Grafos

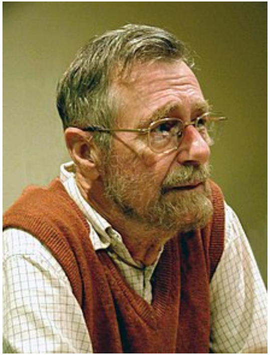
Algoritmo de Dijkstra

Prof. Edson Alves

Faculdade UnB Gama

Proponente

Proponente



Edsger Wybe Dijkstra
(1956)

Características do algoritmo de Dijkstra

Características do algoritmo de Dijkstra

$\star$ **Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s**

Características do algoritmo de Dijkstra

- $\star$ Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s
- $\star$ Processa corretamente apenas grafos com arestas não-negativas

Características do algoritmo de Dijkstra

- ⋆ Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s
- ⋆ Processa corretamente apenas grafos com arestas não-negativas
- ⋆ Eficiência: cada aresta é processada uma única vez

Características do algoritmo de Dijkstra

- $\star$ **Computa o caminho mínimo de todos os vértices de $G(V, E)$ a um dado nó s**
- $\star$ **Processa corretamente apenas grafos com arestas não-negativas**
- $\star$ **Eficiência: cada aresta é processada uma única vez**
- $\star$ **Complexidade:** $O(E + V \log V)$

Pseudocódigo

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

1. Faça $d[s] = 0$, $d[u] = \infty$ se $u \neq s$ e seja $U = V$

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

1. **Faça** $d[s] = 0$, $d[u] = \infty$ se $u \neq s$ e seja $U = V$

2. **Enquanto** $U \neq \emptyset$:
   (a) Seja $u \in U$ o vértice mais próximo de s em U
   (b) Relaxe as distâncias usando as arestas que partem de u
   (c) Remova u de U

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$ e um vértice $s \in V$
Saída: um vetor d tal que $d[u]$ é a distância mínima em G entre s e u

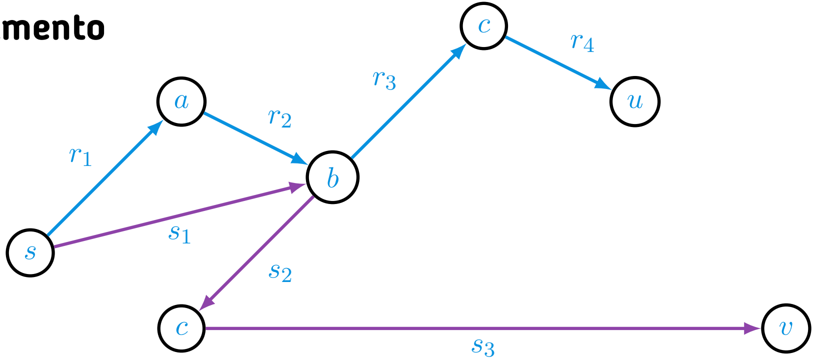
1. Faça $d[s] = 0$, $d[u] = \infty$ se $u \neq s$ e seja $U = V$

2. Enquanto $U \neq \emptyset$:
    (a) Seja $u \in U$ o vértice mais próximo de s em U
    (b) Relaxe as distâncias usando as arestas que partem de u
    (c) Remova u de U

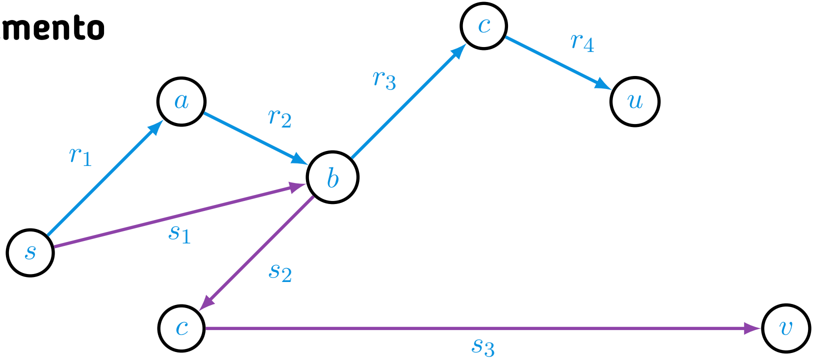
3. Retorne d

Relaxamento

Relaxamento

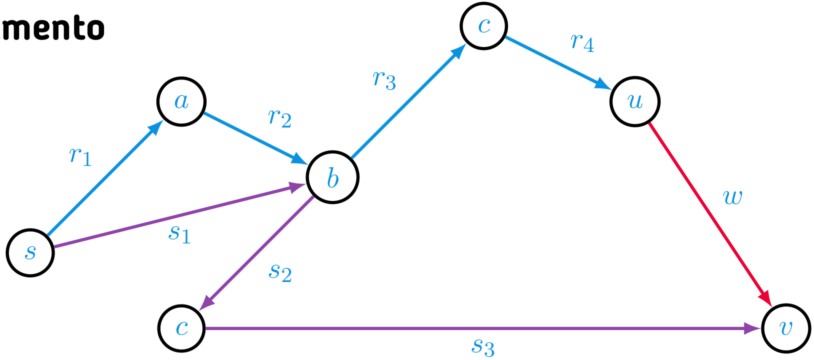


Relaxamento

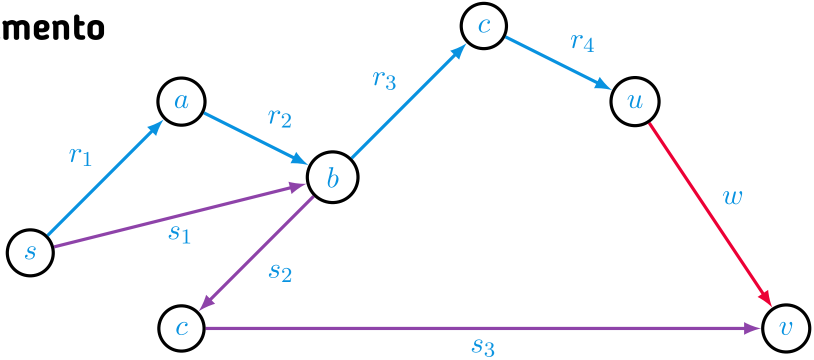


$$\text{dist}(s, u) = \sum_{i=1}^{4} r_i \qquad \text{dist}(s, v) = \sum_{j=1}^{3} s_i$$

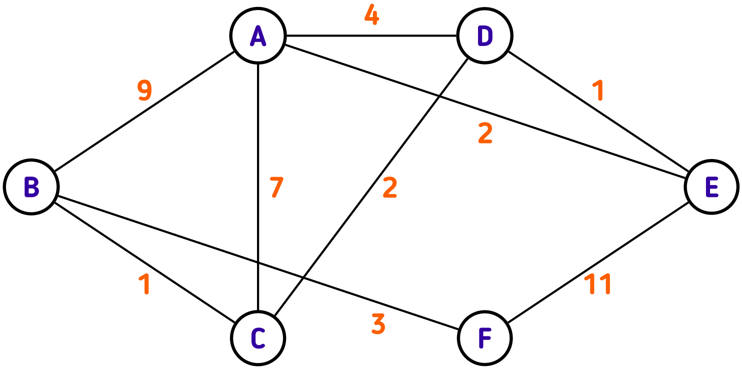
Relaxamento

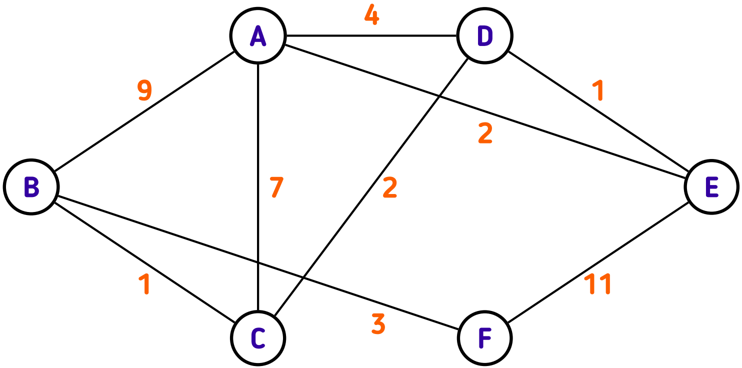


Relaxamento



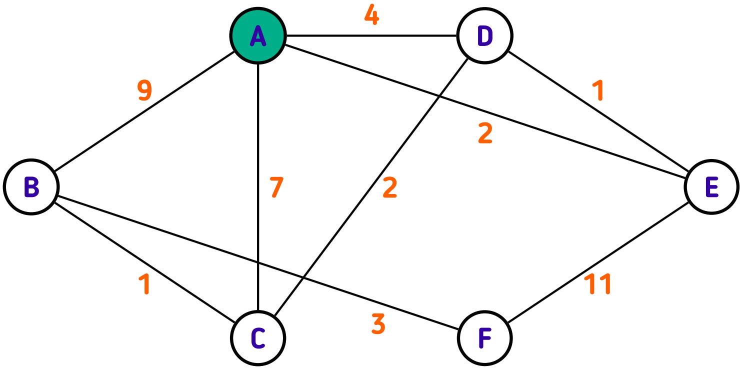
Se $\text{dist}(s, u) + w < \text{dist}(s, v)$**, faça** $\text{dist}(s, v) = \text{dist}(s, u) + w$


A
B
C
D
E
F
4
9
1
2
7
2
1
3
11



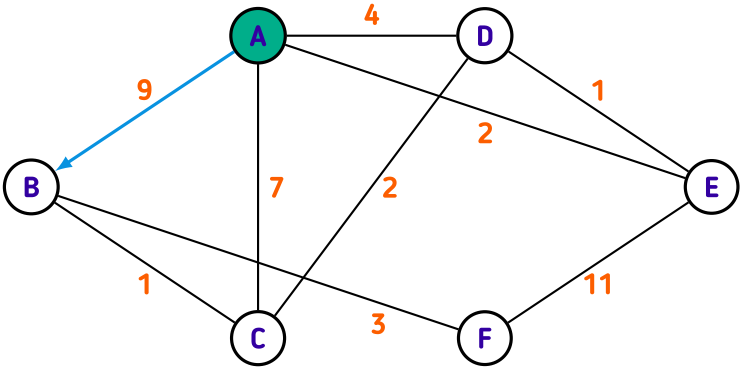
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	∞	∞	∞	∞	∞

$U = \{$ **A, B, C, D, E, F** $\}$



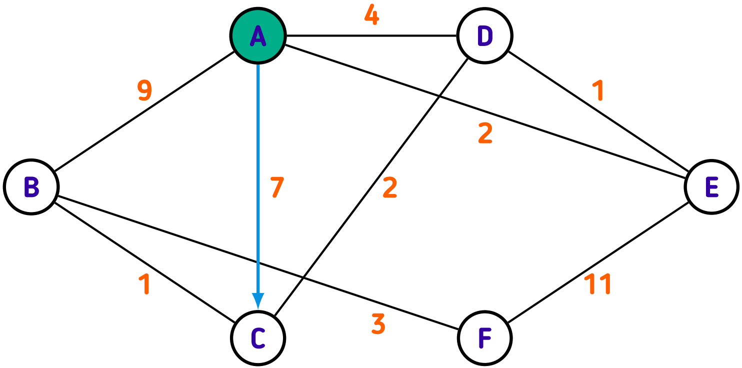
	A	B	C	D	E	F
dist$(u, \textbf{A})$	0	∞	∞	∞	∞	∞

$U = \{$ **B, C, D, E, F** $\}$



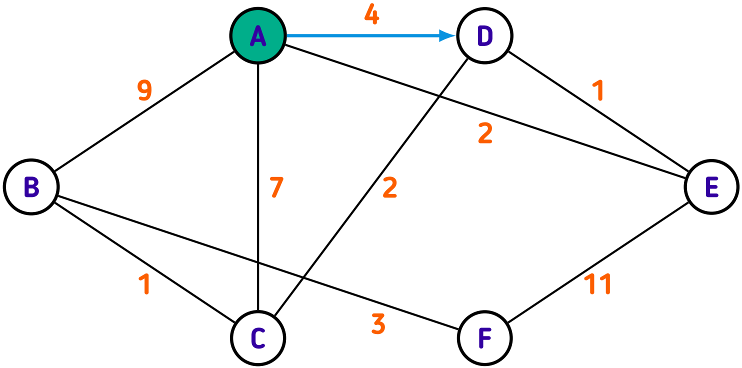
	A	B	C	D	E	F
dist(u, **A**)	0	**9**	∞	∞	∞	∞

$U = \{$ **B, C, D, E, F** $\}$



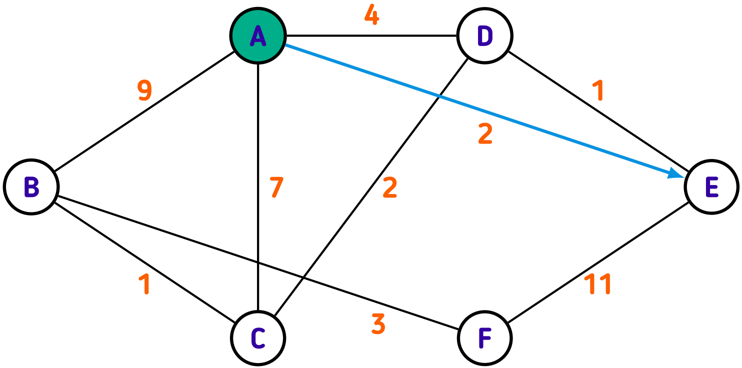
	A	B	C	D	E	F
dist(u, **A**)	0	9	**7**	∞	∞	∞

$U = \{$ **B, C, D, E, F** $\}$



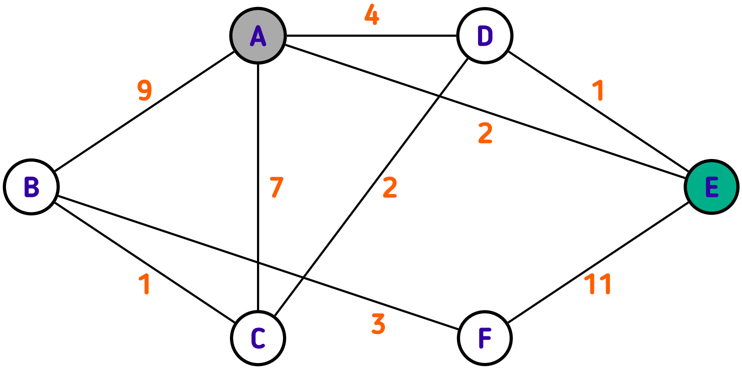
	A	B	C	D	E	F
$\mathsf{dist}(u, \mathbf{A})$	0	9	7	$\mathbf{4}$	∞	∞

$U = \{\ \mathbf{B, C, D, E, F}\ \}$



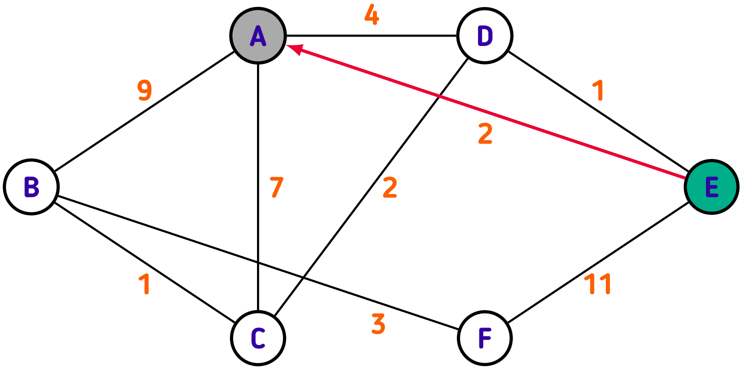
	A	B	C	D	E	F
dist(u, **A**)	0	9	7	4	**2**	∞

$U = \{$ **B, C, D, E, F** $\}$



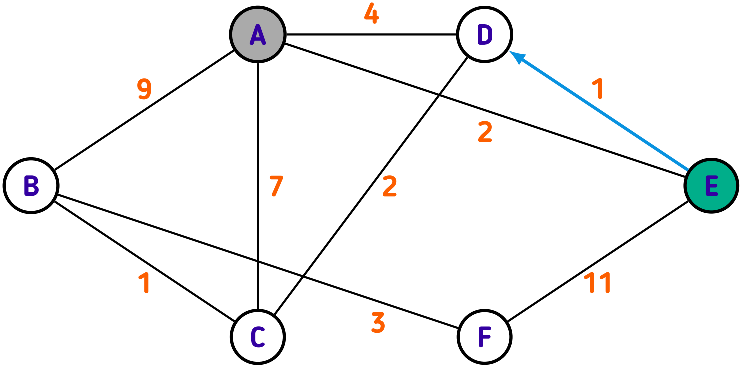
	A	B	C	D	E	F
dist$(u, \textbf{A})$	0	9	7	4	2	∞

$U = \{$ **B, C, D, F** $\}$



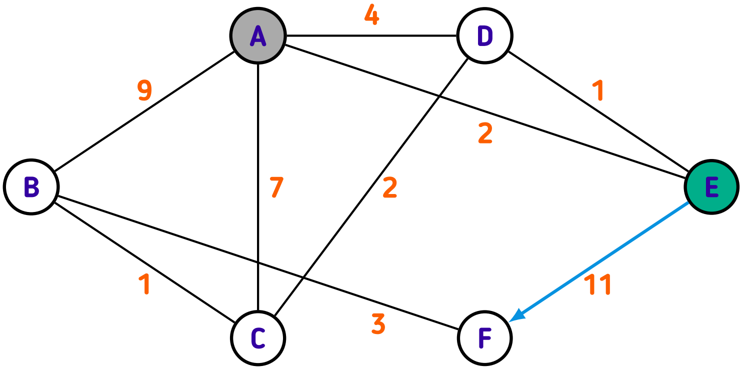
	A	B	C	D	E	F
$\text{dist}(u, \textbf{A})$	0	9	7	4	2	∞

$U = \{\ \textbf{B, C, D, F}\ \}$



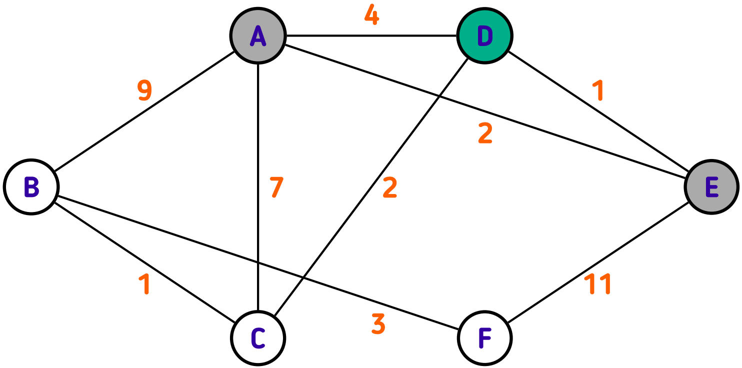
	A	B	C	D	E	F
dist$(u, \textbf{A})$	0	9	7	**3**	2	∞

$U = \{$ **B, C, D, F** $\}$



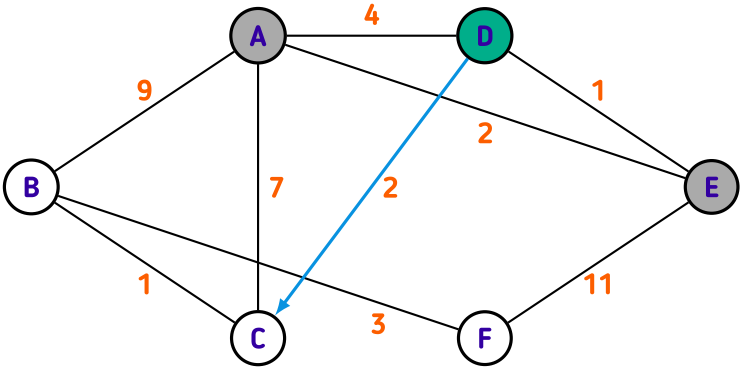
	A	B	C	D	E	F
dist(u, **A**)	0	9	7	3	2	**13**

$U = \{$ **B, C, D, F** $\}$



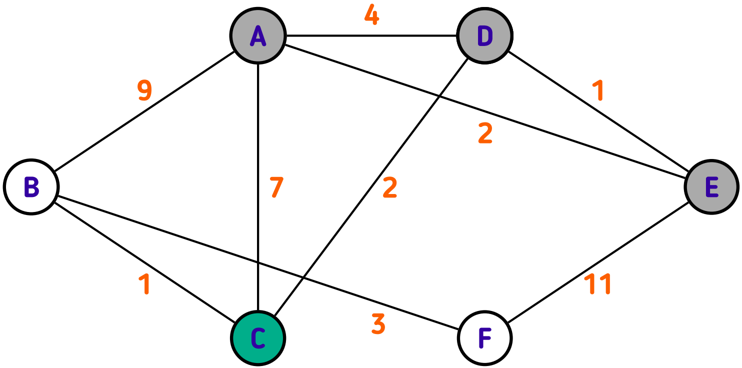
	A	B	C	D	E	F
$\text{dist}(u, \textbf{A})$	0	9	7	3	2	13

$U = \{$ **B, C, F** $\}$



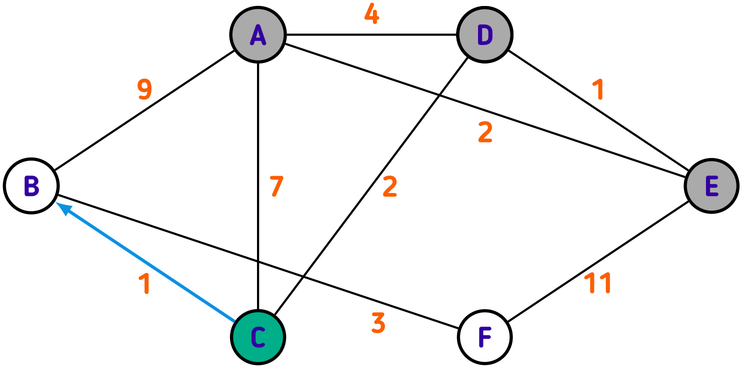
	A	B	C	D	E	F
dist(u, **A**)	0	9	**5**	3	2	13

$U = \{$ **B, C, F** $\}$



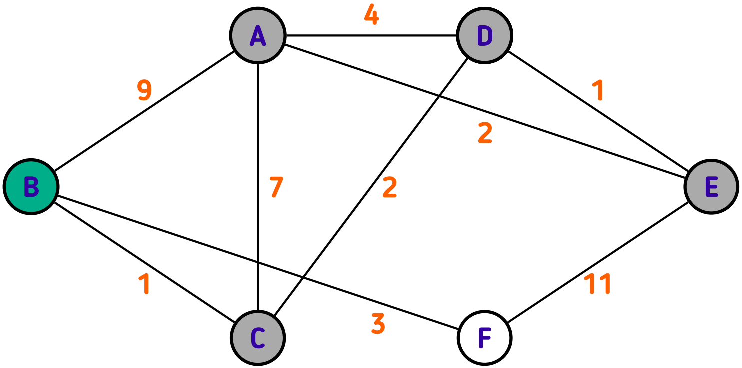
	A	B	C	D	E	F
dist$(u,$ **A**$)$	0	9	5	3	2	13

$U = \{$ **B, F** $\}$



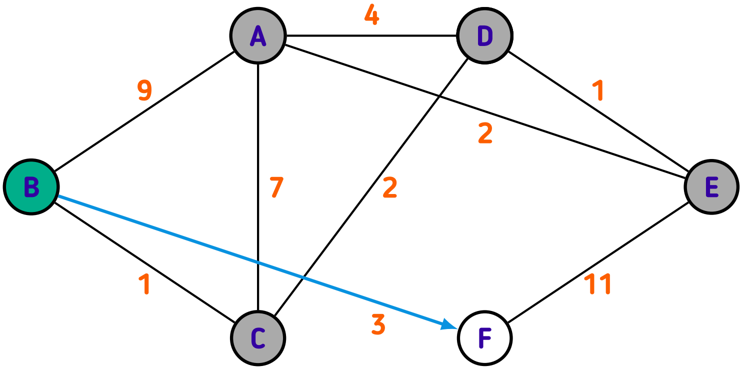
	A	B	C	D	E	F
dist(u, **A**)	0	**6**	5	3	2	13

$U = \{$ **B, F** $\}$



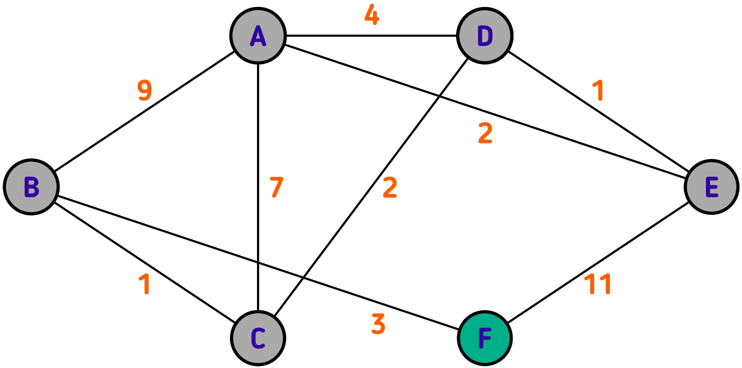
	A	B	C	D	E	F
dist$(u, \mathbf{A})$	0	6	5	3	2	13

$U = \{ \mathbf{F} \}$



	A	B	C	D	E	F
dist(u, **A**)	0	6	5	3	2	**9**

$U = \{ \textbf{F} \}$

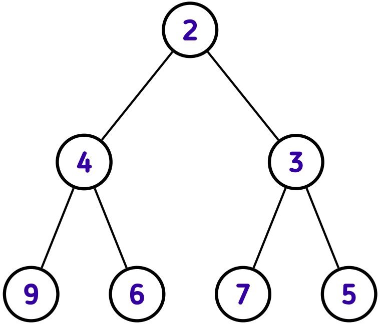


	A	B	C	D	E	F
dist$(u, \textbf{A})$	0	6	5	3	2	9

$U = \emptyset$

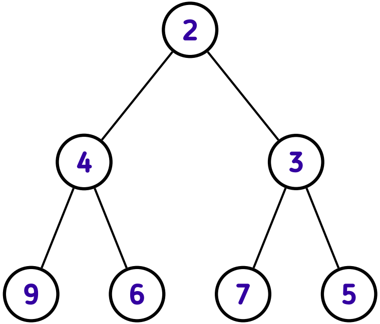
Identificação eficiente do vértice u mais próximo de s

Identificação eficiente do vértice u mais próximo de s



min heap

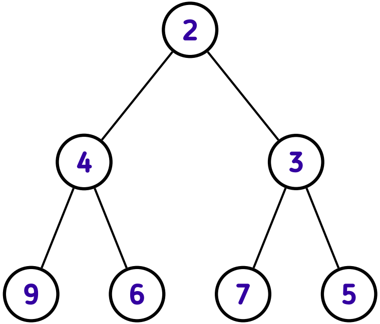
Identificação eficiente do vértice u mais próximo de s



min heap

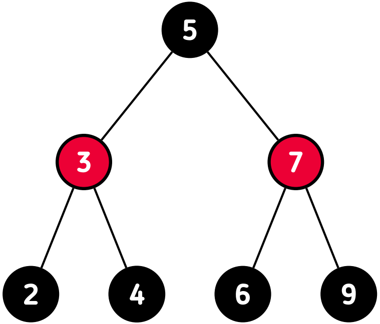
(`priority_queue`)

Identificação eficiente do vértice u mais próximo de s



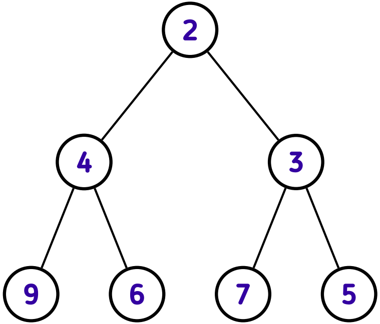
min heap

(`priority_queue`)

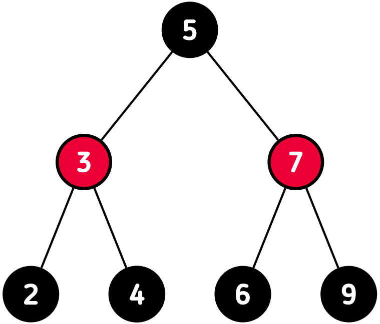


red-black tree

Identificação eficiente do vértice u mais próximo de s



min heap
(`priority_queue`)



red-black tree
(`set`)

```cpp
vector<int> dijkstra(int s, int N)
{
    const int oo { 1000000010 };

    vector<int> dist(N + 1, oo);
    dist[s] = 0;

    set<ii> U;
    U.emplace(0, s);

    while (not U.empty())
    {
        auto [d, u] = *U.begin();
        U.erase(U.begin());
```

```
        for (auto [v, w] : adj[u])
        {
            if (dist[v] > d + w)
            {
                if (U.count(ii(dist[v], v)))
                    U.erase(ii(dist[v], v));

                dist[v] = d + w;
                U.emplace(dist[v], v);
            }
        }
    }

    return dist;
}
```

Identificação do caminho mínimo

Identificação do caminho mínimo

- O algoritmo de Dijkstra computa as distâncias mínimas, mas não os caminhos mínimos

Identificação do caminho mínimo

- $\star$ **O algoritmo de Dijkstra computa as distâncias mínimas, mas não os caminhos mínimos**

- $\star$ **Para determinar um caminho mínimo, é preciso definir o vetor auxiliar** pred, **onde** $\mathsf{pred}[u] =$ **antecessor de** u **no caminho mínimo de** s **a** u

Identificação do caminho mínimo

$\star$ **O algoritmo de Dijkstra computa as distâncias mínimas, mas não os caminhos mínimos**

$\star$ **Para determinar um caminho mínimo, é preciso definir o vetor auxiliar** pred**, onde** $\text{pred}[u] =$ **antecessor de** u **no caminho mínimo de** s **a** u

$\star$ **No início do algoritmo,** $\text{pred}[s] = s$ **e** $\text{pred}[u] = \text{undef}$**, se** $u \neq s$

Identificação do caminho mínimo

- $\star$ **Se** (u, v) **atualizar** $d[v]$**, faça** $\text{pred}[v] = u$

Identificação do caminho mínimo

- **Se** (u, v) **atualizar** $d[v]$**, faça** $\text{pred}[v] = u$

- **A sequência**

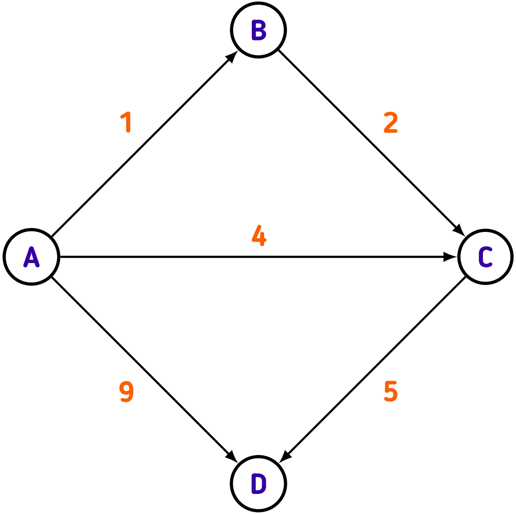
$$p = \{(s, \text{pred}^{k-1}[u]), \ldots, (\text{pred}[\text{pred}[u]], \text{pred}[u]), (\text{pred}[u], u)\}$$

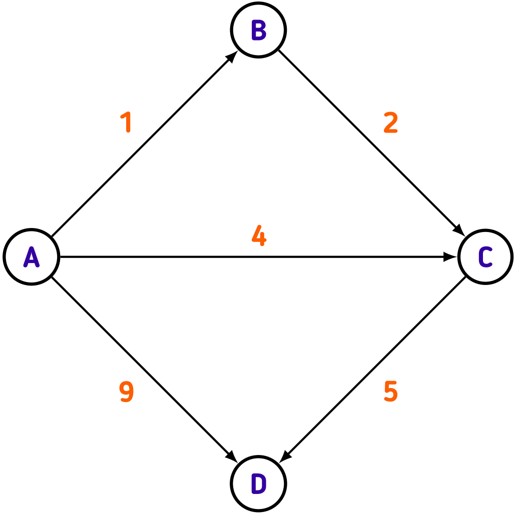
Identificação do caminho mínimo

- $\star$ **Se** (u, v) **atualizar** $d[v]$**, faça** $\mathsf{pred}[v] = u$
- $\star$ **A sequência**

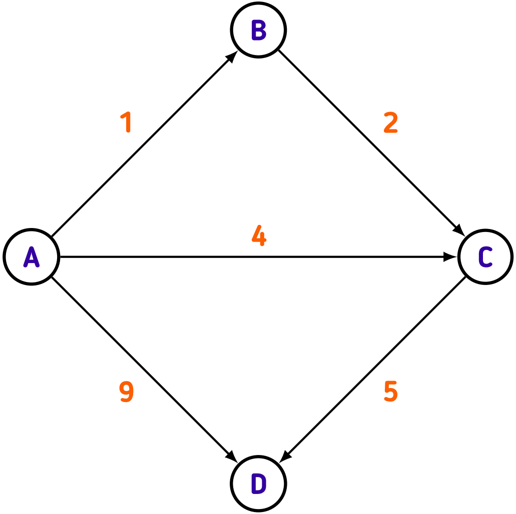
$$p = \{(s, \mathsf{pred}^{k-1}[u]), \dots, (\mathsf{pred}[\mathsf{pred}[u]], \mathsf{pred}[u]), (\mathsf{pred}[u], u)\}$$

é um caminho mínimo de s **a** u **composto de** k **arestas e tamanho** $d[u]$


B
1
2
4
A
C
9
5
D

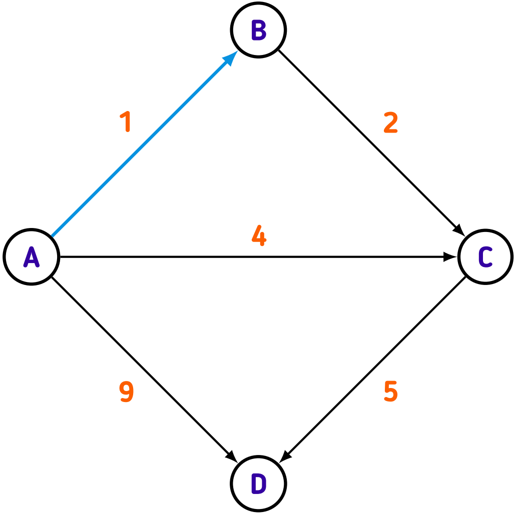


	A	B	C	D
dist$(u, \textbf{A})$	0	∞	∞	∞



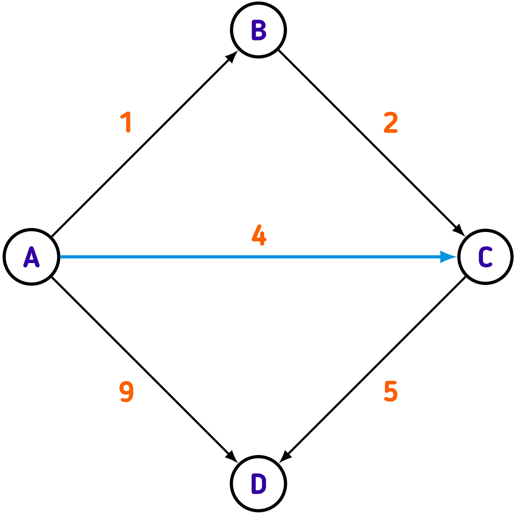
	A	B	C	D
dist$(u, \textbf{A})$	0	∞	∞	∞

	A	B	C	D
pred(u)	A	-	-	-



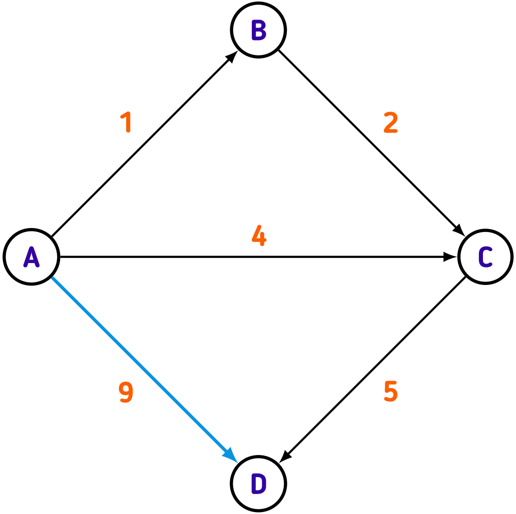
	A	B	C	D
dist$(u,$ **A**$)$	0	1	∞	∞

	A	B	C	D
pred(u)	A	A	-	-



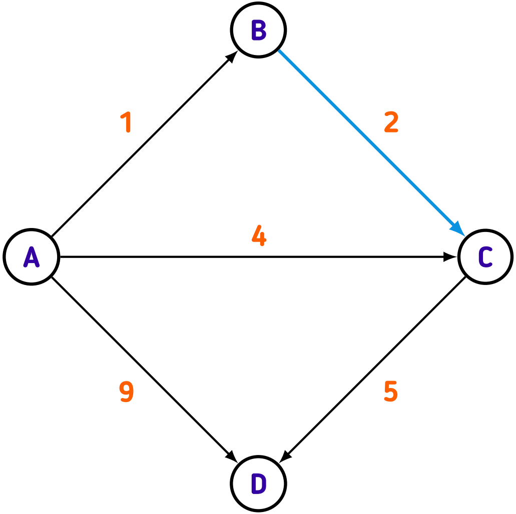
	A	B	C	D
$\mathrm{dist}(u, \mathbf{A})$	0	1	4	∞

	A	B	C	D
$\mathrm{pred}(u)$	A	A	A	-



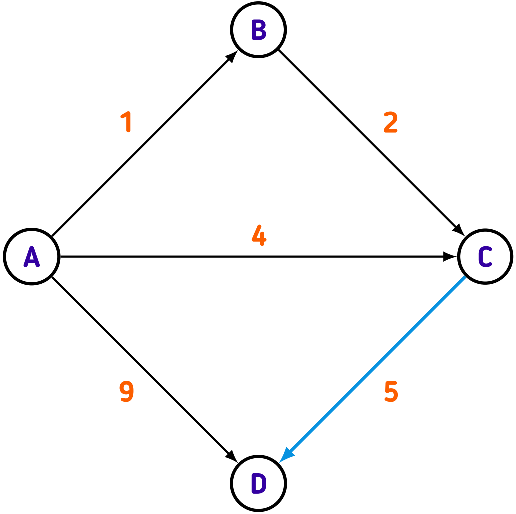
	A	B	C	D
$\mathsf{dist}(u, \mathbf{A})$	0	1	4	**9**

	A	B	C	D
$\mathsf{pred}(u)$	A	A	A	A



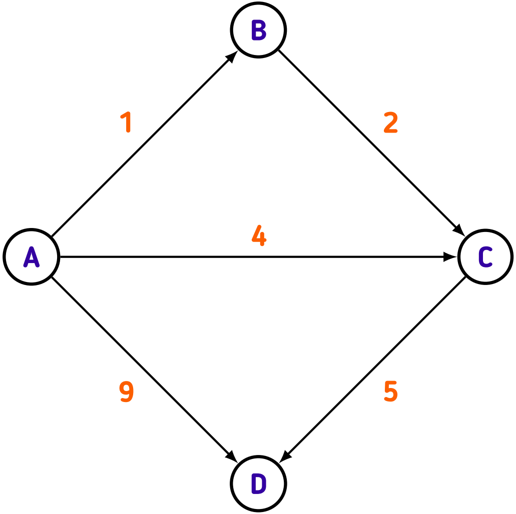
	A	B	C	D
$\text{dist}(u, \textbf{A})$	0	1	**3**	9

	A	B	C	D
$\text{pred}(u)$	A	A	B	A



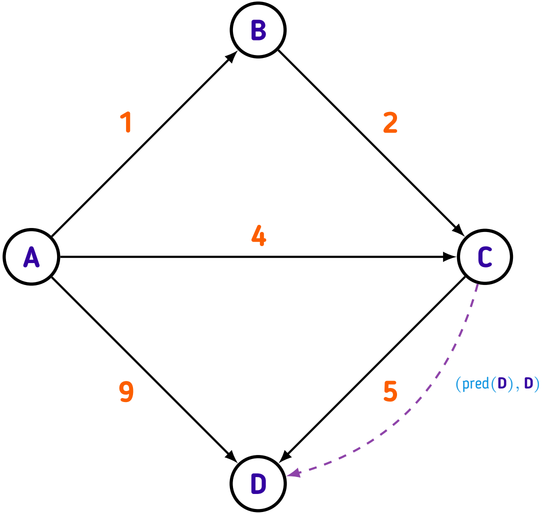
	A	B	C	D
dist$(u,$ **A**$)$	0	1	3	**8**

	A	B	C	D
pred(u)	A	A	B	C



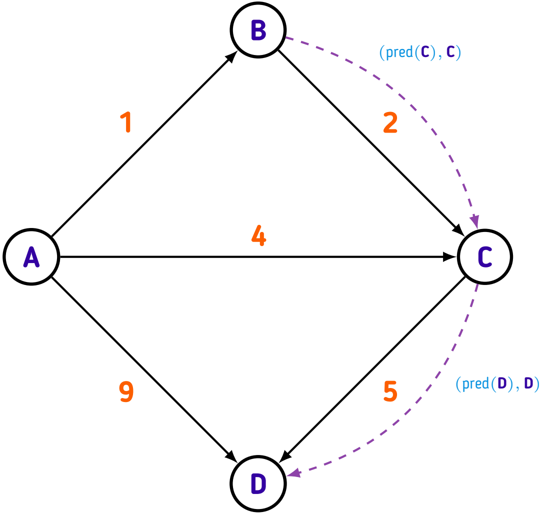
	A	B	C	D
dist(u, **A**)	0	1	3	8

	A	B	C	D
pred(u)	A	A	B	C



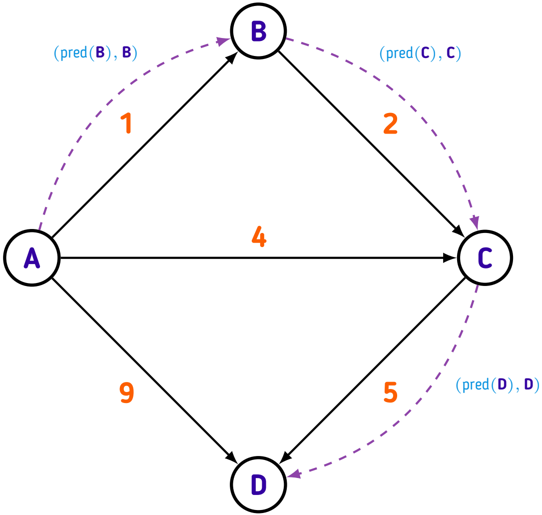
	A	B	C	D
$\mathsf{dist}(u, \mathbf{A})$	0	1	3	8

	A	B	C	D
$\mathsf{pred}(u)$	A	A	B	C



	A	B	C	D
dist(u, A)	0	1	3	8

	A	B	C	D
pred(u)	A	A	B	C



	A	B	C	D
$\mathrm{dist}(u, \mathbf{A})$	0	1	3	8

	A	B	C	D
$\mathrm{pred}(u)$	A	A	B	C

```cpp
pair<vector<int>, vector<int>> dijkstra(int s, int N)
{
    vector<int> dist(N + 1, oo), pred(N + 1, oo);
    dist[s] = 0;
    pred[s] = s;

    processed.reset();

    priority_queue<ii, vector<ii>, greater<ii>> pq;
    pq.emplace(0, s);

    while (not pq.empty())
    {
        auto [d, u] = pq.top();
        pq.pop();
```

```cpp
        if (processed[u])
            continue;

        processed[u] = true;

        for (auto [v, w] : adj[u])
        {
            if (dist[v] > d + w) {
                dist[v] = d + w;
                pred[v] = u;
                pq.emplace(dist[v], v);
            }
        }
    }

    return { dist, pred };
}
```

```
vector<ii> path(int s, int u, const vector<int>& pred)
{
    vector<ii> p;
    int v = u;

    do {
        p.push_back(ii(pred[v], v));
        v = pred[v];
    } while (v != s);

    reverse(p.begin(), p.end());

    return p;
}
```

Problemas sugeridos

1. AtCoder Beginner Contest 143 – Problem E: Travel by Car
2. Codeforces Alpha Round #20 – Problem C: Dijkstra?
3. OJ 1112 – Mice and Maze
4. OJ 10986 – Sending email

Referências

1. **HALIM**, **Felix**; **HALIM**, **Steve**. *Competitive Programming 3*, **2010.**

2. **LAAKSONEN**, **Antti**. *Competitive Programmer's Handbook*, **2018.**

3. **SKIENA**, **Steven**; **REVILLA**, **Miguel**. *Programming Challenges*, **2003.**

4. **Wikipédia**, *Dijkstra's algorithm.* **Acesso em 13/07/2021.**

5. **Wikipédia**, *Edsger W. Dijkstra.* **Acesso em 13/07/2021.**