

Grafos

Algoritmo de Floyd-Warshall

Prof. Edson Alves

Faculdade UnB Gama

Proponentes

Proponentes



Robert W. Floyd
(1962)

Proponentes



Robert W. Floyd
(1962)



Stephen Warshall
(1962)

Proponentes



Robert W. Floyd
(1962)



Stephen Warshall
(1962)



Bernard Roy
(1959)

Características do algoritmo de Floyd-Warshall

Características do algoritmo de Floyd-Warshall

- ★ Computa o caminho mínimo entre todos os pares de vértices de $G(V, E)$

Características do algoritmo de Floyd-Warshall

- ★ Computa o caminho mínimo entre todos os pares de vértices de $G(V, E)$
- ★ É capaz de processar arestas negativas

Características do algoritmo de Floyd-Warshall

- ★ Computa o caminho mínimo entre todos os pares de vértices de $G(V, E)$
- ★ É capaz de processar arestas negativas
- ★ Não processa, mas identifica ciclos negativos

Características do algoritmo de Floyd-Warshall

- ★ Computa o caminho mínimo entre todos os pares de vértices de $G(V, E)$
- ★ É capaz de processar arestas negativas
- ★ Não processa, mas identifica ciclos negativos
- ★ As distâncias são reduzidas por meio do uso de vértices intermediários

Características do algoritmo de Floyd-Warshall

- ★ Computa o caminho mínimo entre todos os pares de vértices de $G(V, E)$
- ★ É capaz de processar arestas negativas
- ★ Não processa, mas identifica ciclos negativos
- ★ As distâncias são reduzidas por meio do uso de vértices intermediários
- ★ Complexidade: $O(V^3)$

Pseudocódigo

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$

Saída: uma matriz d tal que $d[u][v]$ é a distância mínima em G entre u e v

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$

Saída: uma matriz d tal que $d[u][v]$ é a distância mínima em G entre u e v

1. Faça:

(a) $d[u][u] = 0$, para todos $u \in V$

(b) $d[u][v] = w$, se $(u, v, w) \in E$

(c) $d[u][v] = \infty$, caso contrário

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$

Saída: uma matriz d tal que $d[u][v]$ é a distância mínima em G entre u e v

1. Faça:

(a) $d[u][u] = 0$, para todos $u \in V$

(b) $d[u][v] = w$, se $(u, v, w) \in E$

(c) $d[u][v] = \infty$, caso contrário

2. Para cada vértice k e todos os pares $(u, v) \in V^2$, faça

$$d[u][v] = \min(d[u][v], d[u][k] + d[k][v])$$

Pseudocódigo

Entrada: um grafo ponderado $G(V, E)$

Saída: uma matriz d tal que $d[u][v]$ é a distância mínima em G entre u e v

1. Faça:

(a) $d[u][u] = 0$, para todos $u \in V$

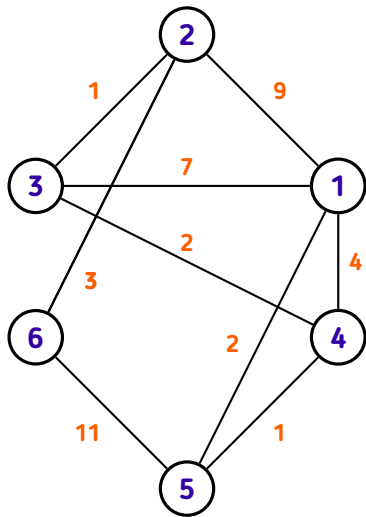
(b) $d[u][v] = w$, se $(u, v, w) \in E$

(c) $d[u][v] = \infty$, caso contrário

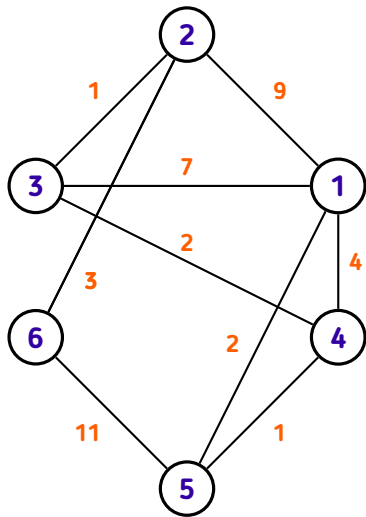
2. Para cada vértice k e todos os pares $(u, v) \in V^2$, faça

$$d[u][v] = \min(d[u][v], d[u][k] + d[k][v])$$

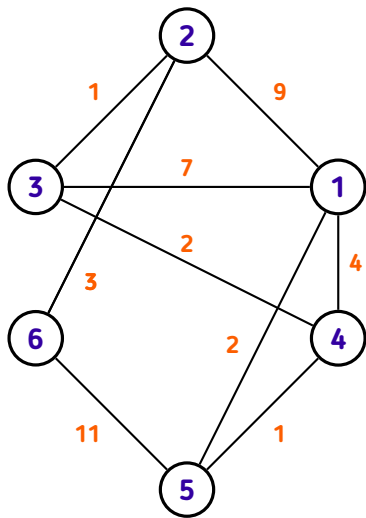
3. Retorne d



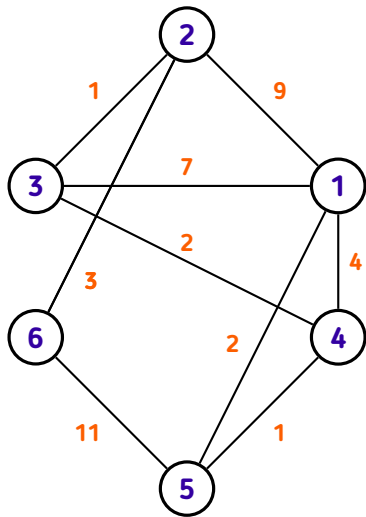
	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						



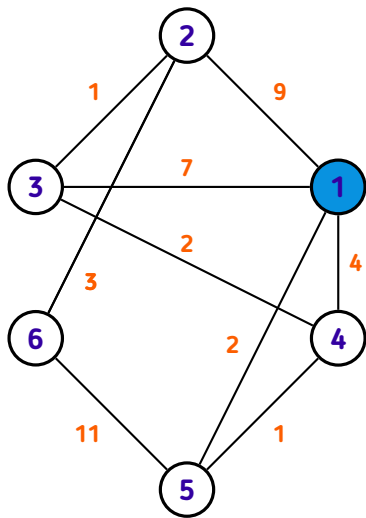
	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0



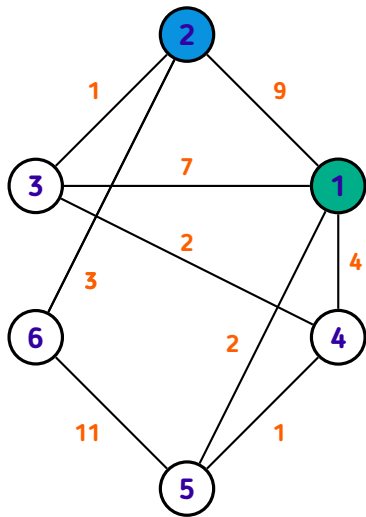
	1	2	3	4	5	6
1	0	9	7	4	2	
2	9	0	1			3
3	7	1	0	2		
4	4		2	0	1	
5	2			1	0	11
6		3			11	0



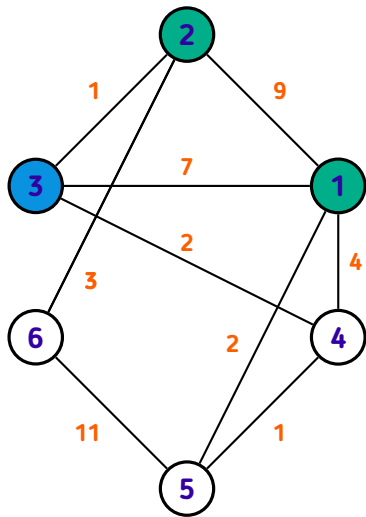
	1	2	3	4	5	6
1	0	9	7	4	2	∞
2	9	0	1	∞	∞	3
3	7	1	0	2	∞	∞
4	4	∞	2	0	1	∞
5	2	∞	∞	1	0	11
6	∞	3	∞	∞	11	0



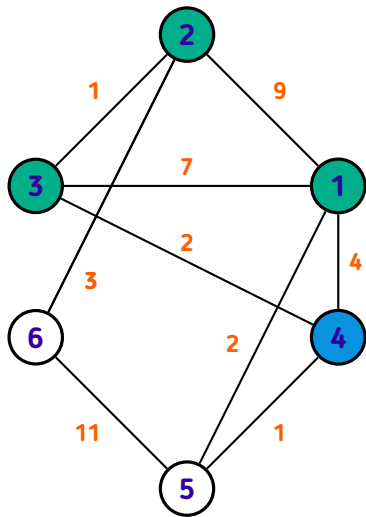
	1	2	3	4	5	6
1	0	9	7	4	2	∞
2	9	0	1	13	11	3
3	7	1	0	2	9	∞
4	4	13	2	0	1	∞
5	2	11	9	1	0	11
6	∞	3	∞	∞	11	0



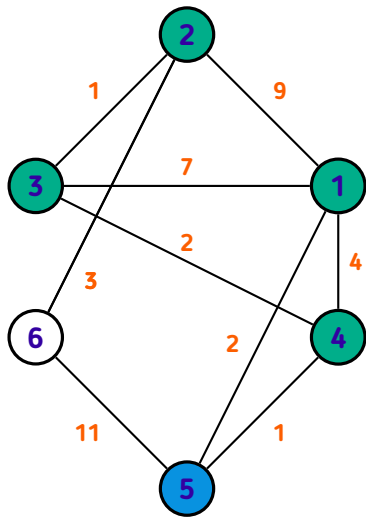
	1	2	3	4	5	6
1	0	9	7	4	2	12
2	9	0	1	13	11	3
3	7	1	0	2	9	4
4	4	13	2	0	1	16
5	2	11	9	1	0	11
6	12	3	4	16	11	0



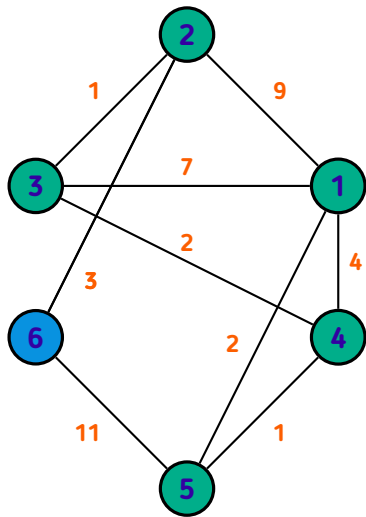
	1	2	3	4	5	6
1	0	8	7	4	2	11
2	8	0	1	3	10	3
3	7	1	0	2	9	4
4	4	3	2	0	1	6
5	2	10	9	1	0	11
6	11	3	4	6	11	0



	1	2	3	4	5	6
1	0	7	6	4	2	10
2	7	0	1	3	4	3
3	6	1	0	2	3	4
4	4	3	2	0	1	6
5	2	4	3	1	0	7
6	10	3	4	6	7	0



	1	2	3	4	5	6
1	0	6	5	3	2	9
2	6	0	1	3	4	3
3	5	1	0	2	3	4
4	3	3	2	0	1	6
5	2	4	3	1	0	7
6	9	3	4	6	7	0



	1	2	3	4	5	6
1	0	6	5	3	2	9
2	6	0	1	3	4	3
3	5	1	0	2	3	4
4	3	3	2	0	1	6
5	2	4	3	1	0	7
6	9	3	4	6	7	0

```
vector<vector<int>> floyd_warshall(int N)
{
    vector<vector<int>> dist(N + 1, vector<int>(N + 1, oo));

    for (int u = 1; u <= N; ++u)
        dist[u][u] = 0;

    for (int u = 1; u <= N; ++u)
        for (auto [v, w] : adj[u])
            dist[u][v] = w;

    for (int k = 1; k <= N; ++k)
        for (int u = 1; u <= N; ++u)
            for (int v = 1; v <= N; ++v)
                if (dist[u][k] < oo and dist[k][v] < oo)
                    dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);

    return dist;
}
```

Identificação de um caminho mínimo

Identificação de um caminho mínimo

- ★ O algoritmo de Floyd-Warshall computa as distâncias mínimas, mas não os caminhos mínimos

Identificação de um caminho mínimo

- ★ O algoritmo de Floyd-Warshall computa as distâncias mínimas, mas não os caminhos mínimos
- ★ Para determinar um caminho mínimo, é preciso definir uma matriz auxiliar pred , onde $\text{pred}[u][v] = \text{antecessor de } v \text{ no caminho mínimo de } u \text{ a } v$

Identificação de um caminho mínimo

★ O algoritmo de Floyd-Warshall computa as distâncias mínimas, mas não os caminhos mínimos

★ Para determinar um caminho mínimo, é preciso definir uma matriz auxiliar pred , onde $\text{pred}[u][v] = \text{antecessor de } v \text{ no caminho mínimo de } u \text{ a } v$

★ No início do algoritmo,

(a) $\text{pred}[u][u] = u, \forall u \in V$

(b) $\text{pred}[u][v] = u, \text{ se } (u, v) \in E$

(c) $\text{pred}[u][v] = \text{undef}, \text{ caso contrário}$

Identificação de um caminho mínimo

★ Se k atualizar $d[u][v]$, faça $\text{pred}[u][v] = \text{pred}[k][v]$

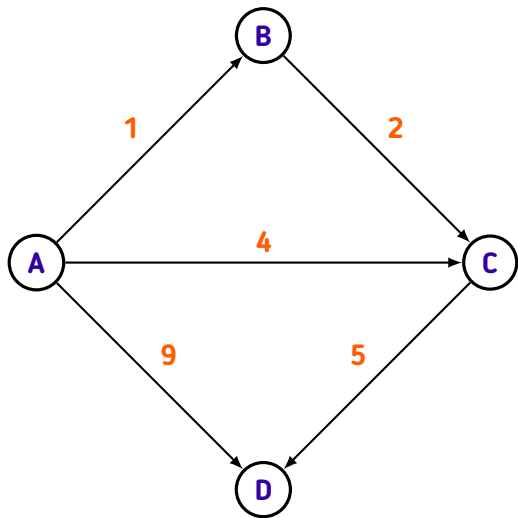
Identificação de um caminho mínimo

★ Se k atualizar $d[u][v]$, faça $\text{pred}[u][v] = \text{pred}[k][v]$

★ A sequência

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

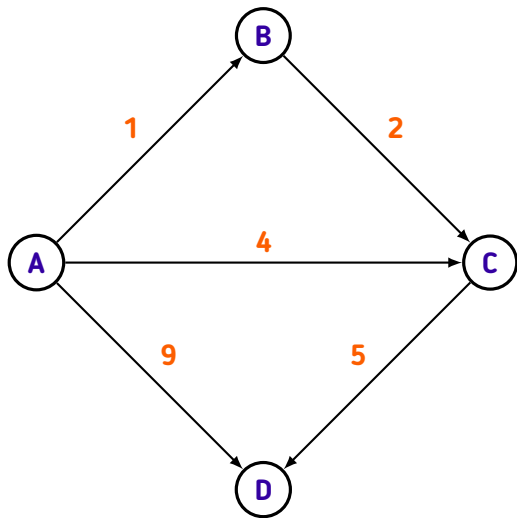
é um caminho mínimo de u a v composto por k arestas e tamanho $d[u][v]$



$M =$

A	B	C	D	
				A
				B
				C
				D

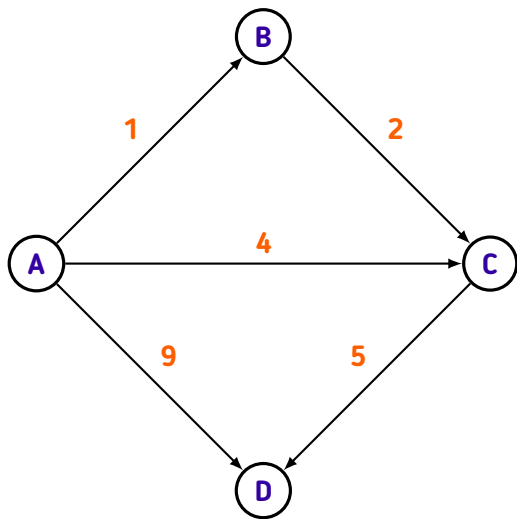
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
A	0_A				A
B		0_B			B
C			0_C		C
D				0_D	D

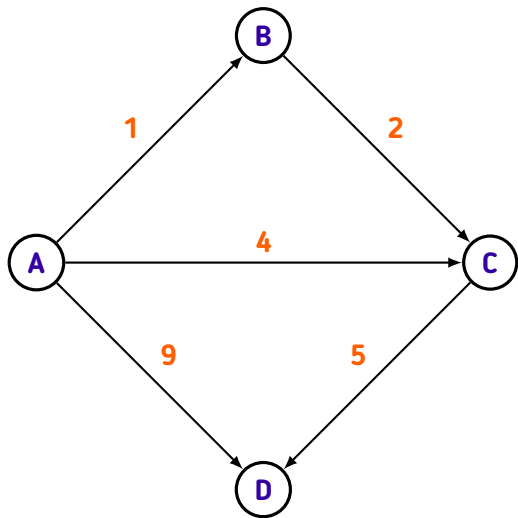
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
A	0_A	1_A	4_A	9_A	A
B		0_B	2_B		B
C			0_C	5_C	C
D				0_D	D

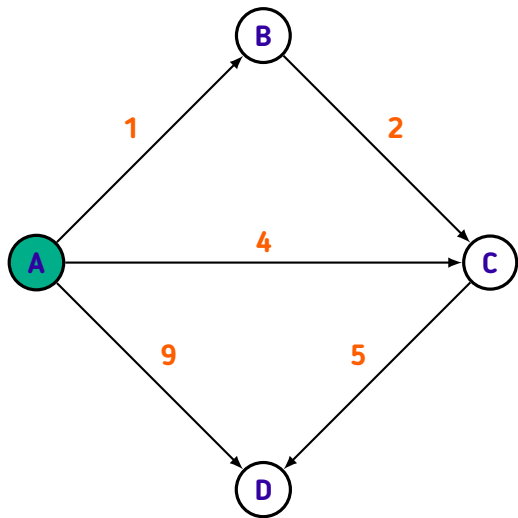
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	4_A	9_A	A
	∞_-	0_B	2_B	∞_-	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

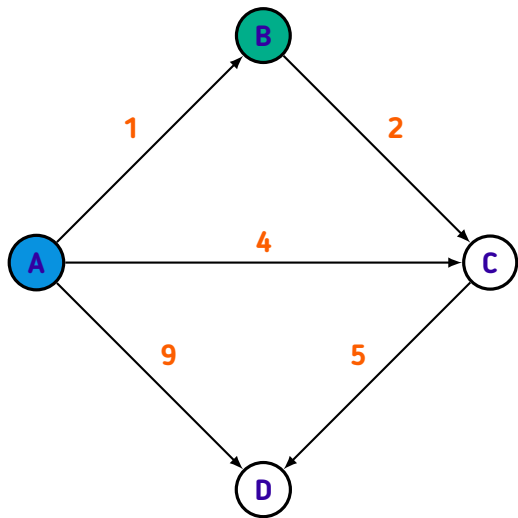
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	4_A	9_A	A
	∞_-	0_B	2_B	∞_-	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

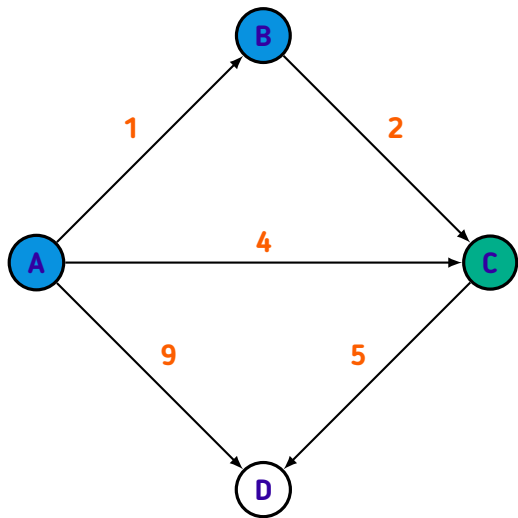
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	3_B	9_A	A
	∞_-	0_B	2_B	∞_-	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

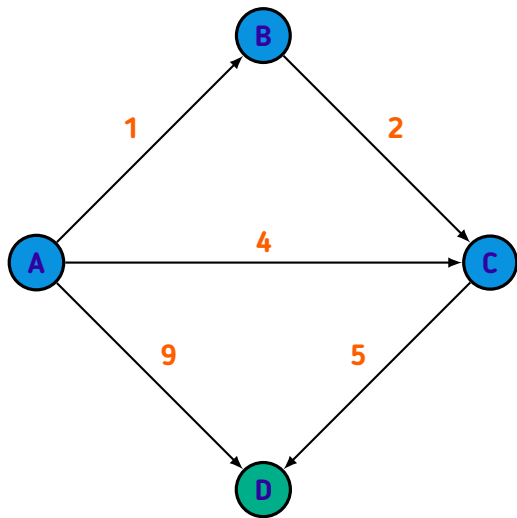
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	3_B	8_C	A
	∞_-	0_B	2_B	7_C	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

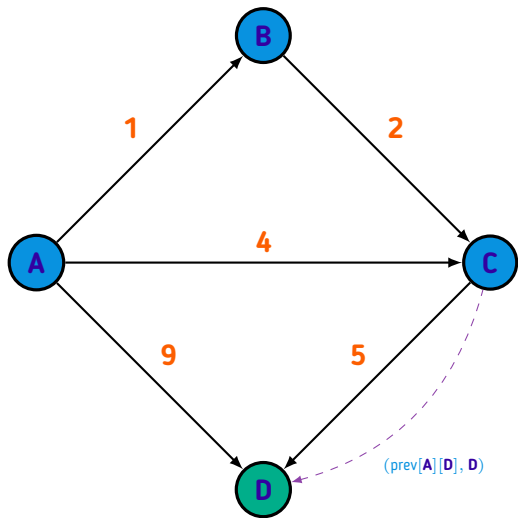
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	3_B	8_C	A
	∞_-	0_B	2_B	7_C	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

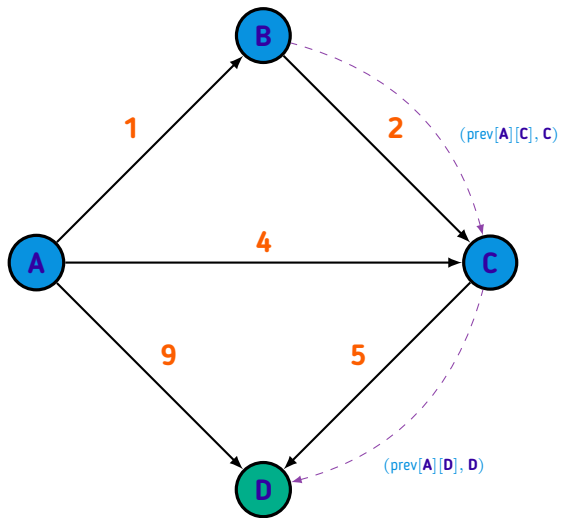
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
A	0_A	1_A	3_B	8_C	A
B	∞_-	0_B	2_B	7_C	B
C	∞_-	∞_-	0_C	5_C	C
D	∞_-	∞_-	∞_-	0_D	D

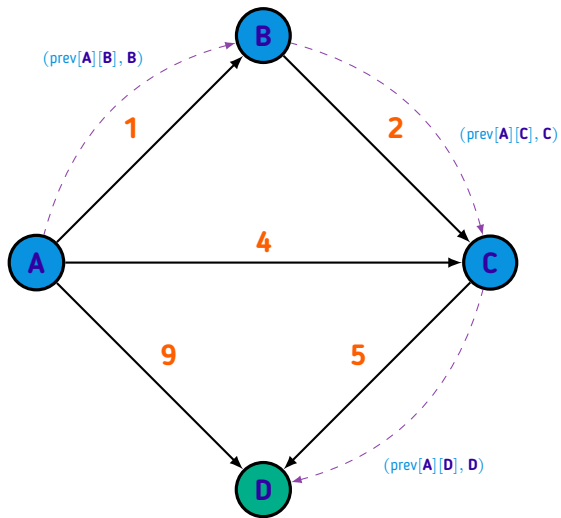
$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	3_B	8_C	A
	∞_-	0_B	2_B	7_C	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$



$M =$

	A	B	C	D	
	0_A	1_A	3_B	8_C	A
	∞_-	0_B	2_B	7_C	B
	∞_-	∞_-	0_C	5_C	C
	∞_-	∞_-	∞_-	0_D	D

$$m_{ij} = \text{dist}[i][j]_{\text{pred}[i][j]}$$

```
pair<vector<vector<int>>, vector<vector<int>>>
floyd_warshall(int N)
{
    vector<vector<int>> dist(N + 1, vector<int>(N + 1, oo));
    vector<vector<int>> pred(N + 1, vector<int>(N + 1, oo));

    for (int u = 1; u <= N; ++u)
    {
        dist[u][u] = 0;
        pred[u][u] = u;
    }

    for (int u = 1; u <= N; ++u)
        for (auto [v, w] : adj[u]) {
            dist[u][v] = w;
            pred[u][v] = u;
        }
}
```

```

for (int k = 1; k <= N; ++k)
{
    for (int u = 1; u <= N; ++u)
    {
        for (int v = 1; v <= N; ++v)
        {
            if (dist[u][k] < oo and dist[k][v] < oo
                and dist[u][v] > dist[u][k] + dist[k][v]) {
                dist[u][v] = dist[u][k] + dist[k][v];
                pred[u][v] = pred[k][v];
            }
        }
    }
}

return { dist, pred };
}

```

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

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

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

    return p;
}
```

Identificação de ciclos negativos

Identificação de ciclos negativos

- ★ O algoritmo de Floyd-Warshall é capaz de detectar ciclos negativos

Identificação de ciclos negativos

- ★ O algoritmo de Floyd-Warshall é capaz de detectar ciclos negativos
- ★ Inicialmente $\text{dist}[u][u] = 0, \forall u \in V$, se G não tem *autoloops*

Identificação de ciclos negativos

- ★ O algoritmo de Floyd-Warshall é capaz de detectar ciclos negativos
- ★ Inicialmente $\text{dist}[u][u] = 0, \forall u \in V$, se G não tem **autoloops**
- ★ Caso exista um ciclo negativo que passe por u , seguir este ciclo de u a u torna $\text{dist}[u][u] < 0$

Identificação de ciclos negativos

- ★ O algoritmo de Floyd-Warshall é capaz de detectar ciclos negativos
- ★ Inicialmente $\text{dist}[u][u] = 0, \forall u \in V$, se G não tem **autoloops**
- ★ Caso exista um ciclo negativo que passe por u , seguir este ciclo de u a u torna $\text{dist}[u][u] < 0$
- ★ Assim, G terá um ciclo negativo se, ao final do algoritmo, $\text{dist}[i][i] < 0$ para algum $i \in V$

```
bool has_negative_cycle(int N) {  
    for (int u = 1; u <= N; ++u)  
        for (int v = 1; v <= N; ++v)  
            dist[u][v] = u == v ? 0 : oo;  
  
    for (int u = 1; u <= N; ++u)  
        for (auto [v, w] : adj[u])  
            dist[u][v] = w;  
  
    for (int k = 1; k <= N; ++k)  
        for (int u = 1; u <= N; ++u)  
            for (int v = 1; v <= N; ++v)  
                if (dist[u][k] < oo and dist[k][v] < oo)  
                    dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);  
  
    for (int i = 1; i <= N; ++i)  
        if (dist[i][i] < 0) return true;  
  
    return false;  
}
```

Problemas sugeridos

1. [Codeforces Round #179 \(Div. 1\) – Problem B: Greg and Graph](#)
2. [LightOJ – Travel Company](#)
3. [OJ 104 – Arbitrage](#)
4. [OJ 10171 – Meeting Prof. Miguel...](#)

Referências

1. CP-Algorithms. *Floyd-Warshall Algorithm*, acesso em 03/08/2021.
2. HALIM, Felix; HALIM, Steve. *Competitive Programming 3*, 2010.
3. LAAKSONEN, Antti. *Competitive Programmer's Handbook*, 2018.
4. SKIENA, Steven; REVILLA, Miguel. *Programming Challenges*, 2003.