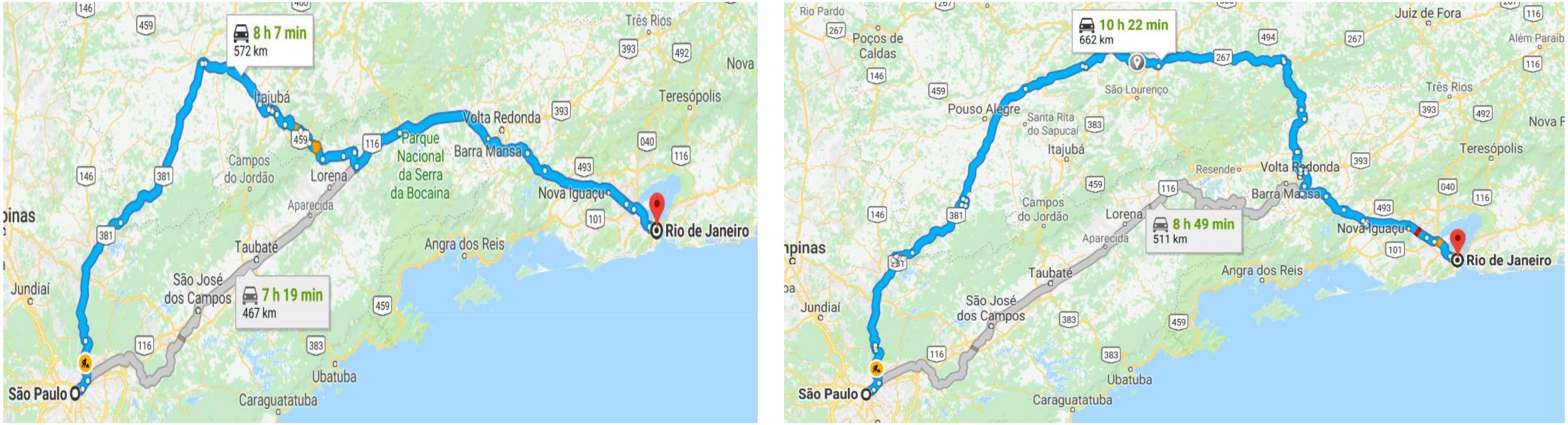


Figura 2: Caminho de São Paulo até o Rio de Janeiro – Fonte: Google Maps

Caminho Mínimo

- Uma abordagem para resolver esse problema seria um algoritmo que liste todas as rotas possíveis
  - Conhecendo todas as possíveis rotas, basta calcular a distância de cada uma delas.
    - Após isso, escolher a rota com a menor distância!

Simples!!!

Caminho Mínimo

- Uma abordagem para resolver esse problema seria um algoritmo que liste todas as rotas possíveis
  - Conhecendo todas as possíveis rotas, basta calcular a distância de cada uma delas.
    - Após isso, escolher a rota com a menor distância!

~~Simples!!!~~

Na verdade não!!!

Essa abordagem para solucionar o problema é simples e intuitiva, porem em casos reais haverá milhões ou bilhões de possibilidades resultando em um tempo muito alto para encontrar uma resposta!

Complexidade Exponencial

Caminho Mínimo

- Outra abordagem:
  - Verificar se a rota incompleta possui distancia maior que a melhor rota conhecida.
    - Se isso acontecer, essa rota incompleta pode ser descartada do conjunto solução.

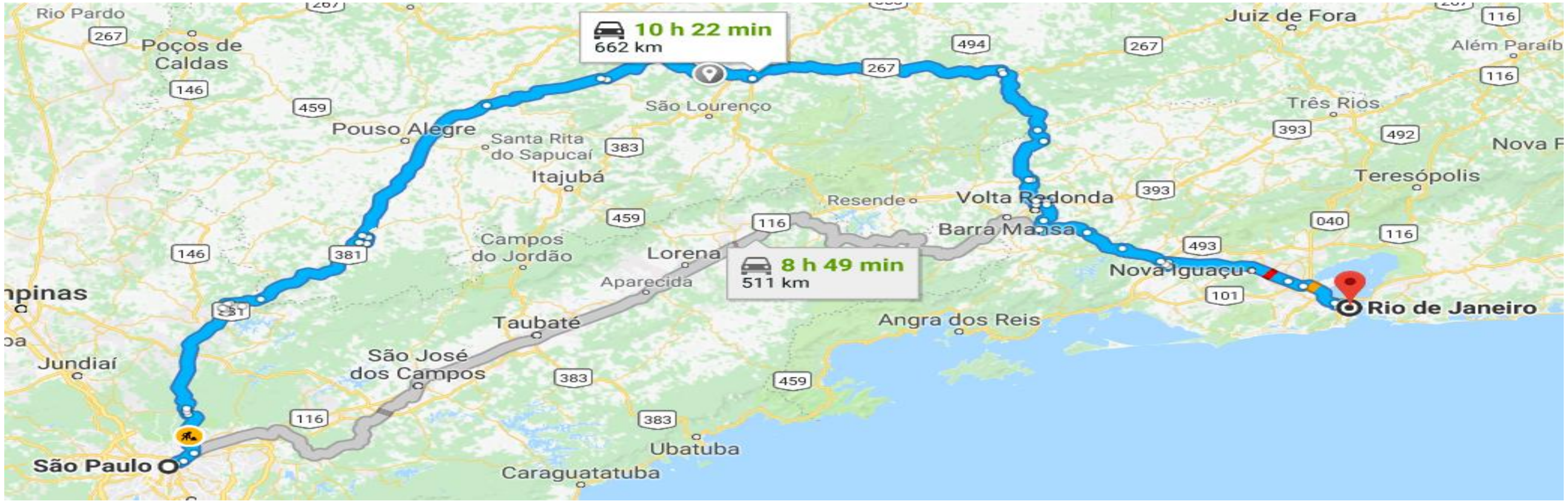


Figura 3: Caminho de São Paulo até o Rio de Janeiro – Fonte: Google Maps

Caminho Mínimo

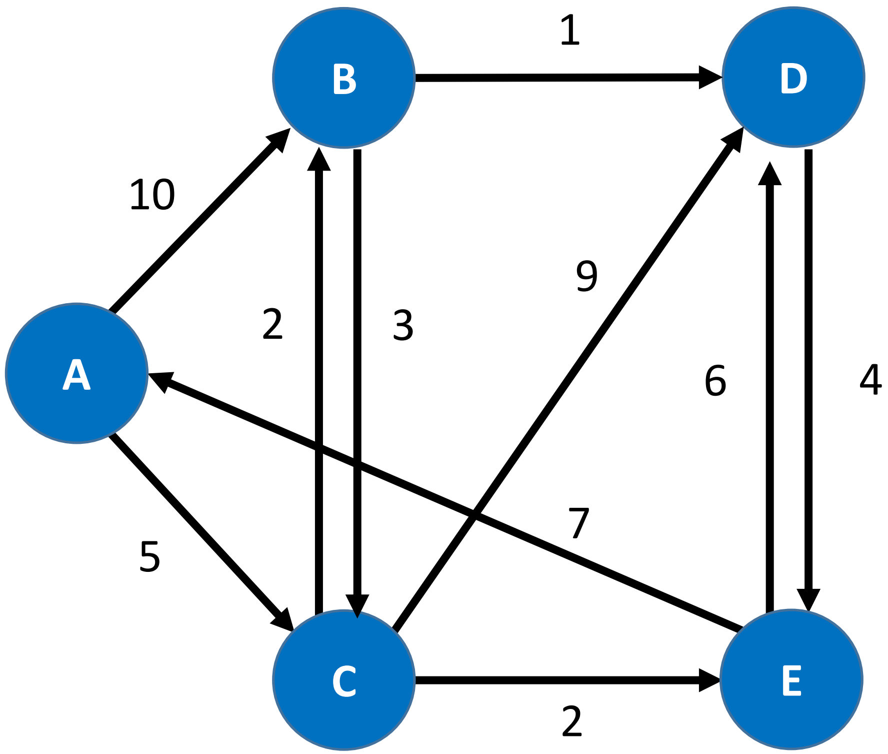
- Variantes do problema [3]:
  - Caminho mínimo de origem única
  - Caminho mínimo de destino único
  - Caminho mínimo de um único par
  - Caminho mínimo de todos para todos

Caminho Mínimo

- Variantes do problema [3]:
  - **Caminho mínimo de origem única**
    - **Algoritmo de Dijkstra**
  - Caminho mínimo de destino único
  - Caminho mínimo de um único par
  - Caminho mínimo de todos para todos

Caminho Mínimo

- Mapeamento do problema através Grafos

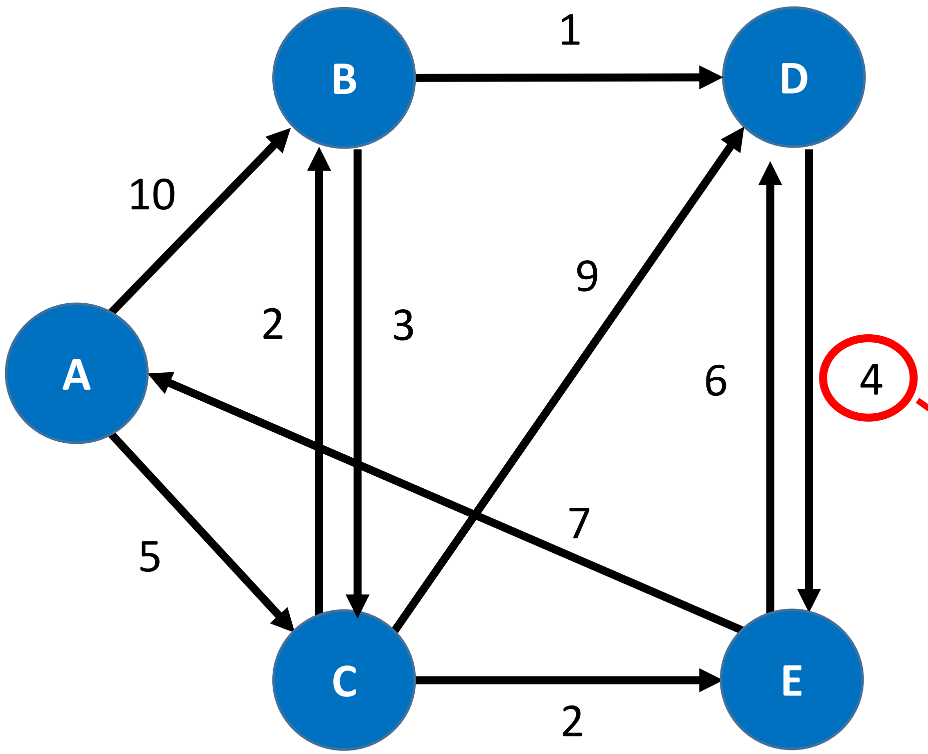


No caso da menor distância entre cidades:

A – Presidente Prudente
B – Martinópolis
C – Rio Claro
D – Bauru
E – São José do Rio Preto

Caminho Mínimo

- Mapeamento do problema através Grafos



No caso da menor distância entre cidades:

A – Presidente Prudente
B – Martinópolis
C – Rio Claro
D – Bauru
E – São José do Rio Preto

Distância, pode ser aplicado métricas diferentes: Tempo, custo, penalidades, perdas.

Caminho Mínimo

- **Caminho mínimo de origem única**

- Considerando um **grafo ponderado**: $G = (V, A)$.
  - Um **caminho**: $\boldsymbol{c} = (v_{0\,,} v_1, \dots, v_n)$

  - Os **pesos de um caminho c** é a soma de todos os pesos das arestas do caminho:

$$p = \sum_{i=1}^{n} p(v_{i-1,} v_i)$$

  - Sendo o **caminho mais curto** definido por:

$$\delta(u, v) = \begin{cases} \min\{p(c) : u \to v\}, & \textit{se existir um caminho de } \boldsymbol{u} \textit{ a } \boldsymbol{v}, \\ \infty, & \textit{caso contrário} \end{cases}$$

Algoritmo de Dijkstra

Algoritmo de Dijkstra

"O algoritmo de Dijkstra resolve o problema de caminho mais curto de fonte única em dígrafos com pesos não-negativos [4]."

- Algoritmo de Dijkstra:
  - Concepção: 1956 – Publicação: 1959 [1]
  - Edsger Wybe Dijkstra (1930 – 2002)
    - Um dos mais influentes na Ciência da Computação.
    - Recebeu o Prêmio ACM Turing.

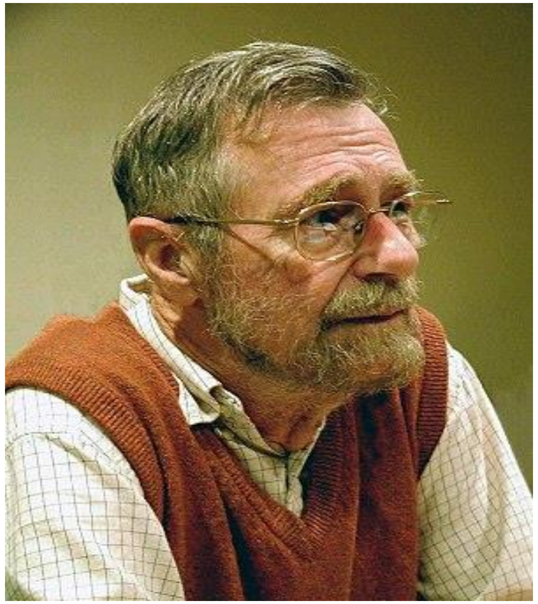


Figura 3: Edsger W. Dijkstra [6]

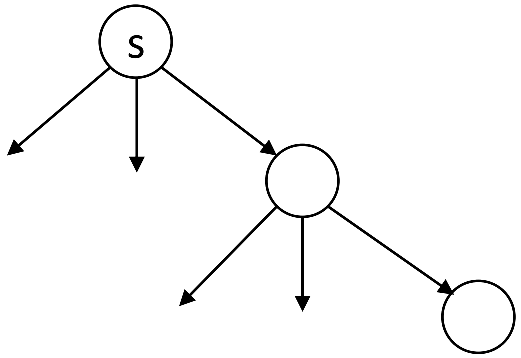
Algoritmo de Dijkstra

- **Propriedades**:
  - Grafo:
    - Direcionado ou não direcionado
    - Ponderado
    - Pode conter ciclos
    - As arestas não podem possuir pesos negativos
  - **Algoritmo Guloso**
    - Exato, sempre **encontra a melhor solução**, no caso, menor caminho da raiz para todos os nós do grafo.

Algoritmo de Dijkstra

- Resultado do algoritmo:

“Ao final da execução do algoritmo de Dijkstra é produzido uma arvore de caminhos mais curtos de um vértice origem **s** para todos os vértices alcançáveis a partir de **s** [2].”



Algoritmo de Dijkstra

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
      u = ExtrairMinimo(Q);
      S += u;
      para cada v ϵ Adj(u)
         Relaxamento(u,v,peso);
      fim para
   fim enquanto
fim
```

- **Funções auxiliares:**
  - ✓ **Inicializar();**
  - ✓ **Relaxamento();**
  - ✓ **ExtrairMinimo();**
- **Estruturas auxiliares:**
  - ✓ **Grafo**
  - ✓ **Fila, lista ou outra:** Variáveis **S** e **Q**
    - **S:** vértices com distância definitiva
    - **Q:** Vértices com distância provisória

Algoritmo de Dijkstra

Funções auxiliares

```
Inicializar(grafo,verticeInicial)
  para cada v ϵ grafo
    dis[v] = ∞;
    pai[v] = null;
  fim para
  dis[verticeInicial] = 0;
fim
```

```
Relaxamento(u, v, peso)
  se dis[v] > (dis[u] + peso(u,v)) então
       dis[v] = dis[u] + peso(u,v);
       pai[v] = u;
  fim se
fim
```

- ✓ **dis[v]:** Distância da origem até v
- ✓ **pai[v]:** Vértice pai de v

Algoritmo de Dijkstra

Funções auxiliares

```
Inicializar(grafo,verticeInicial)
  para cada v ϵ grafo
    dis[v] = ∞;
    pai[v] = null;
  fim para
  dis[verticeInicial] = 0;
fim
```

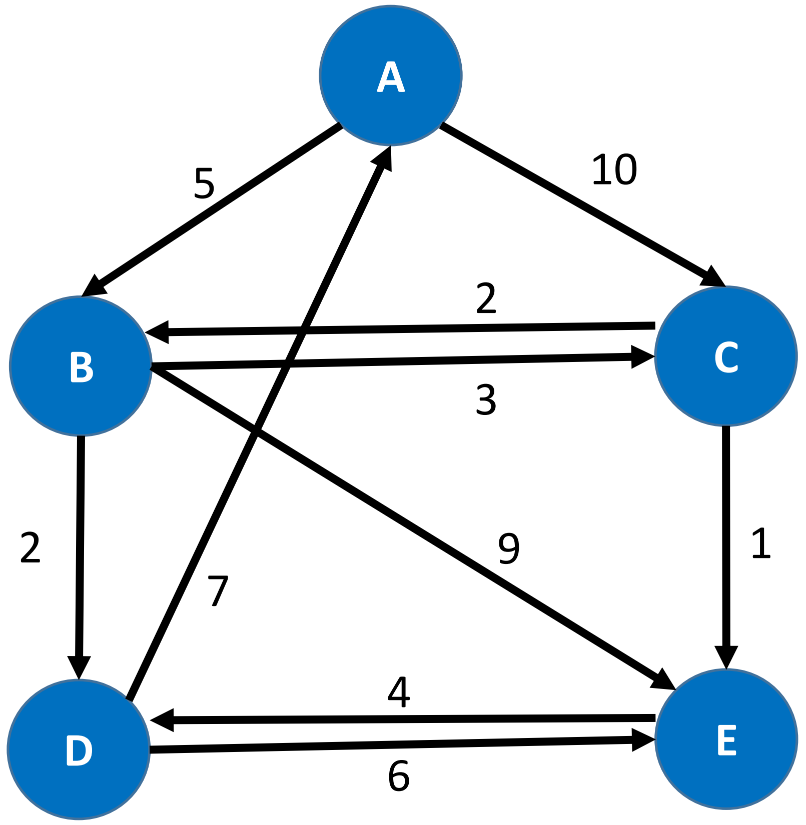
```
Relaxamento(u, v, peso)
  se dis[v] > (dis[u] + peso(u,v)) então
       dis[v] = dis[u] + peso(u,v);
       pai[v] = u;
  fim se
fim
```

Verificar se o caminho atual é melhor que o melhor caminho já conhecido!

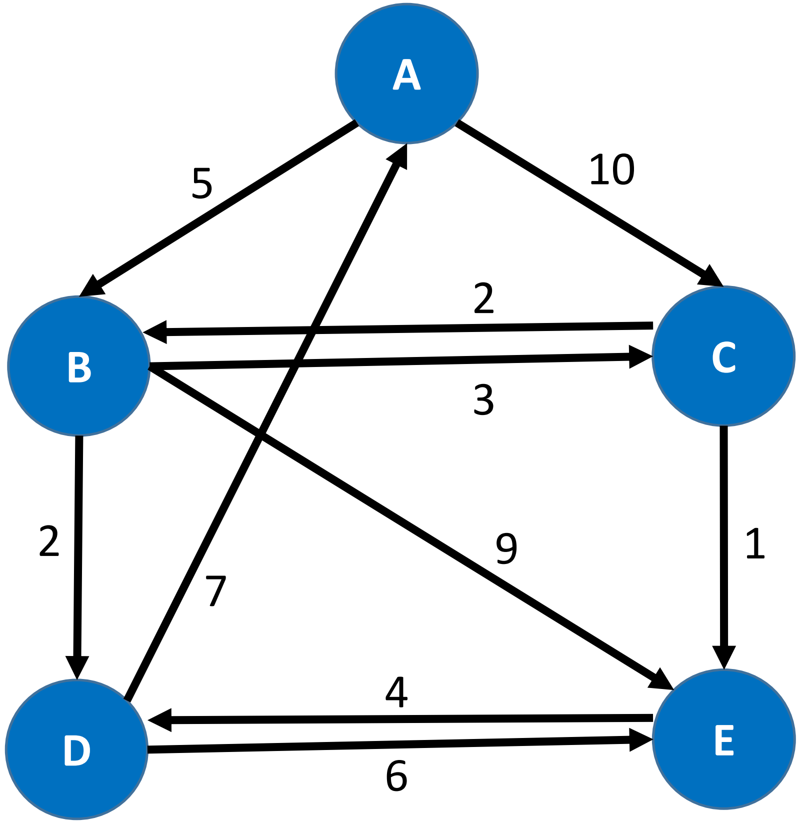
- ✓ **dis[v]:** Distância da origem até v
- ✓ **pai[v]:** Vértice pai de v

Algoritmo de Dijkstra

Exemplificação do algoritmo de Dijkstra



Algoritmo de Dijkstra

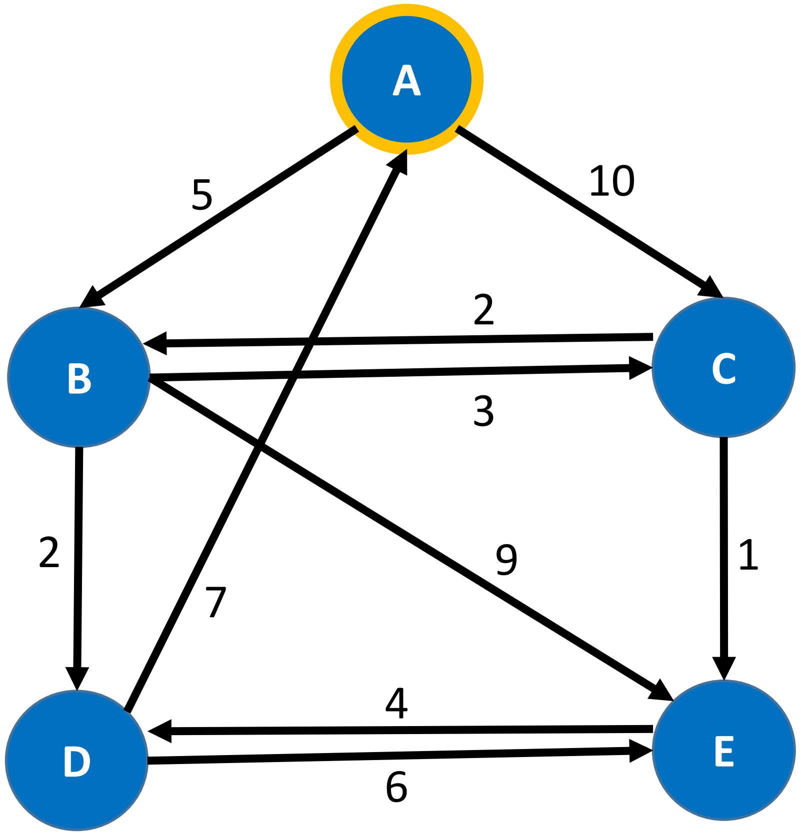


Vértices	A	B	C	D	E
Dis					
Pai					
Q					
S					

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

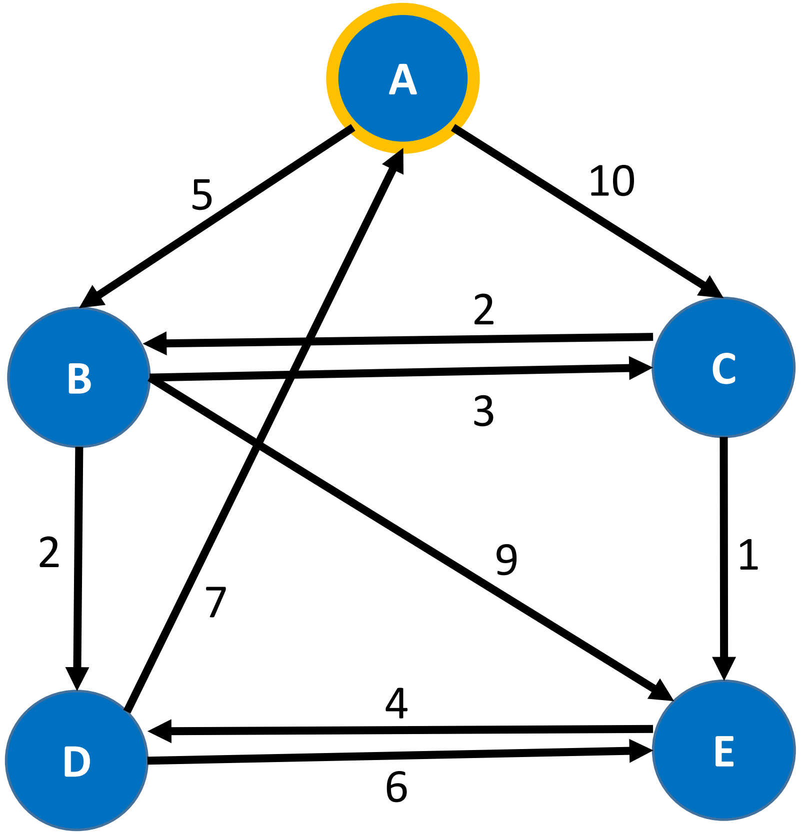
unesp



Vértices	A	B	C	D	E
Dis	0	∞	∞	∞	∞
Pai	null	null	null	null	Null
Q					
S					

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
             u = ExtrairMinimo(Q);
             S += u;
             para cada v ϵ Adj(u)
                Relaxamento(u,v,peso);
             fim para
   fim enquanto
fim
```

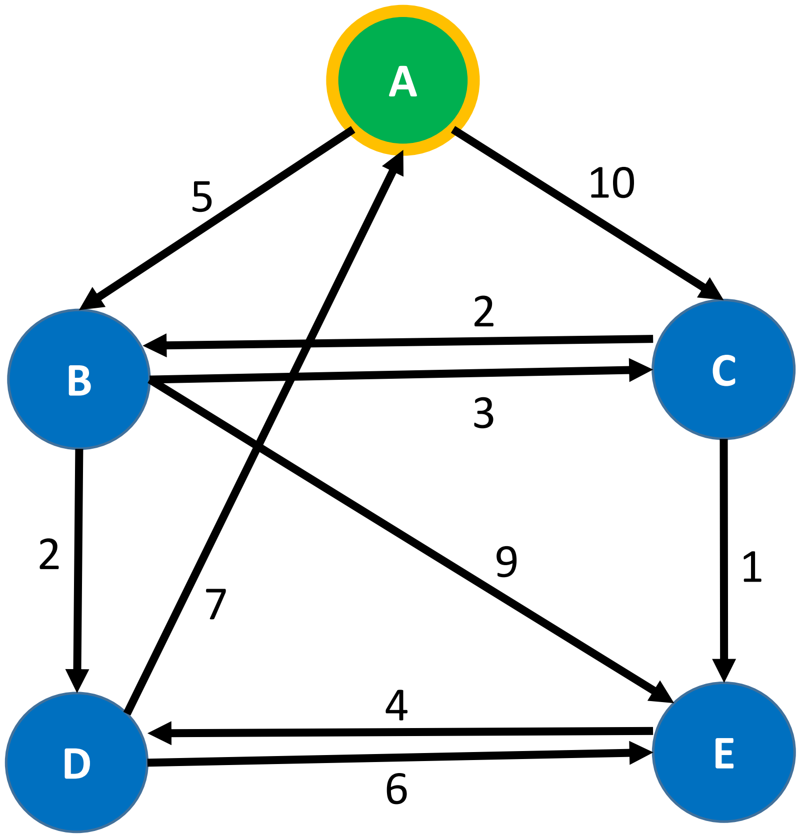
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	∞	∞	∞	∞
Pai	null	null	null	null	Null
Q	X	X	X	X	X
S	-	-	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

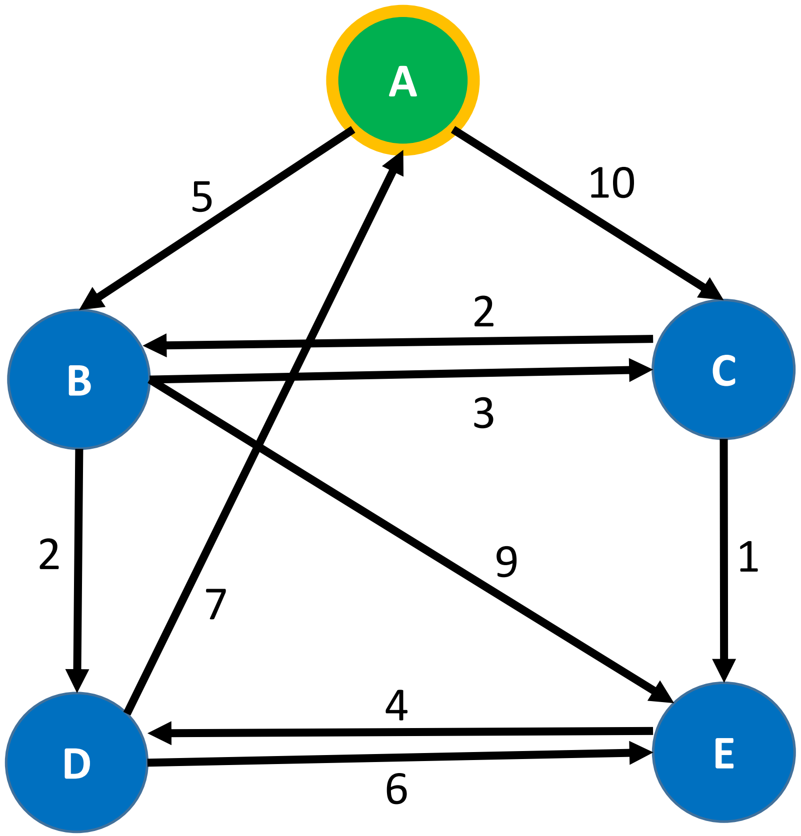
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	∞	∞	∞	∞
Pai	null	null	null	null	Null
Q	X	X	X	X	X
S	-	-	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

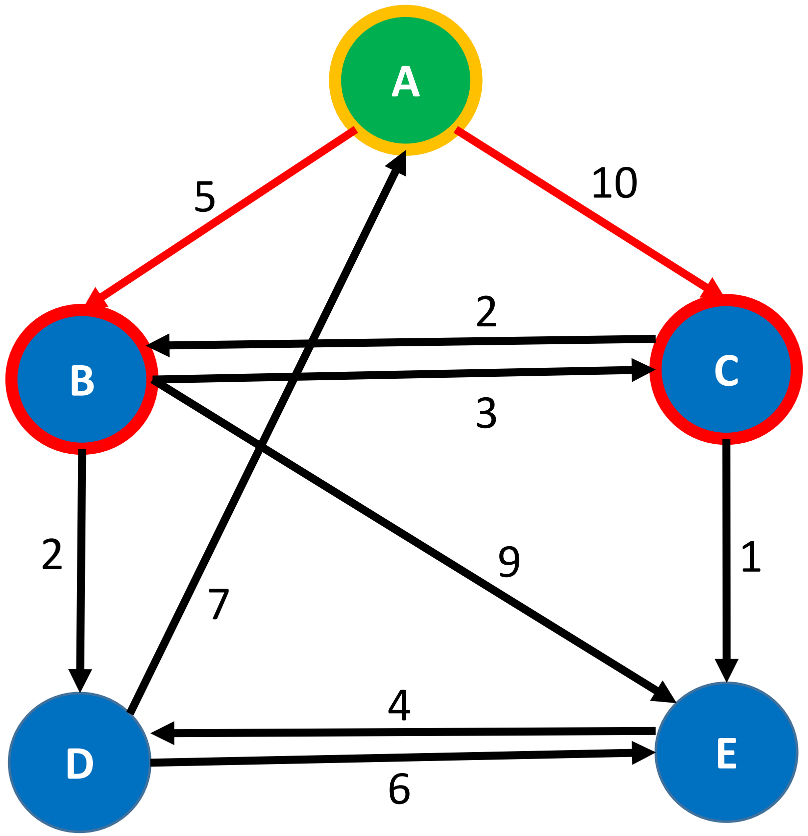


Vértices	A	B	C	D	E
Dis	0	∞	∞	∞	∞
Pai	null	null	null	null	Null
Q	-	X	X	X	X
S	X	-	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
          u = ExtrairMinimo(Q);
          S += u;
          para cada v ϵ Adj(u)
             Relaxamento(u,v,peso);
          fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

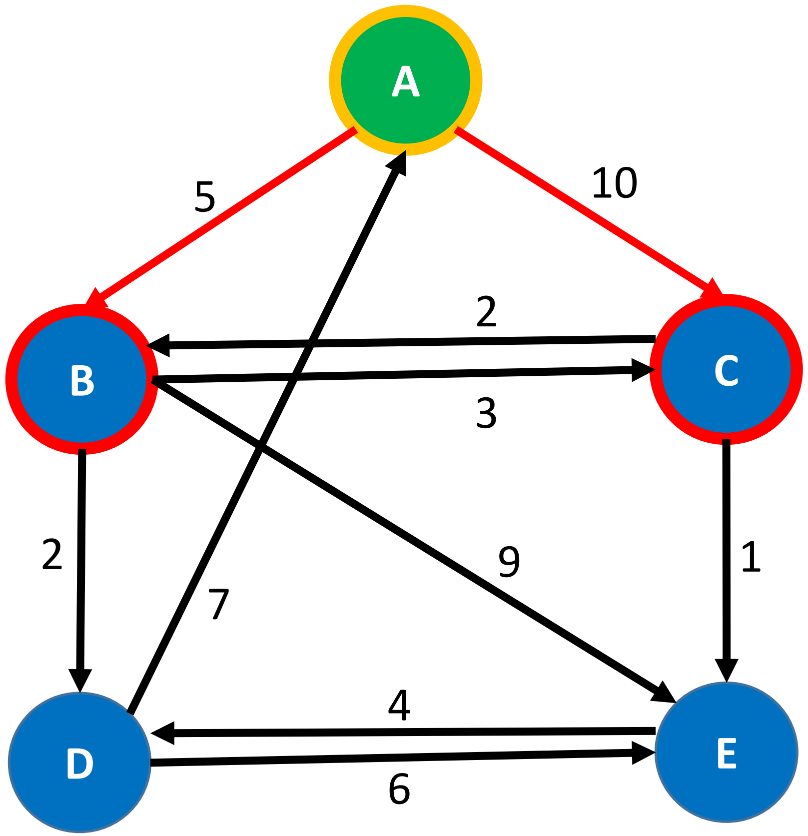
unesp



Vértices	A	B	C	D	E
Dis	0	∞	∞	∞	∞
Pai	null	null	null	null	Null
Q	-	X	X	X	X
S	X	-	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

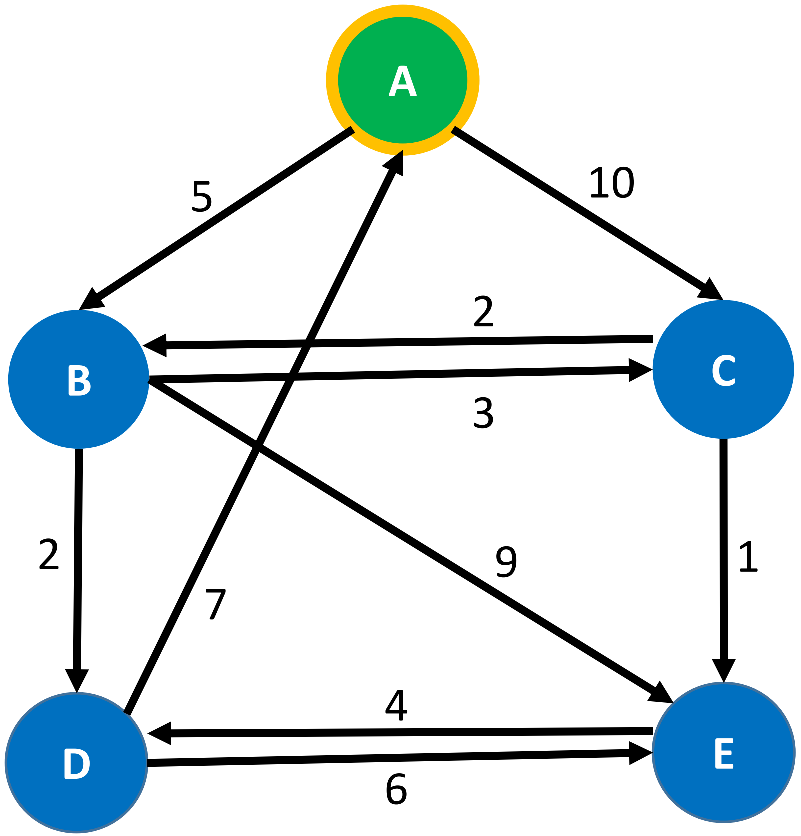


Vértices	A	B	C	D	E
Dis	0	5	10	∞	∞
Pai	null	A	A	null	Null
Q	-	X	X	X	X
S	X	-	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∊ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

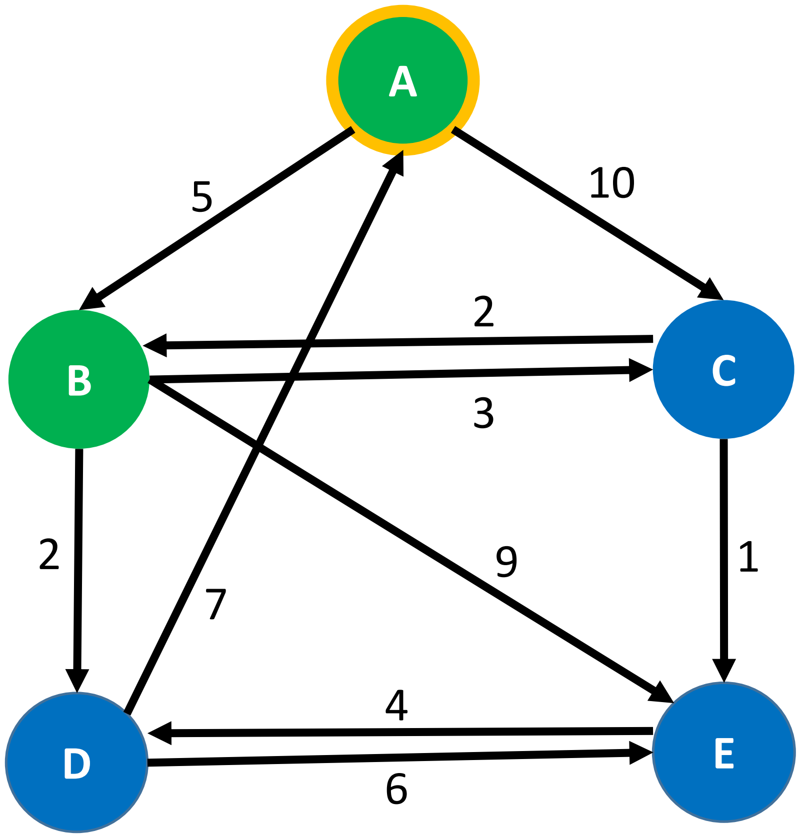
unesp



Vértices	A	B	C	D	E
Dis	0	5	10	∞	∞
Pai	null	A	A	null	Null
Q	-	X	X	X	X
S	X	-	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
           u = ExtrairMinimo(Q);
           S += u;
           para cada v ϵ Adj(u)
              Relaxamento(u,v,peso);
           fim para
   fim enquanto
fim
```

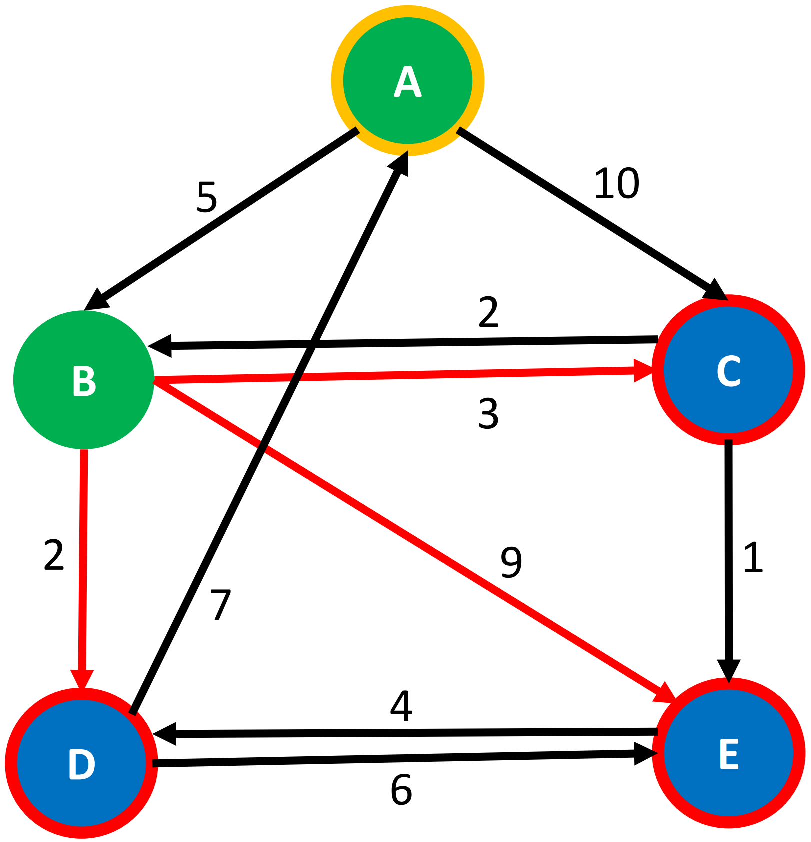
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	10	∞	∞
Pai	null	A	A	null	Null
Q	-	-	X	X	X
S	X	X	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

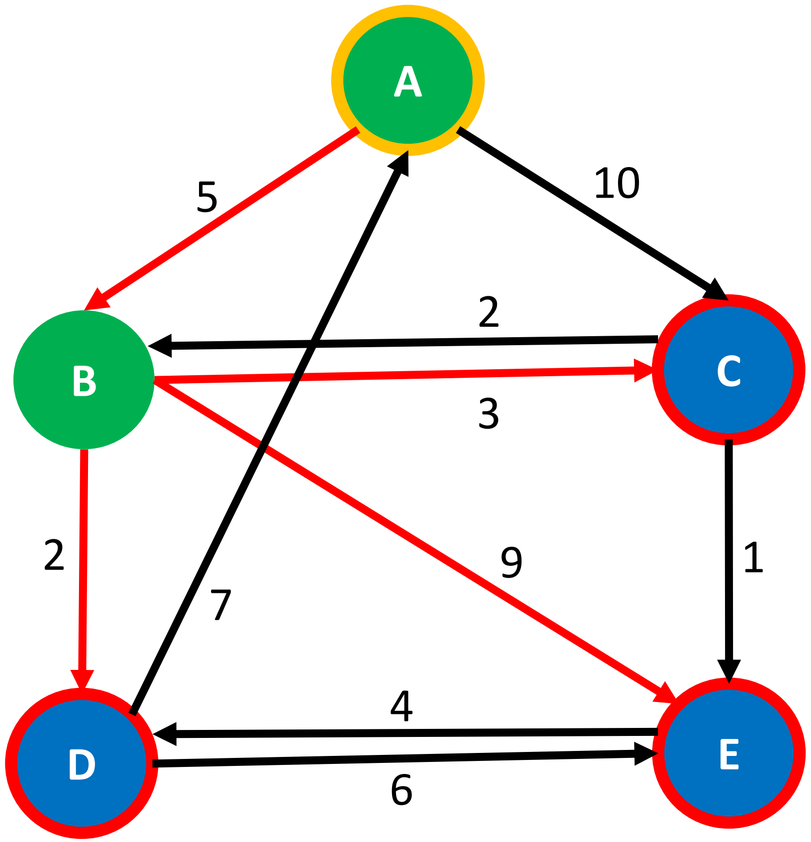


Vértices	A	B	C	D	E
Dis	0	5	10	∞	∞
Pai	null	A	A	null	Null
Q	-	-	X	X	X
S	X	X	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

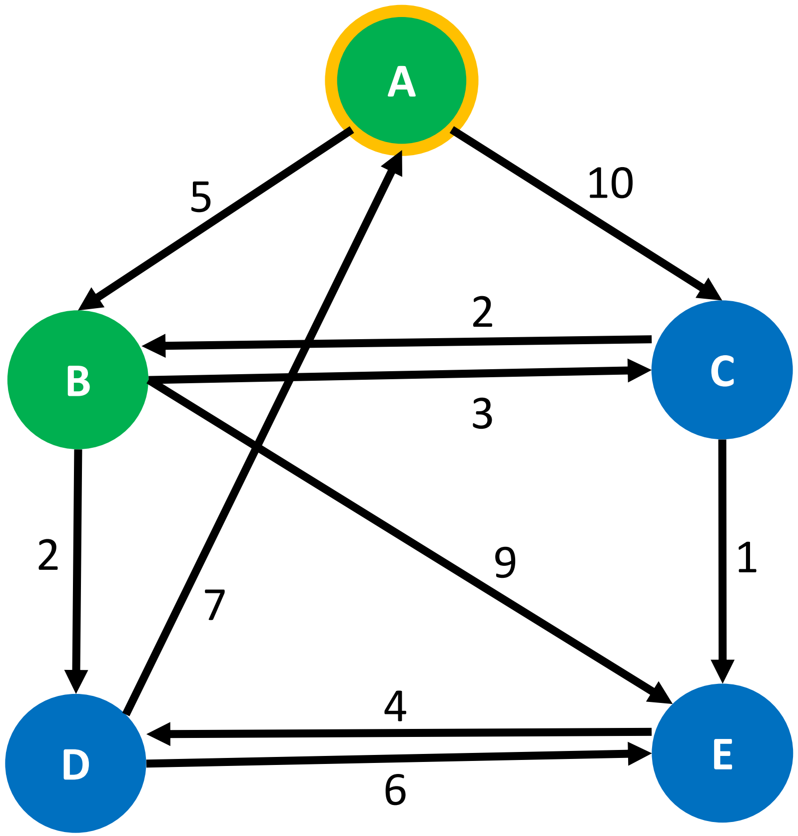
unesp



Vértices	A	B	C	D	E
Dis	0	5	8	7	14
Pai	null	A	B	B	B
Q	-	-	X	X	X
S	X	X	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v є Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

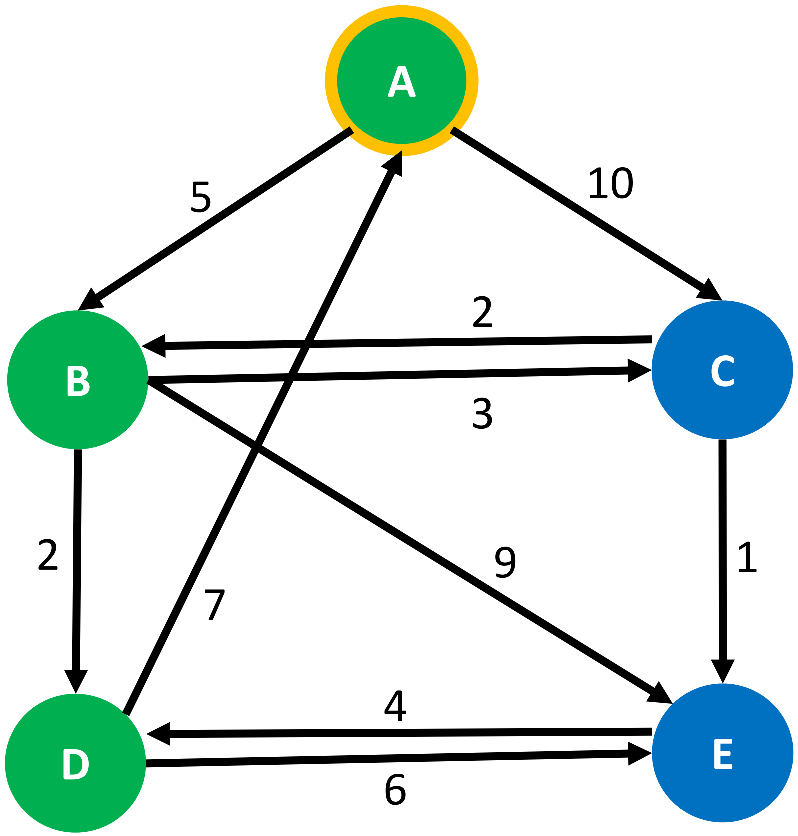
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	14
Pai	null	A	B	B	B
Q	-	-	X	X	X
S	X	X	-	-	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∈ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

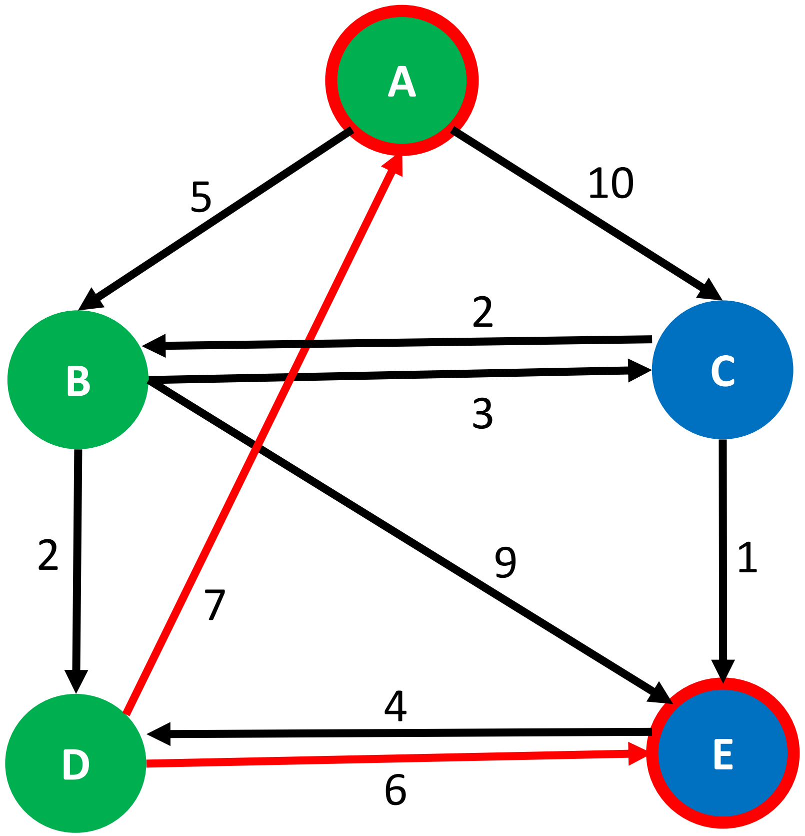
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	14
Pai	null	A	B	B	B
Q	-	-	X	-	X
S	X	X	-	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

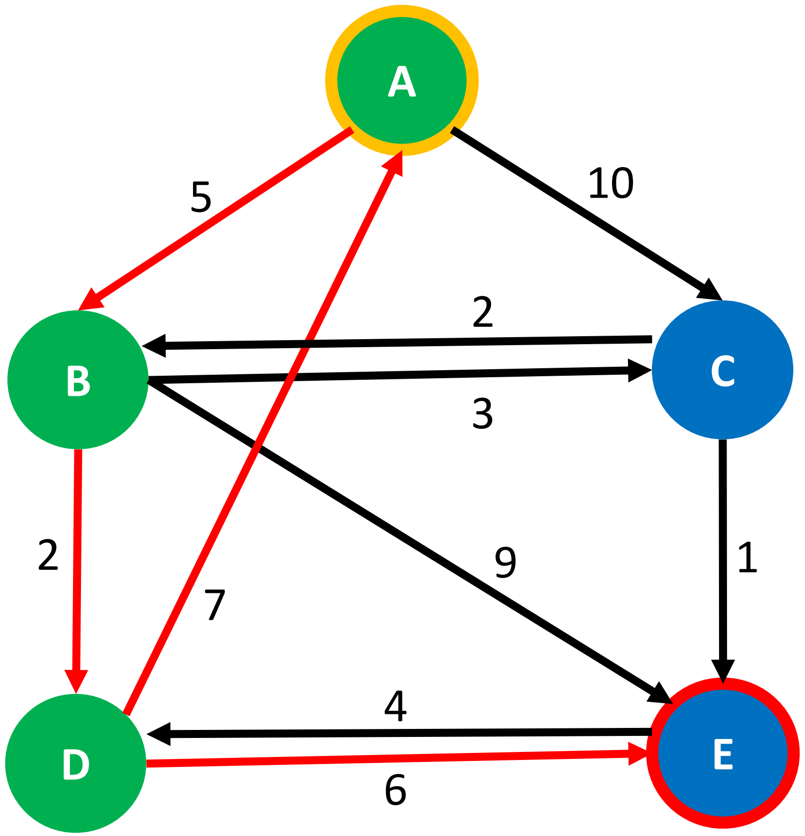


Vértices	A	B	C	D	E
Dis	0	5	8	7	14
Pai	null	A	B	B	B
Q	-	-	X	-	X
S	X	X	-	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∊ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

unesp

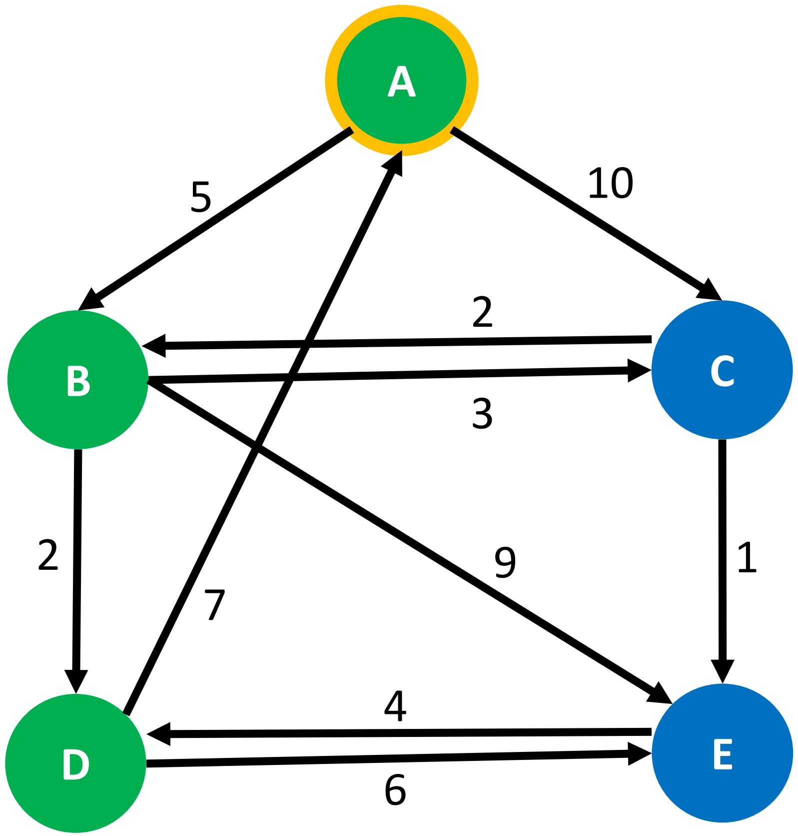


Vértices	A	B	C	D	E
Dis	0	5	8	7	13
Pai	null	A	B	B	D
Q	-	-	X	-	X
S	X	X	-	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∊ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

unesp

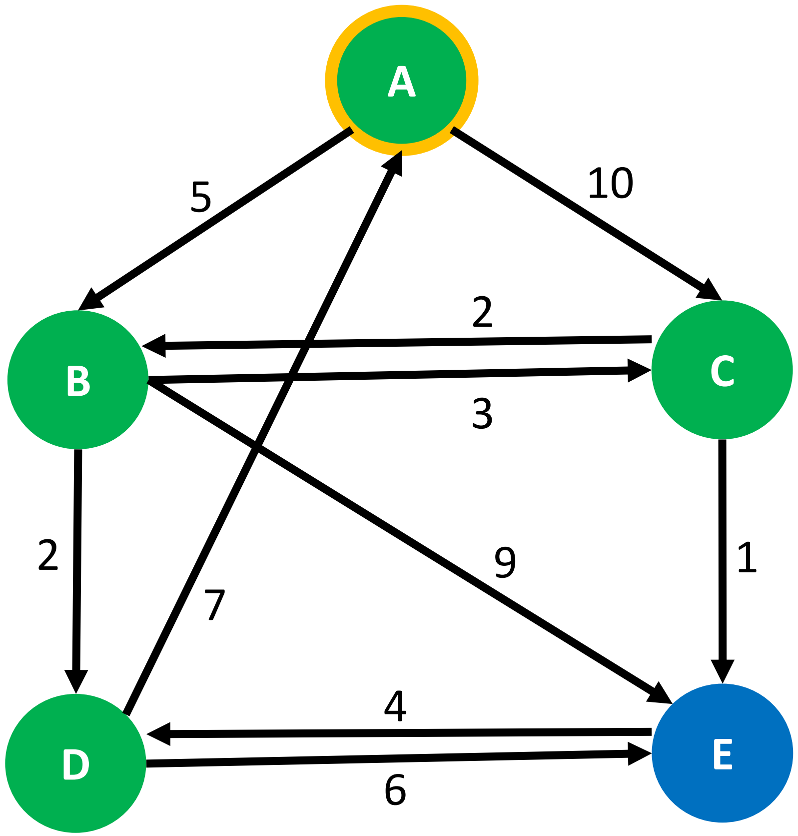


Vértices	A	B	C	D	E
Dis	0	5	8	7	13
Pai	null	A	B	B	D
Q	-	-	X	-	X
S	X	X	-	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra

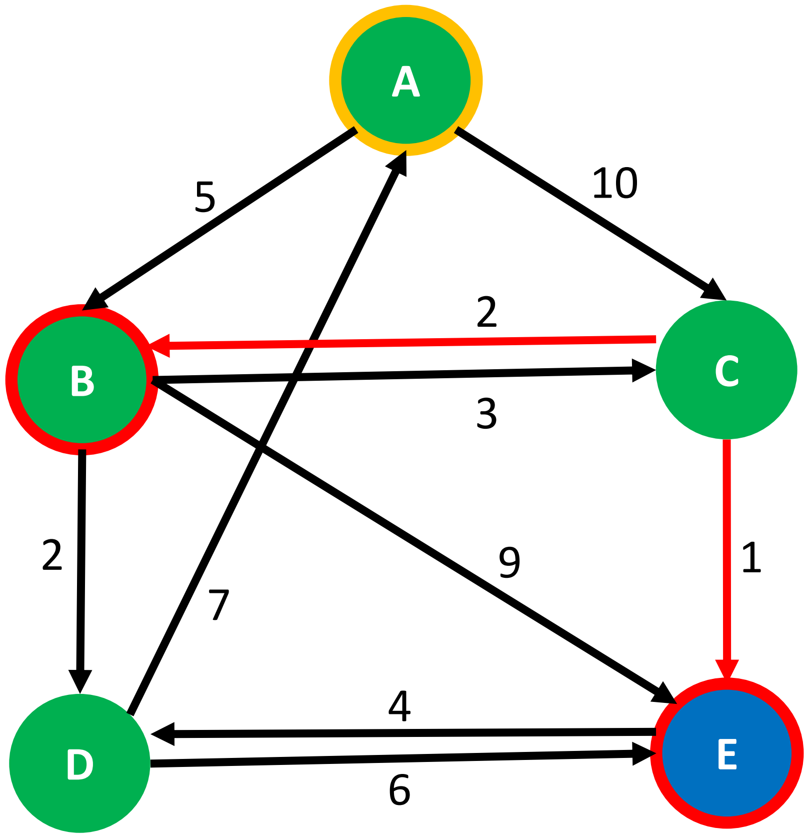
unesp



Vértices	A	B	C	D	E
Dis	0	5	8	7	13
Pai	null	A	B	B	D
Q	-	-	-	-	X
S	X	X	X	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∈ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

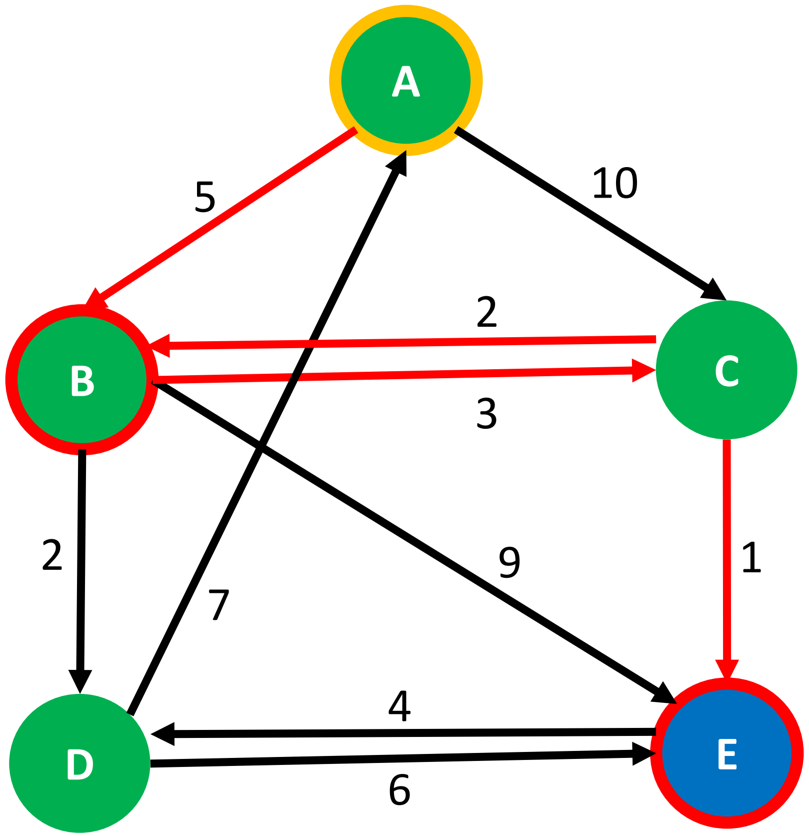
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	13
Pai	null	A	B	B	D
Q	-	-	-	-	X
S	X	X	X	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

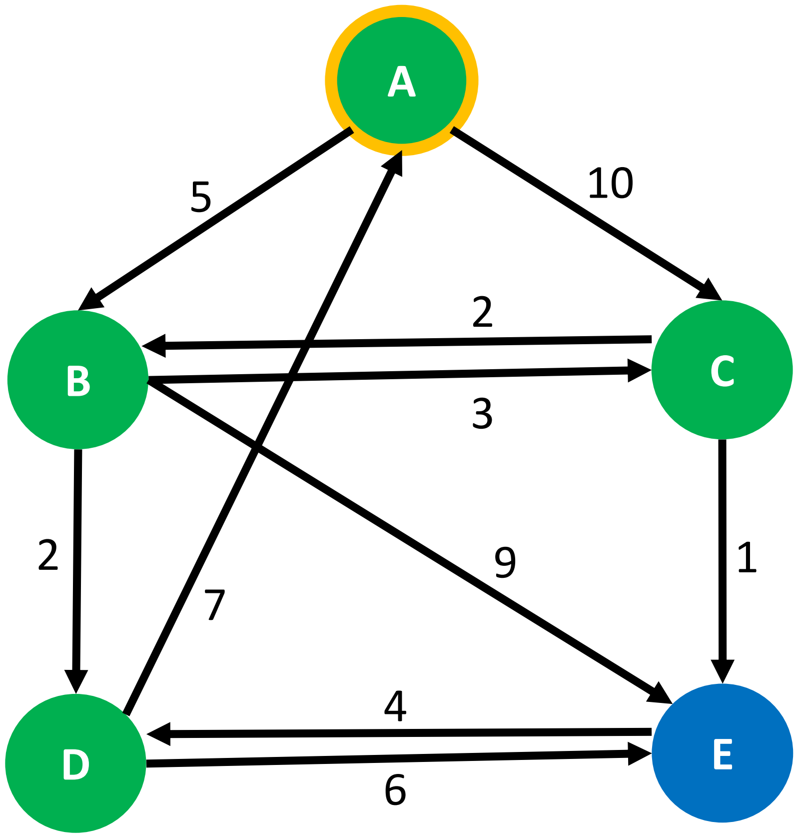
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C
Q	-	-	-	-	X
S	X	X	X	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∈ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

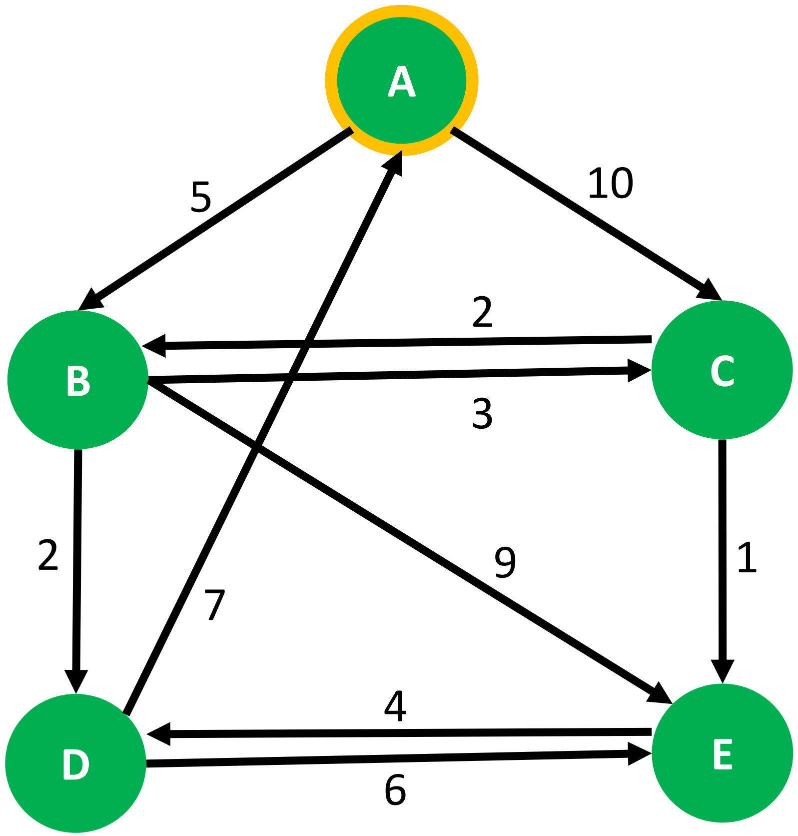
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C
Q	-	-	-	-	X
S	X	X	X	X	-

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
        u = ExtrairMinimo(Q);
        S += u;
        para cada v ∈ Adj(u)
           Relaxamento(u,v,peso);
        fim para
   fim enquanto
fim
```

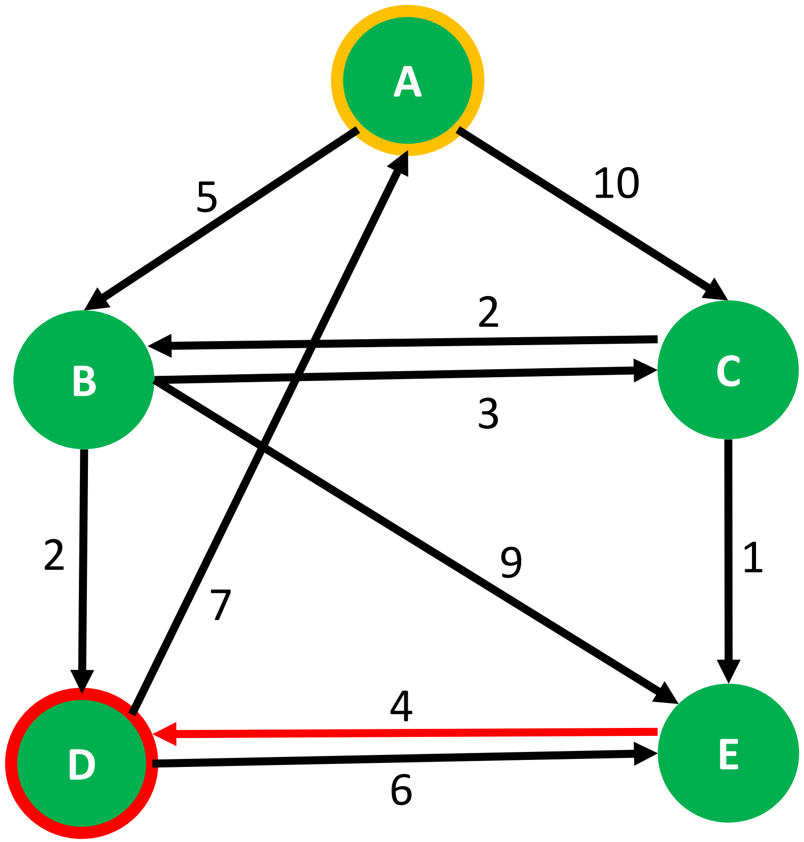
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C
Q	-	-	-	-	-
S	X	X	X	X	X

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ϵ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

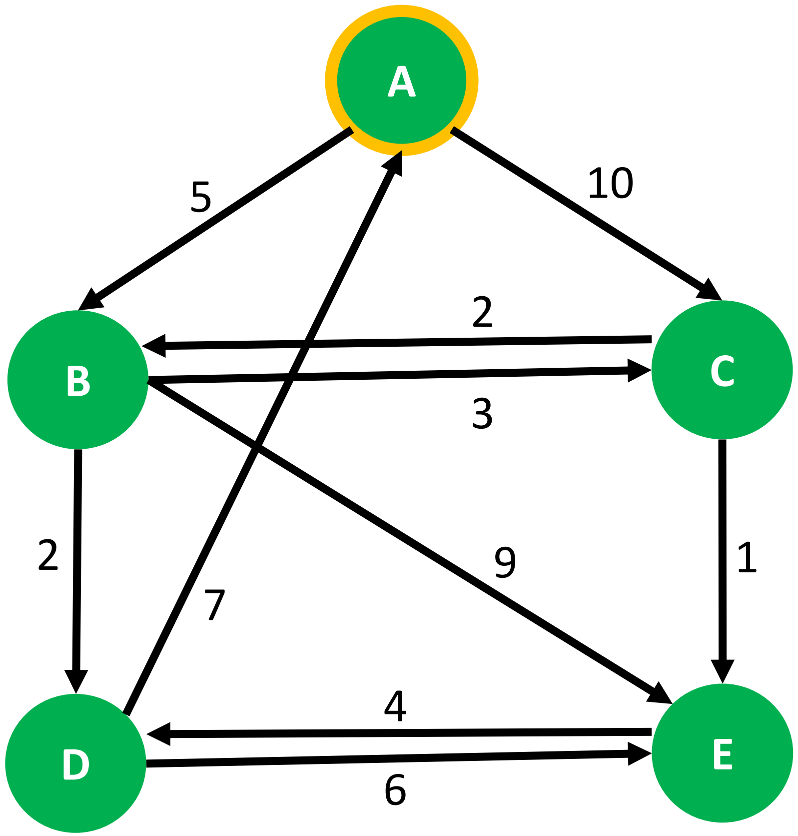
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C
Q	-	-	-	-	-
S	X	X	X	X	X

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∊ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

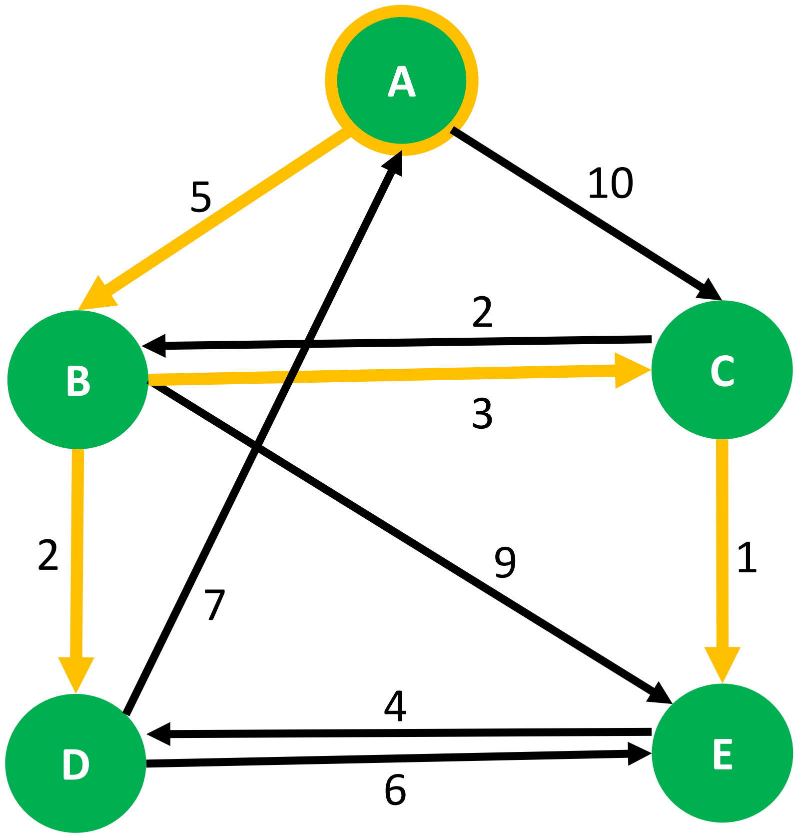
Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C
Q	-	-	-	-	-
S	X	X	X	X	X

```
Dijkstra(grafo,peso,verticeInicial)
  Inicializar(grafo,verticeInicial);
  S = {};
  Q = grafo->getVertices();
  Enquanto Q != vazio
            u = ExtrairMinimo(Q);
            S += u;
            para cada v ∊ Adj(u)
               Relaxamento(u,v,peso);
            fim para
   fim enquanto
fim
```

Algoritmo de Dijkstra



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C

Caminho A para B?

A → B

Custo: 5

Caminho A para C?

A → B → C

Custo: 8

Caminho A para D?

A → B → D

Custo: 7

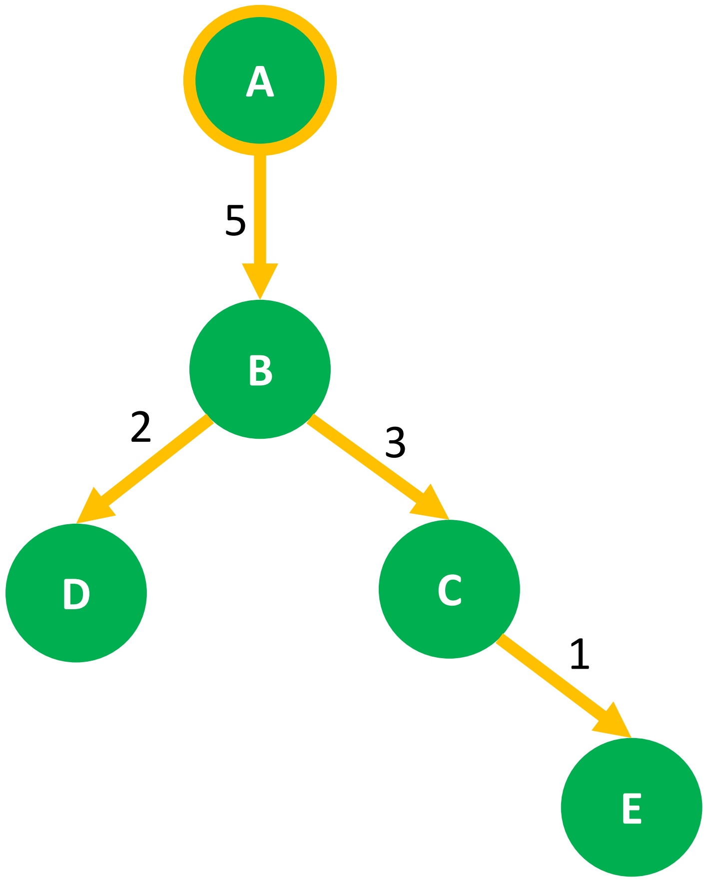
Caminho A para E?

A → B → C → E

Custo: 9

Algoritmo de Dijkstra

unesp



Vértices	A	B	C	D	E
Dis	0	5	8	7	9
Pai	null	A	B	B	C

Caminho A para B?

A → B

Custo: 5

Caminho A para C?

A → B → C

Custo: 8

Caminho A para D?

A → B → D

Custo: 7

Caminho A para E?

A → B → C → E

Custo: 9

Algoritmo de Dijkstra

- Complexidade do algoritmo:
  - <u>Depende de como é implementado a função de extrair o mínimo!!!</u>

    - <u>Abordagem 1:</u>
      - Busca sequencial
      - Custo quadrático: Número de Vértices X busca sequencial: $n^2 = O(v^2)$.

    - <u>Abordagem 2:</u>
      - Estrutura de Heap – Fila de prioridade utilizando Heap.
      - Custo logaritmo: Número de vértices X complexidade Heap = $n * \log n = O(v \log v)$

Aplicações do algoritmo

Aplicações

- Problemas de rotas rodoviárias

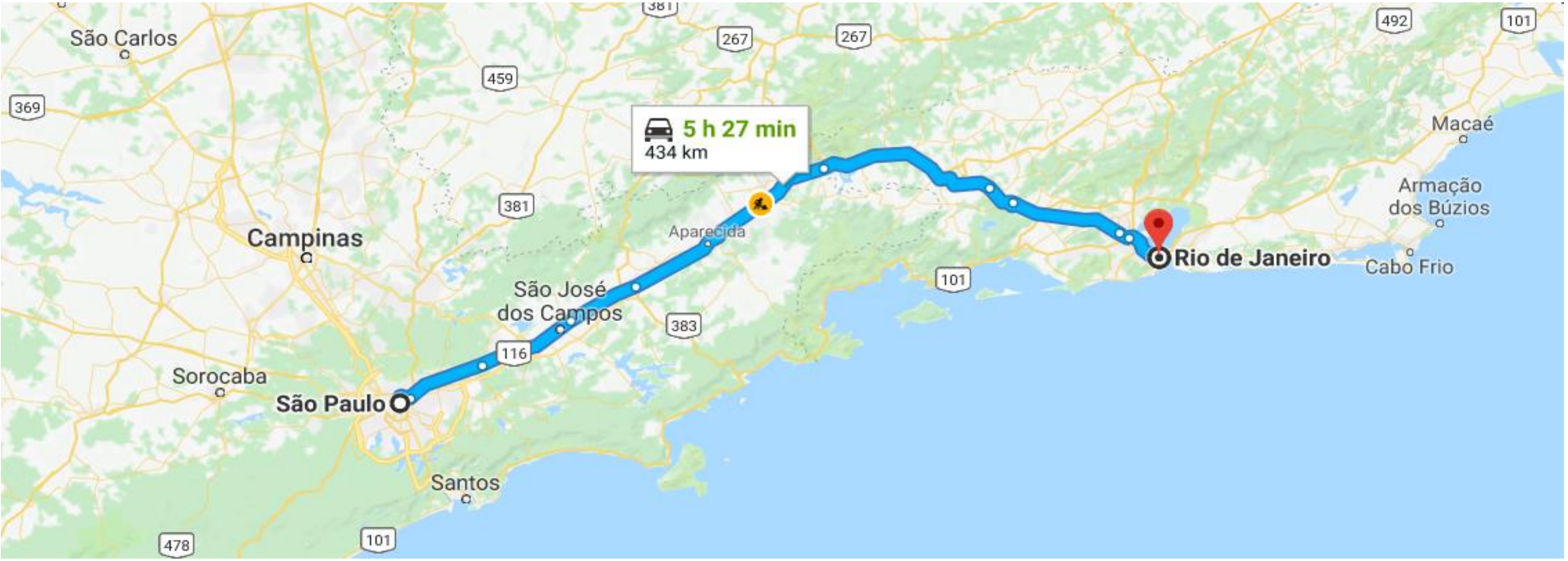


Figura 4: Melhor caminho de custo mínimo entre cidades – Fonte: Google Maps

Aplicações

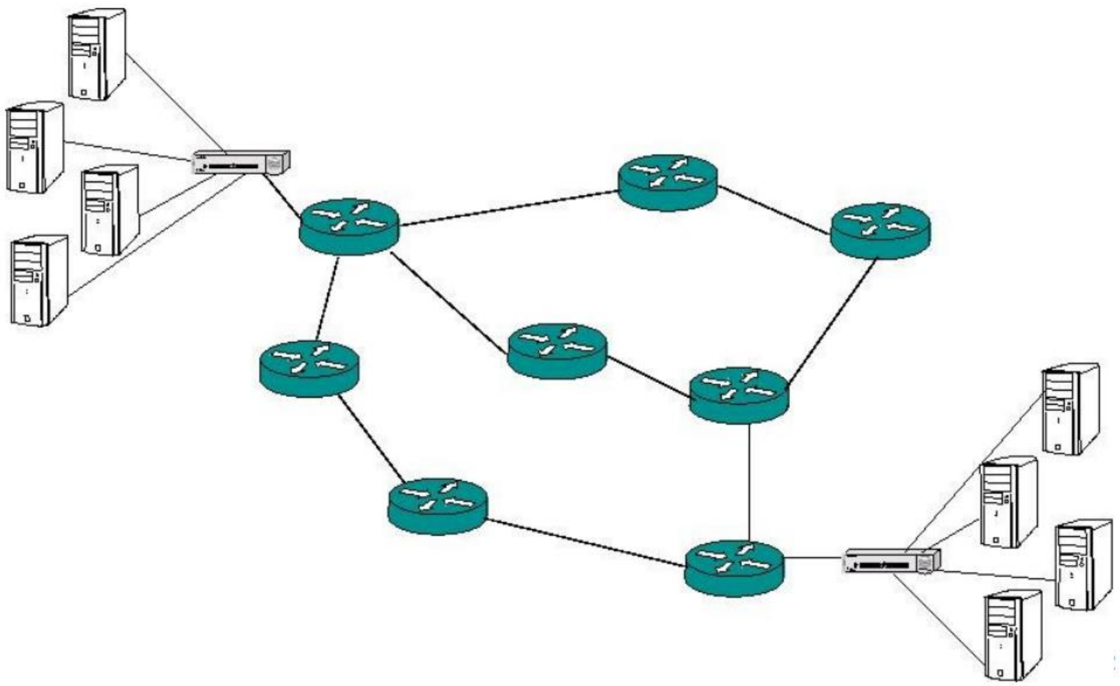


Figura 5: Rotas de roteamento entre redes de computadores

- Redes de computadores
  - Utilizado o algoritmo de Dijkstra para calcular rotas de enlace, a topologia da rede e todos os custos dos enlaces são conhecidos e fornecidos como entrada [5].

Aplicações

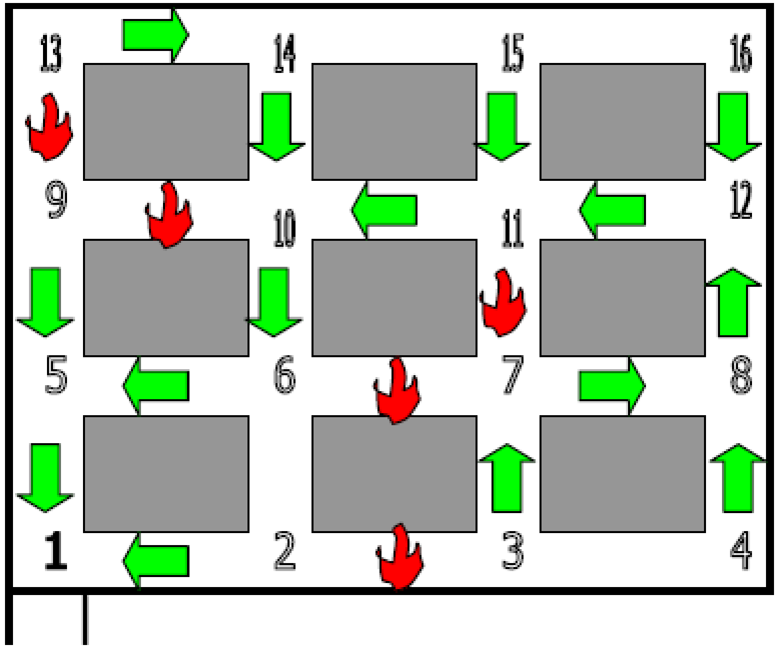


Figura 6: Rota de fuga da cena de incêndio

- Sistema inteligente de evacuação em caso de incêndio.
  - Utilização do algoritmo de Dijkstra para calcular rotas para chegar até a saída em caso de incêndio [8].

Considerações finais

- O Algoritmo de Dijkstra é relativamente simples e poderoso.
  - Algoritmo de abordagem gulosa, com solução sempre ótima.
  - Complexidade pode ser $O(n \log n)$, dependendo da função de extrair mínimo.
    - Aceleração do algoritmo de Dijkstra pode ser realizada com processamento paralelo [7].
  - Não aplicado em problemas que necessitam de pesos negativos.
    - Para este tipo de problema é recomendado o algoritmo de Bellman-Ford.
  - Se o problema utilizar somente grafos acíclicos, melhor algoritmo é o de ordenação topológica

Considerações finais

- Implementações do algoritmo de Dijkstra podem ser acessadas em:
  - Repositório no GitHub:
    - https://github.com/brunoslima/Dijkstra-Algorithm

Referências Bibliográficas

- [1] Dijkstra, E. W. “A Note on Two Problems in Connection with Graphs”, In: Numerische Mathematik, 1, p. 269–271, 1959.

- [2] ZIVIANI, Nivio. Projeto de algoritmos com implementações Pascal e C. 4. Ed. São Paulo : Pioneira, 1999.

- [3] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., & Stein, C. Algoritmos: teoria e prática. Editora Campus, 2 Ed, 2002.

- [4] SEDGEWICK, R, WAYNE, K. Algorithms. 4th Edition. Addison-Wesley, 2011.

Referências Bibliográficas

- [5] KUROSE, J. F. e ROSS, K. Redes de Computadores e a Internet - 5ª Ed., Pearson, 2010

- [6] RICHARDS, H. Manuscripts of Edsger W. Dijkstra, University Texas at Austin. Disponível em: <http://www.cs.utexas.edu/users/EWD/>.

- [7] A. Prasad, S. K. Krishnamurthy and Y. Kim, "Acceleration of Dijkstra's algorithm on multi-core processors," 2018 International Conference on Electronics, Information, and Communication (ICEIC), Honolulu, HI, 2018, pp. 1-5.

- [8] Y. Xu, Z. Wang, Q. Zheng and Z. Han, "The Application of Dijkstra's Algorithm in the Intelligent Fire Evacuation System," 2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics, Nanchang, Jiangxi, 2012, pp. 3-6.