DCA0204, Módulo 9
Caminhos mais curtos

Daniel Aloise

baseado em slides do prof. Leo Liberti, École Polytechnique, França

DCA,UFRN

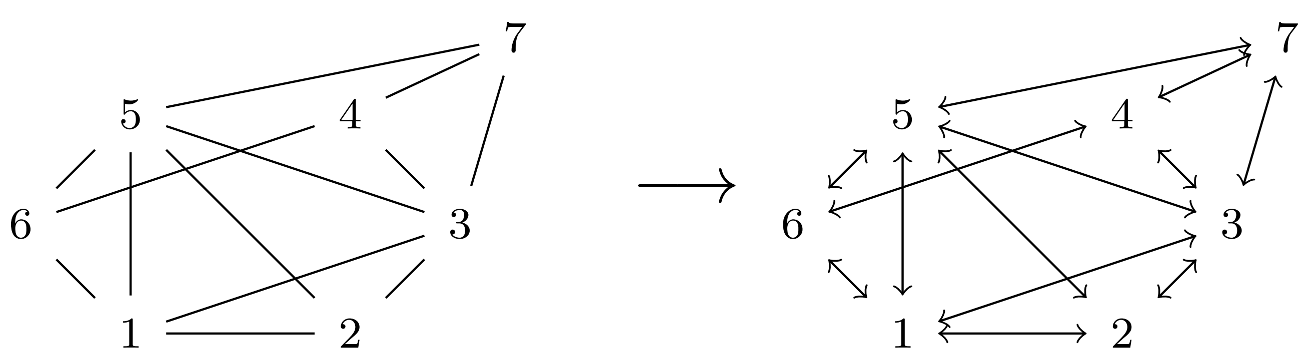
Sumário

- Problemas de caminho mais curto (PCC) e suas variantes
- Algoritmo de Dijkstra
- Algoritmo de Floyd-Warshall

Problemas de caminho mais curto

Grafos ou dígrafos?

- Na maioria das aplicações, o modelo correto para PCCs é dado por **arcos** e **dígrafos** mais do que **arestas** e **grafos**
- PCCs também ocorrem como subproblemas em algoritmos complicados: pode ser necessário resolver PCCs em grafos
- Embora caminhos dirigidos sejam também chamados **percursos** (Módulos 6, 8), nós ainda usamos o termo **caminho** por razões históricas
- Similarmente, usamos o termo **ciclo** também para circuitos
- Um PCC em um grafo é equivalente a um PCC no dígrafo onde cada aresta é substituída por arcos antiparalelos



Motivação

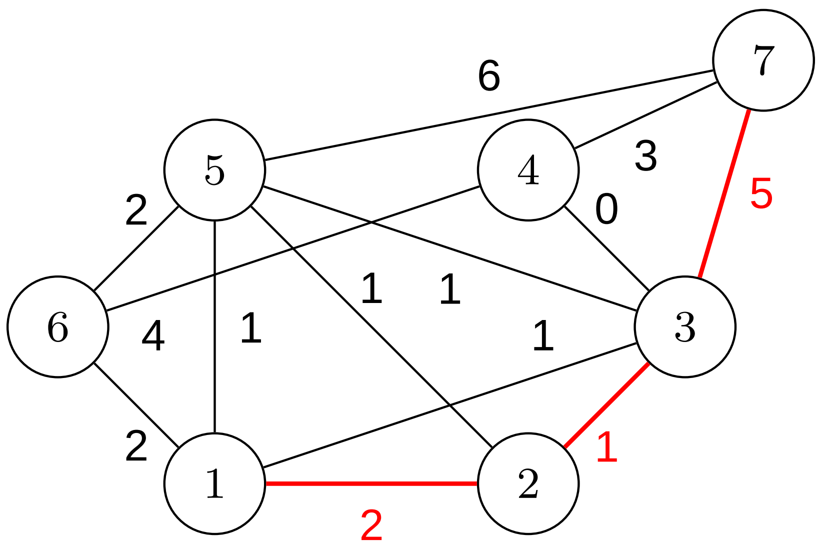
Vários PCCs podem ser resolvidos em tempo polinomial

Custo de um caminho

- Consideramos um **dígrafo ponderado** $G = (V, A)$ com **custos nos arcos**
- I.e. temos uma função $c : A \to \mathbb{Q}$
- Se $P \subseteq G$ é um caminho $u \to v$ em G então

$$c(P) = \sum_{(u,v) \in P} c_{uv},$$

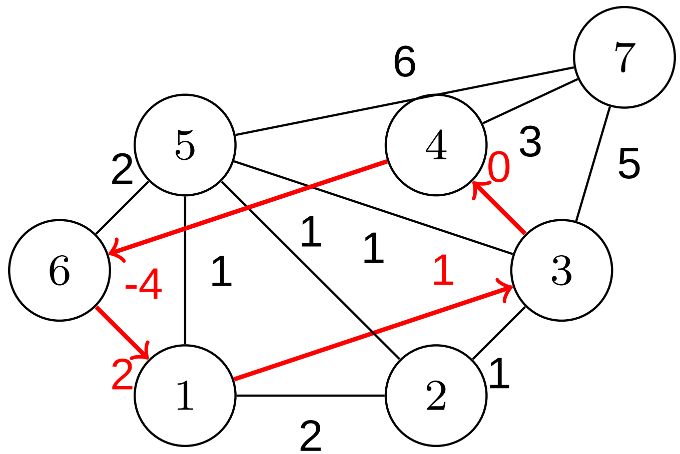
  onde $c_{uv} = c((u, v))$
- Por exemplo, o caminho $1 \to 2 \to 3 \to 7$ tem custo $2 + 1 + 5 = 8$



Caminho mais curto = caminho P tendo custo mínimo $c(P)$

Ciclos negativos

O ciclo vermelho tem *custo negativo* $1 + 0 - 4 + 2 = -1 < 0$



Thm.

Se $G = (V, A)$ tem um ciclo C com $c(C) < 0$, nÃčo existe CC em G

Proof

Suponha P é CC $u \to v$ com custo c^* e seja C um ciclo negativo em G. Podemos repetir C quantas vezes quisermos em um caminho P' de u até v de modo a fazermos o custo de P' menor do que c^*.

$\Rightarrow$ Precisamos assumir que o grafo não tem ciclos negativos

Ciclos negativos: comentários

- A fim de construir Q na prova anterior, nós passamos inúmeras vezes ao redor do ciclo negativo C.
- $\Rightarrow Q$ não é um caminho simples
- Se procuramos pelo *caminho simples mais curto* em grafos então não temos um problema ilimitado
- O problema do CAMINHO MAIS CURTO SIMPLES (CCS), entretanto, é **NP**-árduo para grafos gerais ponderados
- Resolver o problema do CAMINHO MAIS LONGO SIMPLES também é **NP**-árduo

  (Prove isto transformando polinomialmente CCS para o problema do CAMINHO MAIS LONGO SIMPLES)

Hipóteses

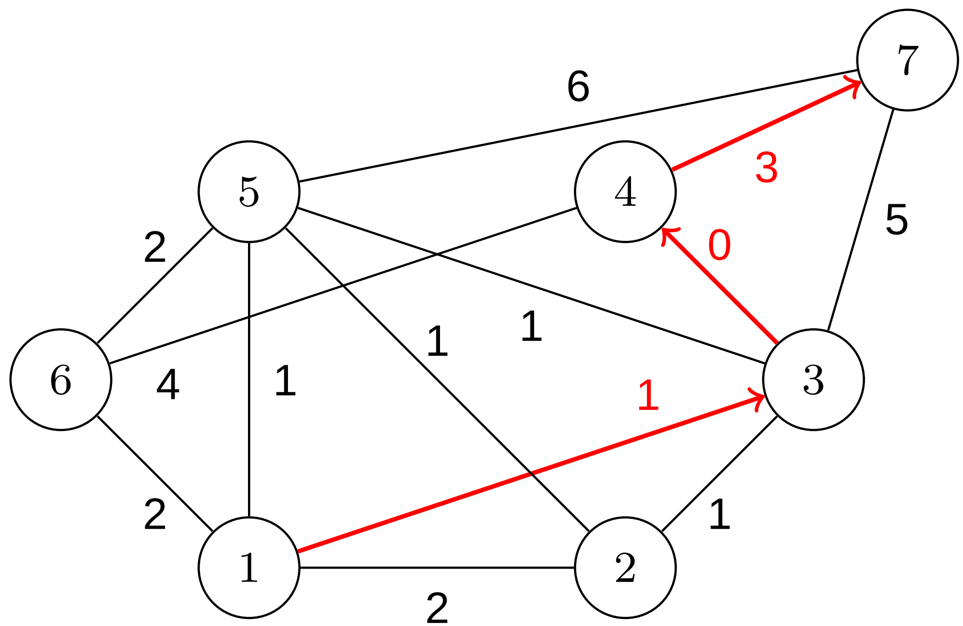
Assuma para o restante dos slides que:

- G é conexo (grafo) ou fortemente conexo (dígrafo)
- O grafo G não possui ciclos negativos

Caminho mais curto ponto a ponto

POINT-TO-POINT SHORTEST PATH (P2PSP). Dado um dígrafo $G = (V, A)$, uma função $c : A \to \mathbb{Q}$ e dois nós distintos $s, t \in V$, encontre um CC $s \to t$

Um caminho mais curto $1 \to 7$



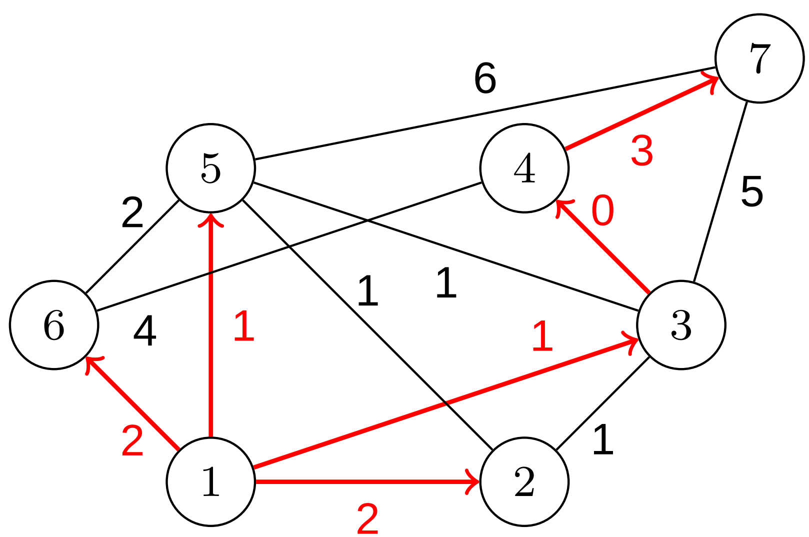
Árvore de caminhos mais curtos

ÁRVORE DE CAMINHOS MAIS CURTOS (ACC). Dado um dígrafo $G = (V, A)$, uma função $c : A \to \mathbb{Q}$ e um nó de origem $s \in V$, encontre CCs $s \to v$ para todo $v \in V \setminus \{s\}$

- *Obs*: pode haver mais de um CC $s \to v$
- **Consistência**: pode-se sempre escolher CC P_{sv} $u \to v$ de modo que $T = \bigcup_{v \neq s} P_{sv}$ é uma árv. geradora orientada ($\Leftrightarrow \forall v \neq s \ (N^-_T(v) = 1)$)
- **Thm. A**

  Se c não contem ciclos negativos, todo subcaminho de um CC é um CC

  (ex. subcaminho $1 \to 4$ de CC $1 \to 7$ abaixo é um CC $1 \to 4$)



Todos os caminhos mais curtos

ALL SHORTEST PATHS (ASP). Dado um dígrafo $G = (V, A)$ e uma função $c : A \rightarrow \mathbb{Q}$, encontre CCs $u \rightarrow v$ para todos os pares u, v de nós distintos em V

Variantes

- **Custos unitários**: para todo $(u, v) \in A$ temos $c_{uv} = 1$
- ACC em custos unitários: use BFS (Módulos 2, 6), $O(m + n)$
- **Custos não-negativos**: para todo $(u, v) \in A$ temos $c_{uv} \geq 0$
- Existem muitas outras variantes!
- *Uma variante importante*: CC em grafos não-dirigidos com $c : E \to \mathbb{N}$ pode ser resolvido em tempo linear [Thorup 1997]

Dijkstra's algorithm

O problema alvo

O Algoritmo de Dijkstra resolve ACC em grafos ponderados $G = (V, A)$ com custos não-negativos (a partir de um dado nó $s \in V$)

- Se $c \geq 0$ então o grafo não tem ciclos negativos (por que?)
- Complexidade de pior caso: $O(n^2)$ em dígrafos gerais, $O(m + n \log n)$ em grafos esparsos, onde $n = |V|$ e $m = |A|$
- Usado como subpasso em inúmeros algoritmos
- Principal aplicação: roteamento em redes (usualmente transporte e comunicação)

Estruturas de dados

- Nós mantemos duas funções
  - $d : V \to \mathbb{Q}_+$

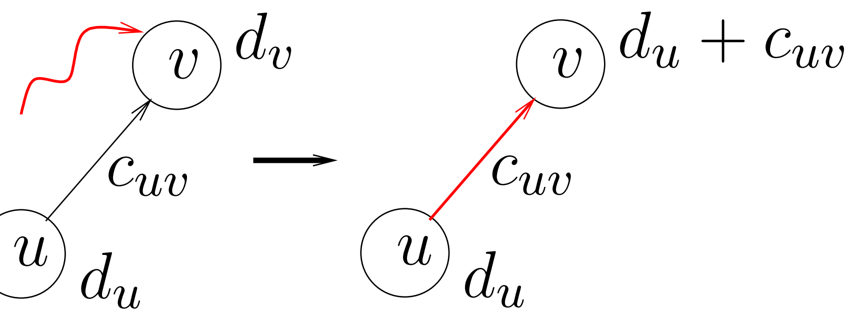
    $d_v = d(v)$ *é o custo de um CC* $s \to v$ *para todo* $v \in V$
  - p $: V \to V$

    p$_v =$ p(v) *é o predecessor de* v *em um CC* $s \to v$ *para todo* $v \in V$
- Inicialização
  - $d_s = 0$ e $d_v = \infty$ para todo $v \in V \setminus \{s\}$
  - p$(v) = s$ para todo $v \in V$

Acomode-se e relaxe

- Um nó $v \in V$ é **acomodado** quando d_v não muda mais
- **Relaxar** um arco $(u, v) \in A$ consiste em:

  if $d_u + c_{uv} < d_v$ **then**
  Seja $d_v = d_u + c_{uv}$;
  Seja $\mathsf{p}_v = u$;
  end if



- Quando (u, v) é relaxado e v ainda não está acomodado, d_v pode mudar

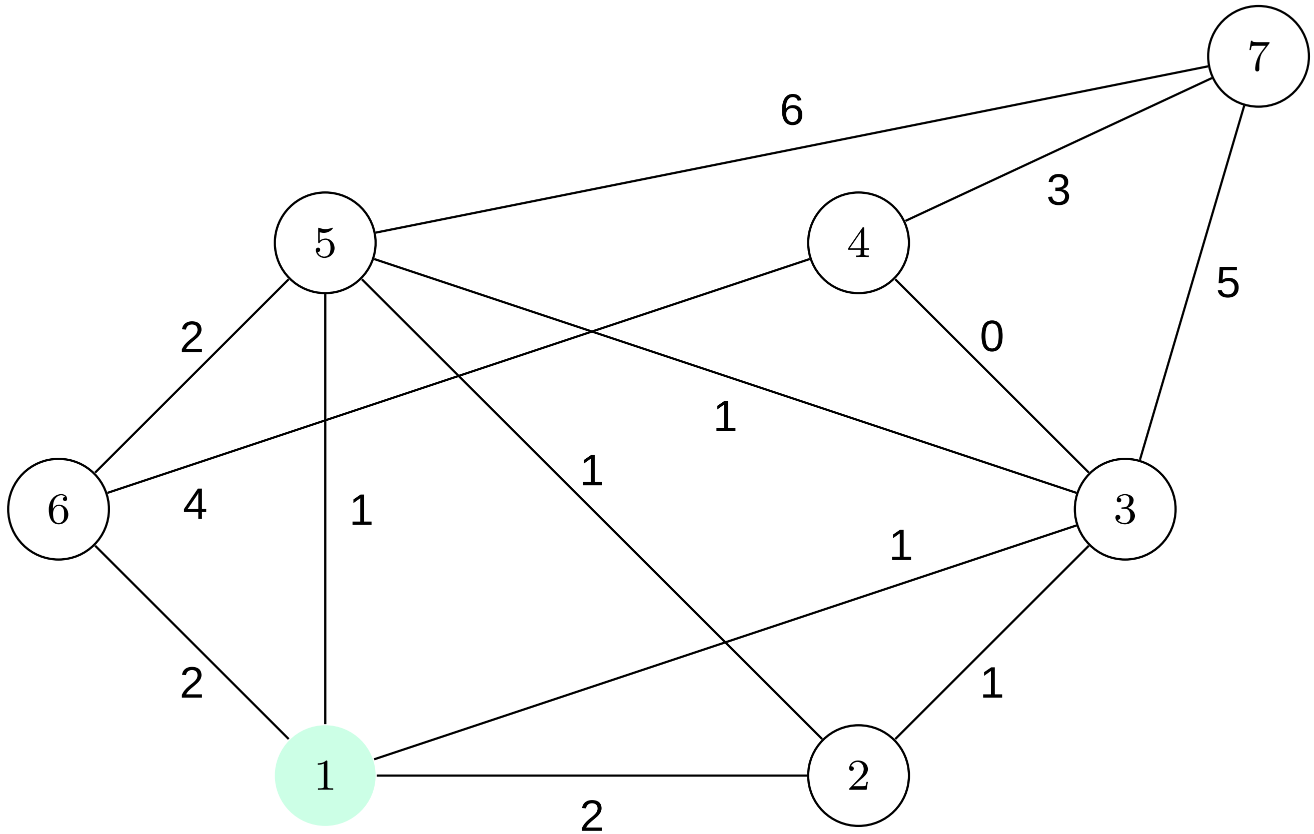
Descrição

- Algoritmo de Dijkstra :

```
1: while ∃ nós não acomodados do
2:   Seja u um nó não acomodado com mínimo d_u;
3:   Acomode u;
4:   for (u, v) ∈ A do
5:     Relaxe (u, v);
6:   end for
7: end while
```

- Se $d_v = \infty$ no Step 4, relaxar (u, v) irá necessariamente mudar d_v (por que?)
- Nós $v \in V$ tal que $d_v < \infty$ são ditos **alcançados**
- Uma implementação simples é $O(n^2)$

Exemplo com $s = 1$

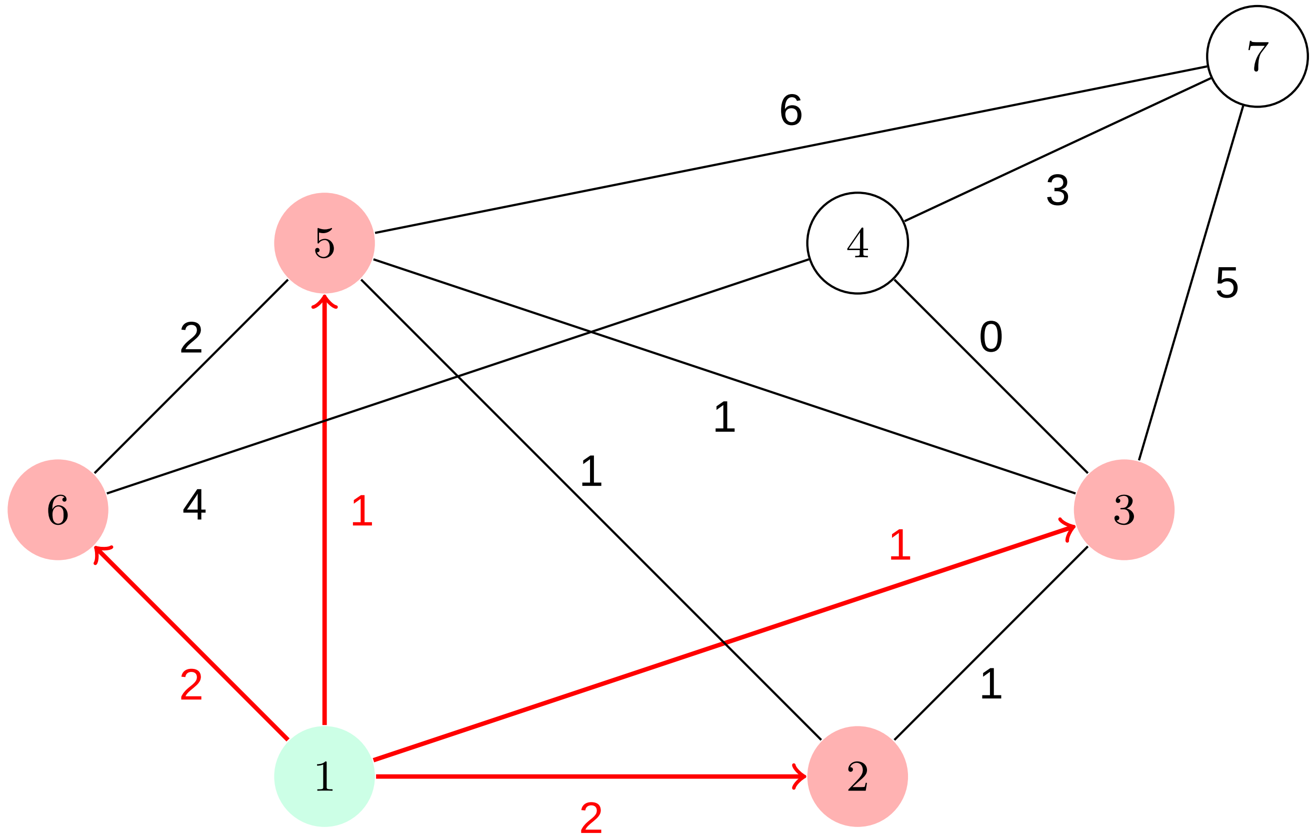


$d:$	1	2	3	4	5	6	7
	0	∞	∞	∞	∞	∞	∞

p :	1	2	3	4	5	6	7
	1	1	1	1	1	1	1

inicialize (acomode) $s = 1$

Exemplo com $s = 1$



d :

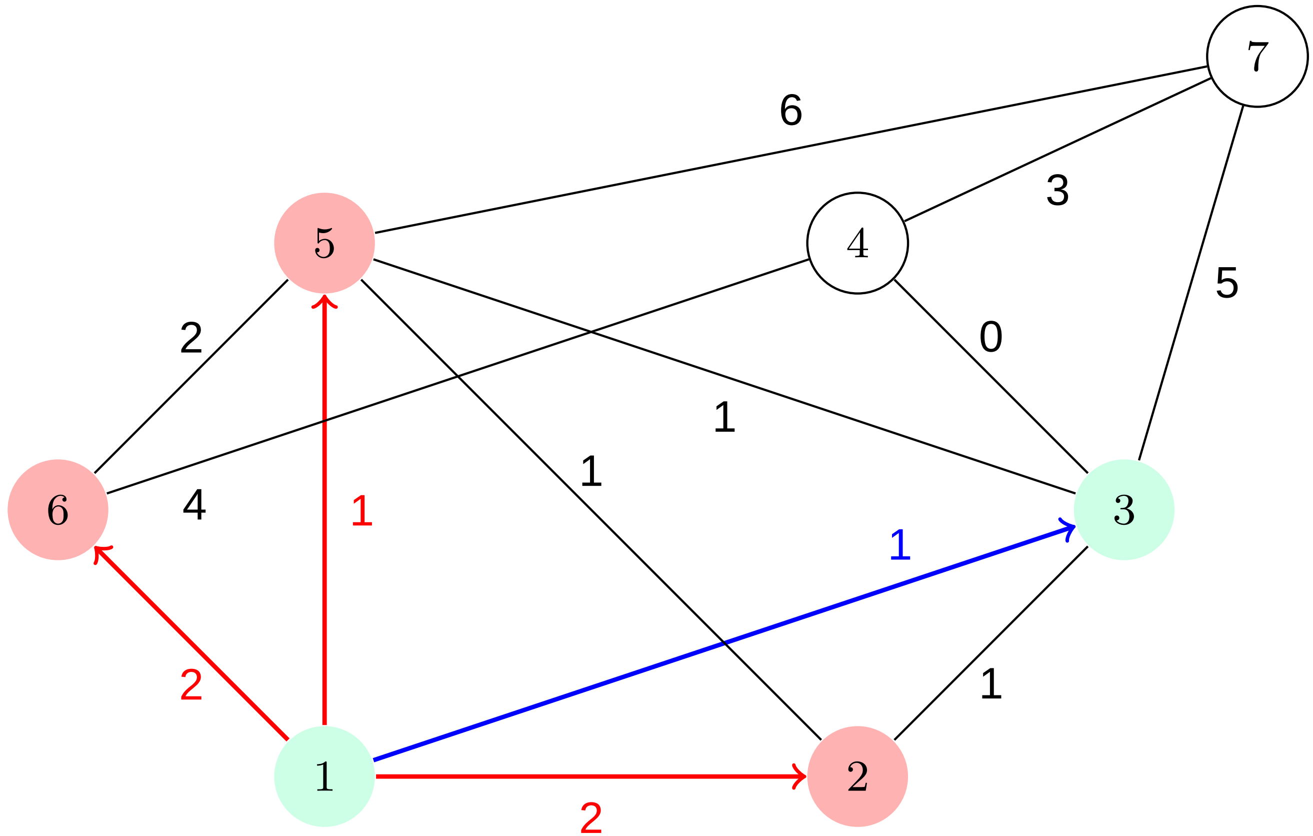
1	2	3	4	5	6	7
0	2	1	∞	1	2	∞

p :

1	2	3	4	5	6	7
1	1	1	1	1	1	1

relaxe $\delta^+(1)$, atualize $2, 3, 5, 6$

Exemplo com $s = 1$



$d:$

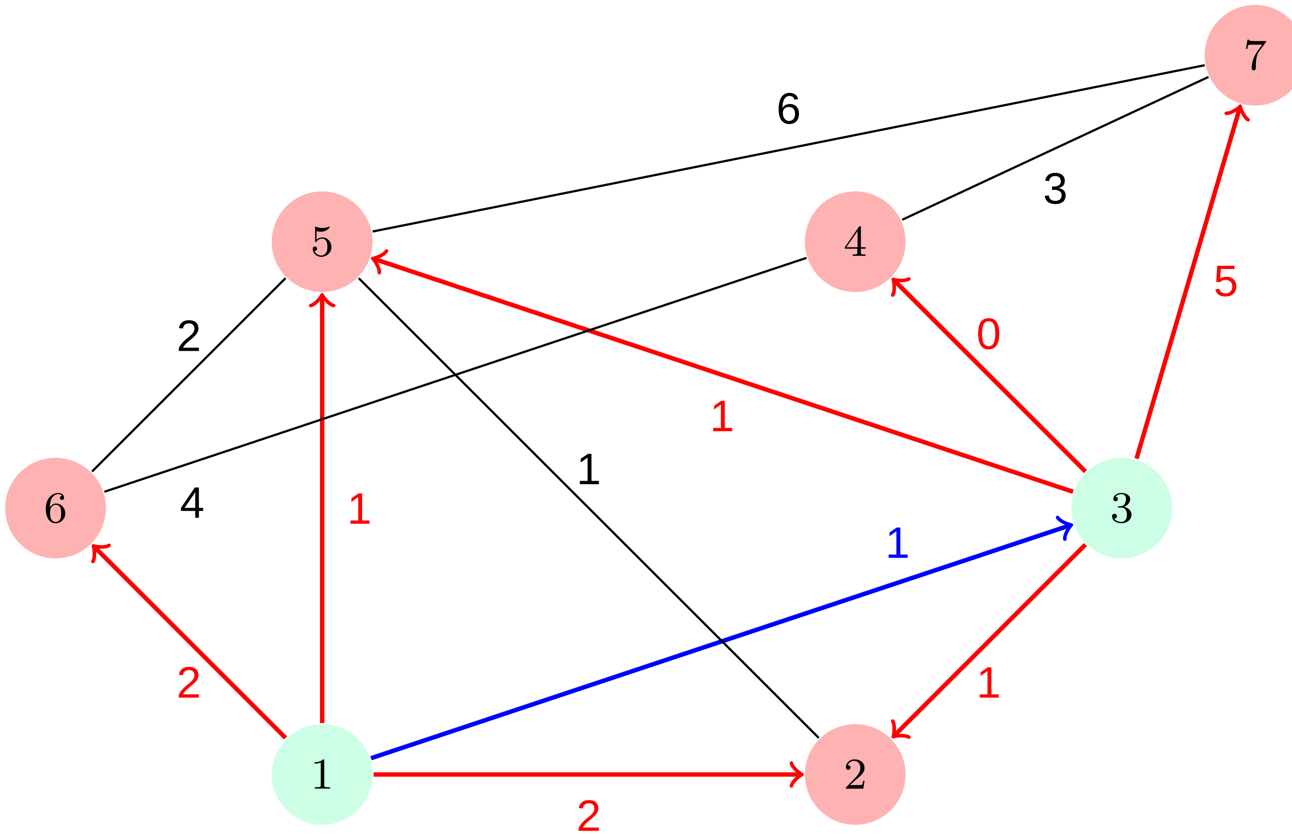
1	2	3	4	5	6	7
0	2	1	∞	1	2	∞

p :

1	2	3	4	5	6	7
1	1	1	1	1	1	1

acomode 3 ($d_3 = 1$ é mínimo)

Exemplo com $s = 1$

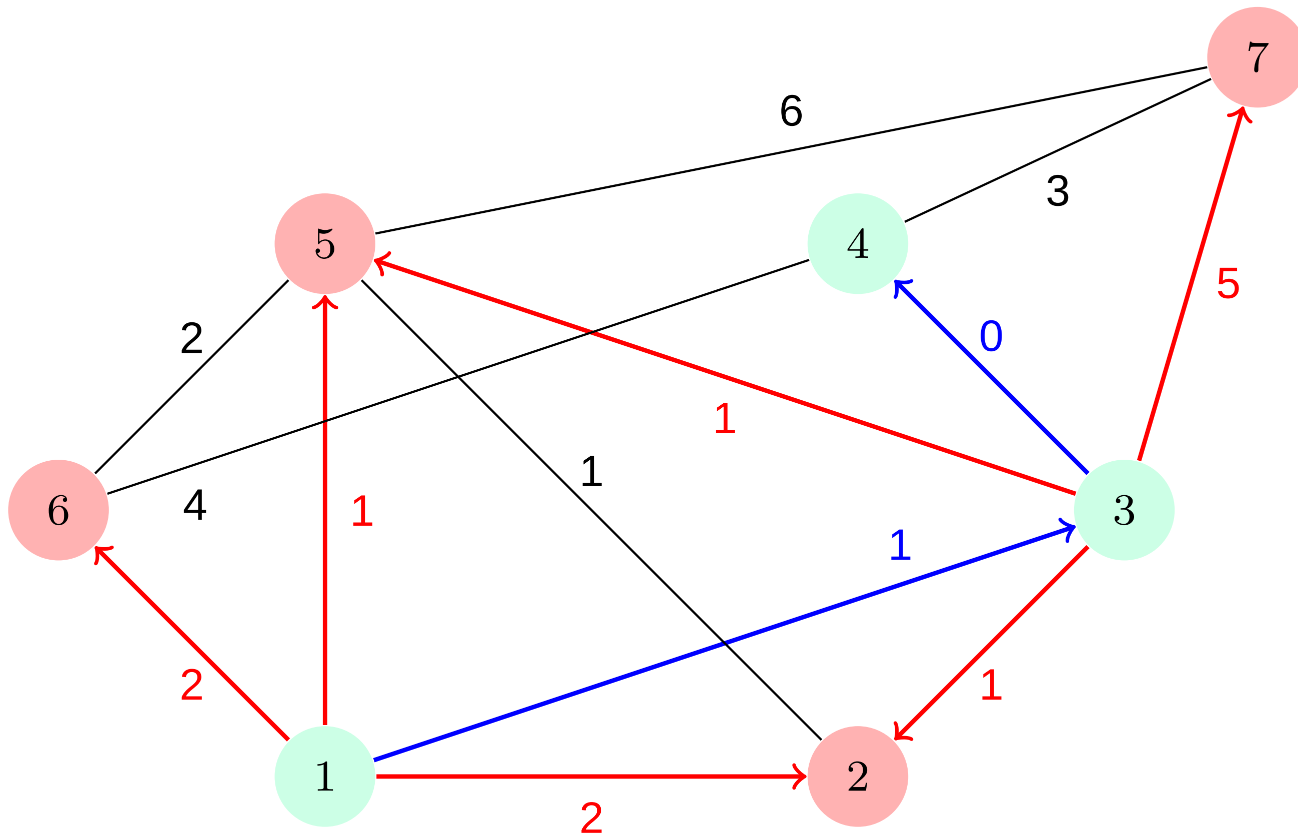


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	6

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	3

relaxe $\delta^+(3)$, atualize $4, 7$

Exemplo com $s = 1$

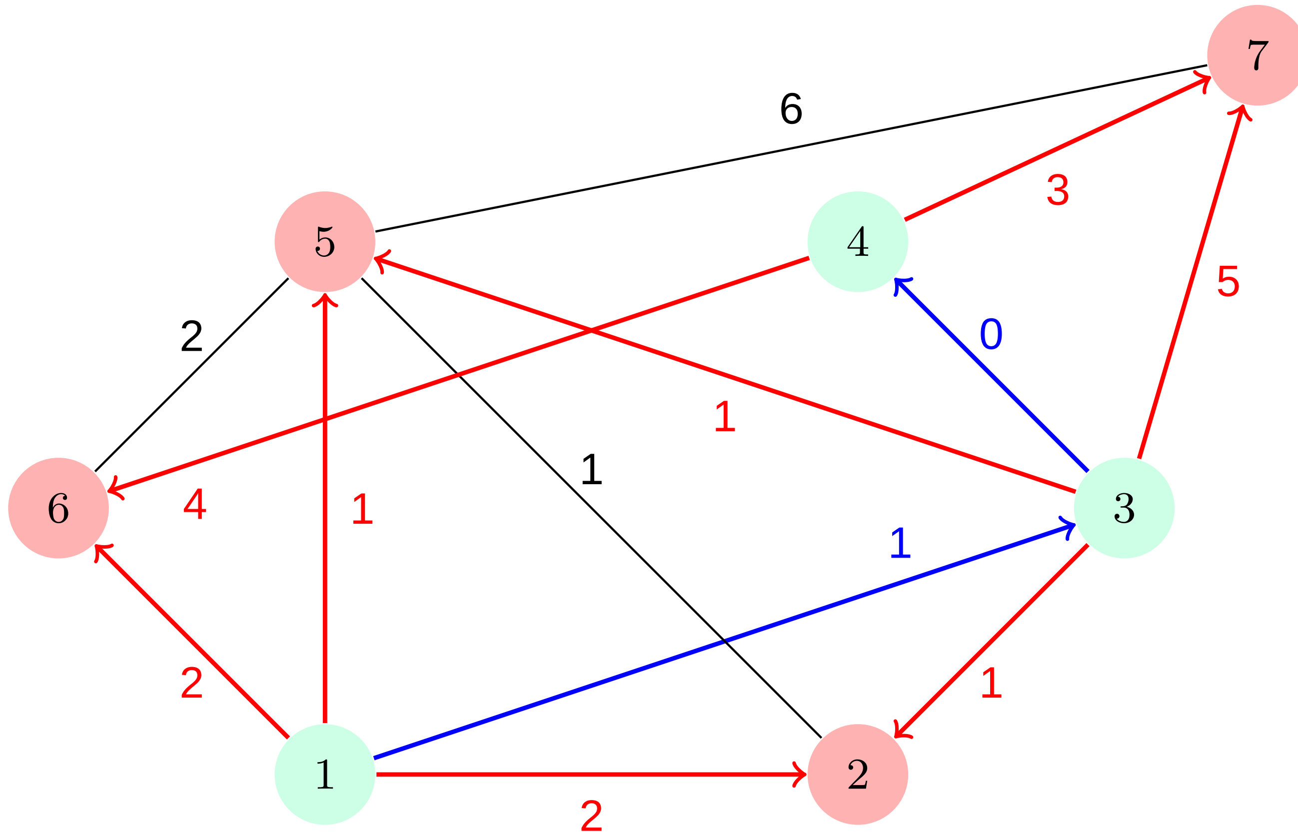


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	6

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	3

acomode 4 ($d_4 = 1$ é mínimo)

Exemplo com $s = 1$



d :

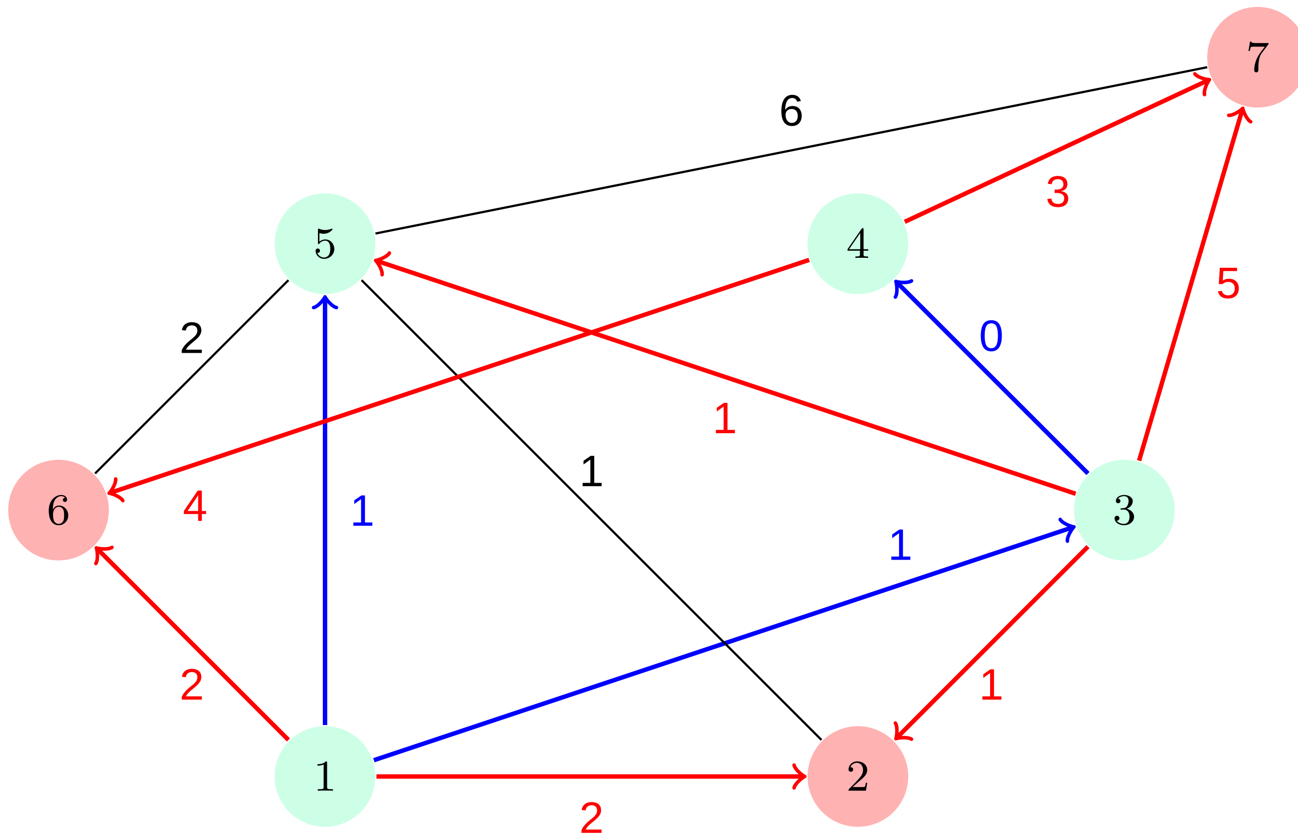
1	2	3	4	5	6	7
0	2	1	1	1	2	4

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	4

relaxe $\delta^+(4)$, atualize 7

Exemplo com $s = 1$

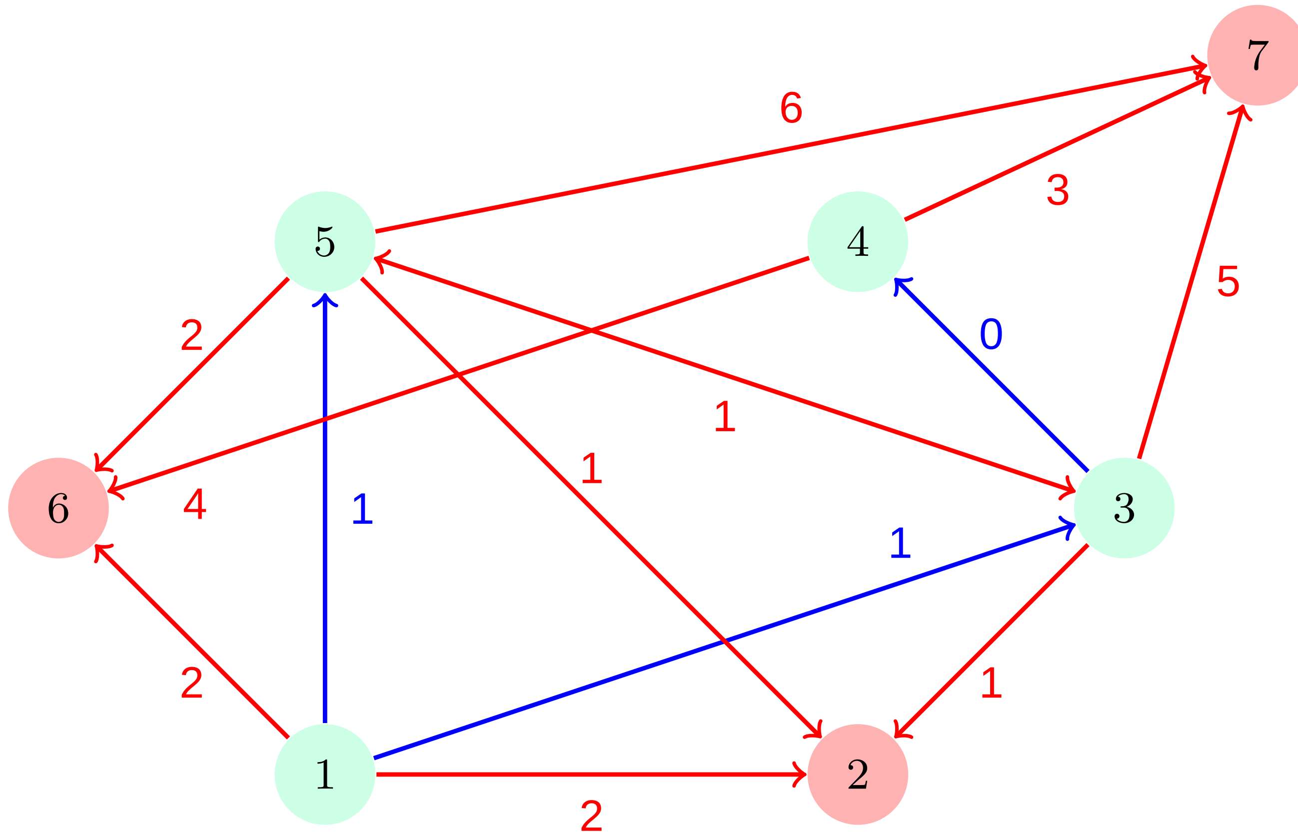


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

acomode 5 ($d_5 = 1$ é mínimo)

Exemplo com $s = 1$

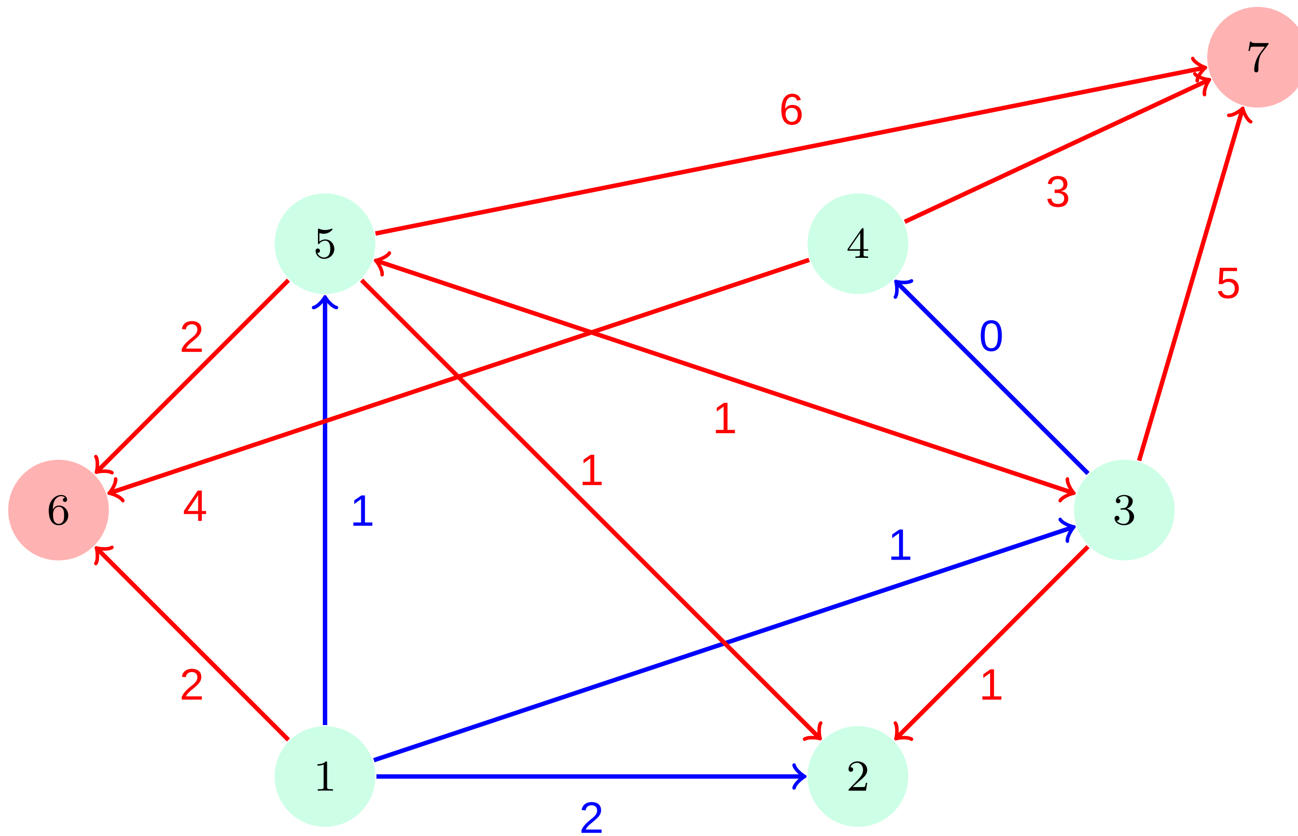


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

relaxe $\delta^{+}(5)$

Exemplo com $s = 1$

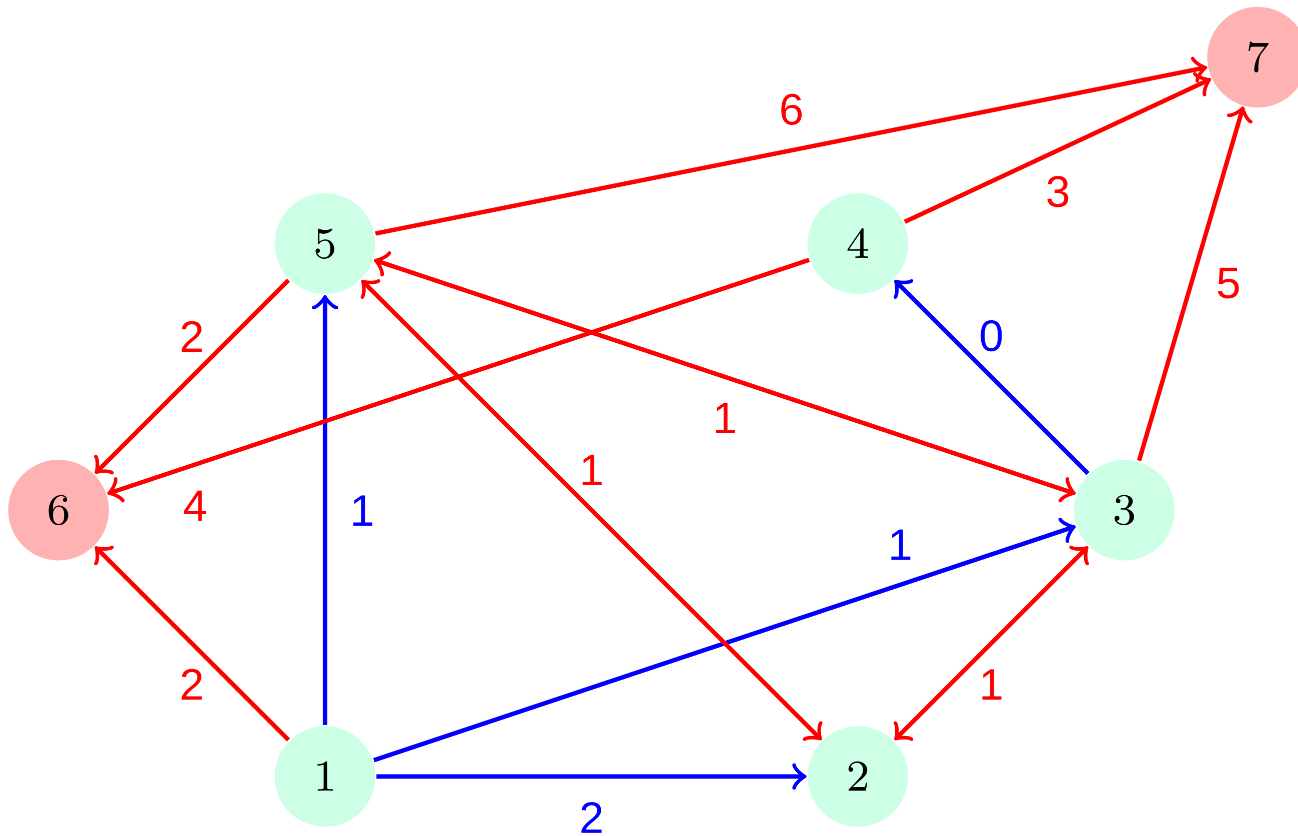


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

acomode 2 ($d_2 = 2$ é mínimo)

Exemplo com $s = 1$



d :

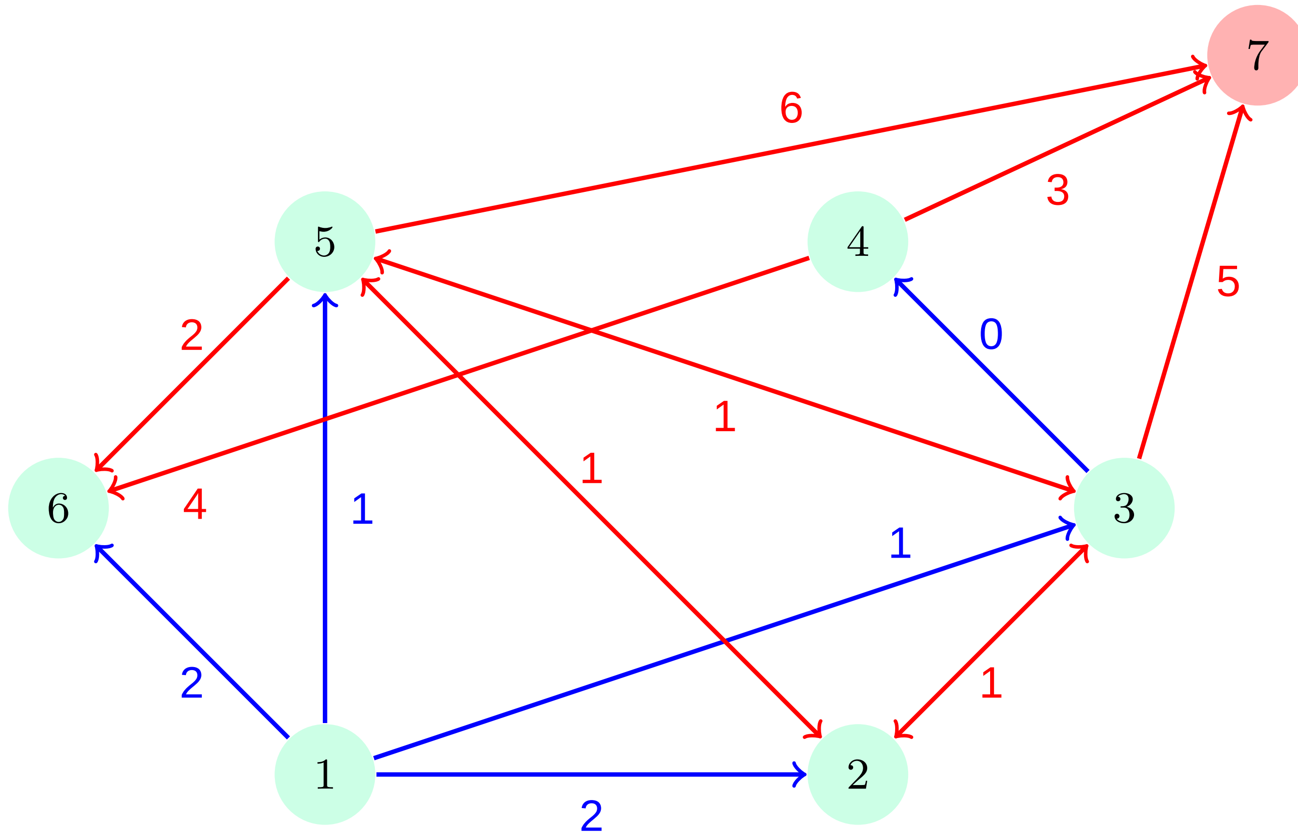
1	2	3	4	5	6	7
0	2	1	1	1	2	4

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	4

relaxe $\delta^+(2)$

Exemplo com $s = 1$

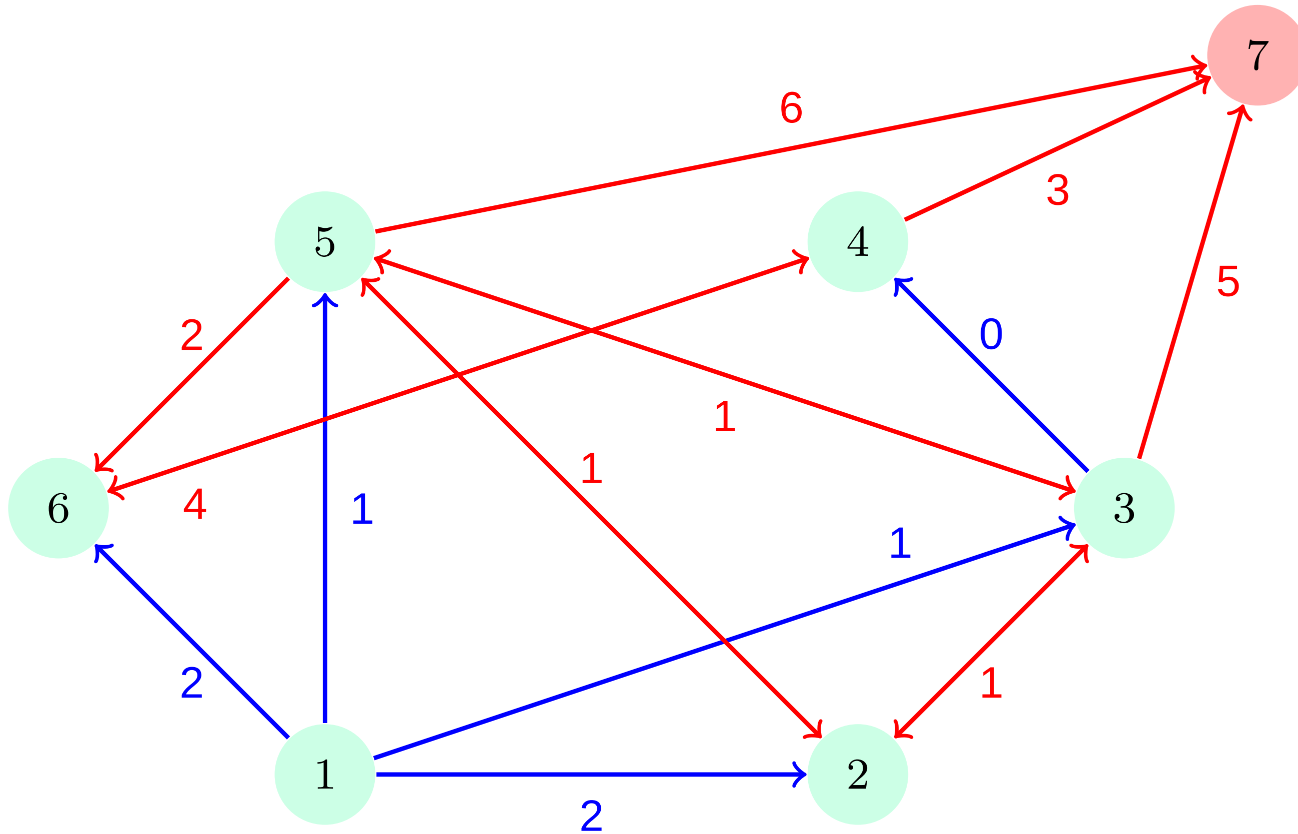


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

acomode 6 ($d_6 = 2$ é mínimo)

Exemplo com $s = 1$

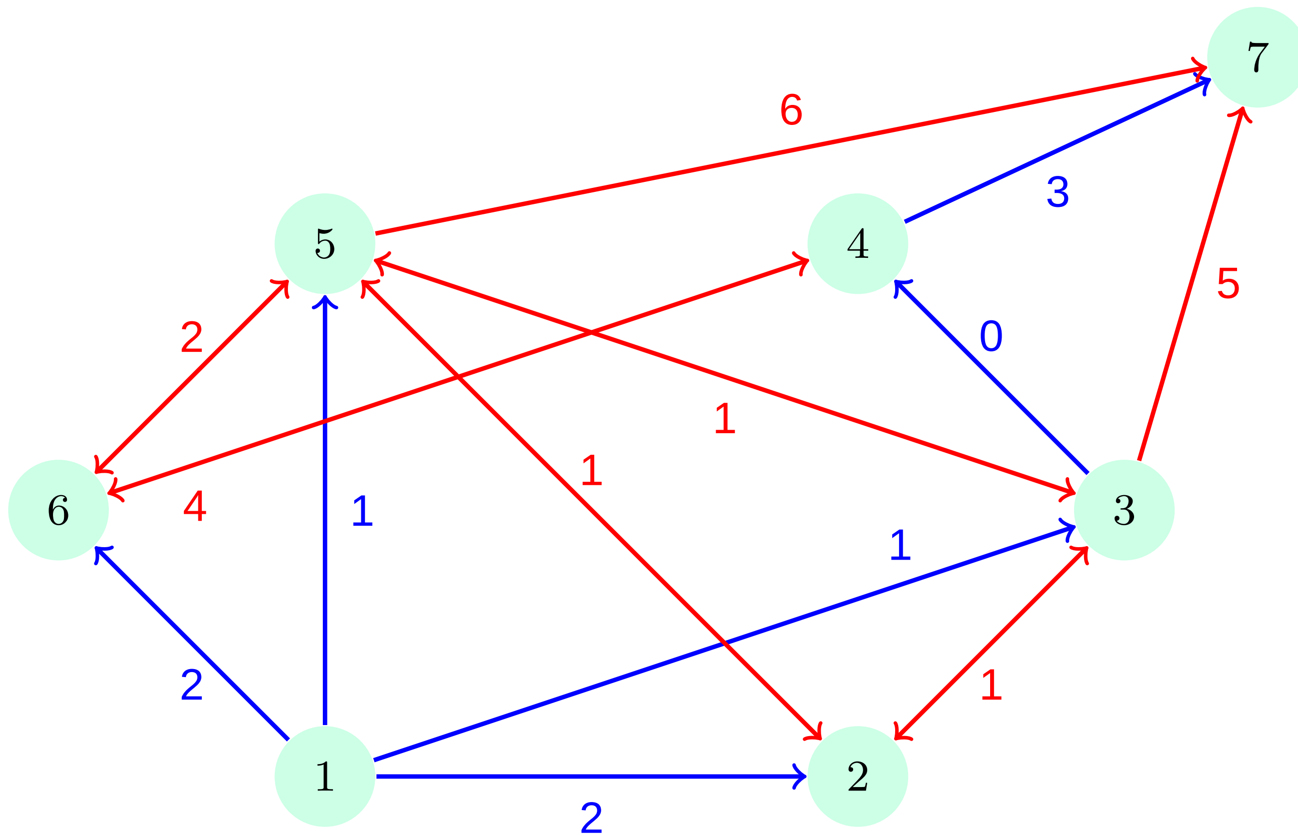


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

relaxe $\delta^+(6)$

Exemplo com $s = 1$

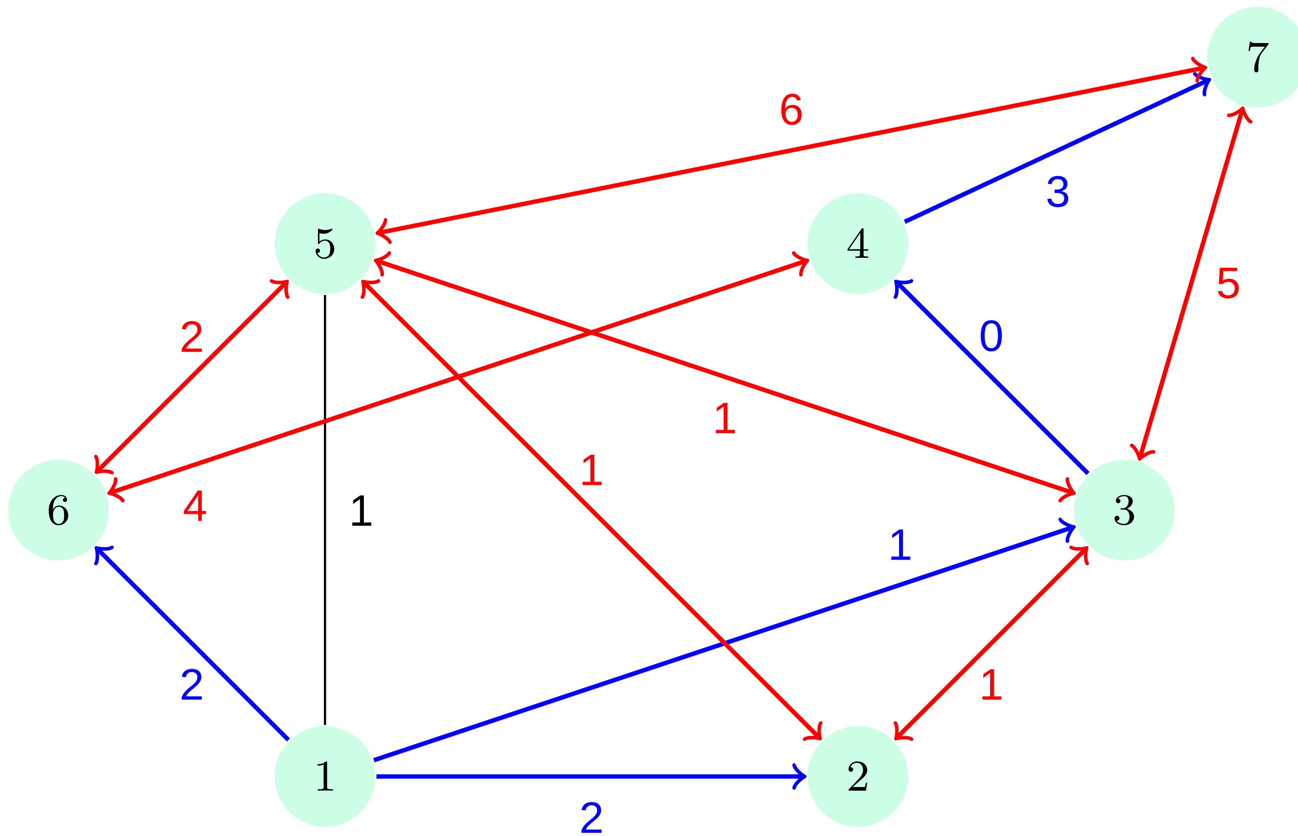


$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

acomode 7 ($d_7 = 4$ é mínimo)

Exemplo com $s = 1$



d :

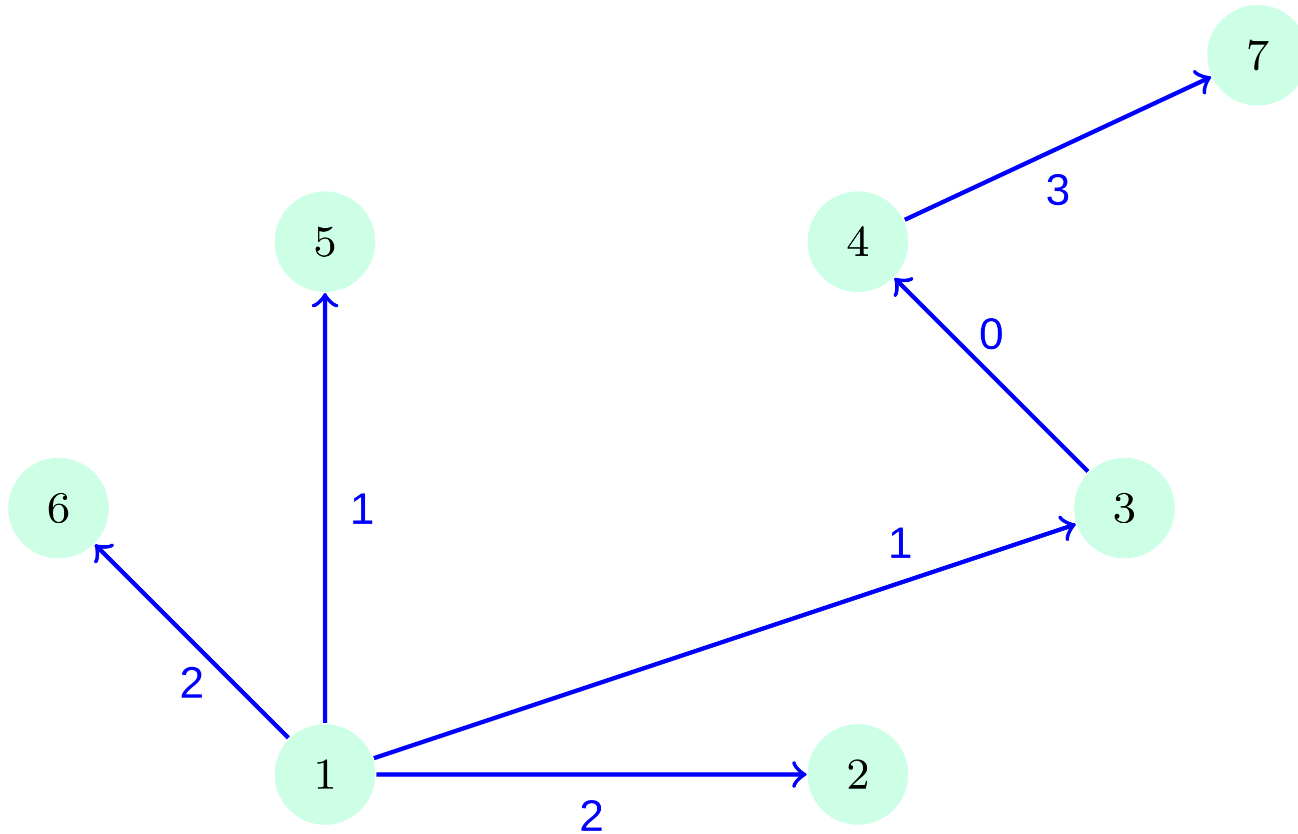
1	2	3	4	5	6	7
0	2	1	1	1	2	4

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	4

relaxe $\delta^+(7)$

Exemplo com $s = 1$



d :

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

p :

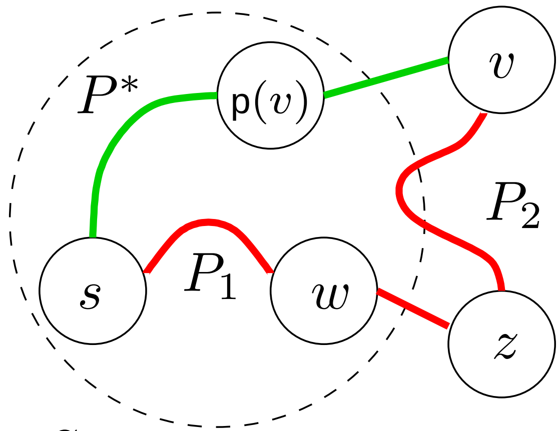
1	2	3	4	5	6	7
1	1	1	3	1	1	4

Uma ACC ótima

O algoritmo está correto 1/2

Thm.

Em qualquer iteração e para cada $v \in V$, d_v é o custo de um CC $s \rightarrow v$ onde todos os predecessores de v estão acomodados



Proof

Por indução na it. k. Seja S o conjunto de nós acomodados na it. $k-1$, seja v escolhido no Step 2 da it. k, e P^* o caminho $s \rightarrow v$ determinado pelo alg. Suponha $\exists$ outro caminho P de s a v com custo $c(P)$. Uma vez que $v \notin S$, tem que existir $(w, z) \in A$ com $w \in S$ e $z \notin S$ s.t. $P = P_1 \cup \{(w, z)\} \cup P_2$, onde $V(P_1) \subseteq S$. Então $c(P) = c(P_1) + c_{wz} + c(P_2) \geq c(P_1) + c_{wz}$ *(porque subtraímos $c(P_2)$)* $= d_w + c_{wz}$ *(pela indução)* $= d_z \geq d_v = c(P^*)$, de modo que P^* é um CC $s \rightarrow v$

O algoritmo está correto 2/2

- Resta provar: no fim do algoritmo, todo nó está acomodado
- Similar à prova que **Graph Scanning** atinge todos os vértices em um grafo (Módulo 6)
- Deixado como exercício

Implementação

- A escolha do mínimo no Step 2 ocorre entre nós não acomodados e alcançados $\Rightarrow$ **é mantida uma estrutura de dados contendo tais nós**
- Estrutura de dados que fornece o mínimo de algo em tempo constante: **fila de prioridades: heap**
- Quando o arco (u, v) é relaxado e v já foi alcançado, a prioridade d_v pode ter que ser atualizada
- A prioridade é atualizada deletando-se e depois reinserindo-se o elemento (nó) com a nova prioridade.

Pseudocódigo

```
∀v ∈ V  d_v = ∞, d_s = 0;
∀v ∈ V  p_v = s;
Q.insert(s, d_s);
while Q ≠ ∅ do
    Let u = Q.popMin();
    for (u, v) ∈ δ⁺(u) do
        Let Δ = d_u + c_uv;
        if Δ < d_v then
            Let d_v = Δ;
            Let p_v = u;
            Q.delete(v); // se v ∉ Q isto não faz nada
            Q.insert(v, d_v);
        end if
    end for
end while
```

Complexidade de pior caso

- Cada nó é acomodado exatamente uma vez (por que?)
  $\Rightarrow$
  1. `popMin()` é chamado $O(n)$ vezes $\Rightarrow O(n \log n)$
  2. cada arco é relaxado exatamente uma vez
     $\Rightarrow O(m \log n)$
- Isto resulta em um algoritmo $O((n + m) \log n)$
- Pior do que $O(n^2)$ se o grafo for denso, entretanto grafos na prática são normalmente esparsos
- Pode ser melhorado para $O(m + n \log n)$ com estruturas de dados mais refinadas

Caminhos mais curtos ponto a ponto

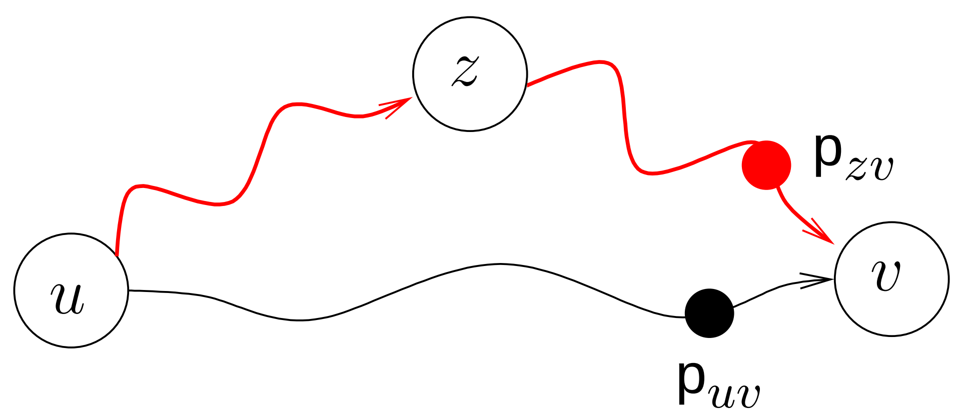
- O P2PSP de s a t em dígrafos ponderados não-negativamente pode ser resolvido pelo algoritmo de Dijkstra.
- Simplesmente termine quando t for acomodado
- Insira o código a seguir entre os Steps 5 e 6:

if $u = t$ **then**
  exit;
end if

Algoritmo de Floyd-Warshall’s

Resolve ASP

- Resolve o ASP em um dígrafo sem ciclos negativos
- Estruturas de dados: duas $n \times n$ matrizes d, p
  - d_{uv} = custo de CC $u \to v$
  - p_{uv} = predecessor de v no CC a partir de u
- Para cada nó z e par u, v de nós, veja se CC $u \to v$ pode ser melhorado passando-se por z



- Se for o caso, atualize d_{uv} para $d_{uz} + d_{zv}$ e p_{uv} para p_{zv}

O algoritmo mais simples!

$\forall u, v \in V \; d_{uv} = \begin{cases} c_{uv} & \text{se } (u,v) \in A \\ \infty & \text{c.c.} \end{cases}$

$\forall u, v \in V \; \mathsf{p}_{uv} = u$

for $z \in V$ **do**

  for $u \in V$ **do**

    for $v \in V$ **do**

      $\Delta = d_{uz} + d_{zv}$;

      if $\Delta < d_{uv}$ **then**

        $d_{uv} = \Delta$;

        $\mathsf{p}_{uv} = \mathsf{p}_{zv}$;

      end if

    end for

  end for

end for

Observações

- **Complexidade de pior caso**: claramente $O(n^3)$
- **Algoritmo está correto**: todas as triangulações possíveis são testadas
- Também pode resolver o problema de identificação de ciclos negativos:
  - Assuma que exista um ciclo negativo passando por u
  - Quando $u = v$, as triangulações irão eventualmente resultar em $d_{uu} < 0$
  - Sempre que isto acontecer, termine: um ciclo negativo foi encontrado
  - Após Step 6, insira o código:

    if $\Delta < 0$ **then**
      exit;
    end if

Fim do Módulo 9

e FIM DO CURSO!

Obrigado pela sua atenção