ESCOLA DE PRIMAVERA DA MARATONA SBC DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG

UFMG

UFV

UNIFEI

USP Universidade de São Paulo

UFU

APOIO:

beecrowd

SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

CAMINHO MÍNIMO COM DIJKSTRA

Por: *Eduardo Costa de Souza*

CONTEÚDOS

01 - PROBLEMA MOTIVADOR: ILHAS (OBI 2018)

Os moradores das Ilhas Brasileiras Ocidentais (IBO) são assíduos jogadores do mais recente jogo online, Magos e Guerreiros. Tão competitivas se tornaram as partidas de Magos e Guerreiros na IBO, que a empresa criadora do jogo decidiu instalar em uma das ilhas um servidor dedicado apenas aos jogadores da IBO.

A conexão de internet da IBO funciona através de um sistema de cabos de fibra ótica. Pares de ilhas são conectados por cabos, e cada cabo toma um certo tempo (chamado de ping) para comunicar informação entre as duas partes. Quando duas ilhas se comunicam através de uma série de cabos (portanto, através de ilhas intermediárias), o ping entre elas é a soma dos pings de cada cabo no caminho. A rede da IBO foi implementada por ótimos programadores e, portanto, um par de ilhas sempre se comunica através do caminho com menor ping possível.

01 - PROBLEMA MOTIVADOR: ILHAS (OBI 2018)

Dada a configuração da rede da IBO e a ilha em que a empresa deseja instalar o novo servidor, determine a diferença entre os pings da ilha com menor e maior pings até o servidor.

Exemplo de entrada	Exemplo de saída
4 5 2 1 5 1 3 4 2 3 6 4 2 8 3 4 12 1	9

Link:
https://judge.beecrowd.com/pt/problems/view/2784

01 - PROBLEMA MOTIVADOR: ILHAS (OBI 2018)

Pseudo código que resolve o problema:

1. Faça a entrada de *n*, *m*, do grafo não direcionado *g* em uma lista de adjacência e da ilha *s* onde será instalado o servidor
2. Defina *d* o vetor em que *d[v]* é o menor ping da ilha *v* à ilha *s* *(d[s] == 0)*
3. Defina *max_ping = max_element(d)*
4. Altere o valor de *d[s]* para *max_ping*, para desconsiderá-lo ao pegar o mínimo
5. Imprima *max_ping – min_element(d)*

01 - PROBLEMA MOTIVADOR: ILHAS (OBI 2018)

Pseudo código que resolve o problema:

1. Faça a entrada de *n*, *m*, do grafo não direcionado *g* em uma lista de adjacência e da ilha *s* onde será instalado o servidor
2. Defina *d* o vetor em que *d[v]* é o menor ping da ilha *v* à ilha *s* *(d[s] == 0)* *Como calcular d de maneira eficiente?*
3. Defina *max_ping = max_element(d)*
4. Altere o valor de *d[s]* para *max_ping*, para desconsiderá-lo ao pegar o mínimo
5. Imprima *max_ping – min_element(d)*

02 – ENTENDENDO O PROBLEMA

Dado um grafo $G = (V, A)$, definimos um caminho $c = (v_1, \ldots, v_k)$ em que $v_i \in V$ e $(v_i, v_{i+1}) \in A$. O peso do caminho é definido como:

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

Definimos o caminho mínimo de um vértice u até um vértice v como sendo:

$$\delta(u, v) = \begin{cases} min\left\{p(c) : u \overset{c}{\rightsquigarrow} v\right\}, & \text{se existir um caminho de } u \text{ a } v, \\ \infty, & \text{caso contrário.} \end{cases}$$

O caminho de menor distância é tal que $p(c) = \delta(u, v)$.

02 – ENTENDENDO O PROBLEMA

O caminho mais curto não contém ciclos, dado que removendo o ciclo teremos um caminho mais curto.

Dessa forma, dado um vértice de origem $s \in V$ temos a árvore de caminhos mais curtos partindo de s. Essa é, de G, um subgrafo direcionado $G' = (V', A')$ em que $V' \subseteq V$ e $A' \subseteq A$, tal que:

- V' é o conjunto de vértices alcançáveis partindo de s no grafo G,
- G' é uma árvore de raiz s,
- Para todos os vértices $v \in V$, o único caminho de s até v em G' é o mais curto em G.

03 – ALGORITMO DE DIJKSTRA

Dado um grafo direcionado ponderado G somente de arestas positivas e um vértice de origem s, gera a árvore de caminhos mais curtos de raiz em s para todos os vértices que se pode alcançar partido de s.
O algoritmo se baseia numa estratégia gulosa, a cada iteração é escolhido um vértice u para ser adicionado no conjunto S. Quando isso ocorre já temos calculado pelo algoritmo a distância mínima de s até u.

03 – ALGORITMO DE DIJKSTRA: RELAXAMENTO

Defina d como sendo um vetor em que $d[v]$, para todo $v \in V$, é o limite superior do peso do caminho mais curto de s a v.

- Podemos entender $d[v]$ como sendo até o momento a melhor estimativa para o peso do caminho mais curto,
- Ao final da execução do algoritmo d será o vetor dos pesos dos caminhos mais curtos,
- Iniciamos d da forma $d[v] \coloneqq \infty \; \forall \; v \in V - \{s\}$ e $d[s] \coloneqq 0$.

Defina a como sendo o vetor dos antecessores de cada vértice $v \in V$ no caminho mais curto de s até v.

- Ao final da execução poderemos recuperar através desse vetor os vértices percorridos no caminho mais curto,
- Iniciamos a da forma $a[v] \coloneqq -1 \; \forall \; v \in V$.

03 – ALGORITMO DE DIJKSTRA: RELAXAMENTO

O processo de relaxamento de uma aresta (u, v) consiste em verificar se é possível melhorar o caminho obtido até o momento.

Tentamos melhorar o caminho usando o caminho de s até u, que já nos é garantidamente mínimo pelo algoritmo de Dijkstra, e passarmos pela aresta (u, v).

```
if (d[v] > d[u] + peso_da_aresta_u_v) {
    d[v] = d[u] + peso_da_aresta_u_v;
    a[v] = u;
}
```

03 – ALGORITMO DE DIJKSTRA

- Seleciona cada vértice $u \in V$ somente uma vez, tal que a escolha é sempre do vértice de menor $d[u]$.
- Ao visitarmos um vértice u fazemos o relaxamento de todas as arestas (u, v) em que v é adjacente à u.
- Ao escolhermos um vértice u, o adicionamos em um conjunto S que descreve os vértices já visitados.
- Se $u \in S$, então já temos o caminho mínimo até u, nesse sentido não ocorrerá mais o relaxamento de nenhuma aresta chegando ao vértice u.
- Como iteramos sobre um vértice somente uma vez, teremos que os vértices que podemos escolher estão somente no conjunto $V - S$.
- Podemos escolher o vértice $u \in V - S$ de menor $d[u]$, a cada iteração, utilizando uma fila de prioridades. Em $O(\log n)$, conseguimos remover ou inserir vértices na fila ordenados por menor distância.

03 – ALGORITMO DE DIJKSTRA

- Podemos escolher o vértice $u \in V - S$ de menor $d[u]$, a cada iteração, utilizando uma fila de prioridades,
- Adicionamos, à princípio, todos os vértices na fila, dessa forma o conjunto de vértices inicialmente na fila será V,
- Ao escolhermos um vértice u e o adicionarmos no conjunto S o removeremos da fila, para que o conjunto de elementos na fila seja, em qualquer ponto da execução, $V - S$,
- Ao realizarmos o relaxamento de uma aresta (u, v) devemos atualizar a distância do vértice v também na fila,
- As operações da fila de prioridades tem complexidade temporal de $O(\log n)$.

03 – ALGORITMO DE DIJKSTRA

```cpp
vector<int> d, a;

void dijkstra(int s, vector<vector<pair<int, int>>> &g) {
    d = vector(g.size(), 1'000'000'000); d[s] = 0;
    a = vector(g.size(), -1);
    set<pair<int, int>> pq;
    for (int u {0}; u < g.size(); u++) pq.emplace(d[u], u);

    while (not pq.empty()) {
        auto [dist, u] = *pq.begin(); pq.erase(pq.begin());

        for (auto [v, w] : g[u])
            if (d[u] + w < d[v]) {
                pq.erase(pq.find({d[v], v}));
                d[v] = d[u] + w;
                a[v] = u;
                pq.emplace(d[v], v);
            }
    }
}
```

03 – ALGORITMO DE DIJKSTRA

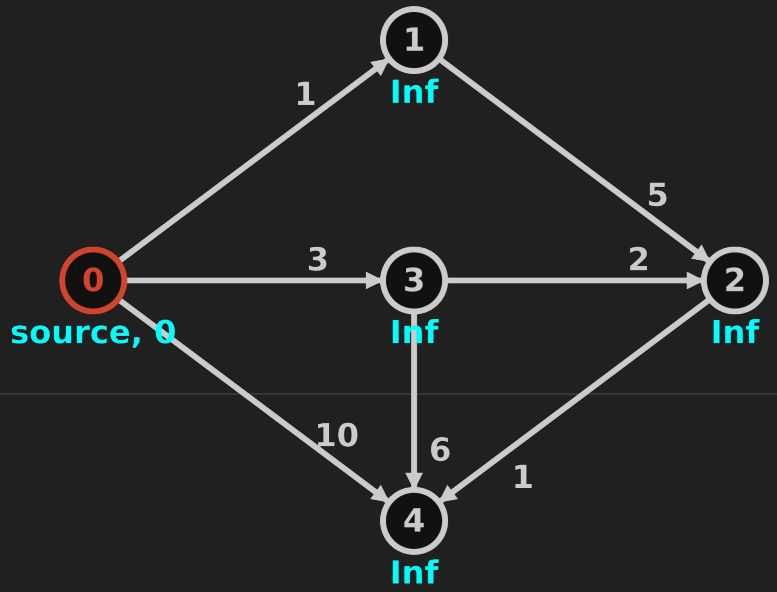
- Continuamos utilizando uma fila de prioridades,
- Começamos inserindo na fila somente o vértice de origem,
- Ao fazermos o relaxamento de uma aresta (u, v) inserimos na fila de prioridades, de forma ordenada, o par $(d[v],\ v)$, após já ter feito a atualização de $d[v]$,
- Essa abordagem pode causar que um vértice u apareça várias vezes na fila. Na primeira iteração de um vértice o adicionamos em S, as demais serão ignoradas,
- O algoritmo continua tendo a mesma complexidade assintótica, todavia, na prática, costuma ser levemente mais veloz.

03 – ALGORITMO DE DIJKSTRA

```cpp
vector<int> d, a;

void dijkstra(int s, vector<vector<pair<int, int>>> &g) {
    d = vector(g.size(), 1'000'000'000); d[s] = 0;
    a = vector(g.size(), -1);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
    pq.emplace(0, s);

    while (not pq.empty()) {
        auto [dist, u] = pq.top(); pq.pop();
        if (dist > d[u]) continue;

        for (auto [v, w] : g[u])
            if (d[u] + w < d[v]) {
                d[v] = d[u] + w;
                a[v] = u;
                pq.emplace(d[v], v);
            }
    }
}
```

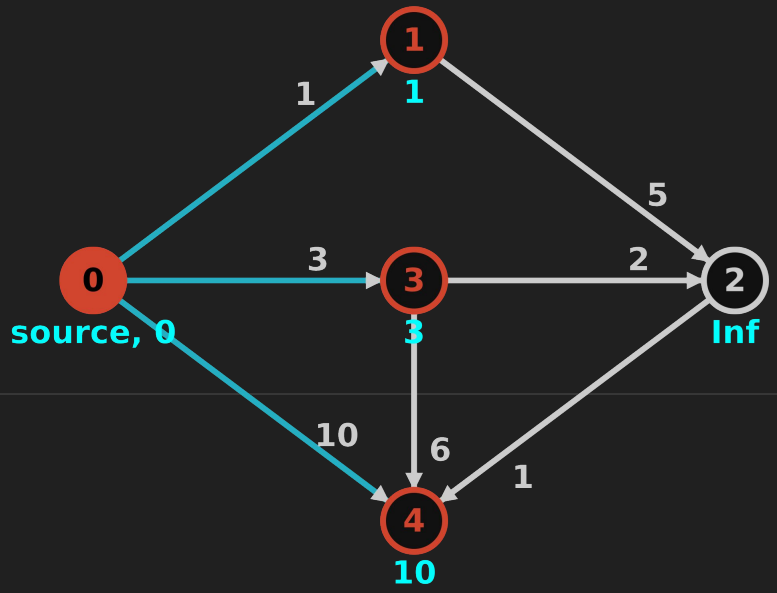
04 – EXECUTANDO O ALGORITMO

$S = \{\}$
$d[0] = \infty$
$d[1] = \infty$
$d[2] = \infty$
$d[3] = \infty$
$d[4] = \infty$



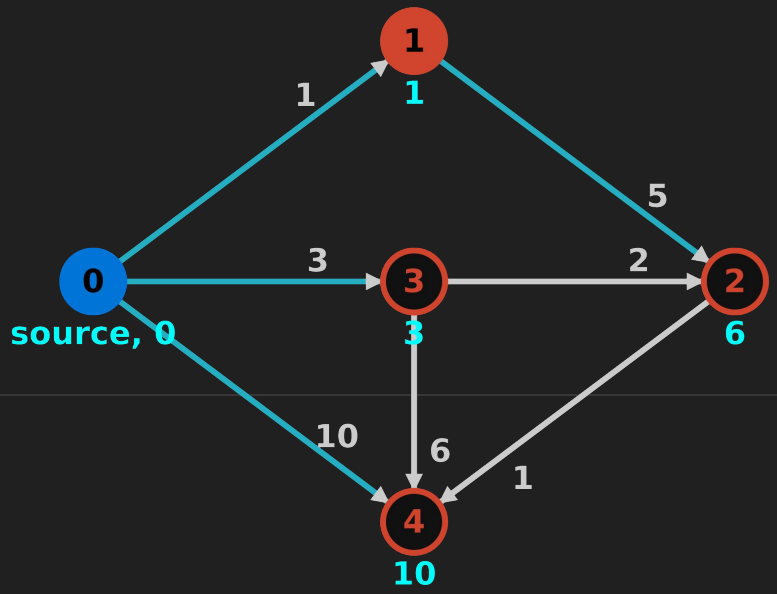
04 – EXECUTANDO O ALGORITMO

$S = \{0\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = \infty$
$d[3] = 3$
$d[4] = 10$



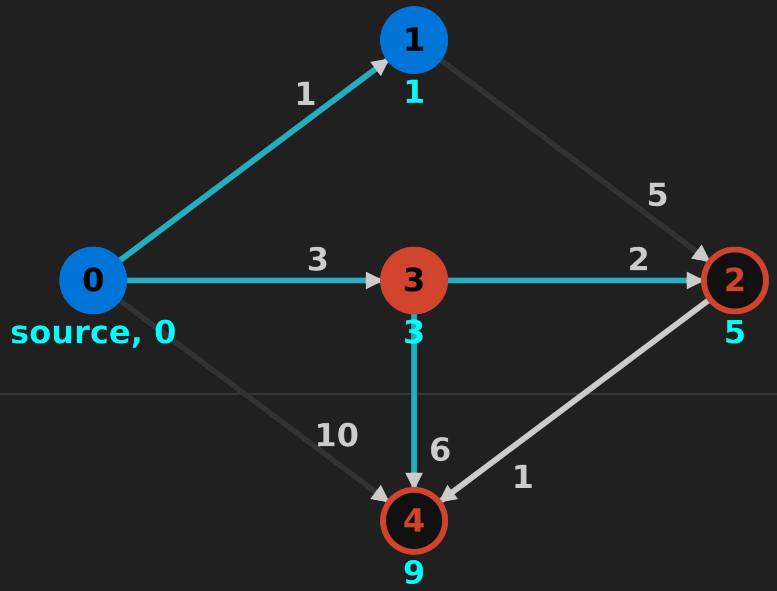
04 – EXECUTANDO O ALGORITMO

$S = \{0, 1\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = 6$
$d[3] = 3$
$d[4] = 10$



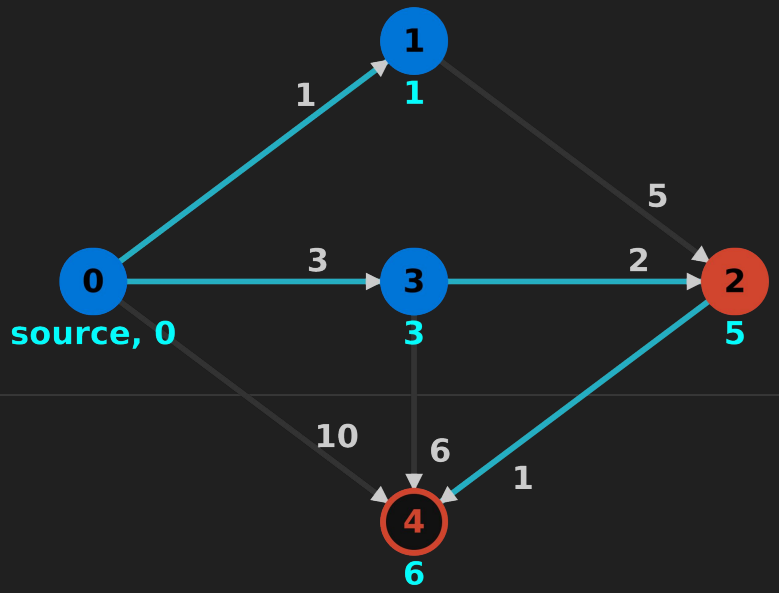
04 – EXECUTANDO O ALGORITMO

$S = \{0, 1, 3\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = 5$
$d[3] = 3$
$d[4] = 9$



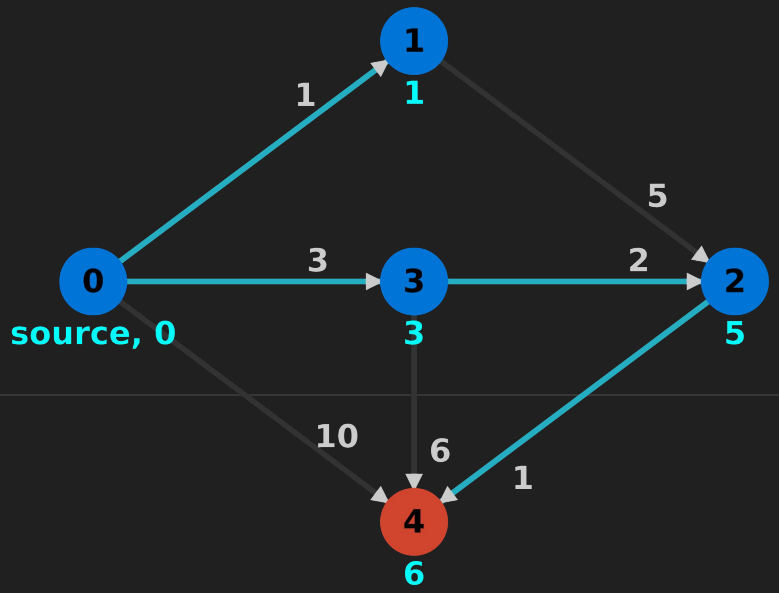
04 – EXECUTANDO O ALGORITMO

$S = \{0, 1, 3, 2\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = 5$
$d[3] = 3$
$d[4] = 6$



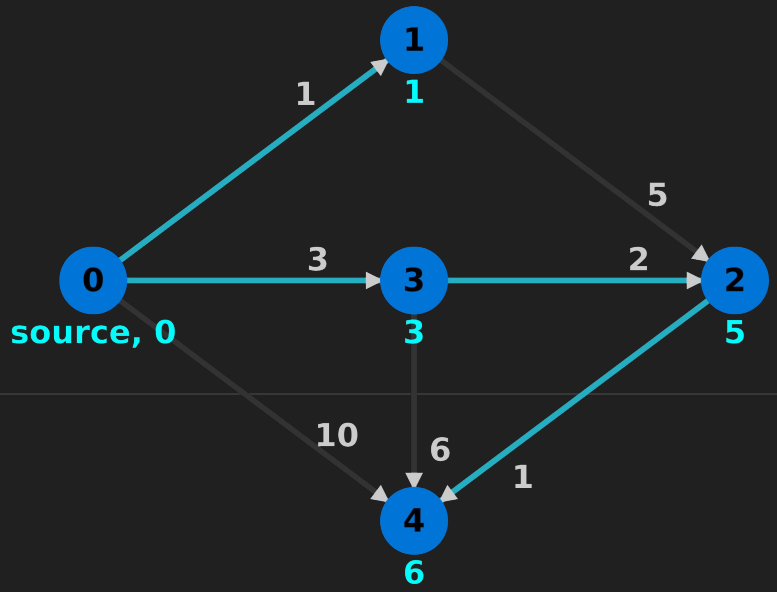
04 – EXECUTANDO O ALGORITMO

$S = \{0, 1, 3, 2, 4\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = 5$
$d[3] = 3$
$d[4] = 6$



04 – EXECUTANDO O ALGORITMO

$S = \{0, 1, 3, 2, 4\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = 5$
$d[3] = 3$
$d[4] = 6$



05 – RESOLVENDO O PROBLEMA MOTIVADOR: ILHAS

Pseudo código que resolve o problema:

1. Faça a entrada de *n*, *m*, do grafo não direcionado *g* em uma lista de adjacência e da ilha *s* onde será instalado o servidor
2. Defina *d* o vetor em que *d[v]* é o menor ping da ilha *v* à ilha *s* *(d[s] == 0)* *Calcularemos d por meio do algoritmo de Dijkstra*
3. Defina *max_ping = max_element(d)*
4. Altere o valor de *d[s]* para *max_ping*, para desconsiderá-lo ao pegar o mínimo
5. Imprima *max_ping – min_element(d)*

06 – ANÁLISE DE COMPLEXIDADE

A complexidade do algoritmo de Dijkstra pode ser analisada em três partes:

1. Inicialização: Tem complexidade de $O(|V| \log |V|)$, pois insere todos os vértices na fila de prioridades que tem tempo de inserção $O(\log n)$,
2. Loop sobre todos os vértices em V: Apresenta complexidade $O(|V| \log |V|)$, pois percorre cada vértice removendo-o da fila de prioridade,
3. Loop sobre os vértices adjacentes: Executa em $O(|A| \log |V|)$, dado que, executando o loop de fora para cada vértice e tendo no pior caso que o grafo não é direcionado, executaremos esse loop para um total de $2|A|$ arestas, o $\log |V|$ vêm da complexidade da fila de prioridades.

Concluímos, portanto, que a complexidade do algoritmo será: $O(|V| \log|V| + |V| \log|V| + |A| \log|V|) = O(|A| \log|V|)$, sabendo que $|A| + 1 \geq |V|$ para qualquer grafo conexo.

07 – DEMONSTRAÇÃO

Tese: Quando adicionamos u ao conjunto S, ou seja, quanto retiramos u da fila, não será mais alterado o valor de $d[u]$, pois o caminho mínimo já foi encontrado.

1. **Caso base: Ao removermos o primeiro vértice, já teremos uma distância $d[s] = 0$ que é mínima.**
2. **Passo indutivo: Assuma que $d[v]$ é mínimo para todo v já pertencente à S.**
3. **Seja $c = (s, \dots, u)$ o menor caminho de s até u e sejam $c_1 = (s, \dots, q)$ e $c_2 = (p, \dots, u)$ caminhos tais que unindo c_1 e c_2 pela aresta (q, p) temos o próprio caminho c, considere que os vértices em c_1 estão todos em S e em c_2 ao menos p e u não estão em S.**
4. **Como $q \in S$ e $p(c) = \delta(s, u)$, ocorreu o relaxamento da aresta (q, p) de forma que $d[p] = \delta(s, p) \leq \delta(s, u) \leq d[u]$.**
5. **Como estamos iterando sobre u, então $d[p] \geq d[u]$, por 4, decorre que $d[p] = d[u] \Rightarrow d[u] = \delta(s, u)$.**

08 – RECUPERANDO O CAMINHO MAIS CURTO

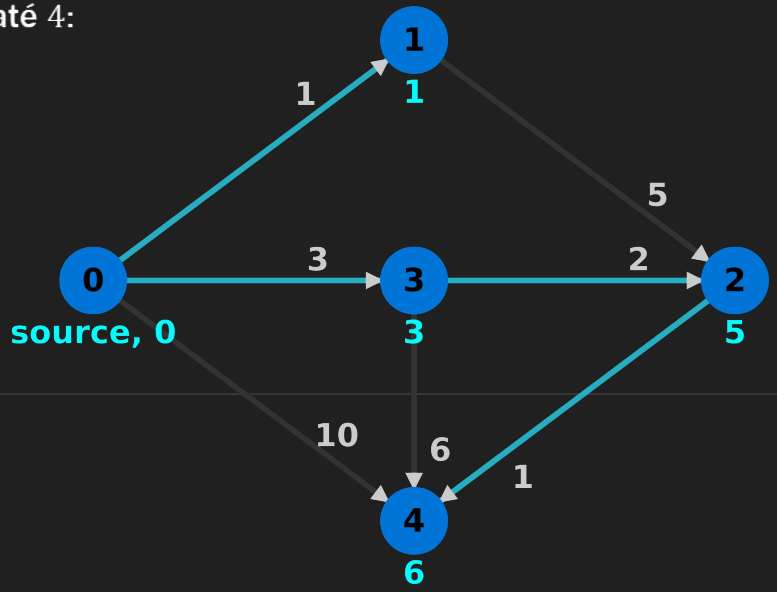
Temos calculado, após executar Dijkstra, o vetor a dos antecessores no menor caminho, o usaremos para encontrar em $O(|V|)$ o caminho do vértice de origem até um vértice v.

```cpp
vector<int> recuperar_caminho(int v) {
    if (a[v] == -1)
        return {};

    vector<int> c;
    for (; v != -1; v = a[v])
        c.push_back(v);

    reverse(c.begin(), c.end());
    return c;
}
```

08 – RECUPERANDO O CAMINHO MAIS CURTO

Caminho mais curto de 0 até 4:

$$c = (0, 3, 2, 4)$$



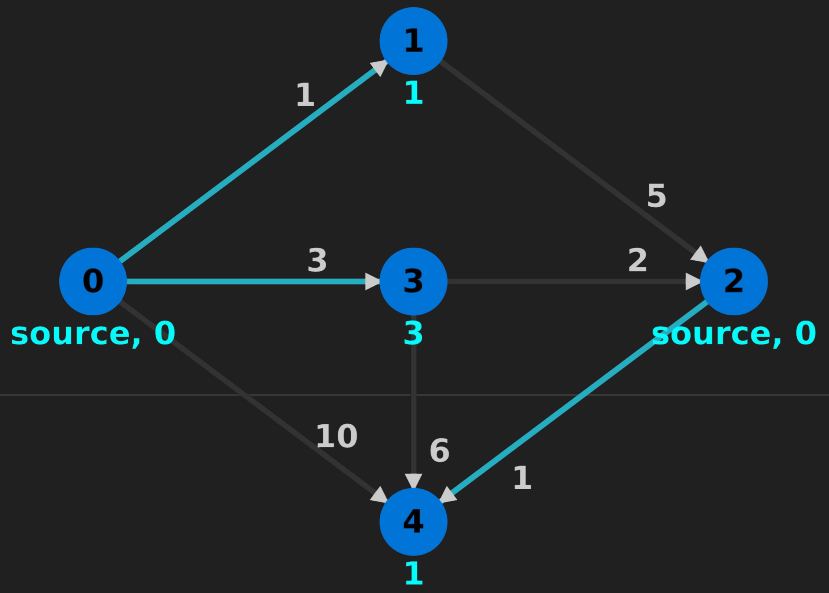
09 – DIJKSTRA COM MÚLTIPLAS ORIGENS

Dado um conjunto de origens O queremos $d[u] = min\{\delta(s,u) \; \forall \; s \in O\}$. Para tanto, podemos ligar todos os vértices em O por arestas de peso 0, que é equivalente à adicionar os vértices em O na fila de prioridades com peso 0 e fazer $d[s] \coloneqq 0 \; \forall \; s \in O$. O resto do algoritmo se mantém inalterado.

```cpp
void dijkstra(vector<int> &origens, vector<vector<pair<int, int>>> &g) {
    d = vector(g.size(), INF);
    a = vector(g.size(), -1);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;

    for (int s : origens) {
        pq.emplace(0, s);
        d[s] = 0;
    }

    // resto do algoritmo
}
```

09 – DIJKSTRA COM MÚLTIPLAS ORIGENS

$S = \{0, 2, 1, 4, 3\}$
$d[0] = 0$
$d[1] = 1$
$d[2] = 0$
$d[3] = 3$
$d[4] = 1$



10 – CALCULANDO DISTÂNCIAS ATÉ A ORIGEM

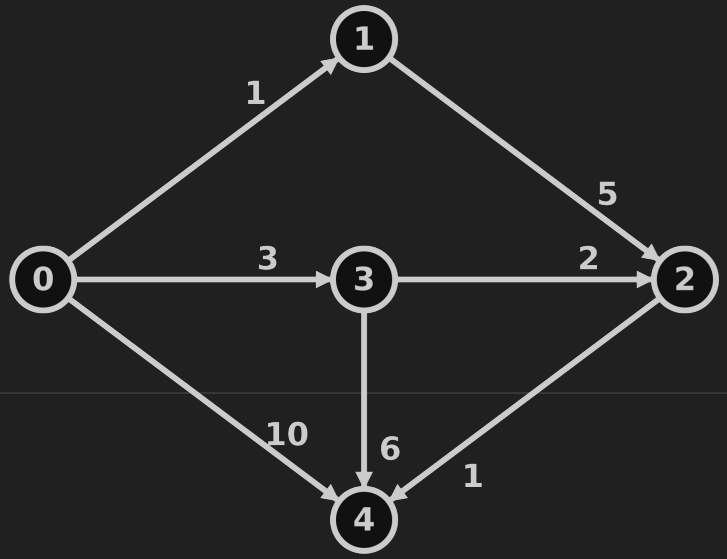
O algoritmo de Dijkstra calcula os valores de $\delta(s, u)\ \forall\ u \in V$, queremos calcular os valores de $\delta(u, s)\ \forall\ u \in V$.

Para tal, basta executarmos o algoritmo de Dijkstra sobre o grafo transposto de $G = (V, A)$, seja esse:

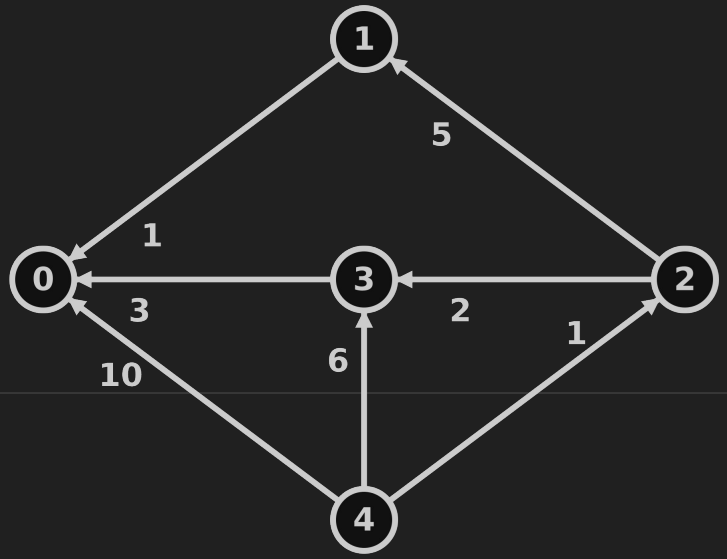
$$G^T = (V, A^T), \text{ em que } A^T = \{(v, u) : (u, v) \in A\}.$$

```cpp
vector<vector<pair<uint, uint>>> transpor(const vector<vector<pair<uint, uint>>> &g) {
    vector<vector<pair<uint, uint>>> gt(g.size());
    for (uint u {0}; u < g.size(); u++)
        for (const auto &[v, w] : g[u])
            gt[v].emplace_back(u, w);
    return gt;
}
```

10 – CALCULANDO DISTÂNCIAS ATÉ A ORIGEM

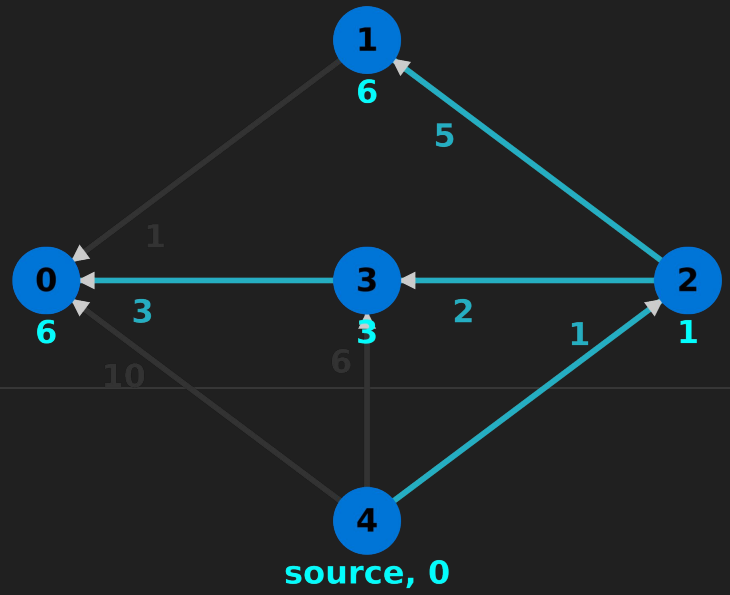


10 – CALCULANDO DISTÂNCIAS ATÉ A ORIGEM



10 – CALCULANDO DISTÂNCIAS ATÉ A ORIGEM

$S = \{4, 2, 3, 1, 0\}$
$d[0] = 6$
$d[1] = 6$
$d[2] = 1$
$d[3] = 3$
$d[4] = 0$



11 – REFERÊNCIAS BIBLIOGRÁFICAS

ZIVIANI, N. Projeto de algoritmos com implementação em java e c++. Cengage Learning, 2006. ISBN 9788522105250. Disponível em: https://books.google.com.br/books?id=sNBIMgAACAAJ.

ALGORITHMS FOR COMPETITIVE PROGRAMMING. Dijkstra - finding shortest paths from given vertex.2024. Disponível em: https://cp-algorithms.com/graph/dijkstra.html. Acesso em: 13 oct. 2024.

VISUALGO. Single-Source Shortest Paths. 2024. Disponível em: https://visualgo.net/en/sssp.Acesso em: 13 oct. 2024.

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

GCC.CEFETMG

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg