Explicação dos Exercícios de Grafo

Exercícios B, D e G

Programação Competitiva Unesp Bauru
apresenta

Arissa “TokiDebug” Yoshida
Professora, programadora, musicista, futeboleira, 42

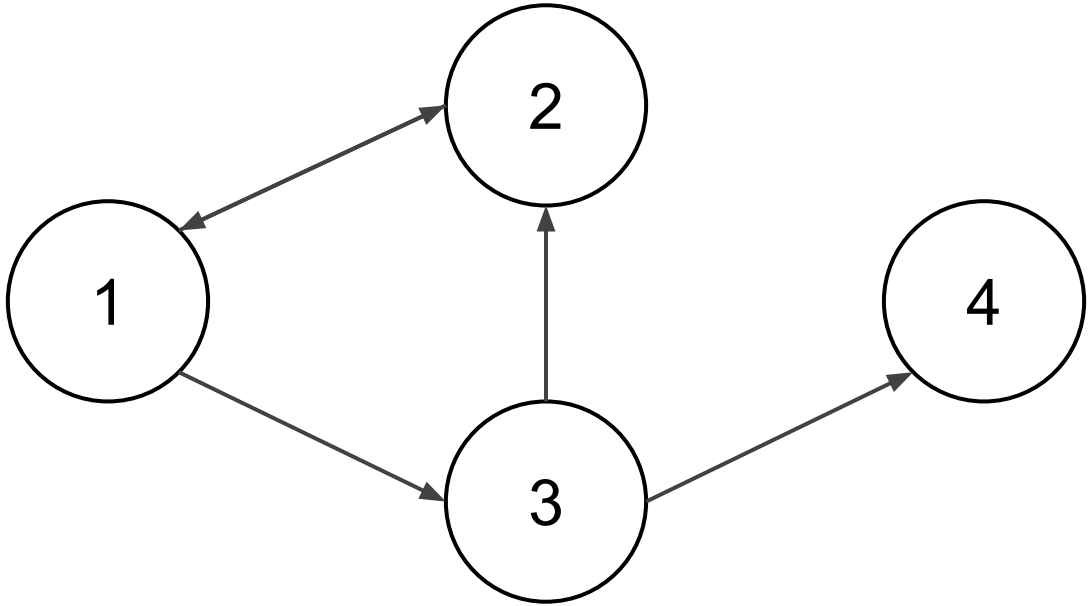
B - Find the Treasure

“Tesouros não são apenas ouro e prata, amigo.”
- Capitã Toki Sparrow

04/Julho ○ **Local:** aqui ○ **Horário:** agora

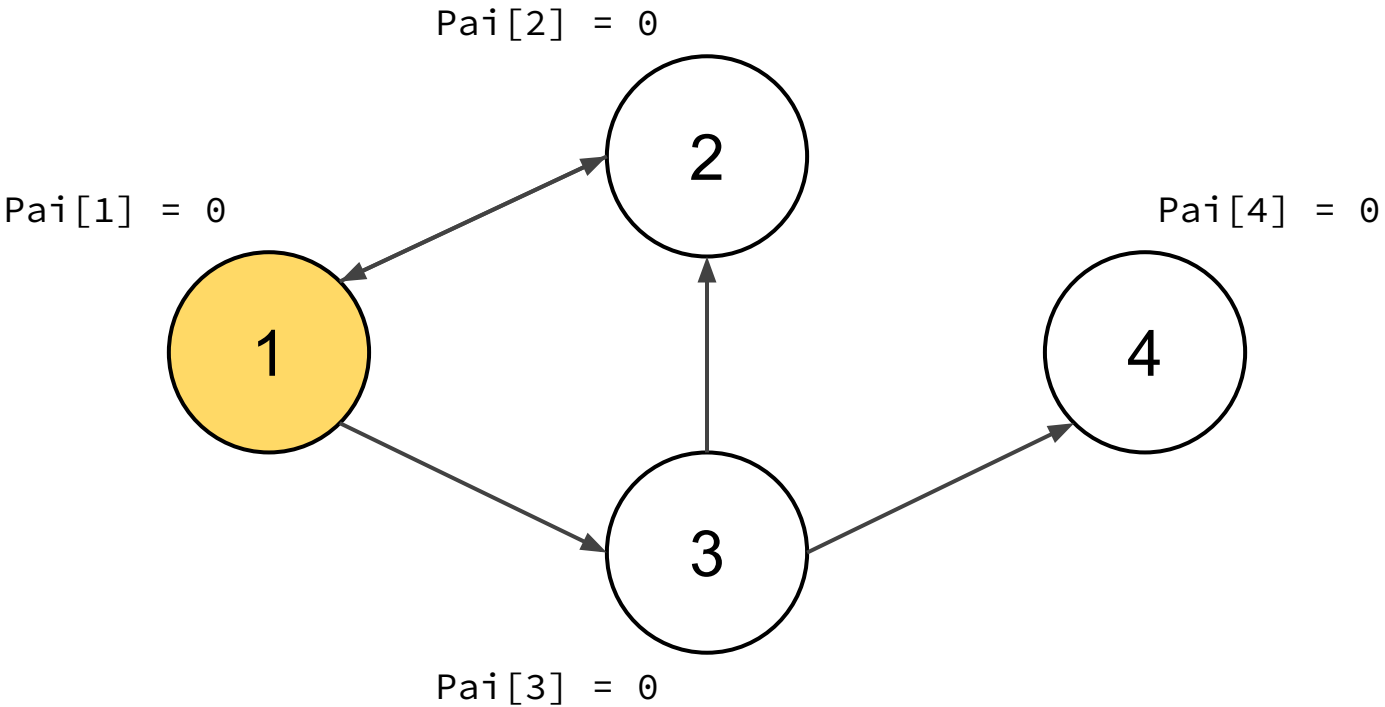
Valor: R$ 100.000.000,00
(Ingressos a venda com o **Pai**ola)

B - Find the Treasure



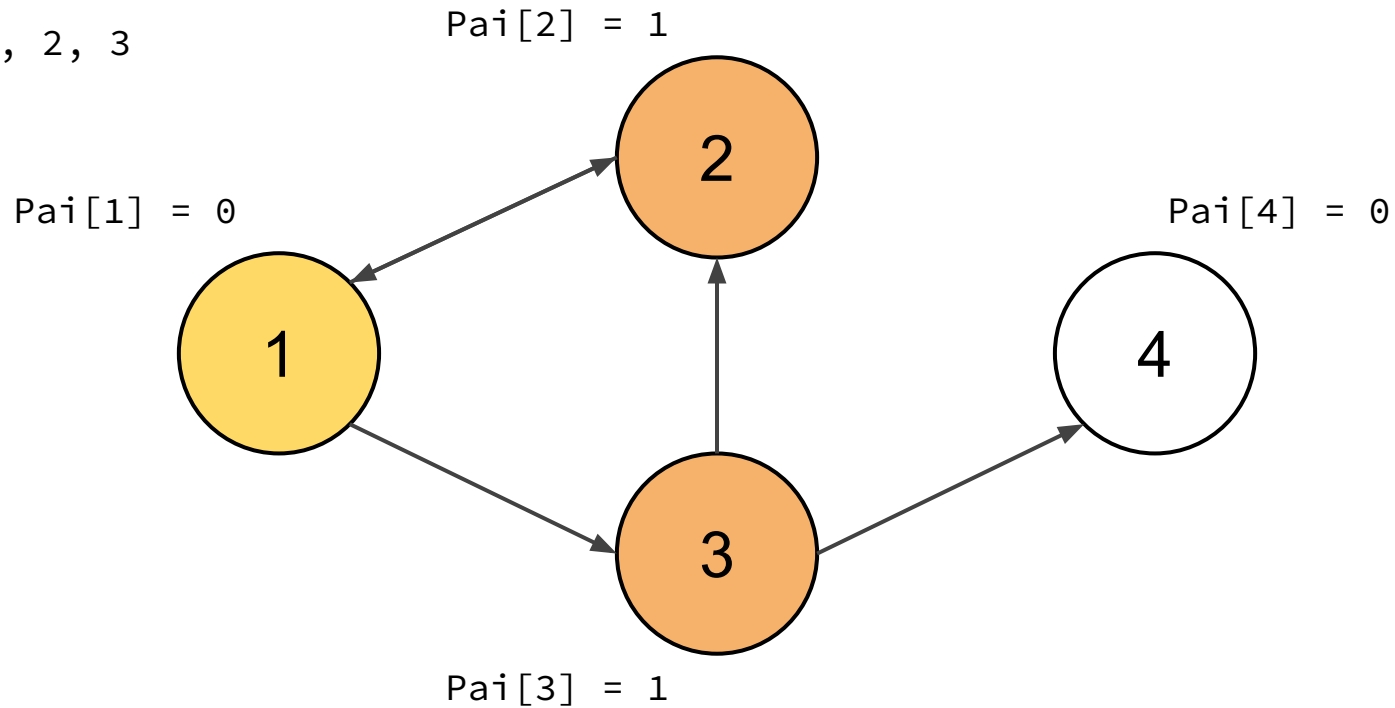
B - Find the Treasure

Fila: 1



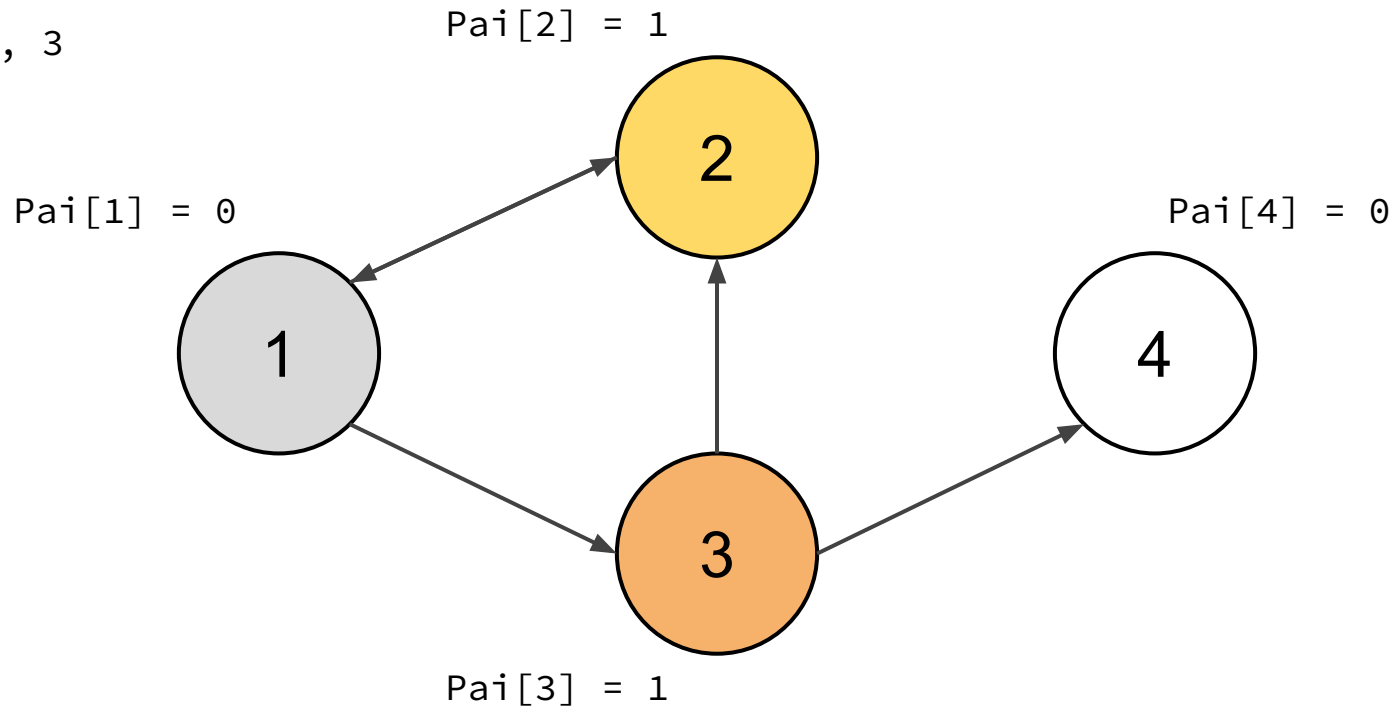
B - Find the Treasure

Fila: 1, 2, 3



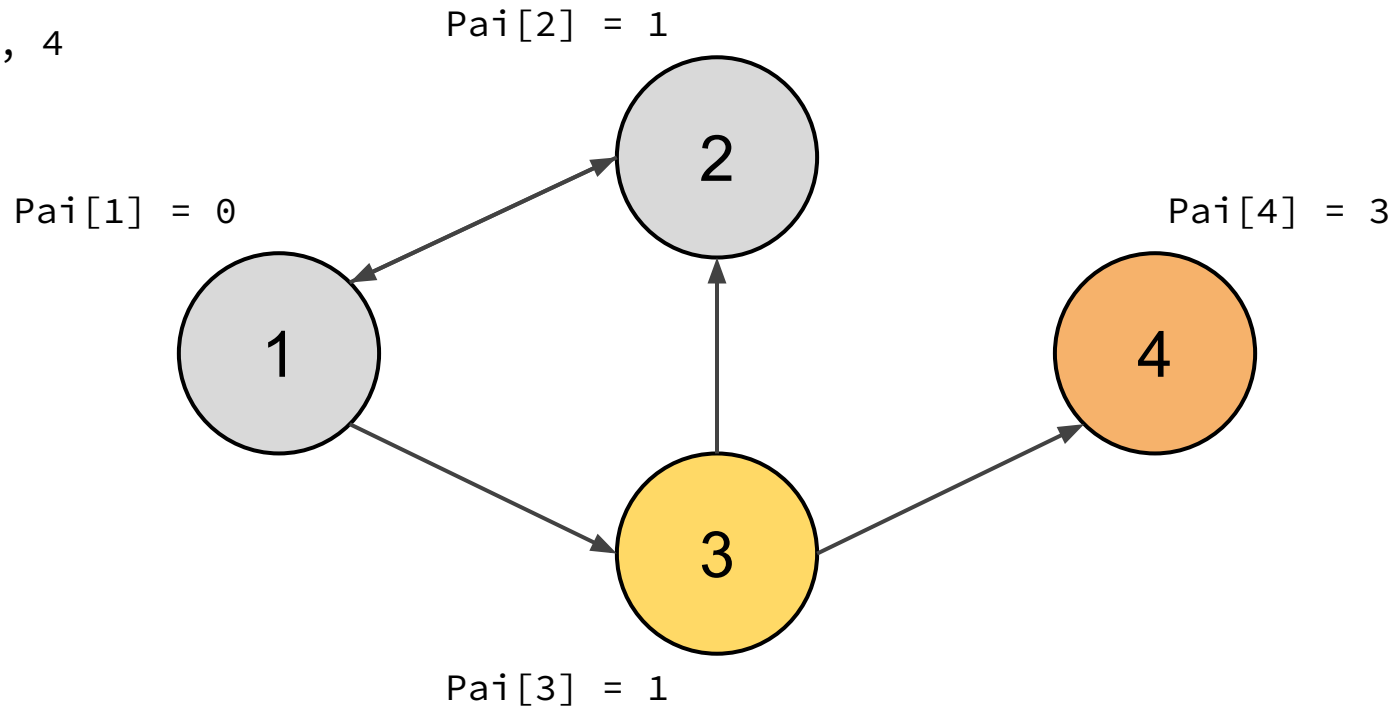
B - Find the Treasure

Fila: 2, 3



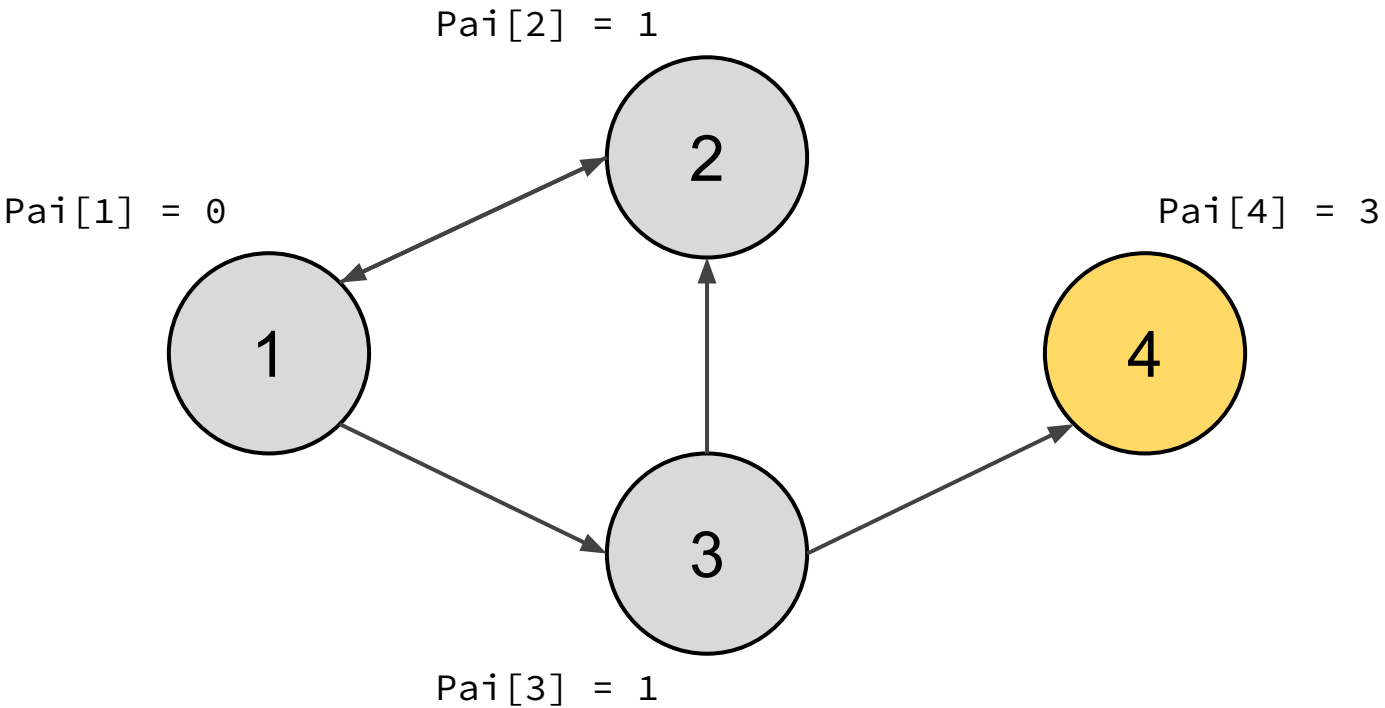
B - Find the Treasure

Fila: 3, 4

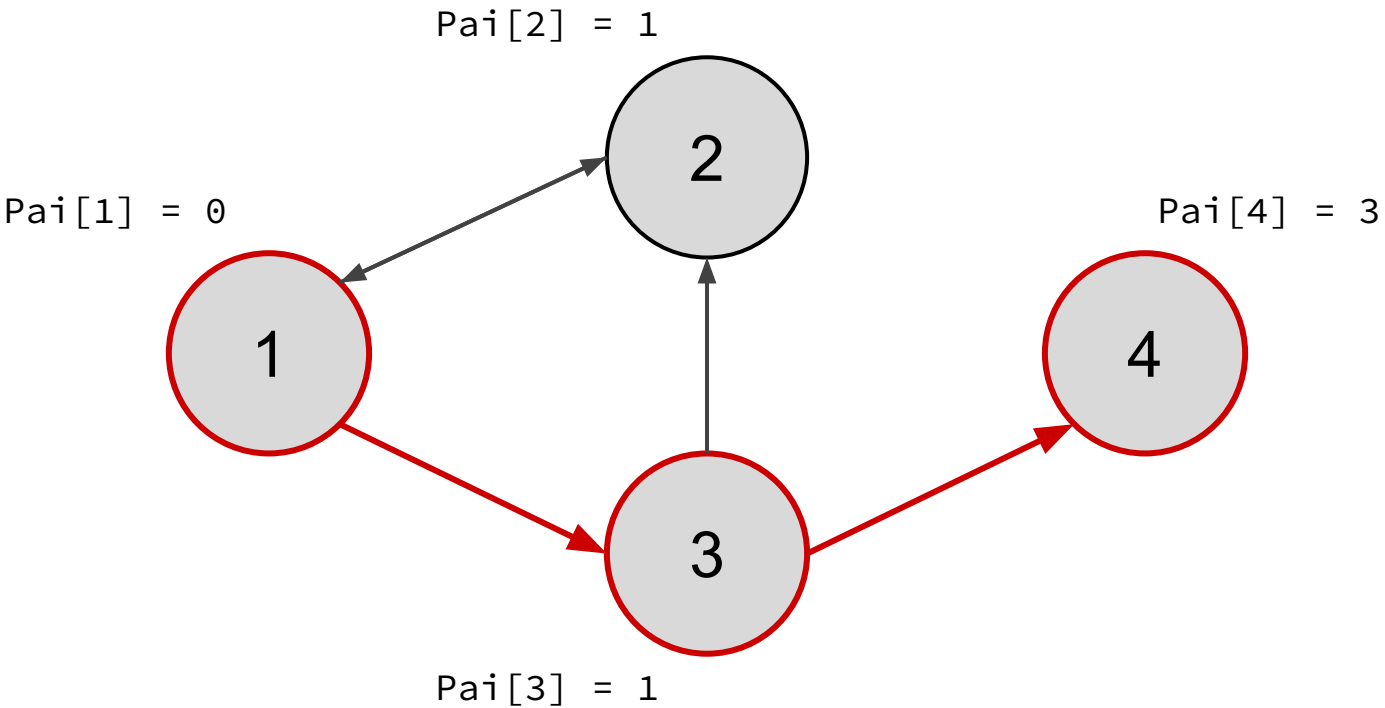


B - Find the Treasure

Fila: 4



B - Find the Treasure



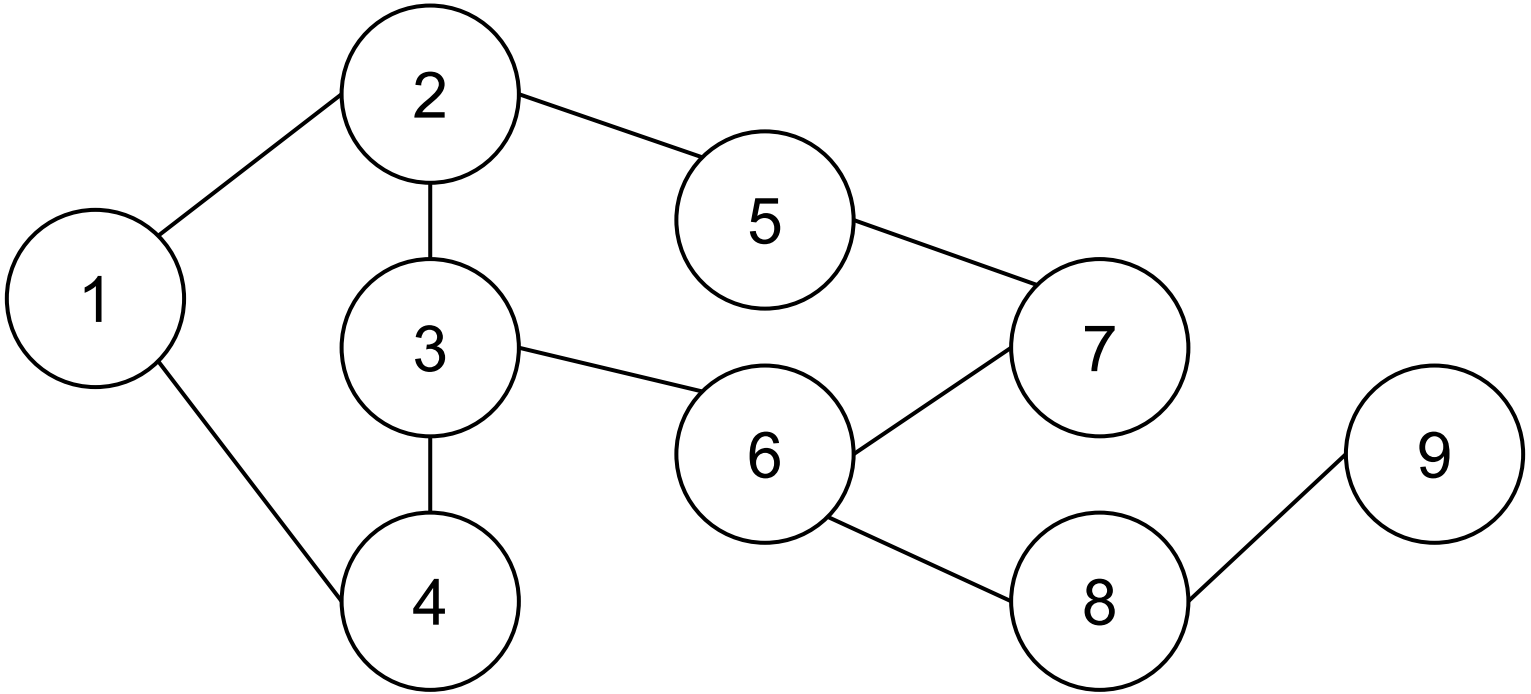
BFS partindo de múltiplas fontes

- Seu funcionamento é exatamente igual ao da BFS.

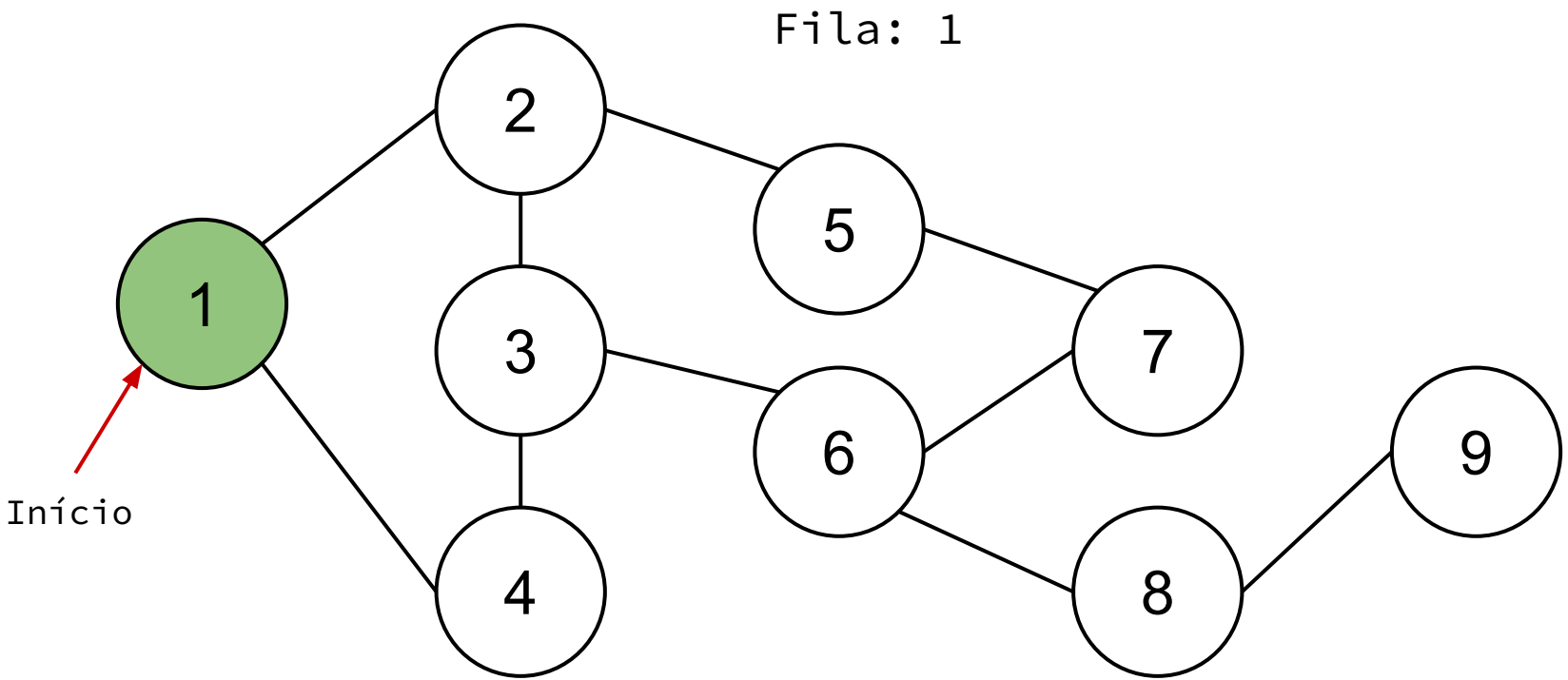
- No entanto, ao invés de começar em um único nó, **todos os vértices** que considerarmos como **ínicio** serão **inseridos na fila** de processamento.

- Útil para resolução de problemas que requerem o cálculo da **mínima distância** ou **custo** para se chegar a um vértice.

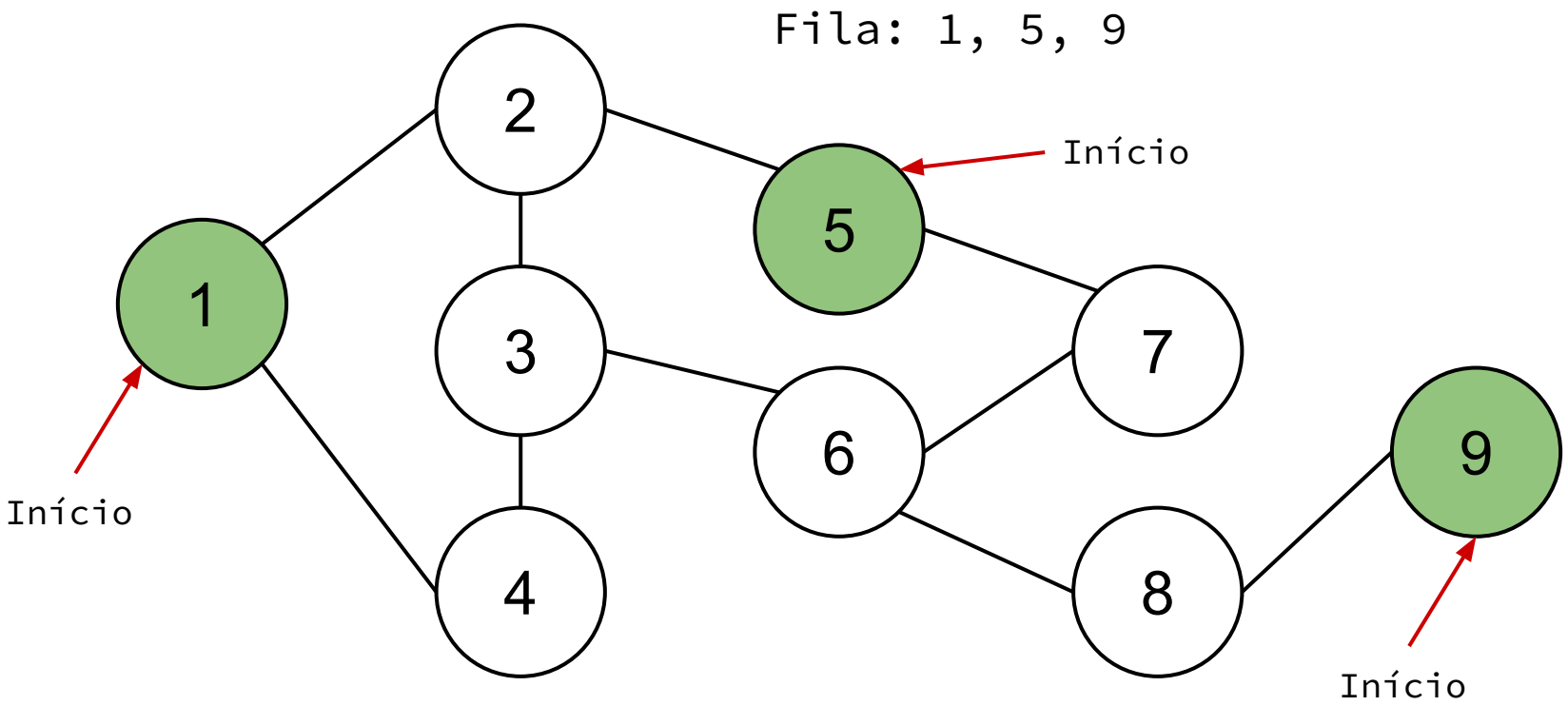
BFS partindo de múltiplas fontes



BFS partindo de múltiplas fontes

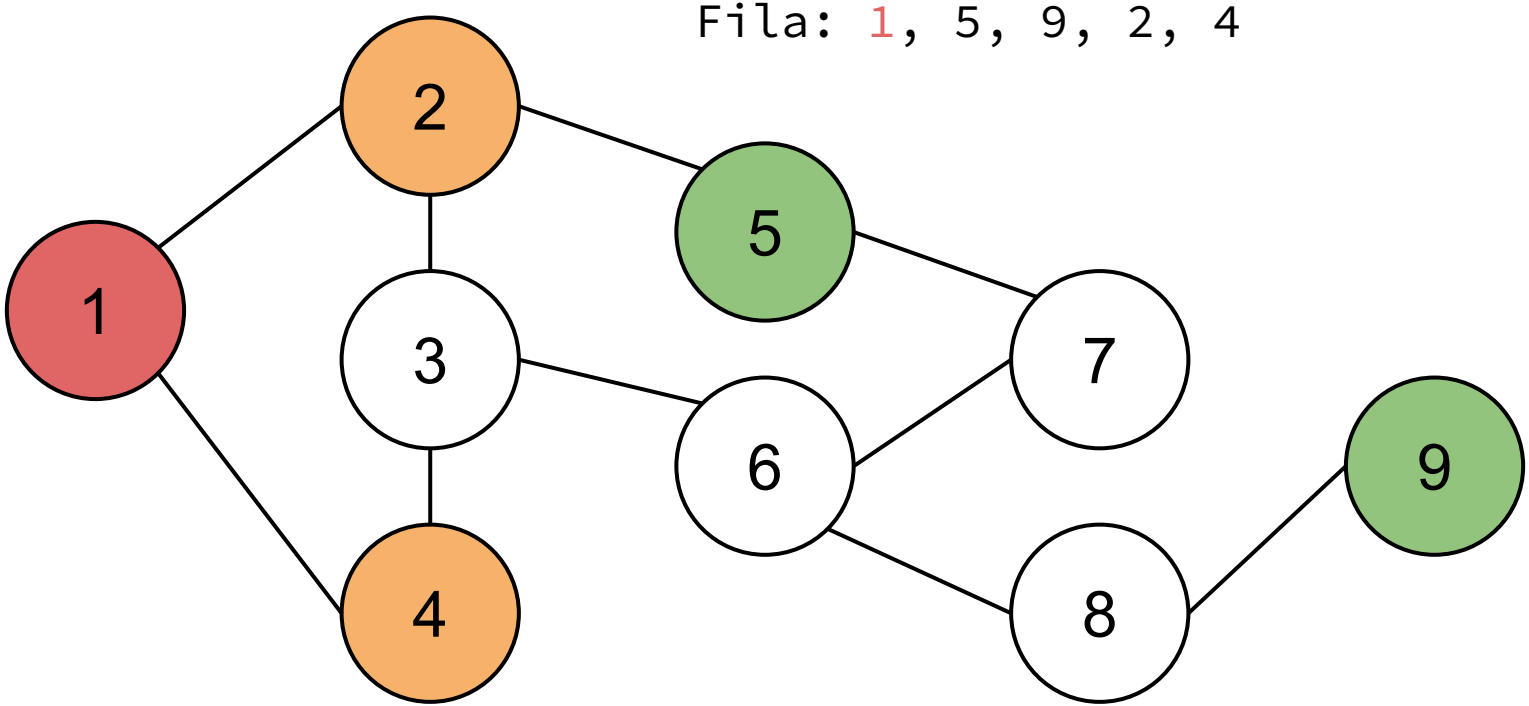


BFS partindo de múltiplas fontes



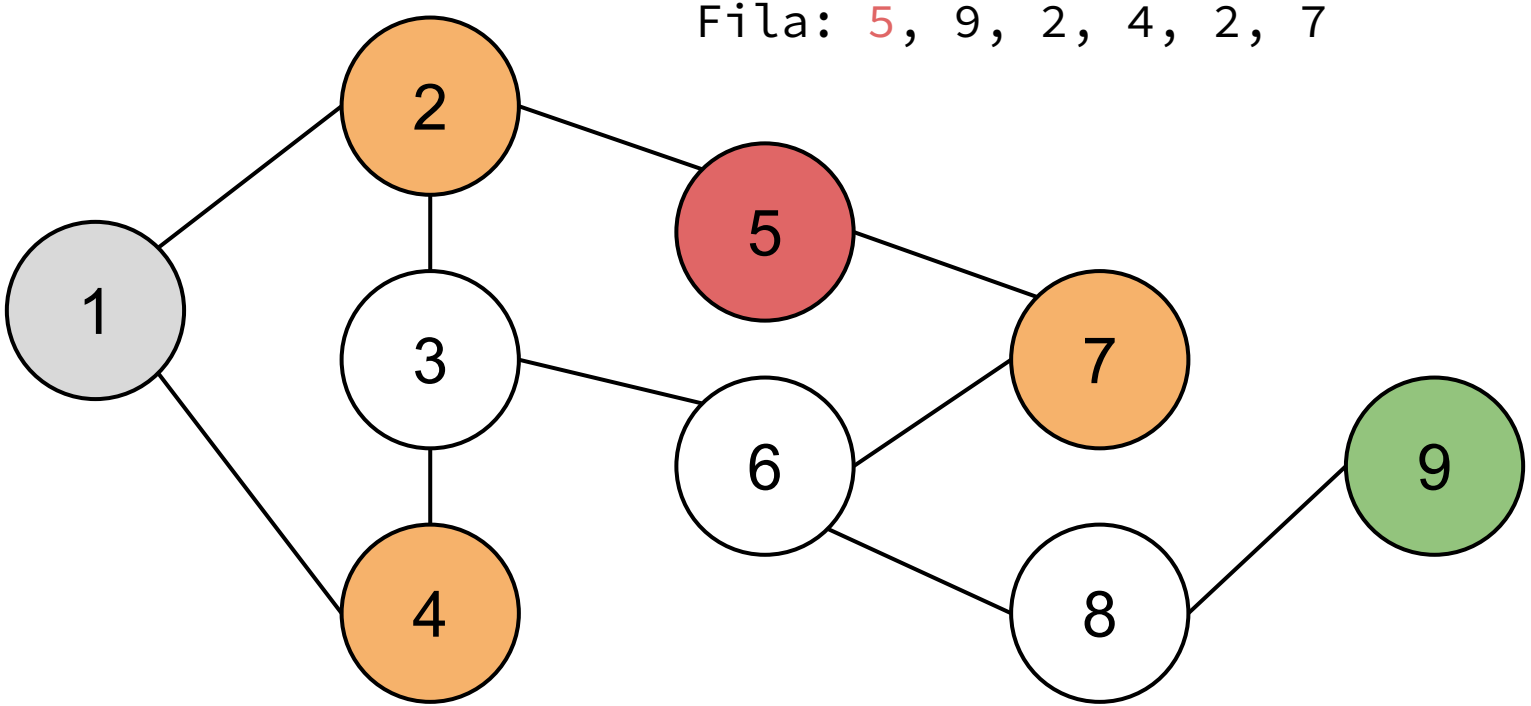
BFS partindo de múltiplas fontes

Fila: 1, 5, 9, 2, 4



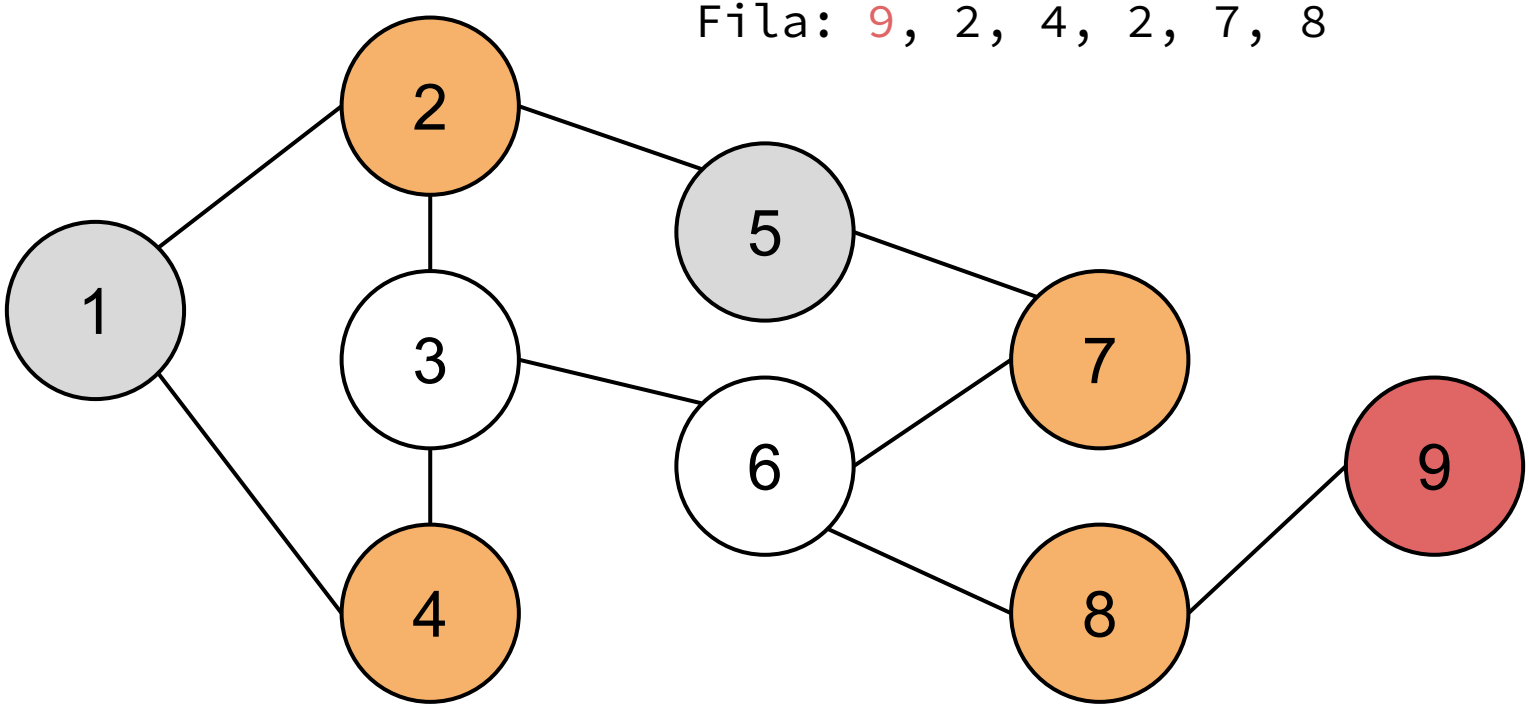
BFS partindo de múltiplas fontes

Fila: 5, 9, 2, 4, 2, 7



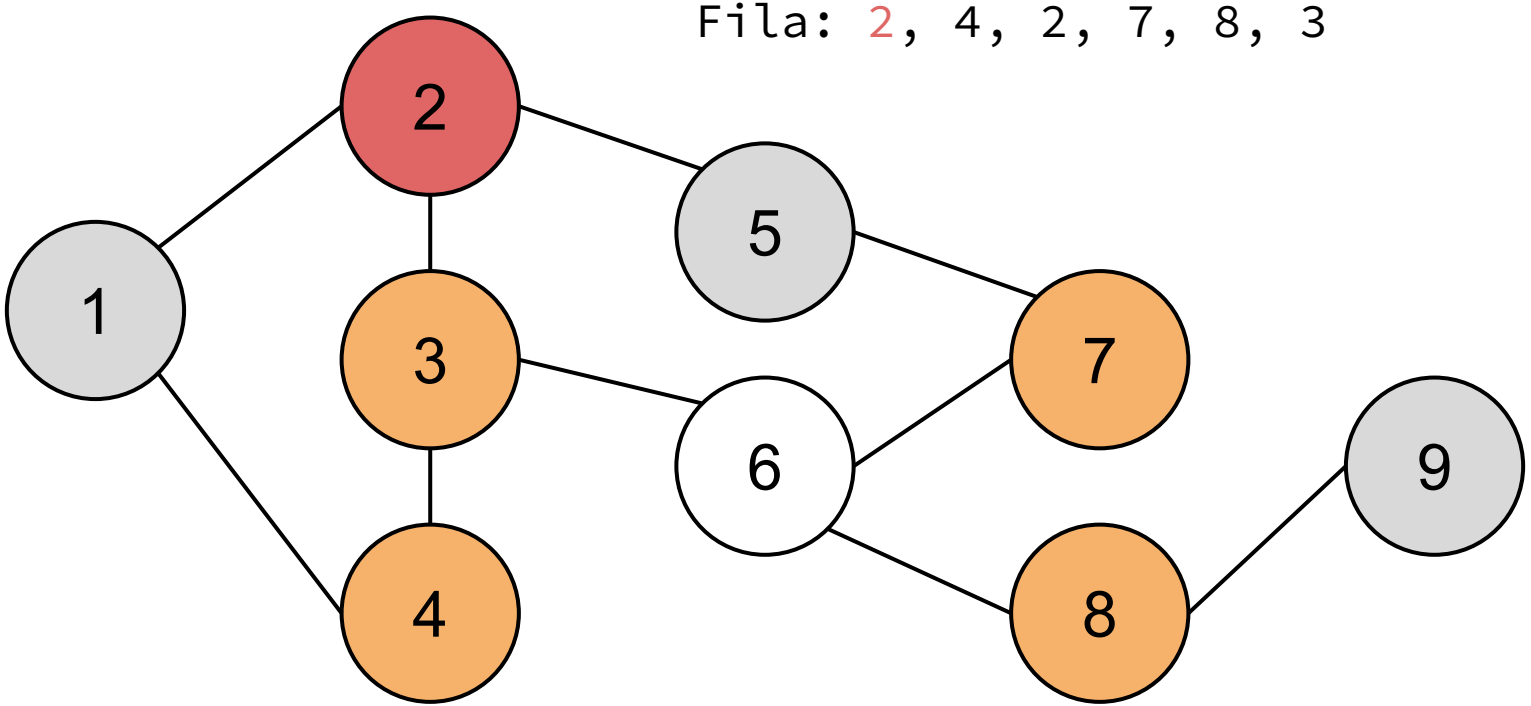
BFS partindo de múltiplas fontes

Fila: 9, 2, 4, 2, 7, 8



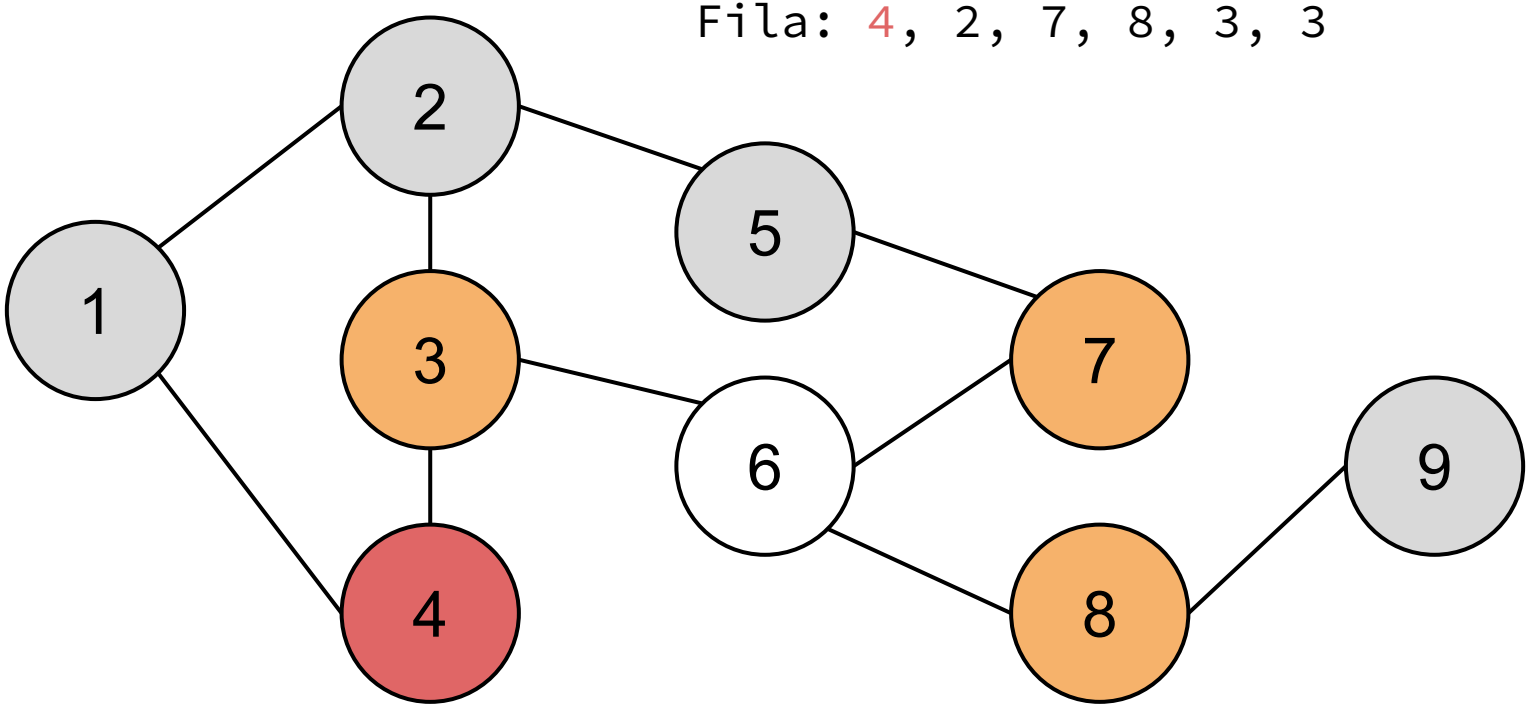
BFS partindo de múltiplas fontes

Fila: 2, 4, 2, 7, 8, 3



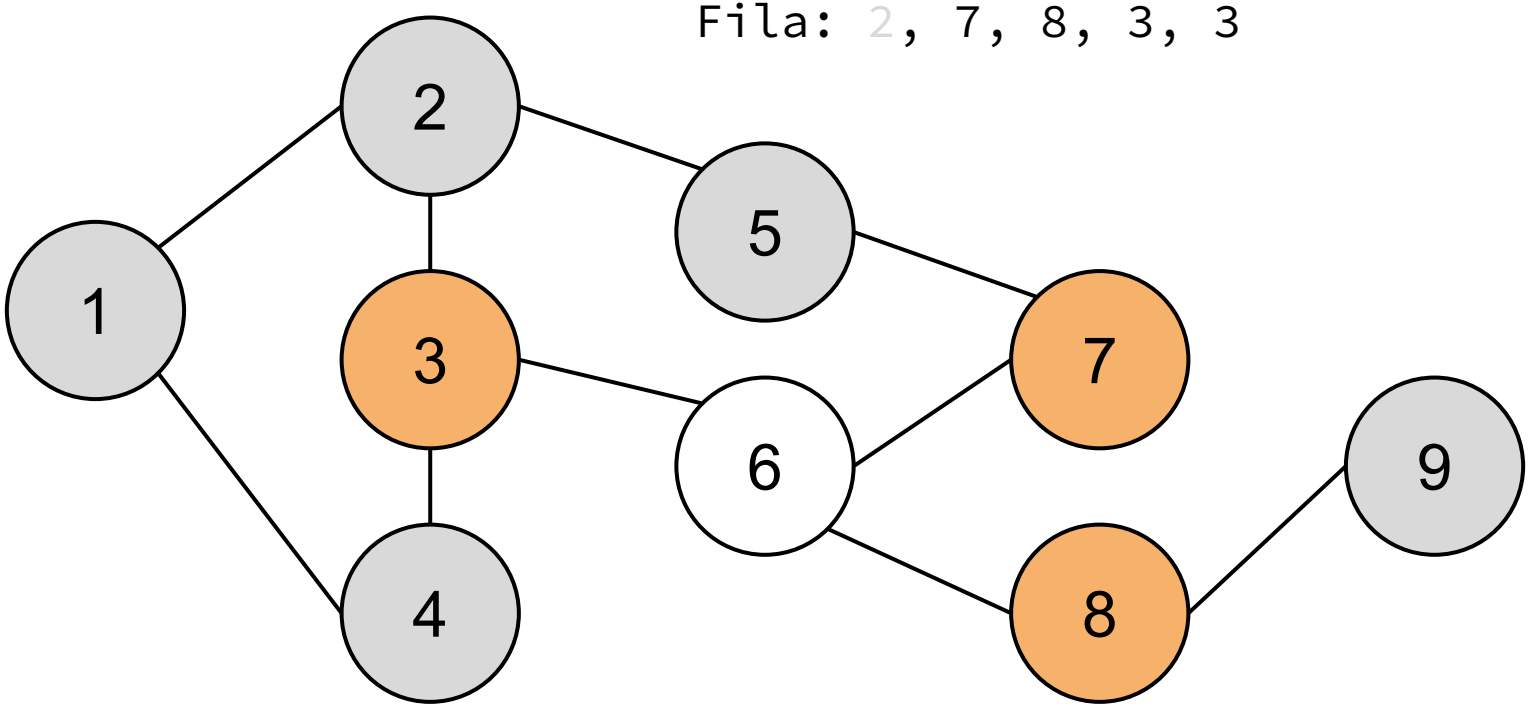
BFS partindo de múltiplas fontes

Fila: 4, 2, 7, 8, 3, 3



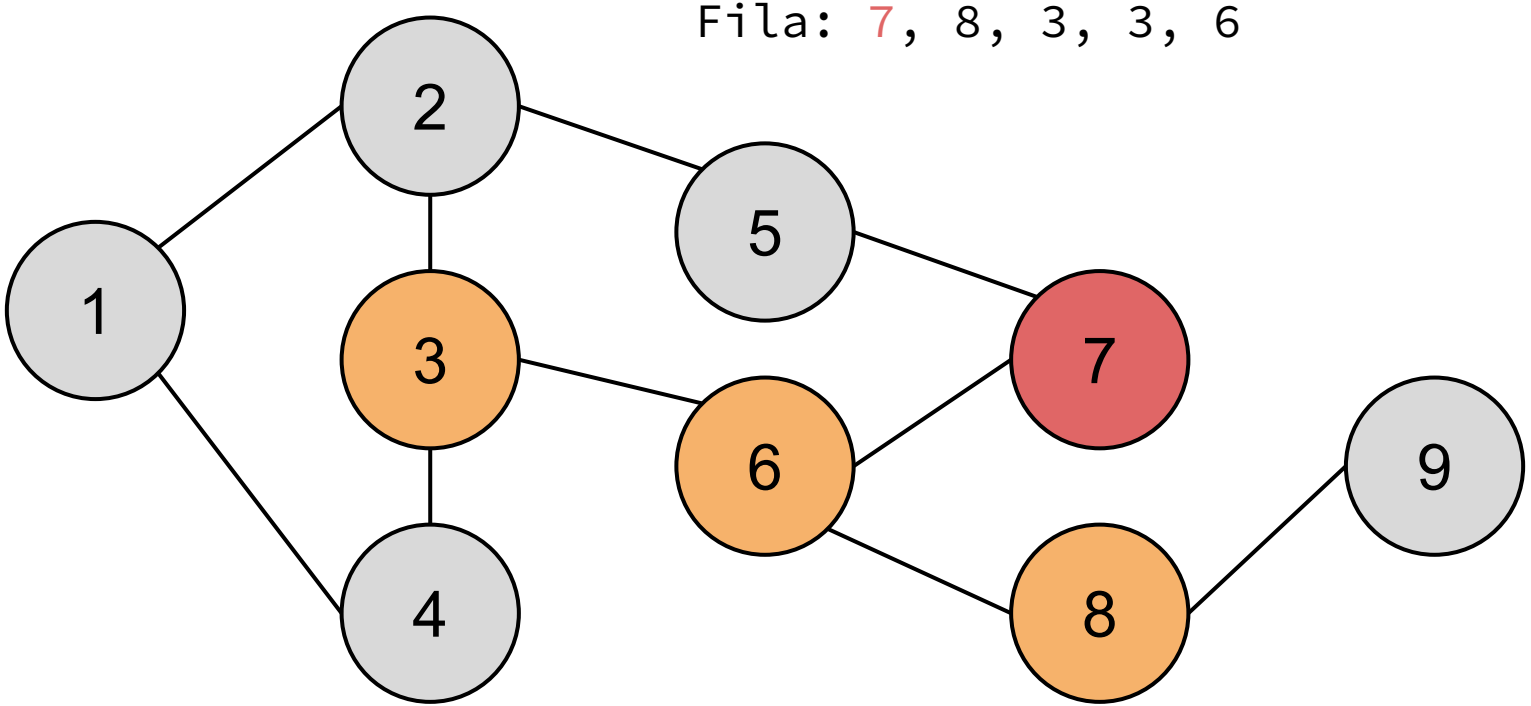
BFS partindo de múltiplas fontes

Fila: 2, 7, 8, 3, 3



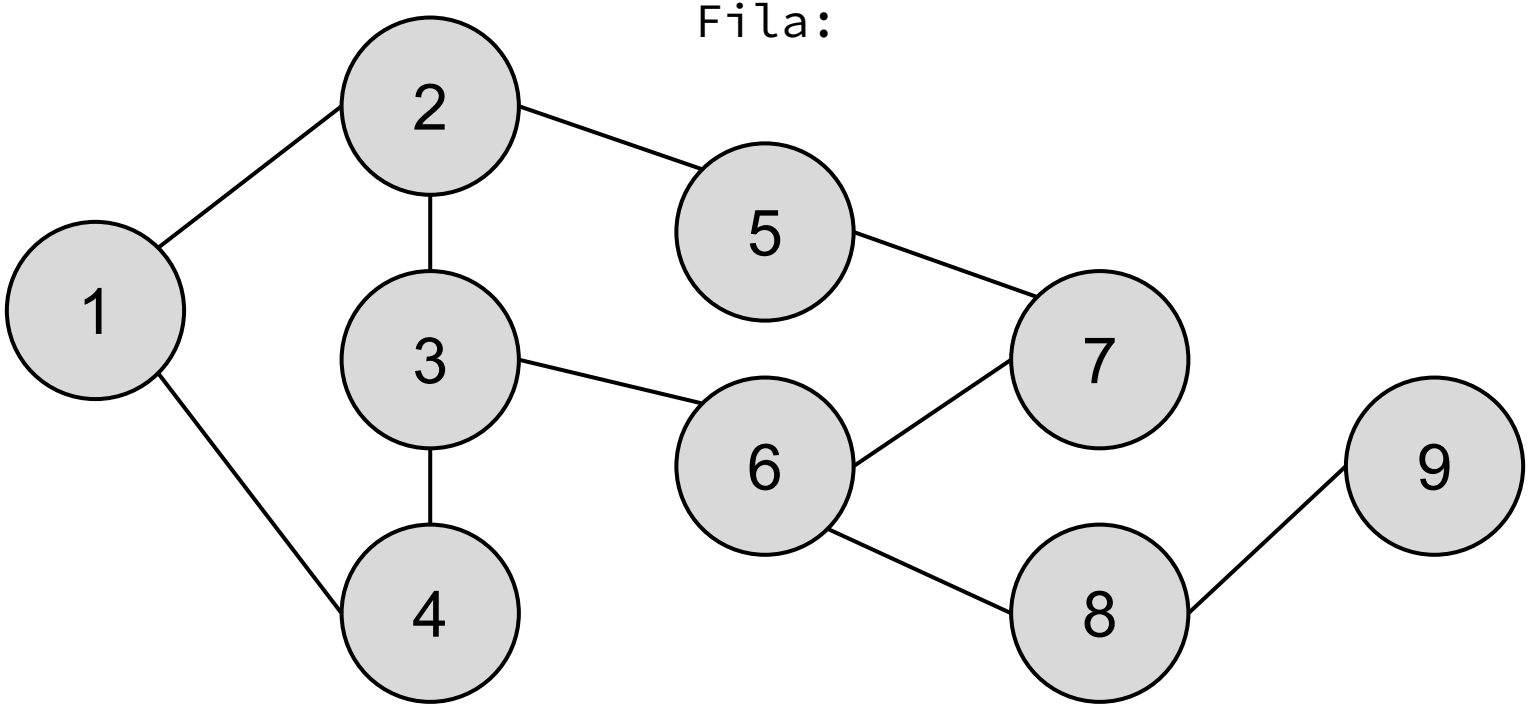
BFS partindo de múltiplas fontes

Fila: 7, 8, 3, 3, 6



BFS partindo de múltiplas fontes

Fila:



BFS partindo de múltiplas fontes

```
void bfs(const vector<int> &sources) {
    queue<int> q;
    vector<int> dist;

    for (auto s : sources) {
        dist[s] = 0;
        q.push(s);
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto v : adj[u]) {
            if (dist[u] + 1 < dist[v]) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

BFS partindo de múltiplas fontes

```
void bfs(const vector<int> &sources) {
    queue<int> q;
    vector<int> dist;

    for (auto s : sources) {
        dist[s] = 0;
        q.push(s);
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto v : adj[u]) {
            if (dist[u] + 1 < dist[v]) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

Vetor com todos os vértices de ínicio

BFS partindo de múltiplas fontes

```
void bfs(const vector<int> &sources) {
    queue<int> q;
    vector<int> dist;

    for (auto s : sources) {
        dist[s] = 0;
        q.push(s);
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto v : adj[u]) {
            if (dist[u] + 1 < dist[v]) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

Vetor com todos os vértices de ínicio

Colocar cada fonte na fila e iniciar suas informações adequadamente para o problema

BFS partindo de múltiplas fontes

```
void bfs(const vector<int> &sources) {
    queue<int> q;
    vector<int> dist;

    for (auto s : sources) {
        dist[s] = 0;
        q.push(s);
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto v : adj[u]) {
            if (dist[u] + 1 < dist[v]) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

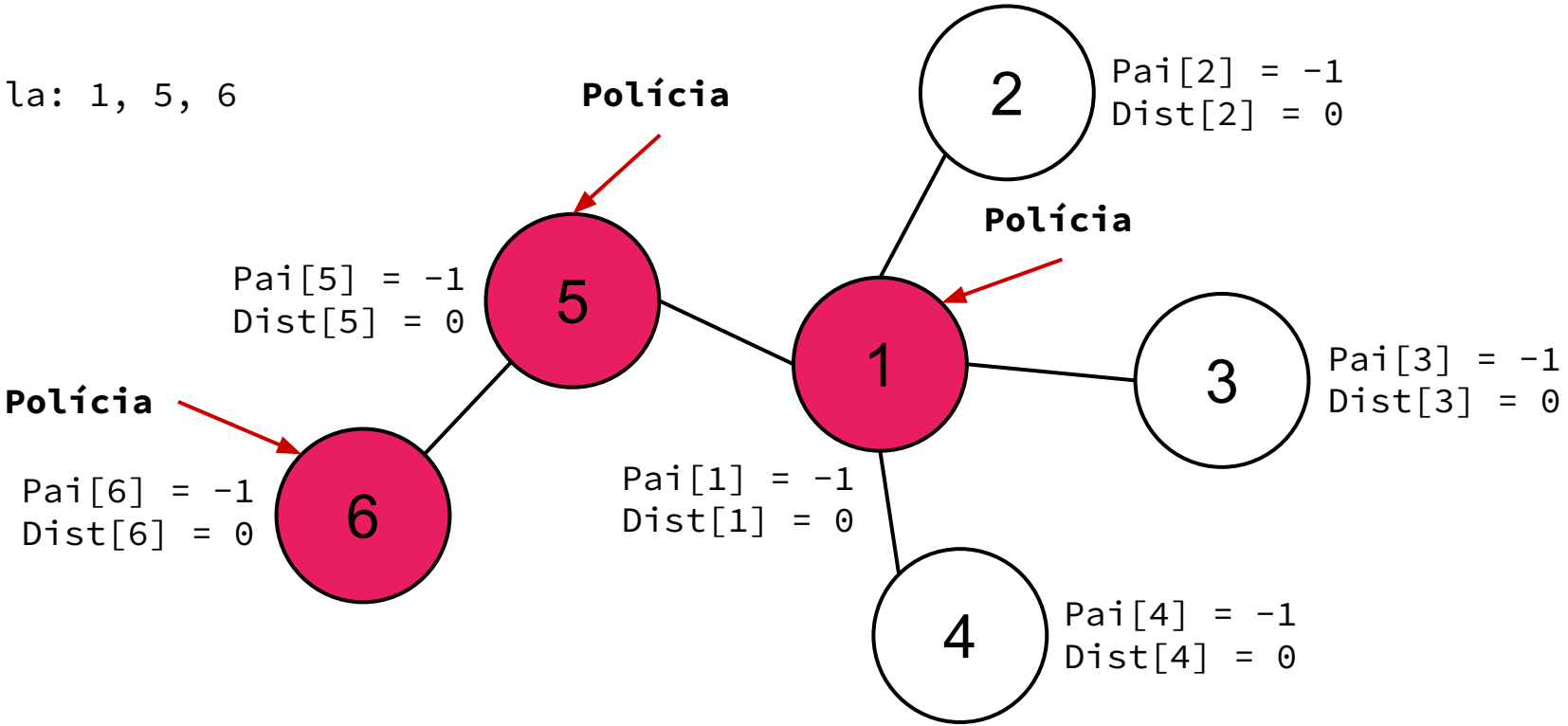
Vetor com todos os vértices de ínicio

Colocar cada fonte na fila e iniciar suas informações adequadamente para o problema

Processar cada vértice visitado da mesma forma que uma BFS normal

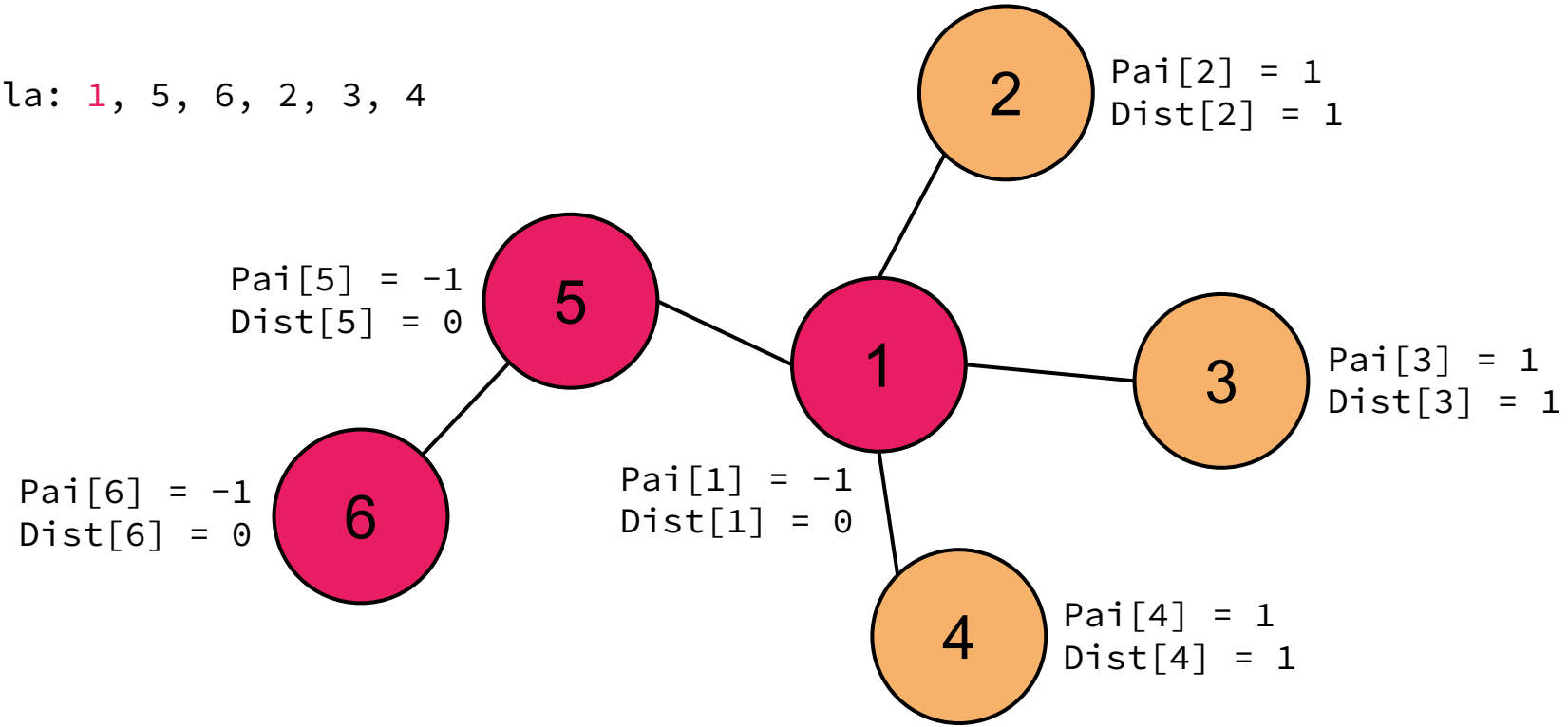
D - Police Stations

Fila: 1, 5, 6



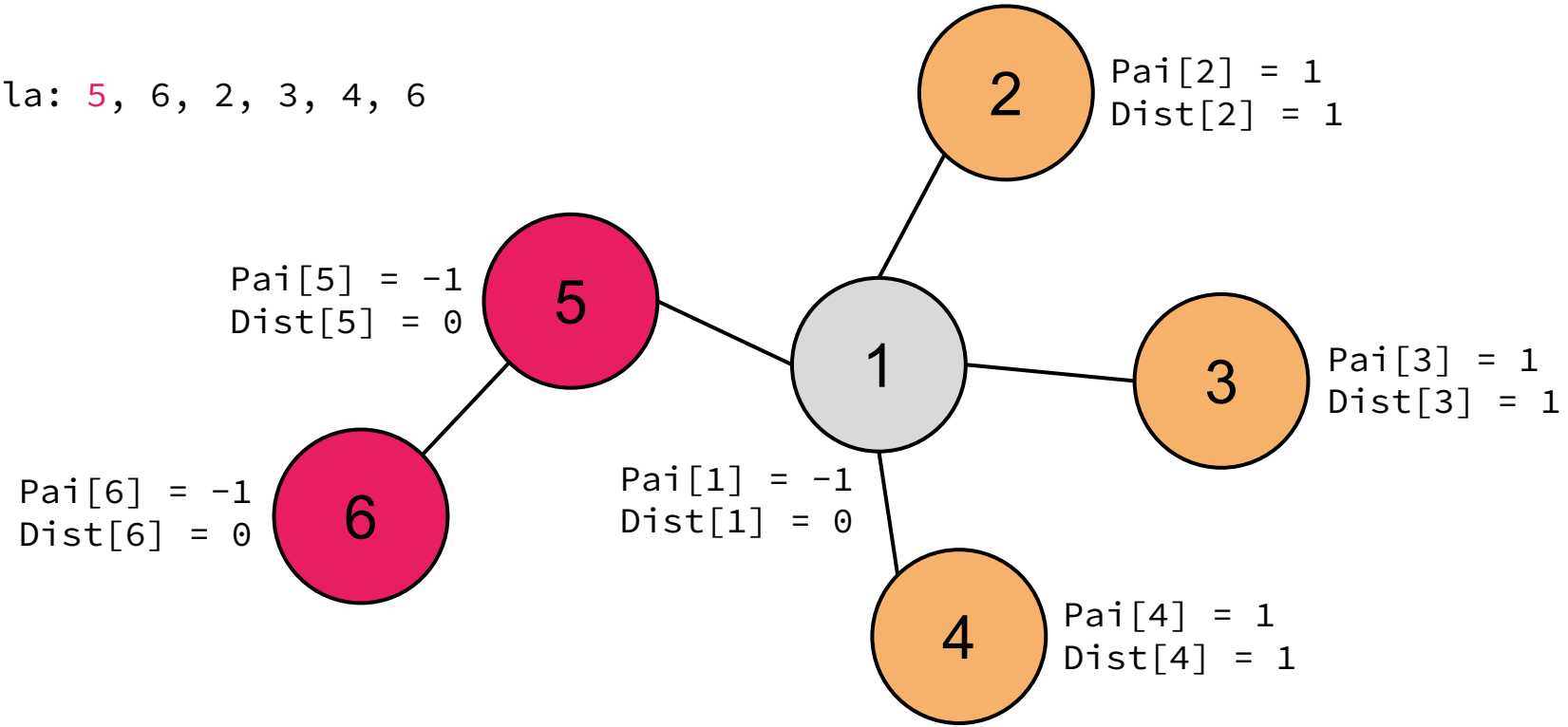
D - Police Stations

Fila: 1, 5, 6, 2, 3, 4



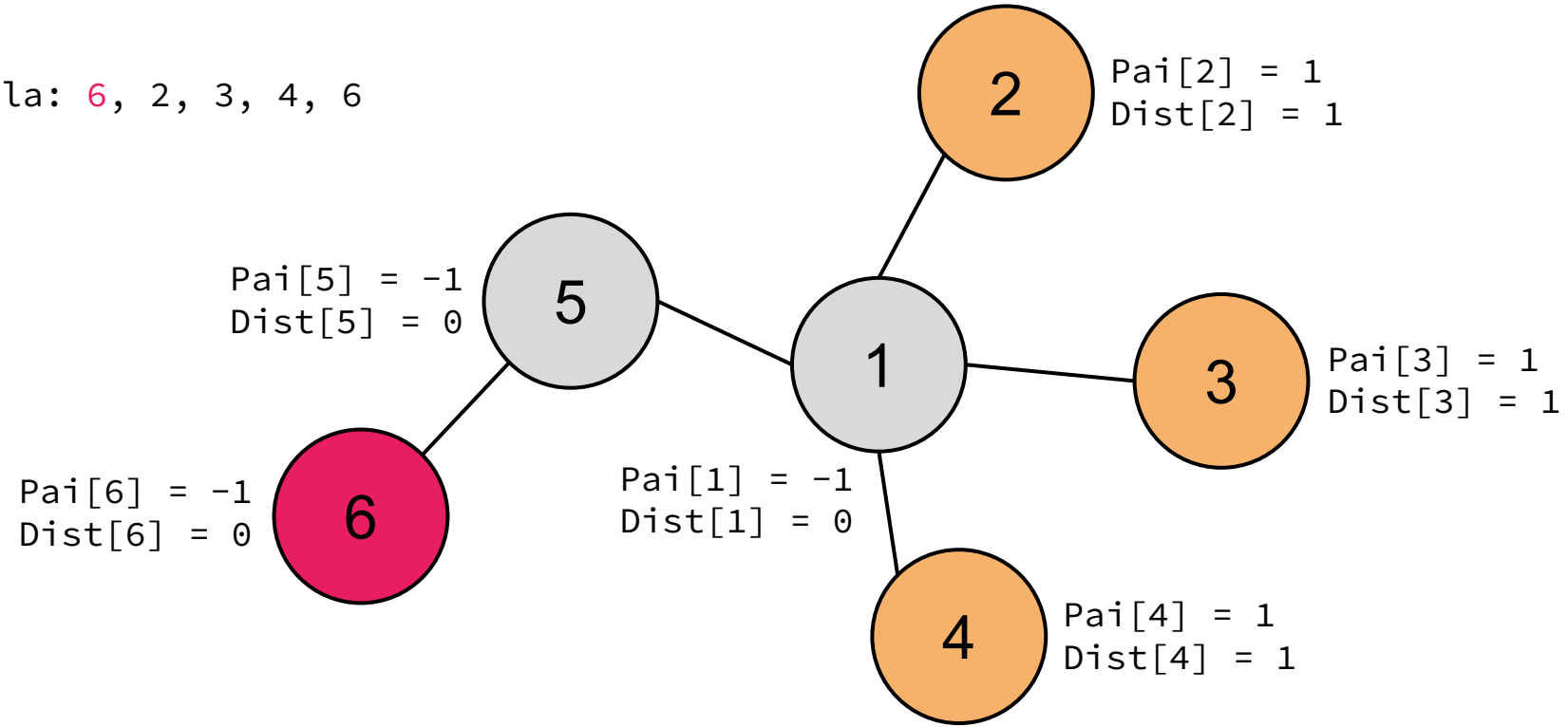
D - Police Stations

Fila: 5, 6, 2, 3, 4, 6



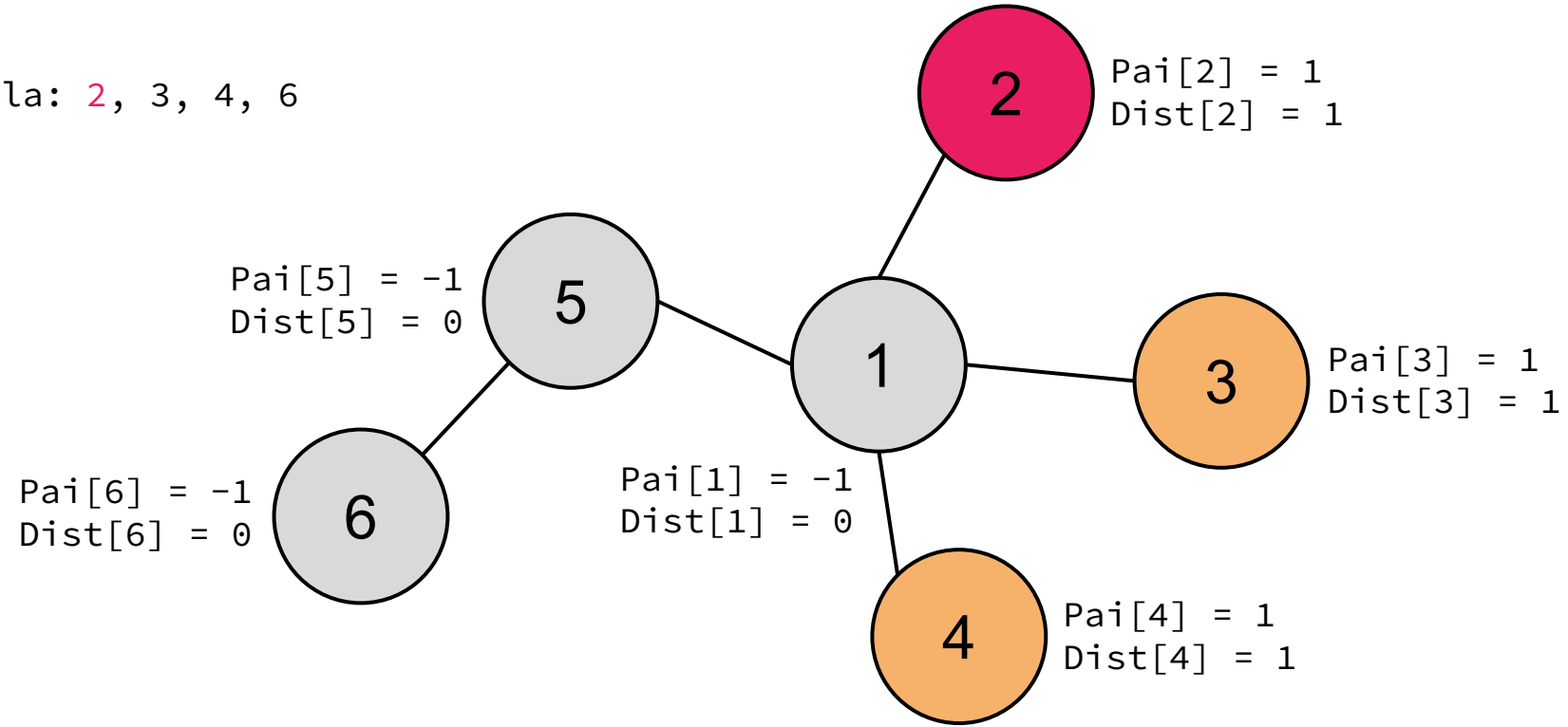
D - Police Stations

Fila: 6, 2, 3, 4, 6



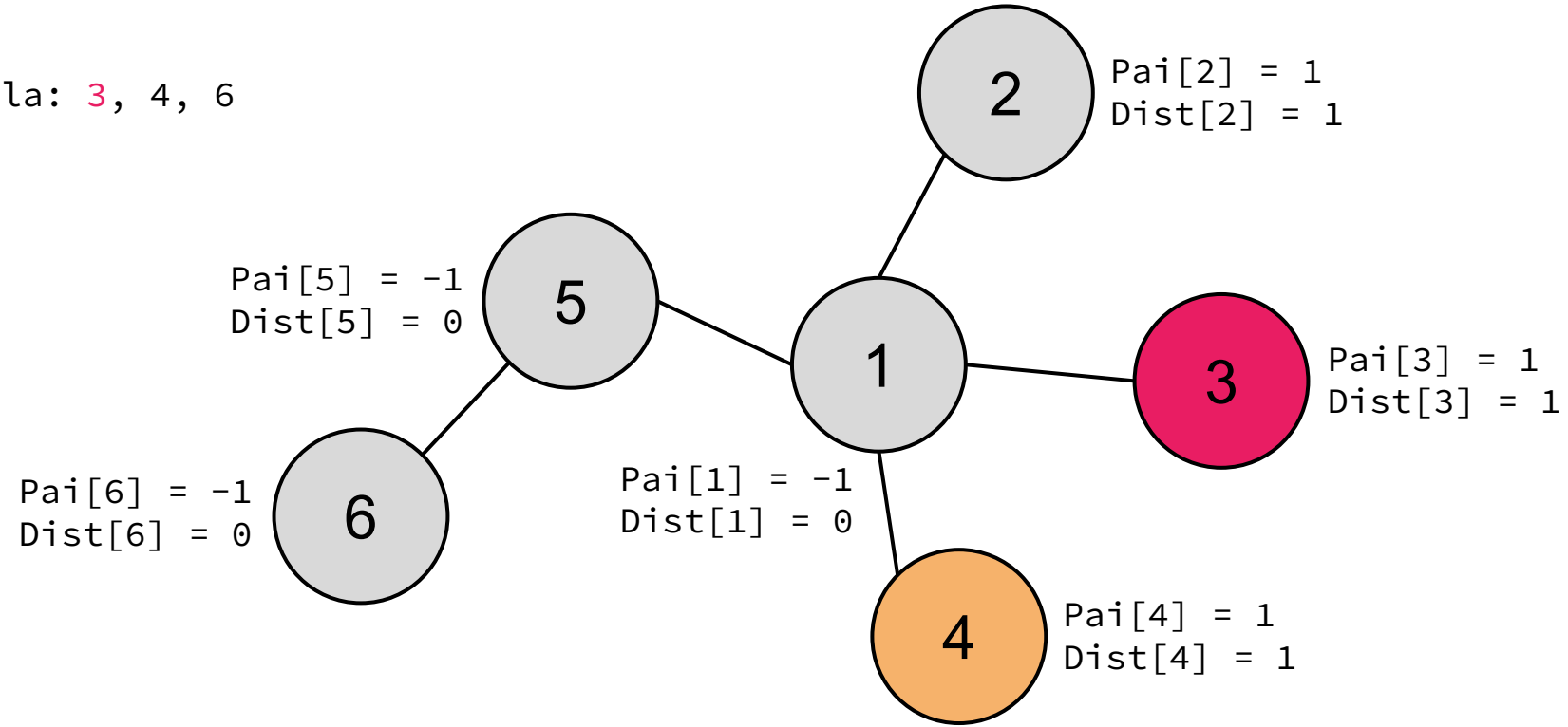
D - Police Stations

Fila: 2, 3, 4, 6



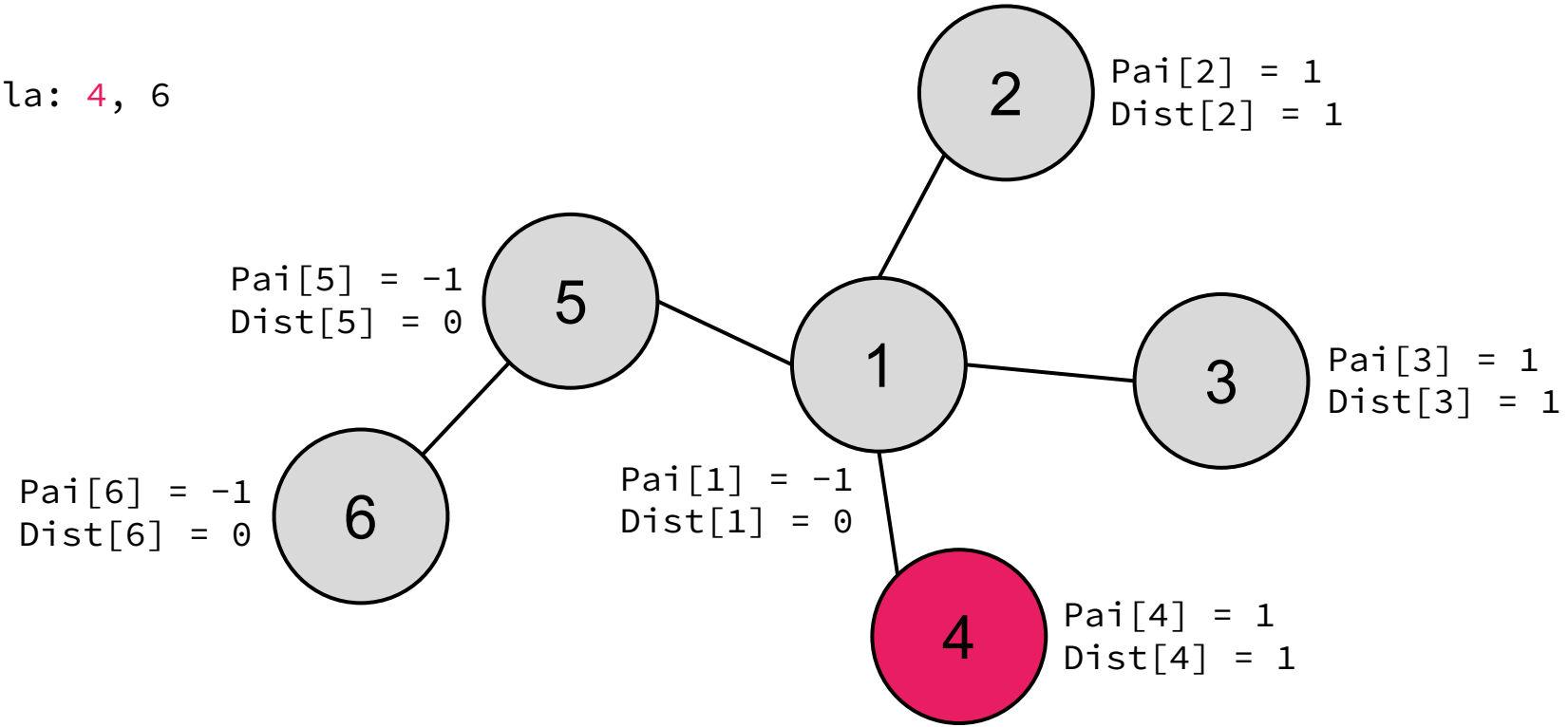
D - Police Stations

Fila: 3, 4, 6

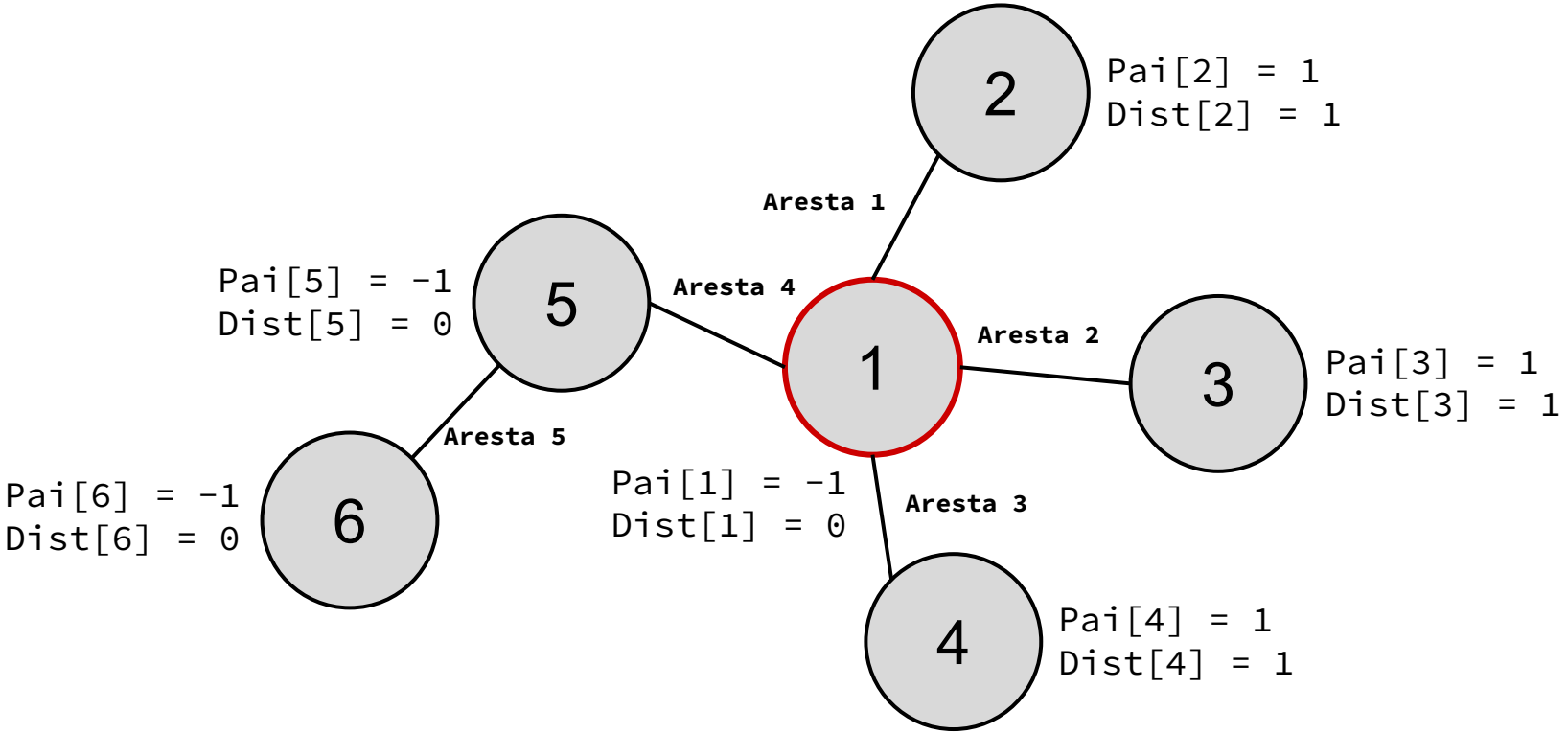


D - Police Stations

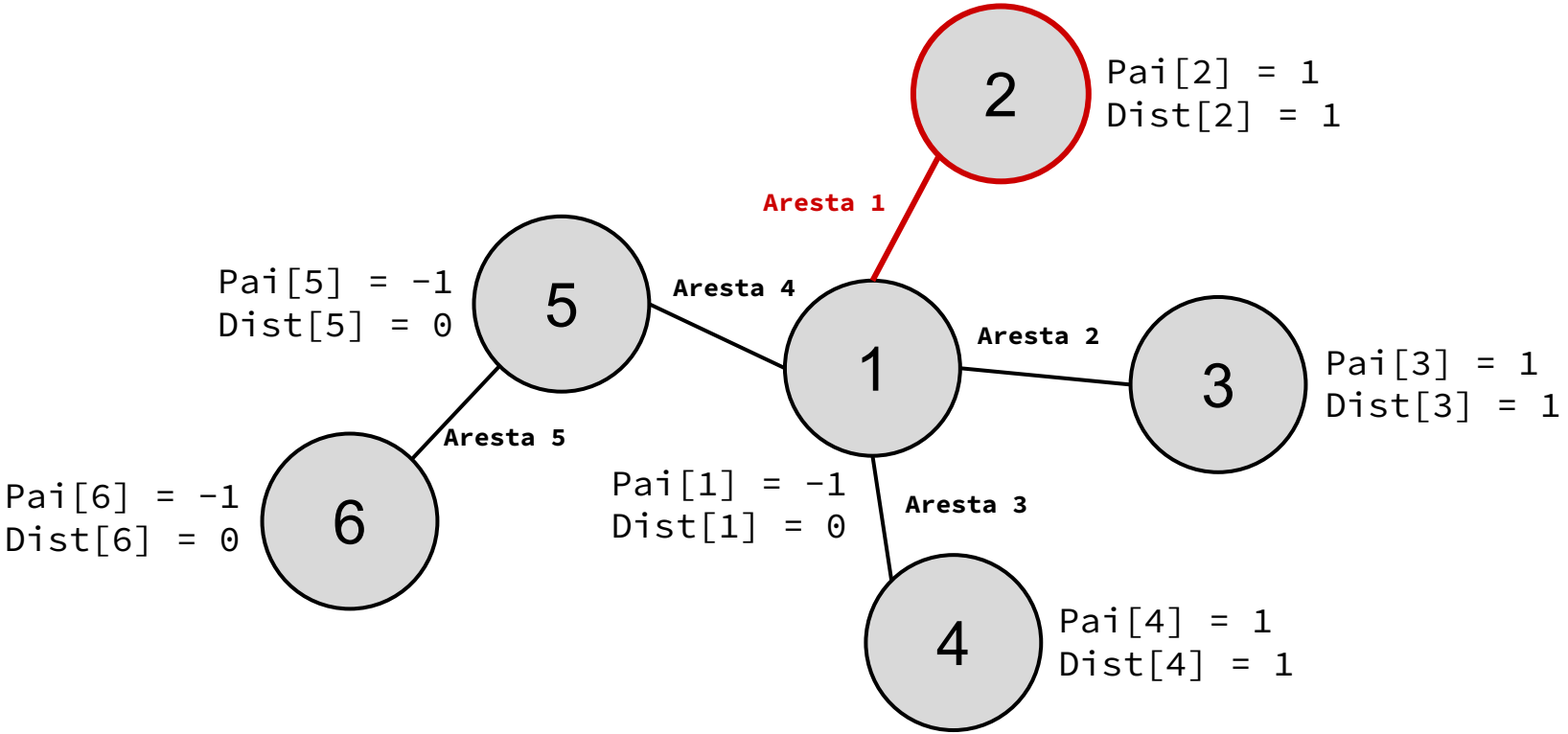
Fila: 4, 6



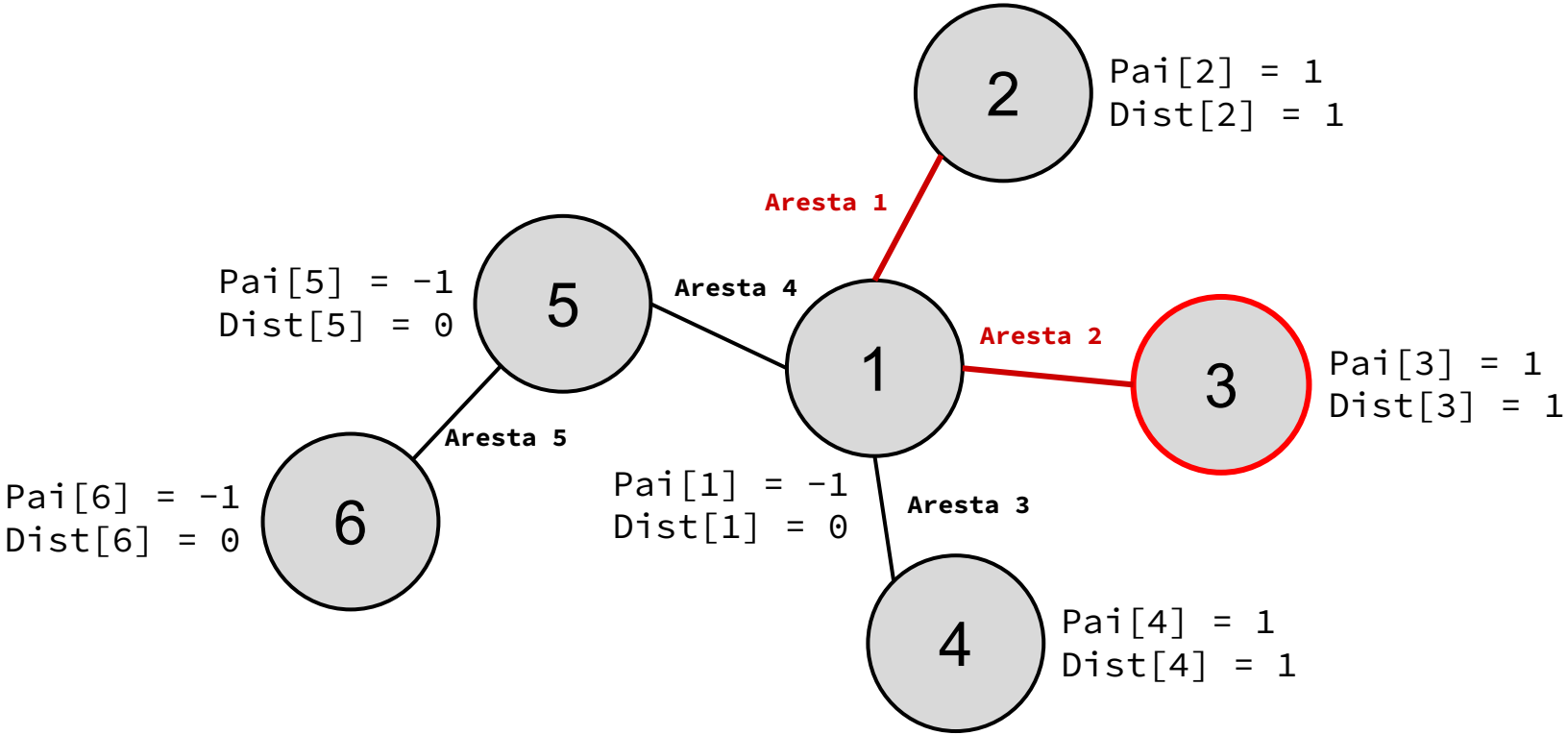
D - Police Stations



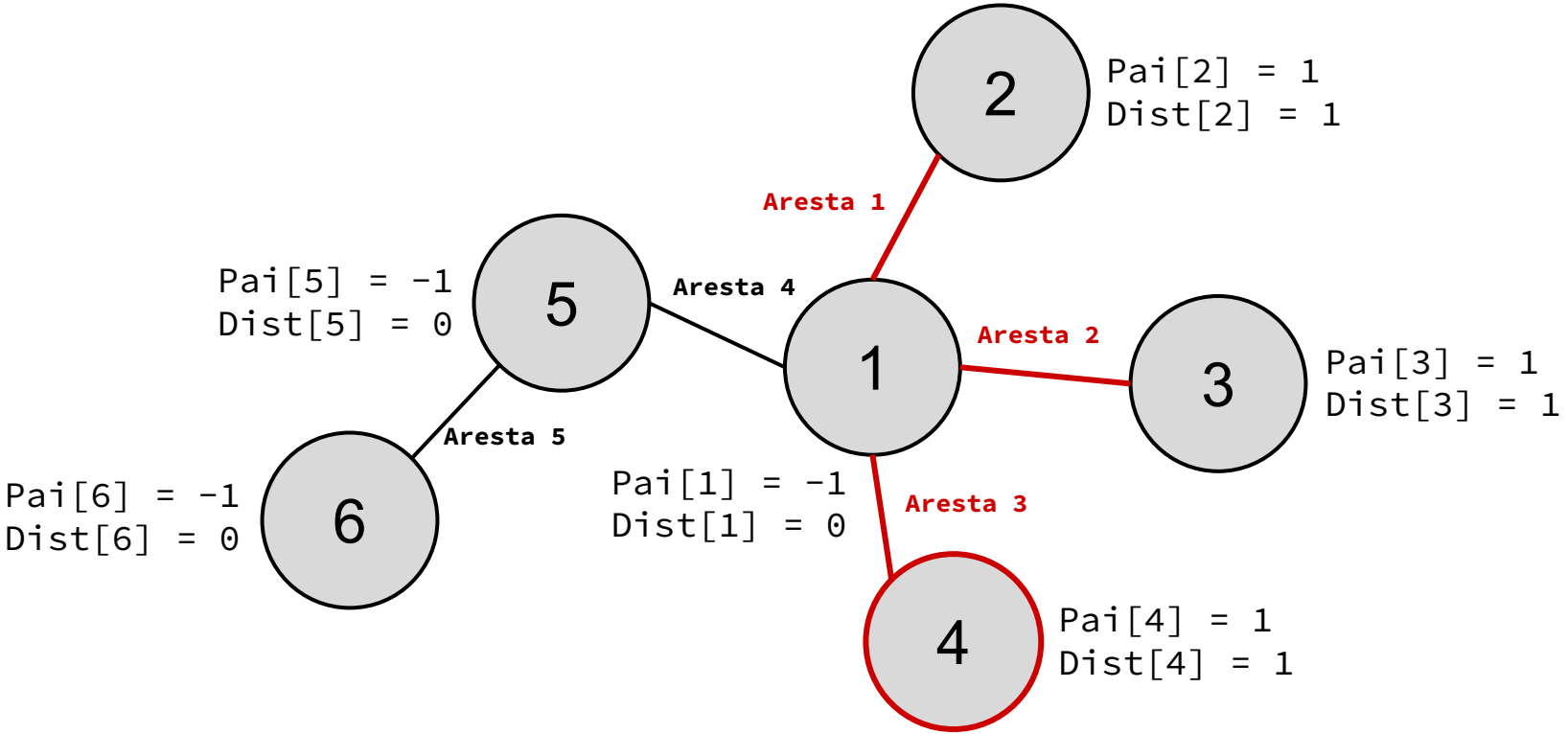
D - Police Stations



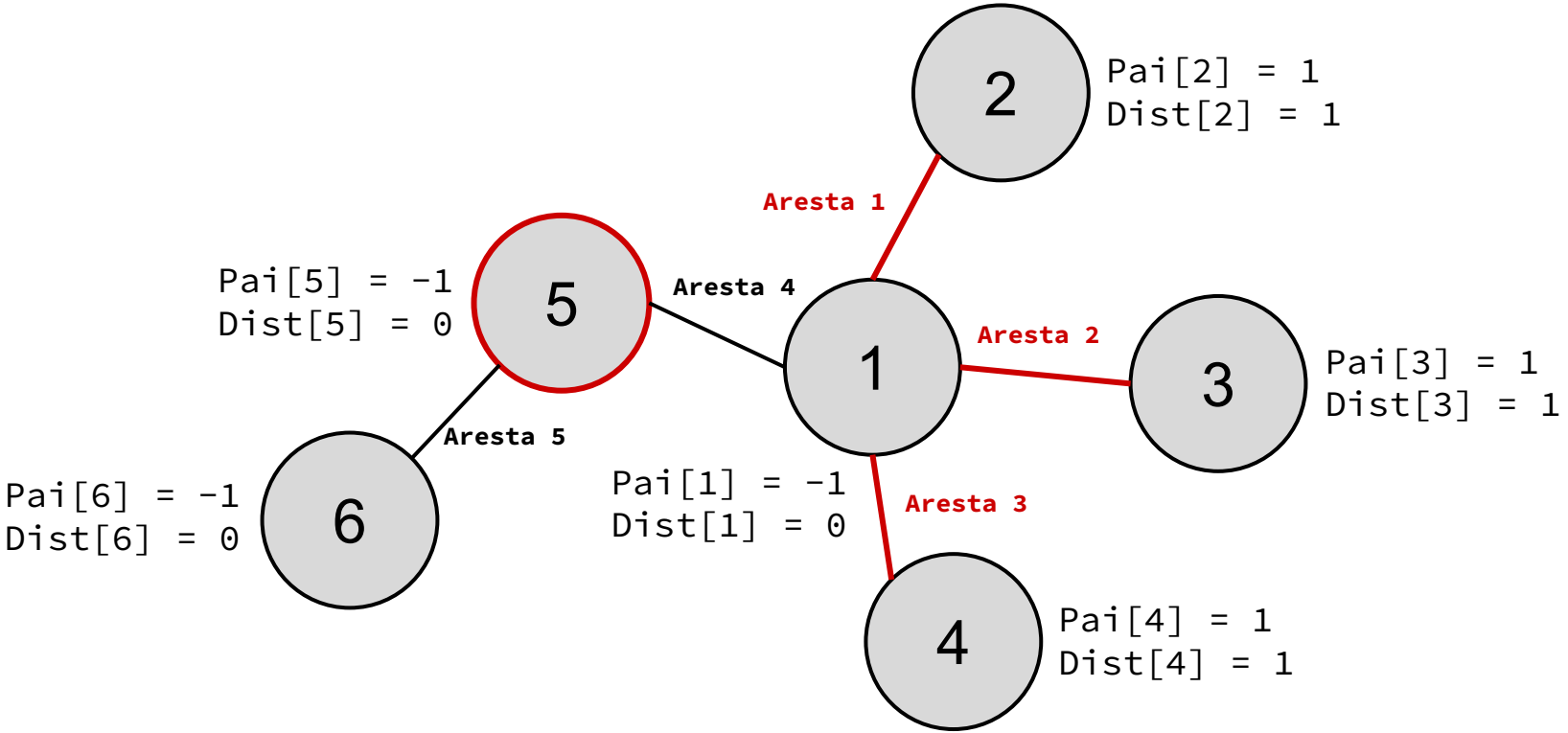
D - Police Stations



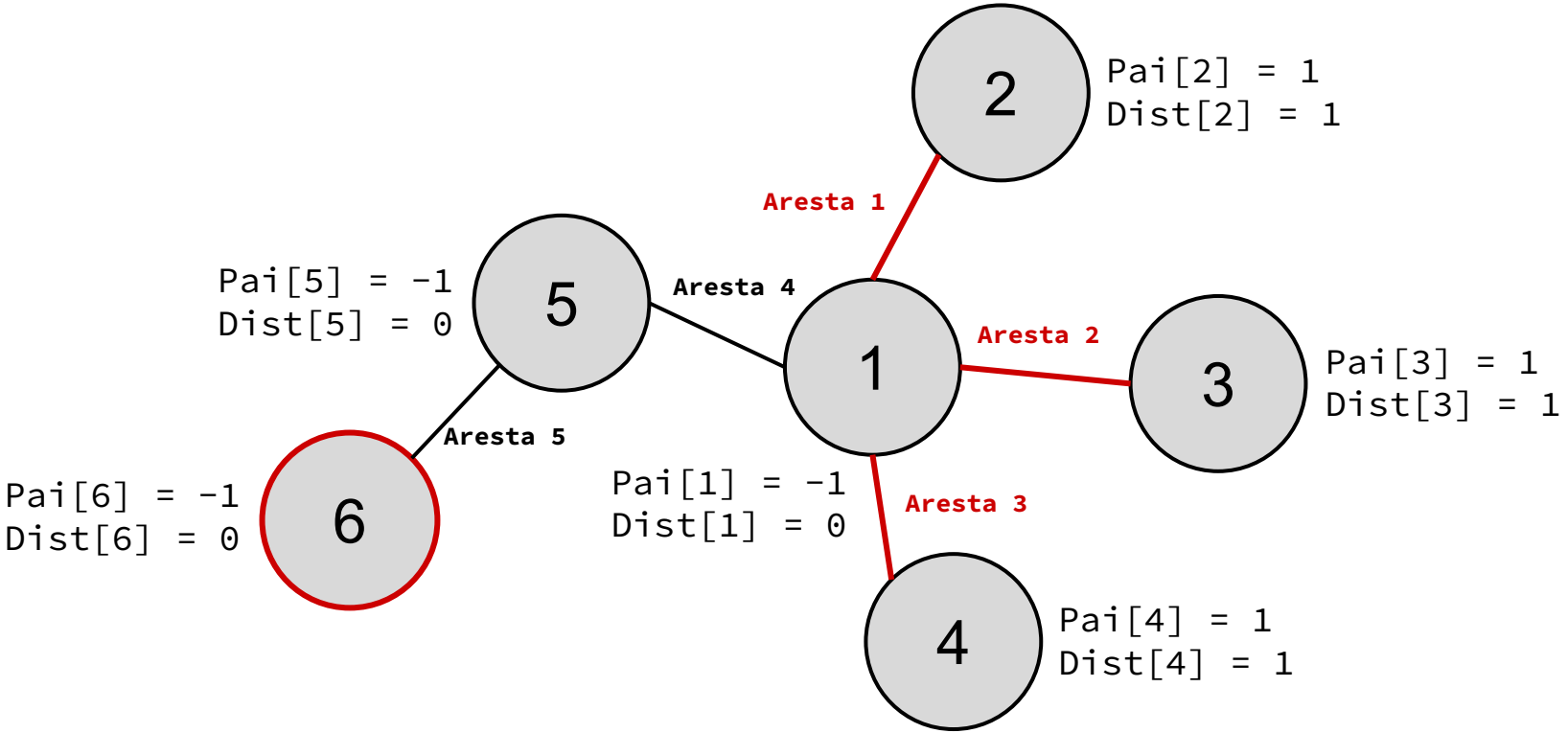
D - Police Stations



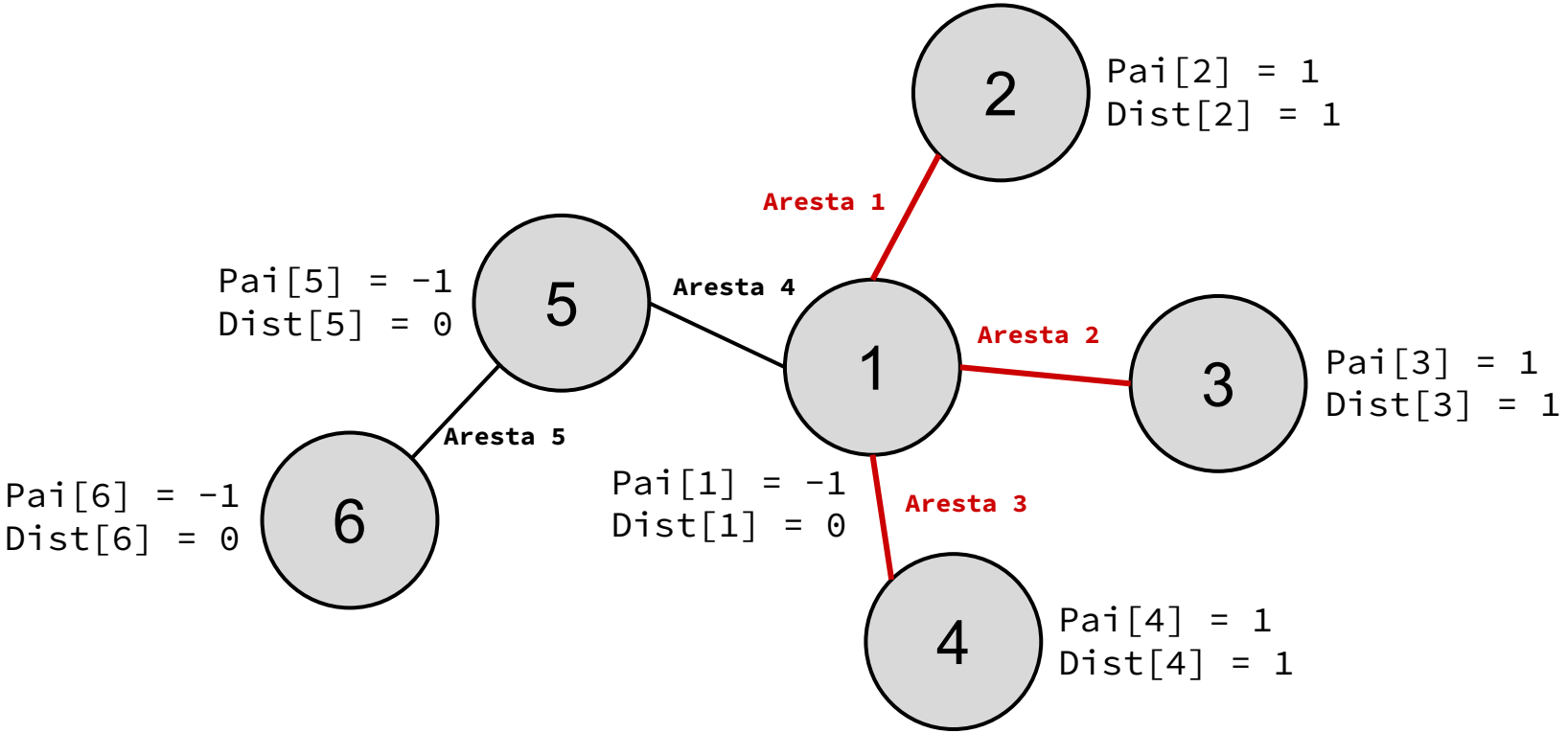
D - Police Stations



D - Police Stations

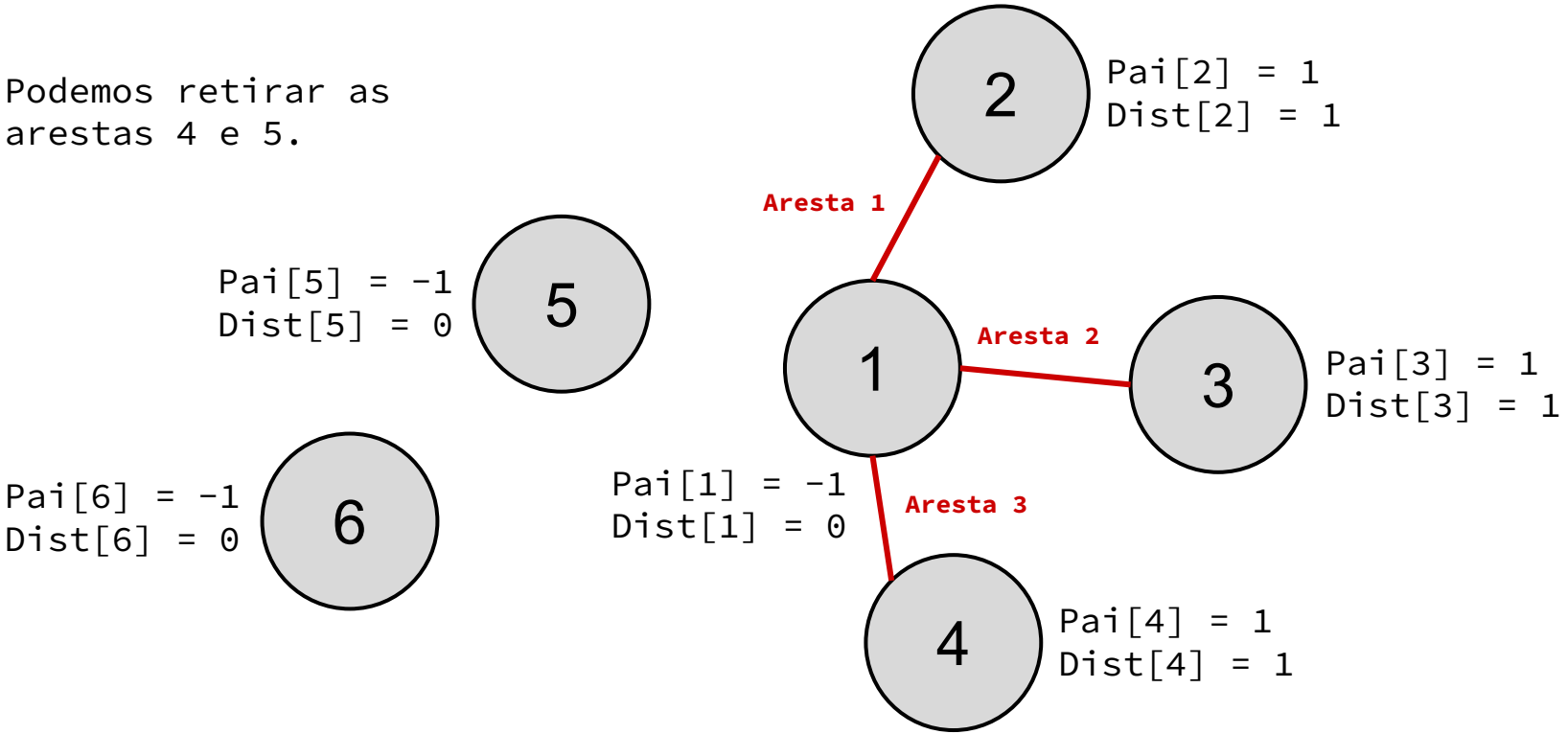


D - Police Stations



D - Police Stations

Podemos retirar as arestas 4 e 5.



G - Jane and the Frost Giants

- Jane está presa em um labirinto que contém armadilhas de fogo, espalhando **uma célula** para o lado**,** nas direções **vertical** e **horizontal,** a **cada minuto**.

- Jane leva **um minuto** para andar para uma **célula adjacente.**

- Determinar o **tempo mínimo** para Jane escapar **por uma das bordas** do labirinto, **se possível**.

G - Jane and the Frost Giants

- **Estratégia: computar o tempo mínimo** que demora para uma **célula pegar fogo** a partir de **cada armadilha.**

- Após isso, **computar o tempo mínimo** que leva para **Jane alcançar uma célula adjacente** e **se é possível alcançá-la antes do fogo.**

- **Verificar** se ela consegue **chegar em alguma borda.**

G - Jane and the Frost Giants

– – –

#	#	●	#	#
#	J	F	F	#
#	●	●	●	#
#	●	●	●	#

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

G - Jane and the Frost Giants

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	∞	∞	∞
∞	∞	0	0	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

G - Jane and the Frost Giants

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	∞	1	∞	∞
∞	∞	∞	∞	∞

G - Jane and the Frost Giants

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	∞	1	1	∞
∞	∞	∞	∞	∞

G - Jane and the Frost Giants

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	2	1	1	∞
∞	∞	2	∞	∞

G - Jane and the Frost Giants

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	2	1	1	∞
∞	∞	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	2	1	1	∞
∞	3	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	2	1	1	∞
∞	3	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	1	0	0	∞
∞	2	1	1	∞
∞	3	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J	F	F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	0	0	0	∞
∞	2	1	1	∞
∞	3	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J		F	#
#	•	•	•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	0	0	0	∞
∞	1	1	1	∞
∞	3	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J		F	#
#	•		•	#
#	•	•	•	#

∞	∞	1	∞	∞
∞	0	0	0	∞
∞	1	1	1	∞
∞	2	2	2	∞

G - Jane and the Frost Giants

– – –

#	#	•	#	#
#	J		F	#
#	•		•	#
#	•		•	#

∞	∞	1	∞	∞
∞	0	0	0	∞
∞	1	1	1	∞
∞	2	2	2	∞

G - Jane and the Frost Giants

```
int main() {
    int T;
    cin >> T;

    for (int t = 1; t <= T; t++) {
        cin >> r >> c;

        grid = vector<string>(r + 1);

        for (int i = 0; i < r; i++)
            cin >> grid[i];

        vii fire, jane;

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                if (grid[i][j] == 'F')
                    fire.push_back({i, j});
                else if (grid[i][j] == 'J')
                    jane.push_back({i, j});
            }
        }

        int minutes;
        dist = vector<vi>(r + 1, vi(c + 1, INF));

        bfs(fire, false);
        minutes = bfs(jane, true);

        cout << "Case " << t << ": ";

        if (minutes != INF)
            cout << minutes + 1 << "\n";
        else
            cout << "IMPOSSIBLE\n";
    }

    return 0;
}
```

G - Jane and the Frost Giants

```
int bfs(const vii &sources, bool jane) {
    int minutes = INF;
    queue<ii> q;

    for (auto [i, j] : sources) {
        dist[i][j] = 0;
        q.push({i, j});
    }

    while (!q.empty()) {
        int i, j;
        tie(i, j) = q.front();
        q.pop();

        if (jane && (i == 0 || i == r - 1 || j == 0 || j == c -
1))
            minutes = min(dist[i][j], minutes);

        for (auto [x, y] : dir) {
            int a, b;
            a = i + x;
            b = j + y;

            if (a >= 0 && a < r && b >= 0 && b < c && grid[a][b]
== '.' && dist[i][j] + 1 < dist[a][b]) {
                dist[a][b] = dist[i][j] + 1;
                q.push({a, b});
            }
        }
    }

    return minutes;
}
```