

Grafos

Programação Dinâmica em DAGs

Prof. Edson Alves

Faculdade UnB Gama

Relação entre DAGs e programação dinâmica

Relação entre DAGs e programação dinâmica

- ★ É possível utilizar algoritmos de programação (DP) dinâmica em um grafo G , se G é direcionado e acíclico (DAG – *directed acyclic graph*)

Relação entre DAGs e programação dinâmica

- ★ É possível utilizar algoritmos de programação (DP) dinâmica em um grafo G , se G é direcionado e acíclico (DAG – *directed acyclic graph*)
- ★ De fato, há uma relação direta entre uma DP e um DAG

Relação entre DAGs e programação dinâmica

- ★ É possível utilizar algoritmos de programação (DP) dinâmica em um grafo G , se G é direcionado e acíclico (DAG – *directed acyclic graph*)
- ★ De fato, há uma relação direta entre uma DP e um DAG
- ★ Uma DP induz um DAG cujos vértices são os estados e as arestas indicam as transições entre estados

Aplicação: número de caminhos de u a v

Aplicação: número de caminhos de u a v

- ★ É possível computar o número de caminhos distintos entre os vértices $u, v \in V$ de um DAG $G(V, E)$

Aplicação: número de caminhos de u a v

★ É possível computar o número de caminhos distintos entre os vértices $u, v \in V$ de um DAG $G(V, E)$

★ Seja $p_u(x)$ o número de caminhos de u a x

Aplicação: número de caminhos de u a v

★ É possível computar o número de caminhos distintos entre os vértices $u, v \in V$ de um DAG $G(V, E)$

★ Seja $p_u(x)$ o número de caminhos de u a x

★ Há um único caminho de u a u : permanecer onde está

Aplicação: número de caminhos de u a v

★ É possível computar o número de caminhos distintos entre os vértices $u, v \in V$ de um DAG $G(V, E)$

★ Seja $p_u(x)$ o número de caminhos de u a x

★ Há um único caminho de u a u : permanecer onde está

★ Assim, $p_u(u) = 1$ é o caso base

Aplicação: número de caminhos de u a v

★ Para os demais vértices $x \in V$, vale que

$$p_u(x) = \sum_{(v_i, x) \in E} p_u(v_i)$$

Aplicação: número de caminhos de u a v

- ★ Para os demais vértices $x \in V$, vale que

$$p_u(x) = \sum_{(v_i, x) \in E} p_u(v_i)$$

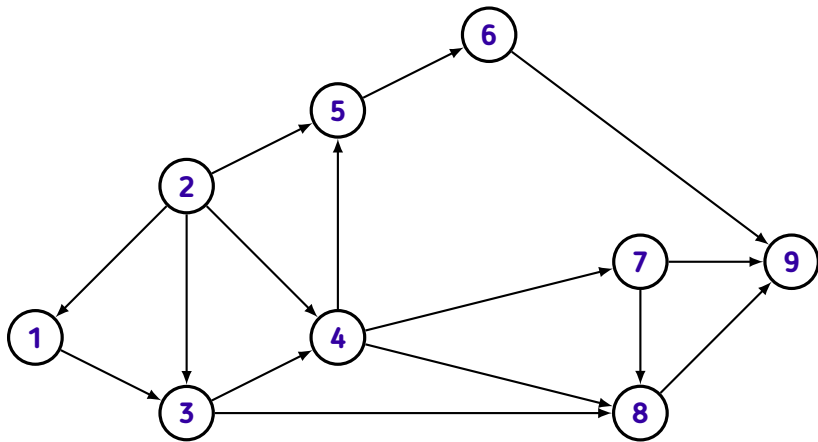
- ★ O cálculo de $p_u(x)$ depende da ordem de processamento dos vértices

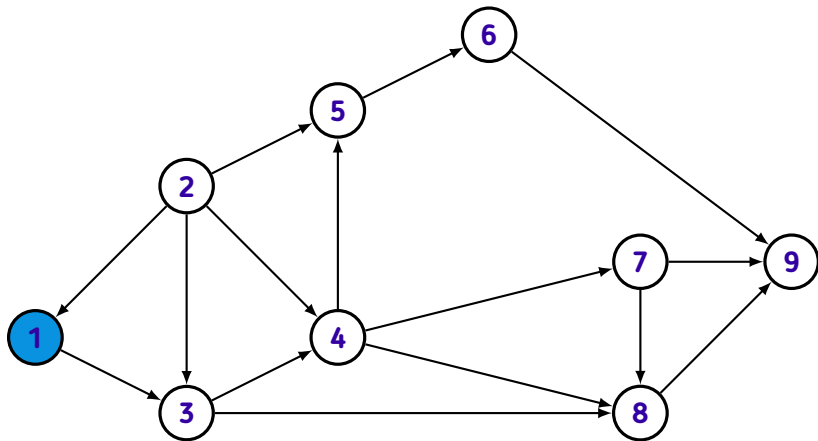
Aplicação: número de caminhos de u a v

- ★ Para os demais vértices $x \in V$, vale que

$$p_u(x) = \sum_{(v_i, x) \in E} p_u(v_i)$$

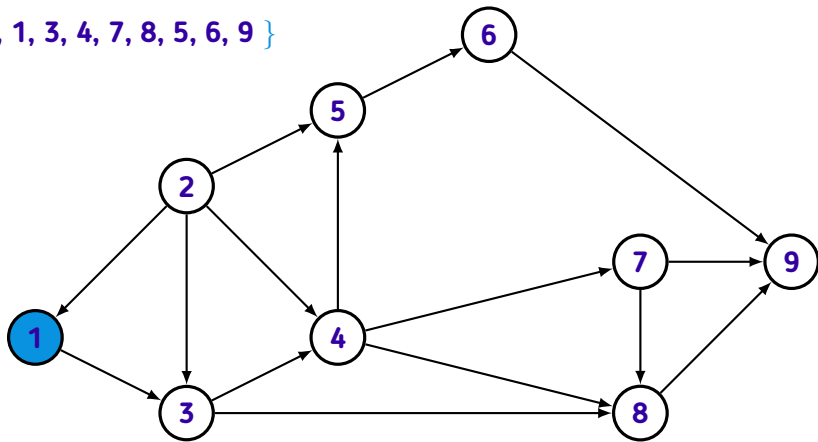
- ★ O cálculo de $p_u(x)$ depende da ordem de processamento dos vértices
- ★ Como G é um DAG, basta usar a ordenação topológica para computar $p_u(x)$





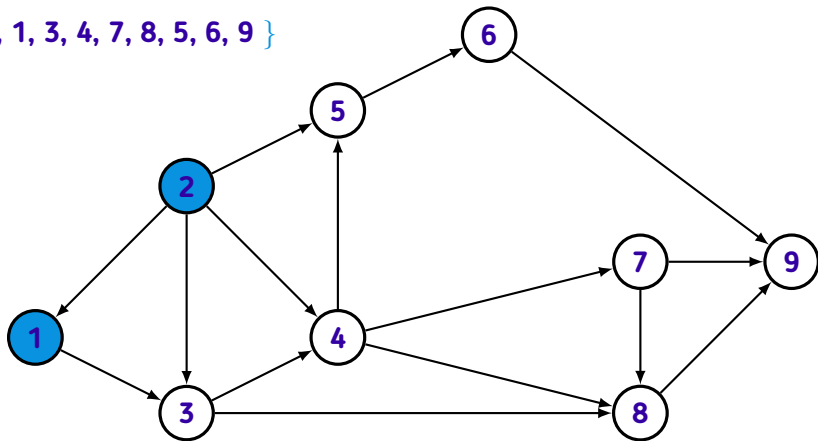
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	-	-	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



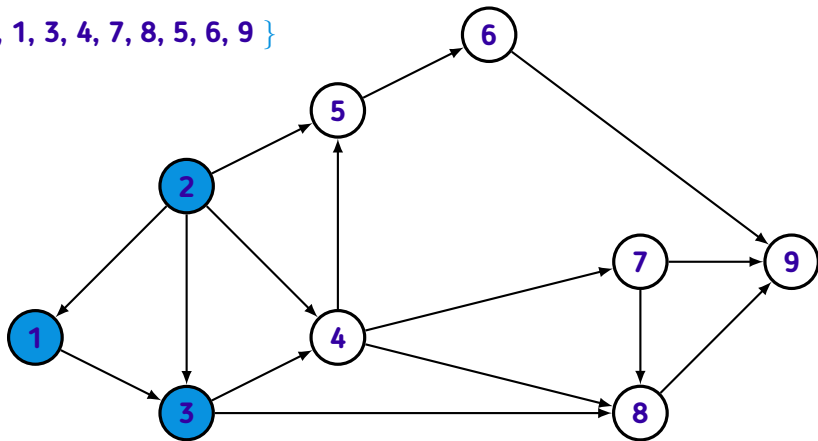
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	-	-	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



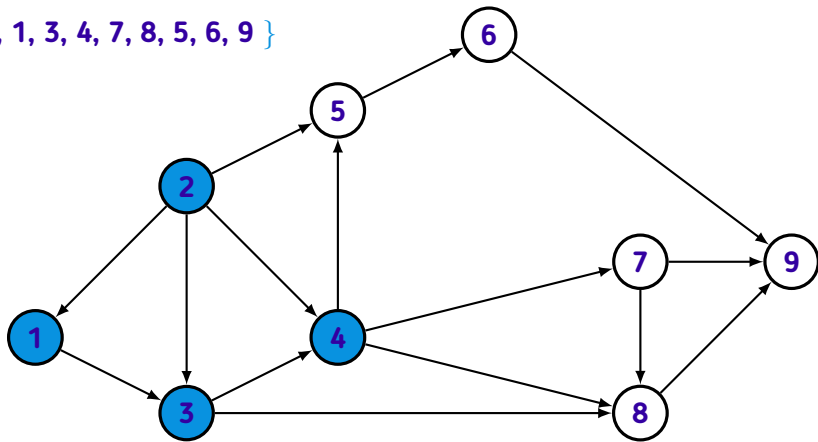
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	-	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



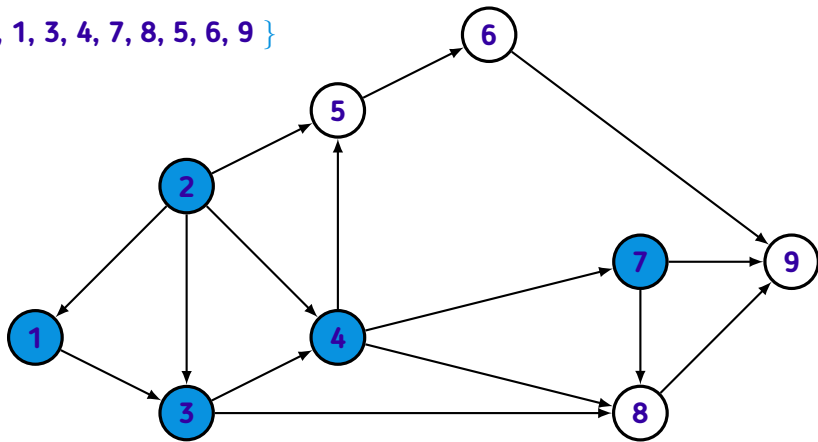
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



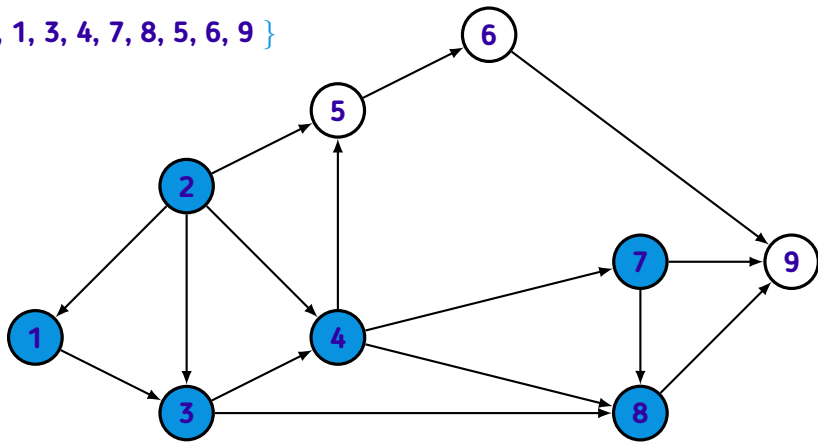
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	1	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



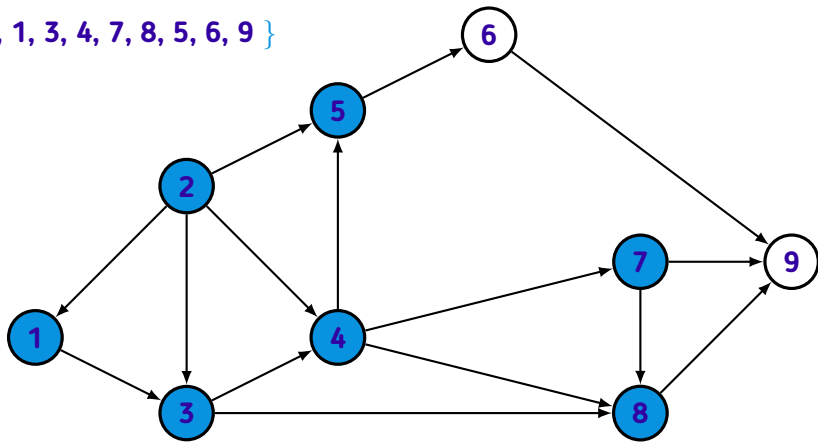
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	1	-	-	1	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



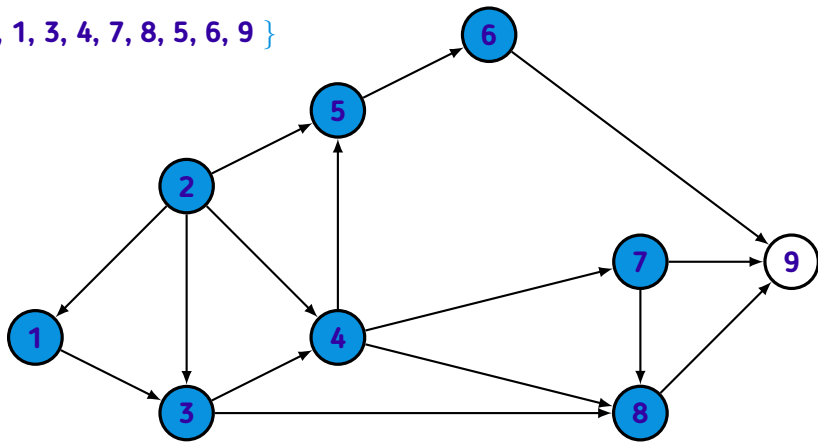
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	1	-	-	1	3	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



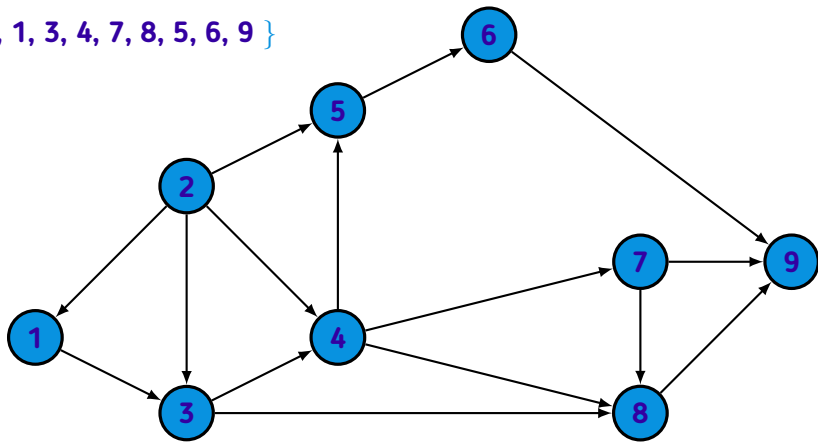
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	1	1	-	1	3	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



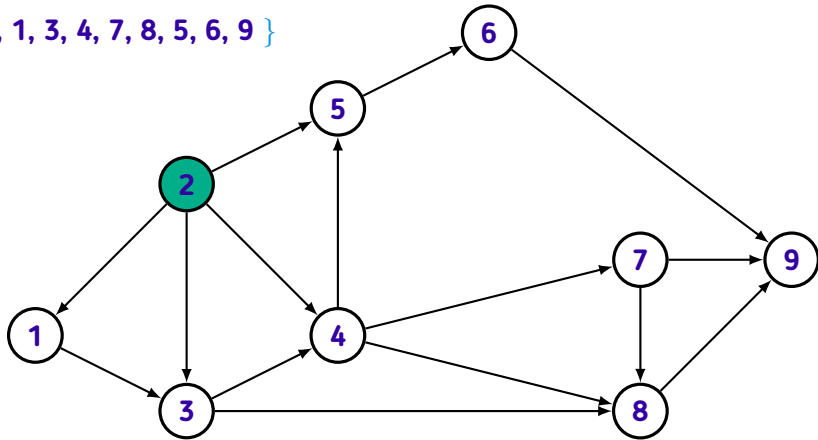
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	1	1	1	1	3	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



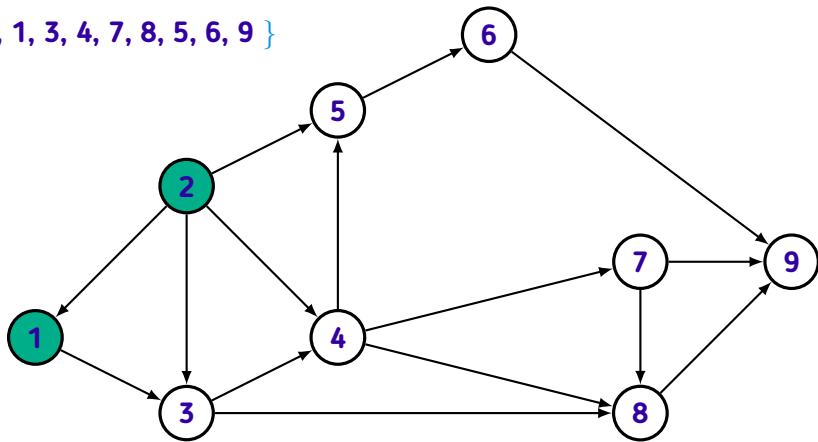
	1	2	3	4	5	6	7	8	9
$p_1(x) =$	1	0	1	1	1	1	1	3	5

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



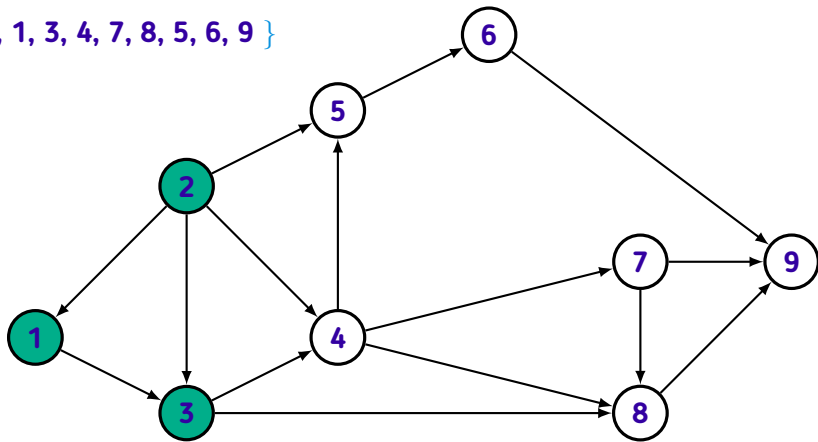
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	-	1	-	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



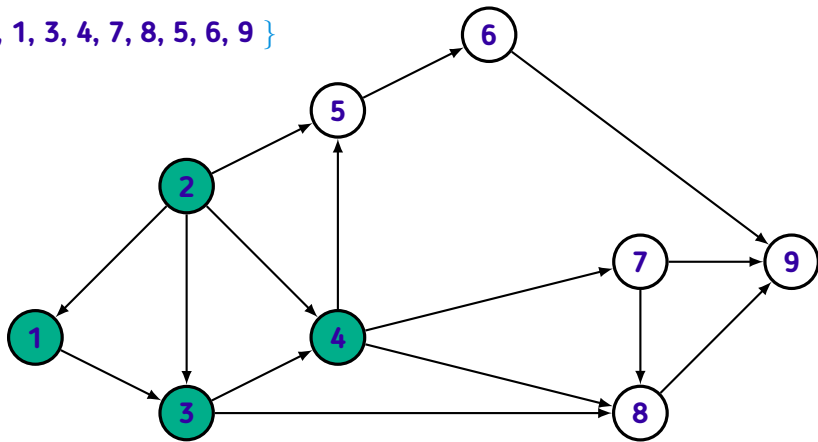
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	-	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



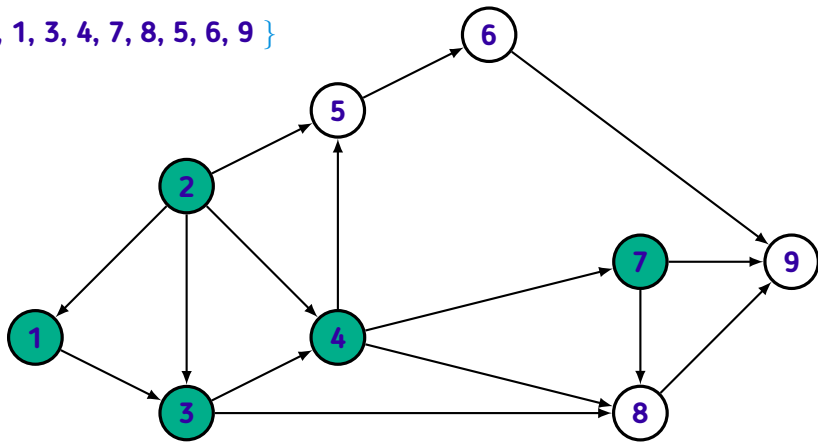
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	-	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



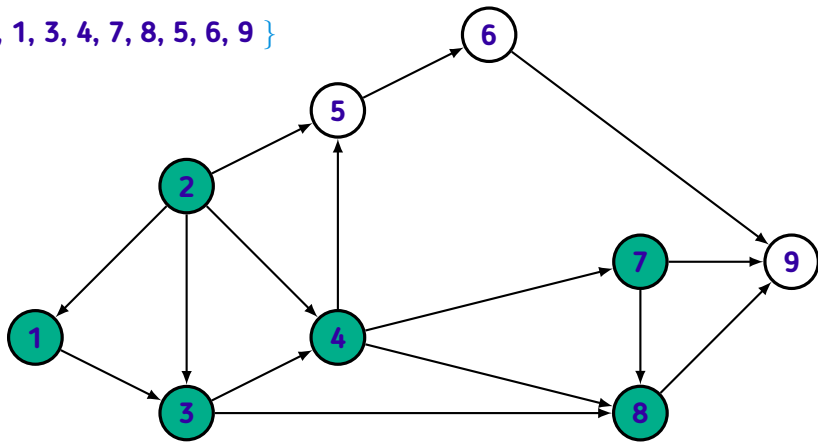
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	3	-	-	-	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



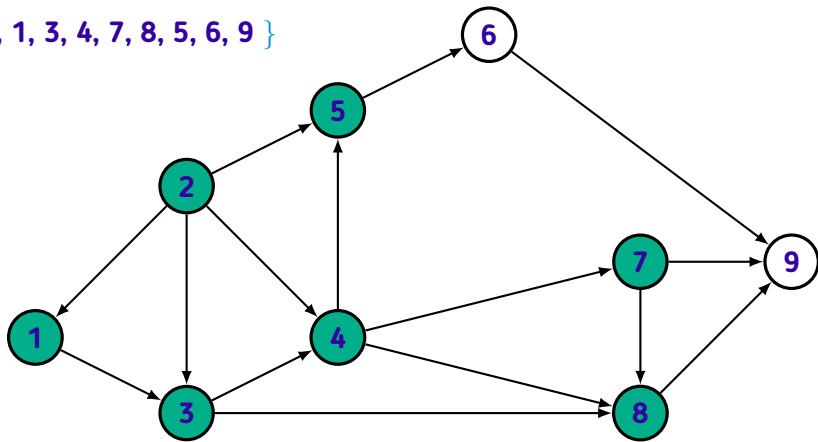
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	3	-	-	3	-	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



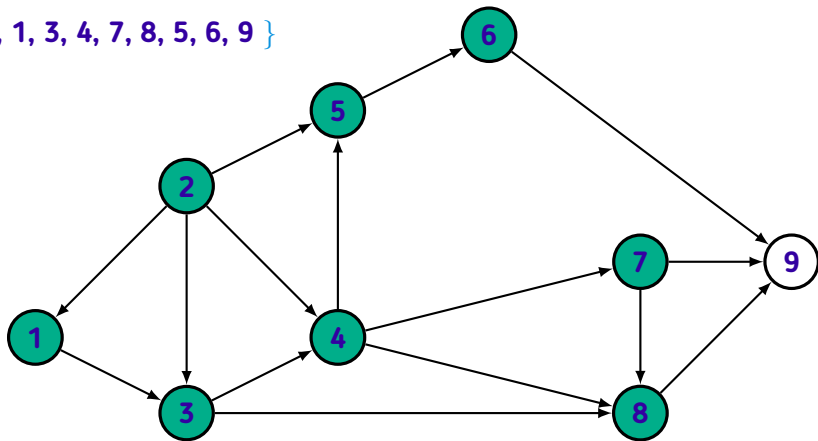
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	3	-	-	3	8	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



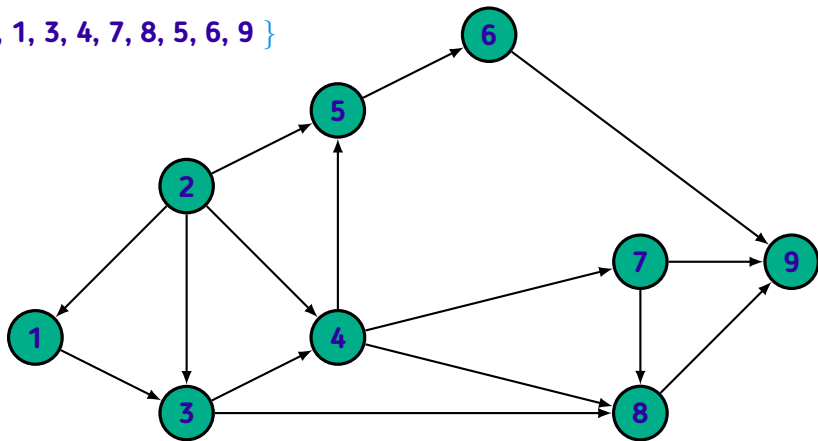
	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	3	4	-	3	8	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	3	4	4	3	8	-

$$O = \{ 2, 1, 3, 4, 7, 8, 5, 6, 9 \}$$



	1	2	3	4	5	6	7	8	9
$p_2(x) =$	1	1	2	3	4	4	3	8	15

```
vector<int> paths(int u, int N)
{
    auto o = topological_sort(N);

    vector<int> ps(N + 1, 0);
    ps[u] = 1;

    for (auto x : o)
        for (auto v : in[x])
            ps[x] += ps[v];

    return ps;
}
```

Aplicação: número de caminhos mínimos de u a v

Aplicação: número de caminhos mínimos de u a v

★ O algoritmo de Dijkstra induz um DAG que indica, para cada vértice do grafo original, como atingir os demais vértices por caminhos mínimos

Aplicação: número de caminhos mínimos de u a v

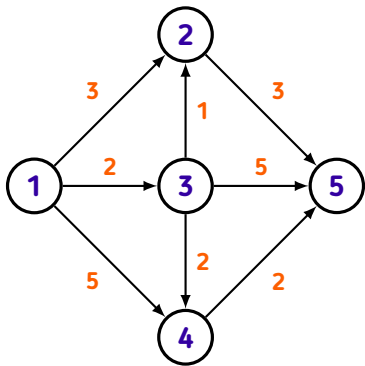
- ★ O algoritmo de Dijkstra induz um DAG que indica, para cada vértice do grafo original, como atingir os demais vértices por caminhos mínimos
- ★ Assim, pode-se usar DP neste grafo

Aplicação: número de caminhos mínimos de u a v

- ★ O algoritmo de Dijkstra induz um DAG que indica, para cada vértice do grafo original, como atingir os demais vértices por caminhos mínimos
- ★ Assim, pode-se usar DP neste grafo
- ★ Assim seria possível, por exemplo, computar o número de caminhos mínimos de u a v

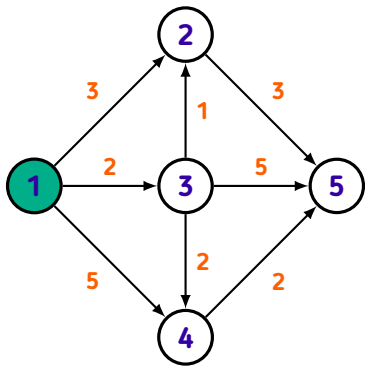
Aplicação: número de caminhos mínimos de u a v

- ★ O algoritmo de Dijkstra induz um DAG que indica, para cada vértice do grafo original, como atingir os demais vértices por caminhos mínimos
- ★ Assim, pode-se usar DP neste grafo
- ★ Assim seria possível, por exemplo, computar o número de caminhos mínimos de u a v
- ★ O algoritmo seria o mesmo usado para computar o número de caminhos, com a ordenação O sendo substituída pela ordenação de Dijkstra



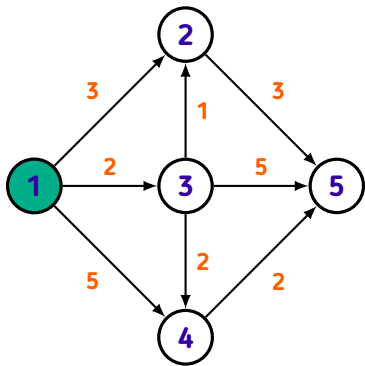
$d_1(x) =$

1	2	3	4	5
-	-	-	-	-



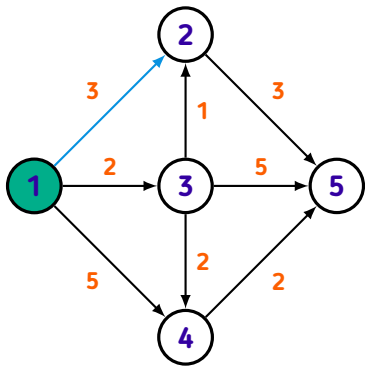
$d_1(x) =$

1	2	3	4	5
0	-	-	-	-



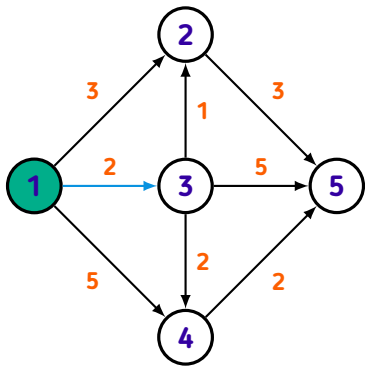
$d_1(x) =$

1	2	3	4	5
0	-	-	-	-



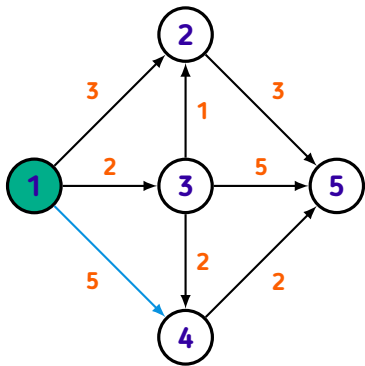
$$d_1(x) =$$

	1	2	3	4	5
	0	3	-	-	-



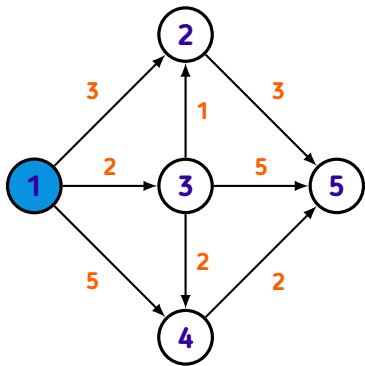
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	-	-



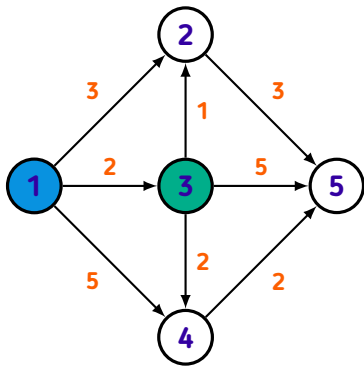
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	5	-



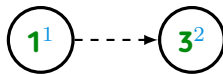
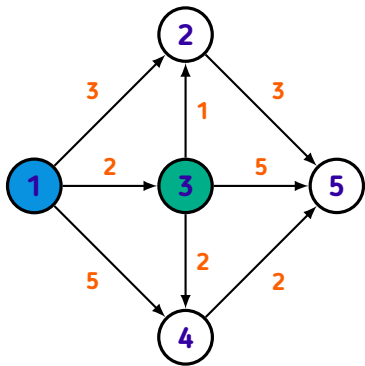
$$d_1(x) =$$

1	2	3	4	5
0	3	2	5	-



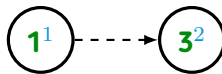
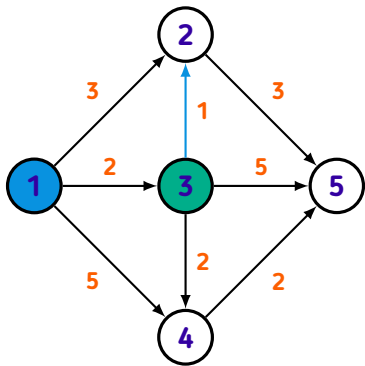
$$d_1(x) =$$

1	2	3	4	5
0	3	2	5	-



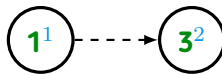
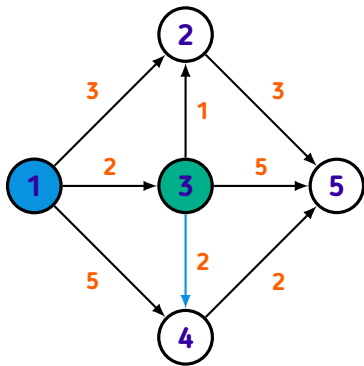
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	5	-



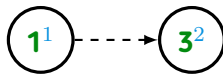
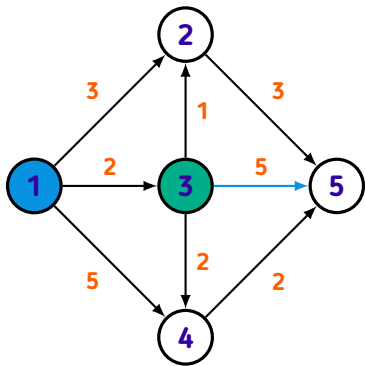
$$d_1(x) =$$

1	2	3	4	5
0	3	2	5	-



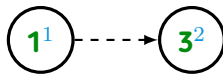
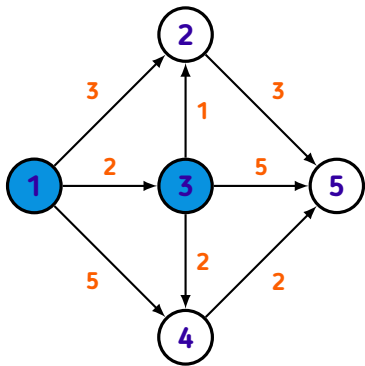
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	4	-



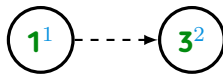
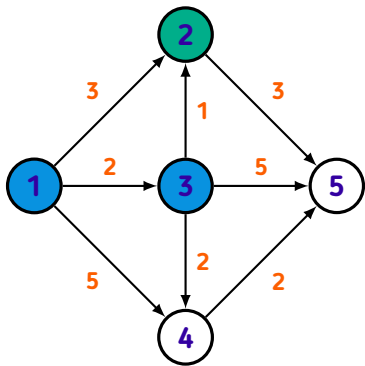
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	4	7



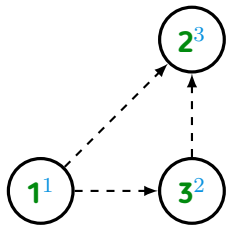
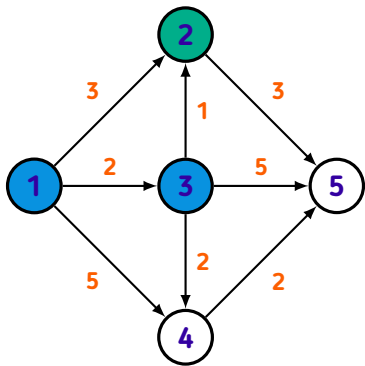
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	4	7



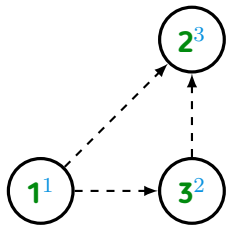
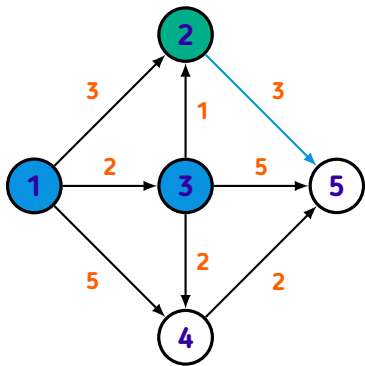
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	4	7



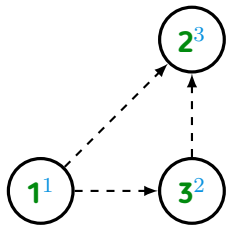
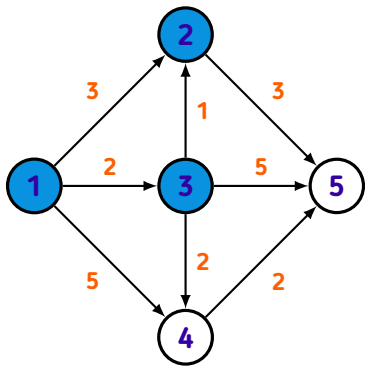
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	7



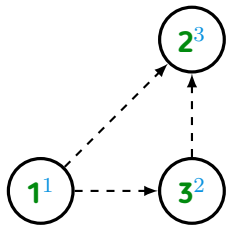
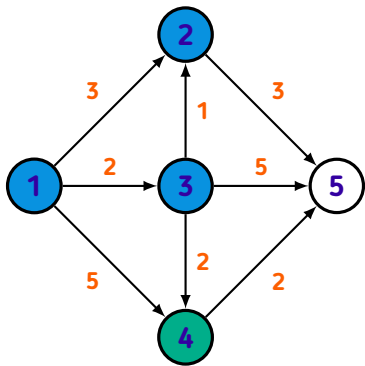
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



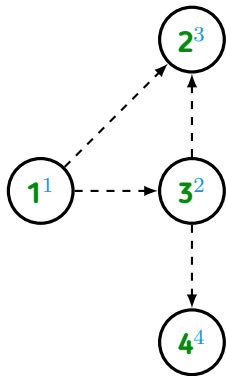
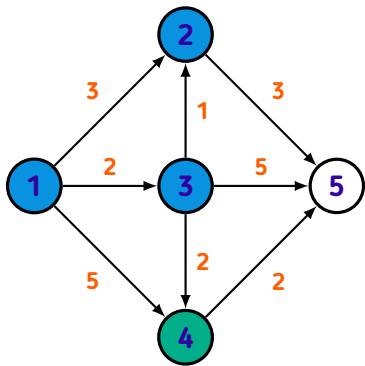
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



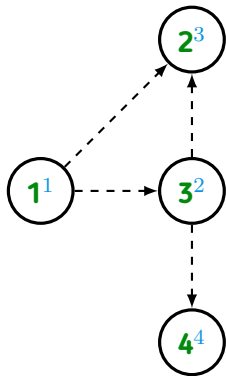
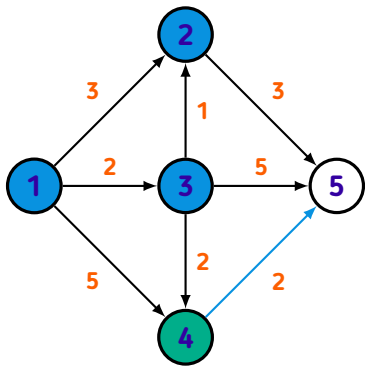
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



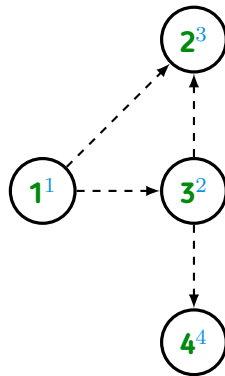
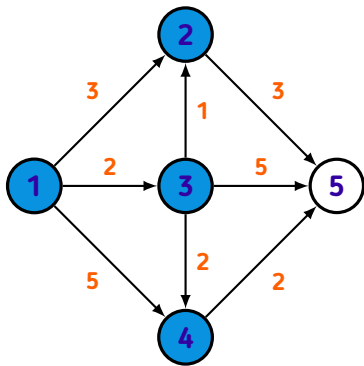
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



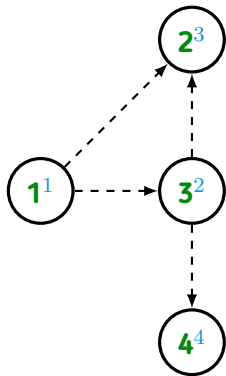
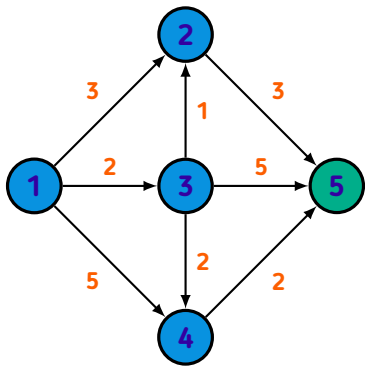
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



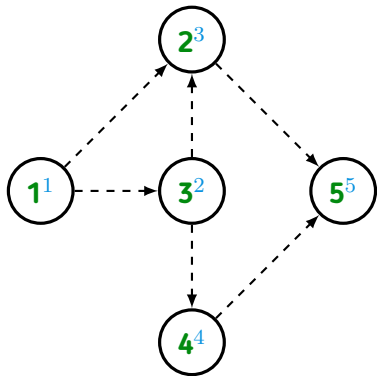
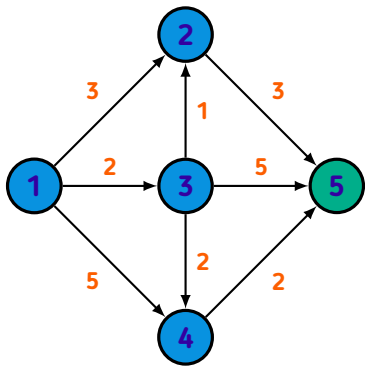
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



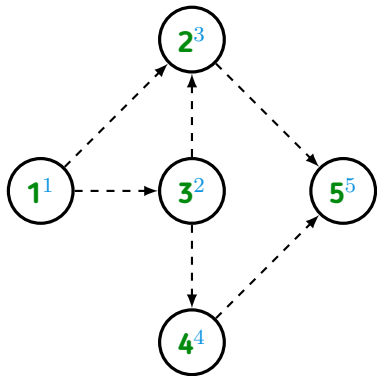
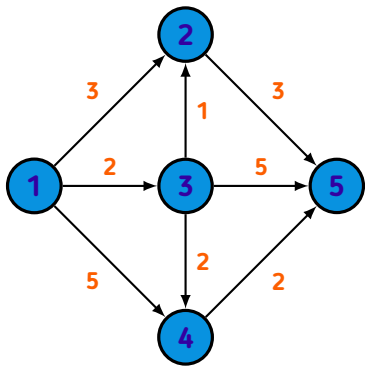
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6



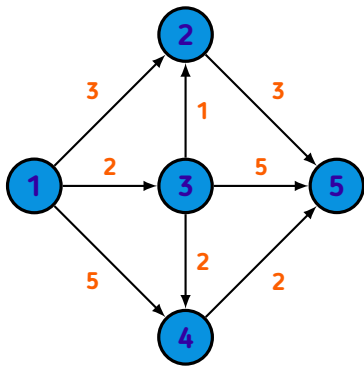
$$d_1(x) =$$

1	2	3	4	5
0	3	2	4	6

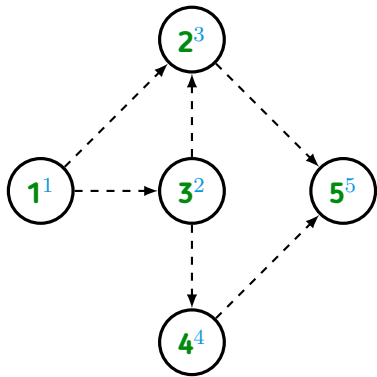


$$d_1(x) =$$

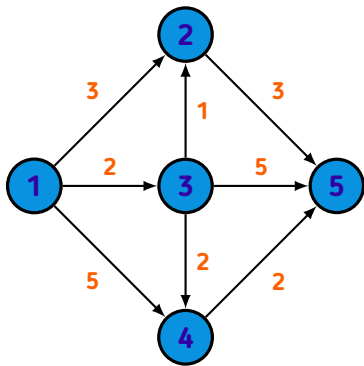
1	2	3	4	5
0	3	2	4	6



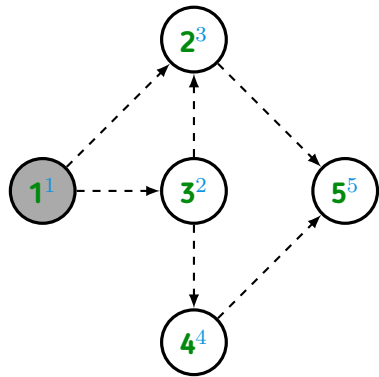
$$d_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 3 & 2 & 4 & 6 \end{array}$$



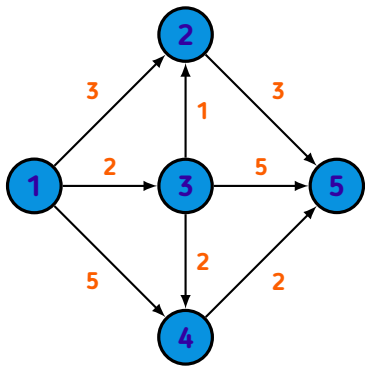
$$p_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline - & - & - & - & - \end{array}$$



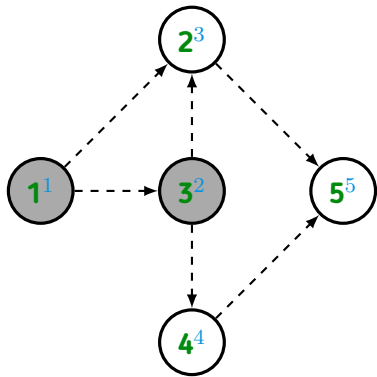
$$d_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 3 & 2 & 4 & 6 \end{array}$$



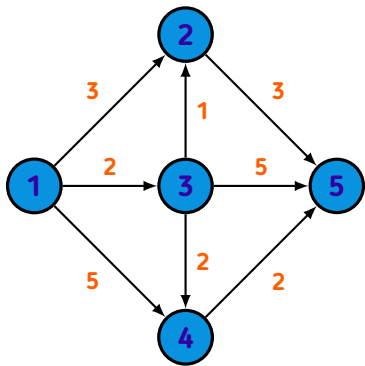
$$p_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & - & - & - & - \end{array}$$



$$d_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 3 & 2 & 4 & 6 \end{array}$$

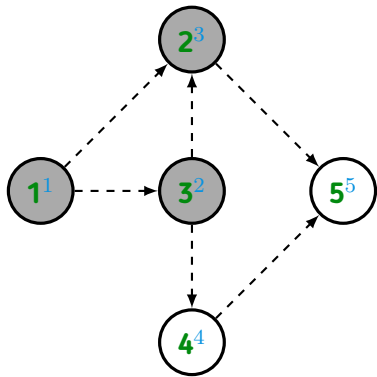


$$p_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & - & 1 & - & - \end{array}$$



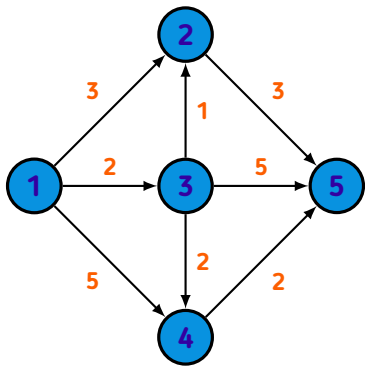
$$d_1(x) =$$

	1	2	3	4	5
	0	3	2	4	6

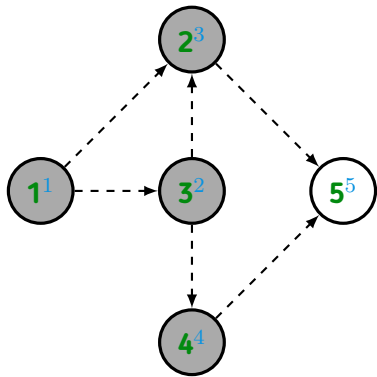


$$p_1(x) =$$

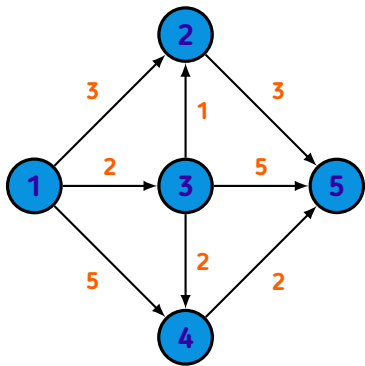
	1	2	3	4	5
	1	2	1	-	-



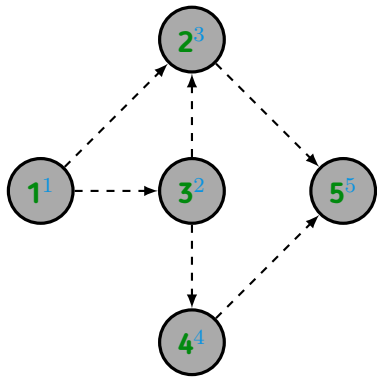
$$d_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 3 & 2 & 4 & 6 \end{array}$$



$$p_1(x) = \begin{array}{c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 2 & 1 & 1 & - \end{array}$$



$$d_1(x) = \begin{array}{c|c|c|c|c} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \hline & 0 & 3 & 2 & 4 & 6 \end{array}$$



$$p_1(x) = \begin{array}{c|c|c|c|c} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \hline & 1 & 2 & 1 & 1 & \mathbf{3} \end{array}$$

```
vector<int> min_paths(int s, int N)
{
    vector<int> ps(N + 1, 0);
    ps[s] = 1;

    auto o = dijkstra_order(s, N);

    for (auto x : o)
        for (auto v : in[x])
            ps[x] += ps[v];

    return ps;
}
```



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

    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();

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

```
    order.emplace_back(u);
    processed[u] = true;

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

    return order;
}
```

Problemas sugeridos

1. CSES 1202 – Investigation
2. HackerEarth – Counting on Tree
3. SPOJ DAGCNT2 – Counting in a DAG
4. Timus 1018 – Binary Apple Tree

Referências

1. Codeforces. *DP on Trees Tutorial*, acesso em 14/09/2021.
2. DUMOL, Tim. *IOI Training Week 5 – DP on Trees and DAGs*.
3. HALIM, Felix; HALIM, Steve. *Competitive Programming 3*, 2010.
4. LAAKSONEN, Antti. *Competitive Programmer's Handbook*, 2018.