Aula 11 - Algoritmo de Dijkstra

Luís Felipe

UFF

30 de Abril de 2026

Luís Felipe

30/04/26

Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10).

Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Como seria para saber a distância dentre todos os pares de vértices do grafo?

- Basta executarmos uma BFS para cada vértice do grafo.

Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Como seria para saber a distância dentre todos os pares de vértices do grafo?

- Basta executarmos uma BFS para cada vértice do grafo. Complexidade: $O(n(n + m))$.

Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Como seria para saber a distância dentre todos os pares de vértices do grafo?

- Basta executarmos uma BFS para cada vértice do grafo. Complexidade: $O(n(n + m))$. Para grafos densos, $m \approx n^2$, assim, essa abordagem tem a complexidade de $O(n^3)$.

Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Como seria para saber a distância dentre todos os pares de vértices do grafo?

- Basta executarmos uma BFS para cada vértice do grafo. Complexidade: $O(n(n + m))$. Para grafos densos, $m \approx n^2$, assim, essa abordagem tem a complexidade de $O(n^3)$.

E se o grafo tiver pesos nas arestas que sejam valores inteiros positivos e "pequenos", é possível adaptar essas abordagens acima? Faça subdivisão de cada aresta com peso, tal como abaixo.

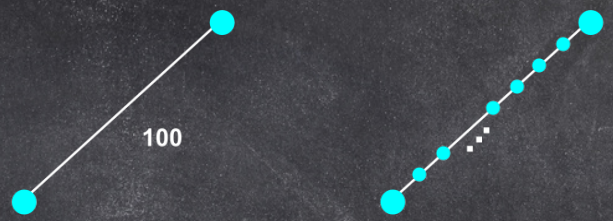
Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Como seria para saber a distância dentre todos os pares de vértices do grafo?

- Basta executarmos uma BFS para cada vértice do grafo. Complexidade: $O(n(n + m))$. Para grafos densos, $m \approx n^2$, assim, essa abordagem tem a complexidade de $O(n^3)$.

E se o grafo tiver pesos nas arestas que sejam valores inteiros positivos e "pequenos", é possível adaptar essas abordagens acima? Faça subdivisão de cada aresta com peso, tal como abaixo.



Obs.: A complexidade passa a ser em função da quantidade de vértices no novo grafo. Isso se torna eficiente? De modo geral não.

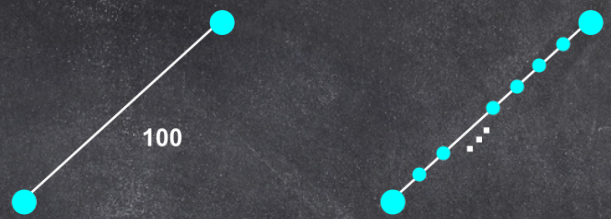
Distâncias entre vértices

Vimos que para um dado vértice x, para sabermos a distância de x para cada vértice do grafo, basta executarmos uma BFS com raiz em x (Aula 10). Complexidade: $O(n + m)$.

Como seria para saber a distância dentre todos os pares de vértices do grafo?

- Basta executarmos uma BFS para cada vértice do grafo. Complexidade: $O(n(n + m))$. Para grafos densos, $m \approx n^2$, assim, essa abordagem tem a complexidade de $O(n^3)$.

E se o grafo tiver pesos nas arestas que sejam valores inteiros positivos e "pequenos", é possível adaptar essas abordagens acima? Faça subdivisão de cada aresta com peso, tal como abaixo.



Obs.: A complexidade passa a ser em função da quantidade de vértices no novo grafo. Isso se torna eficiente? De modo geral não.
É possível ter um algoritmo mais eficiente para determinar distâncias?

Luís Felipe

30/04/26

Algoritmo de Dijkstra

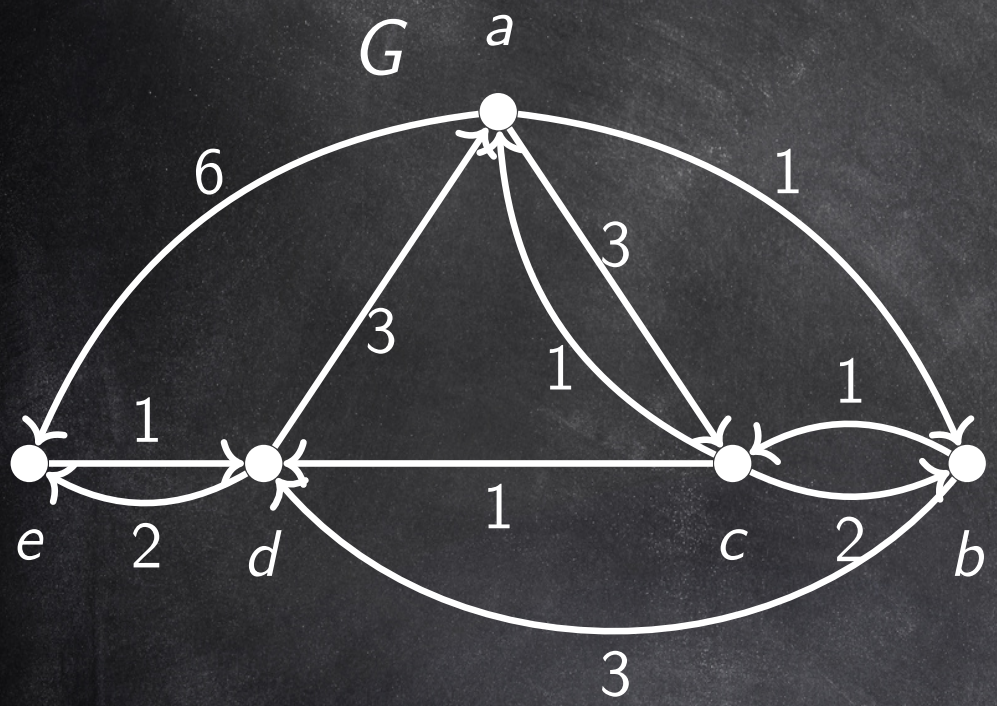
Problema: Dado um dígrafo G com pesos positivos nas arestas, determinar caminhos mínimos de um vértice v a todos os demais vértices do dígrafo.

Luís Felipe

30/04/26

Algoritmo de Dijkstra

Problema: Dado um dígrafo G com pesos positivos nas arestas, determinar caminhos mínimos de um vértice v a todos os demais vértices do dígrafo.



matriz de pesos W:

	a	b	c	d	e
a	0	1	3	∞	6
b	∞	0	1	3	∞
c	1	2	0	1	∞
d	3	∞	∞	0	2
e	∞	∞	∞	1	0

$$W(x,y) = \begin{cases} 0, \text{ se } x = y \\ \text{peso da aresta } xy, \text{ se houver } (x \neq y) \\ \infty, \text{ se não houver aresta } xy \ (x \neq y) \end{cases}$$

Luís Felipe

30/04/26

Ideia do Algoritmo de Dijkstra

Problema:

- Dado um dígrafo G com pesos positivos nas arestas e um vértice raiz r,
- queremos determinar a distância de r a todos os vértices do grafo.

Luís Felipe

30/04/26

Ideia do Algoritmo de Dijkstra

Problema:

- Dado um dígrafo G com pesos positivos nas arestas e um vértice raiz r,
- queremos determinar a distância de r a todos os vértices do grafo.

Ideia central:

- Mantemos para cada vértice v um rótulo $L(v)$ que representa a melhor distância conhecida de r até v.

Luís Felipe

30/04/26

Ideia do Algoritmo de Dijkstra

Problema:

- Dado um dígrafo G com pesos positivos nas arestas e um vértice raiz r,
- queremos determinar a distância de r a todos os vértices do grafo.

Ideia central:

- Mantemos para cada vértice v um rótulo $L(v)$ que representa a melhor distância conhecida de r até v.
- Inicialmente:

$$L(r) = 0 \qquad L(v) = \infty \text{ para } v \neq r$$

Luís Felipe

30/04/26

Ideia do Algoritmo de Dijkstra

Problema:

- Dado um dígrafo G com pesos positivos nas arestas e um vértice raiz r,
- queremos determinar a distância de r a todos os vértices do grafo.

Ideia central:

- Mantemos para cada vértice v um rótulo $L(v)$ que representa a melhor distância conhecida de r até v.
- Inicialmente:

$$L(r) = 0 \qquad L(v) = \infty \text{ para } v \neq r$$

- A cada passo escolhemos o vértice com menor rótulo ainda não processado (abordagem gulosa).

Luís Felipe

30/04/26

Ideia do Algoritmo de Dijkstra

Problema:

- Dado um dígrafo G com pesos positivos nas arestas e um vértice raiz r,
- queremos determinar a distância de r a todos os vértices do grafo.

Ideia central:

- Mantemos para cada vértice v um rótulo $L(v)$ que representa a melhor distância conhecida de r até v.
- Inicialmente:

$$L(r) = 0 \qquad L(v) = \infty \text{ para } v \neq r$$

- A cada passo escolhemos o vértice com menor rótulo ainda não processado (abordagem gulosa).
- Ao processar este vértice v, tentamos melhorar os rótulos de seus vizinhos.

Luís Felipe

30/04/26

Construção da solução

Durante a execução do algoritmo:

- Os vértices já processados formam um conjunto cuja distância mínima a partir da raiz já é conhecida.

Luís Felipe

30/04/26

Construção da solução

Durante a execução do algoritmo:

- Os vértices já processados formam um conjunto cuja distância mínima a partir da raiz já é conhecida.
- Sempre escolhemos o vértice v com menor rótulo entre os ainda não processados.

Luís Felipe

30/04/26

Construção da solução

Durante a execução do algoritmo:

- Os vértices já processados formam um conjunto cuja distância mínima a partir da raiz já é conhecida.
- Sempre escolhemos o vértice v com menor rótulo entre os ainda não processados.
- Esse valor é definitivo: nenhum caminho futuro poderá produzir uma distância menor.

Luís Felipe

30/04/26

Construção da solução

Durante a execução do algoritmo:

- Os vértices já processados formam um conjunto cuja distância mínima a partir da raiz já é conhecida.
- Sempre escolhemos o vértice v com menor rótulo entre os ainda não processados.
- Esse valor é definitivo: nenhum caminho futuro poderá produzir uma distância menor.
- Em seguida, verificamos cada aresta vw e testamos:

$$L(v) + W(v, w) < L(w)$$

Ou seja:

$$L(w) = \min\{L(w), L(v) + W(v, w)\}$$

Luís Felipe

30/04/26

Construção da solução

Durante a execução do algoritmo:

- Os vértices já processados formam um conjunto cuja distância mínima a partir da raiz já é conhecida.
- Sempre escolhemos o vértice v com menor rótulo entre os ainda não processados.
- Esse valor é definitivo: nenhum caminho futuro poderá produzir uma distância menor.
- Em seguida, verificamos cada aresta vw e testamos:

$$L(v) + W(v, w) < L(w)$$

Ou seja:

$$L(w) = \min\{L(w), L(v) + W(v, w)\}$$

- Queremos ver se a distância de r até v adicionando a aresta vw diminui o caminho conhecido de r até w.
- Se a desigualdade for verdadeira, atualizamos o rótulo de w.

Luís Felipe

30/04/26

Uso de fila de prioridades

Durante a execução do algoritmo, cada vértice v possui um rótulo $L(v)$ que representa a melhor distância conhecida da raiz até v.

Luís Felipe

30/04/26

Uso de fila de prioridades

Durante a execução do algoritmo, cada vértice v possui um rótulo $L(v)$ que representa a melhor distância conhecida da raiz até v.

A cada passo precisamos:

- escolher o vértice não processado com menor valor de $L(v)$;

Luís Felipe

30/04/26

Uso de fila de prioridades

Durante a execução do algoritmo, cada vértice v possui um rótulo $L(v)$ que representa a melhor distância conhecida da raiz até v.

A cada passo precisamos:

- escolher o vértice não processado com menor valor de $L(v)$;
- remover esse vértice da estrutura de dados;

Luís Felipe

30/04/26

Uso de fila de prioridades

Durante a execução do algoritmo, cada vértice v possui um rótulo $L(v)$ que representa a melhor distância conhecida da raiz até v.

A cada passo precisamos:

- escolher o vértice não processado com menor valor de $L(v)$;
- remover esse vértice da estrutura de dados;
- atualizar os rótulos de seus vizinhos.

Luís Felipe

30/04/26

Uso de fila de prioridades

Durante a execução do algoritmo, cada vértice v possui um rótulo $L(v)$ que representa a melhor distância conhecida da raiz até v.

A cada passo precisamos:

- escolher o vértice não processado com menor valor de $L(v)$;
- remover esse vértice da estrutura de dados;
- atualizar os rótulos de seus vizinhos.

Para realizar essas operações de forma eficiente, utilizamos uma fila de prioridades.

Luís Felipe

30/04/26

Uso de fila de prioridades

Durante a execução do algoritmo, cada vértice v possui um rótulo $L(v)$ que representa a melhor distância conhecida da raiz até v.

A cada passo precisamos:

- escolher o vértice não processado com menor valor de $L(v)$;
- remover esse vértice da estrutura de dados;
- atualizar os rótulos de seus vizinhos.

Para realizar essas operações de forma eficiente, utilizamos uma fila de prioridades.

Nessa estrutura:

- cada vértice é armazenado com chave $L(v)$;
- o vértice com menor valor de $L(v)$ é removido primeiro.

Luís Felipe

30/04/26

Algoritmo de Dijkstra

```
1  Entrada: dígrafo G, pesos W, raiz r
2
3  para todo vertice v
4      L(v) <- infinito
5      pai(v) <- null
6
7  L(r) <- 0
8  inserir o par (L(r),r) em uma fila de prioridade D
9
10 enquanto D nao estiver vazia faca
11
12     v <- vertice com menor L(v) em D
13     remover v de D
14
15     para todo w em N_out(v) faca
16         se L(v) + W(v,w) < L(w) entao
17             L(w) <- L(v) + W(v,w)
18             pai(w) <- v
19             inserir o par (L(w),w) em D
```

Luís Felipe

30/04/26

Algoritmo de Dijkstra em Python

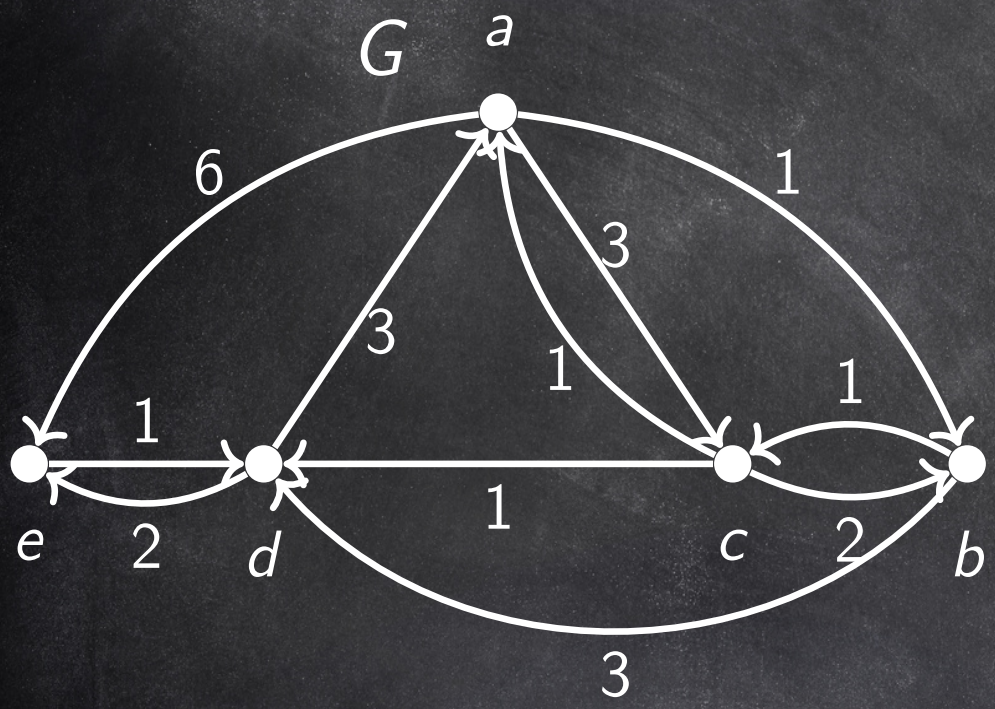
```python
import heapq

def dijkstra(G, W, r):

    n = len(G)

    L = {v: float('inf') for v in G}
    pai = {v: None for v in G}

    L[r] = 0

    D = [(0, r)]                            # fila de prioridades

    while D:

        dist_v, v = heapq.heappop(D)        # remove o par (distância, vértice) com menor distância

        if dist_v > L[v]:                   # entrada desatualizada; já existe um caminho melhor para v
            continue

        for w in G[v]:                      # w pertence a vizinhança de saída de v

            if L[v] + W[(v,w)] < L[w]:

                L[w] = L[v] + W[(v,w)]
                pai[w] = v

                heapq.heappush(D, (L[w], w))

    return L, pai
```

Luís Felipe

30/04/26

Exemplo

Raiz: a



v	a	b	c	d	e
L(v)	0	∞	∞	∞	∞
pai(v)	null	null	null	null	null

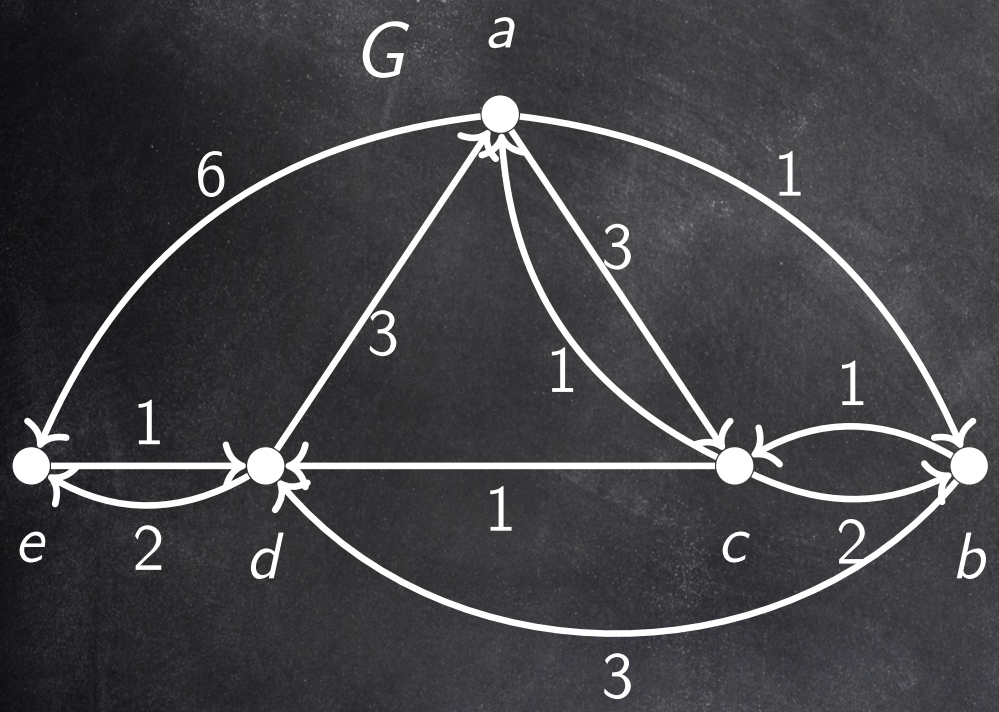
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(0, a)]$



v	a	b	c	d	e
L(v)	0	∞	∞	∞	∞
pai(v)	null	null	null	null	null

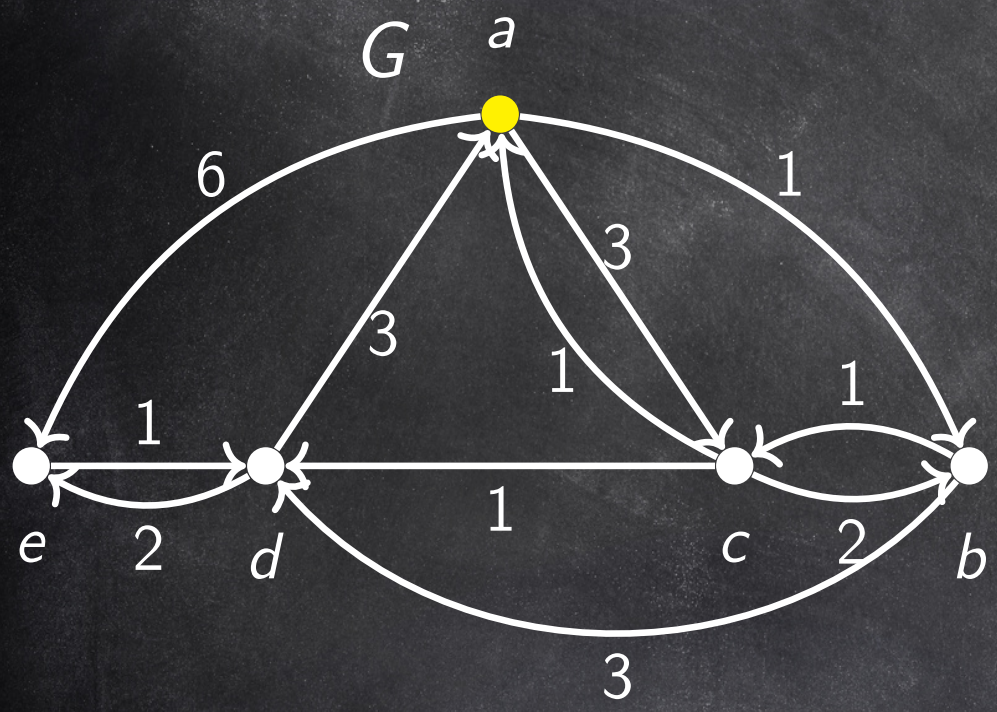
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
[ ]



v	a	b	c	d	e
L(v)	0	∞	∞	∞	∞
pai(v)	null	null	null	null	null

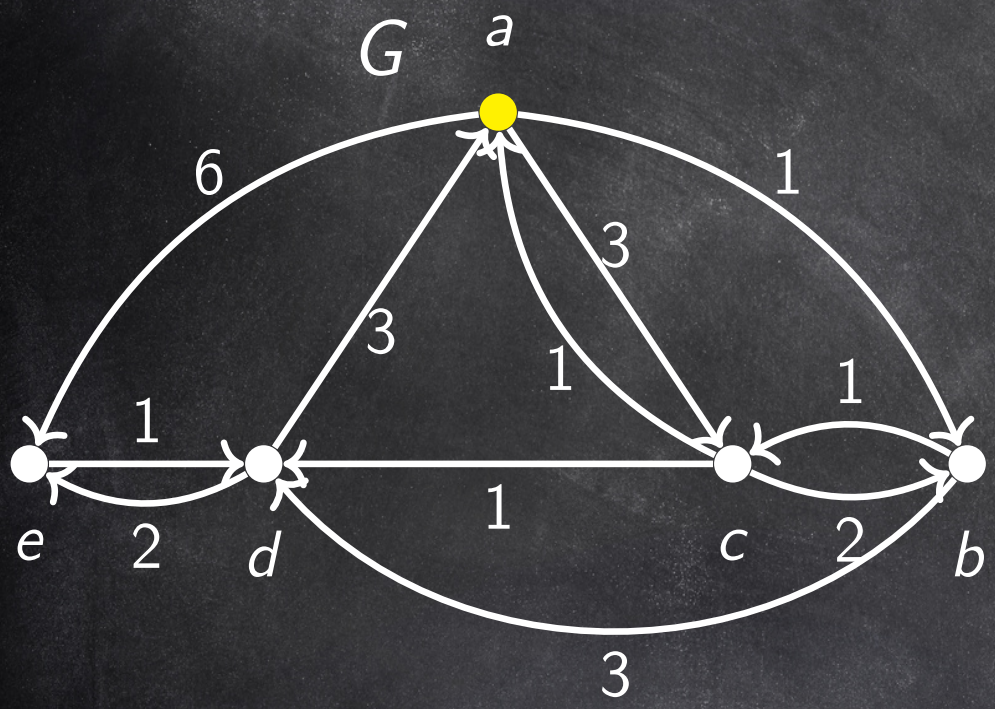
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(1, b)]$



v	a	b	c	d	e
L(v)	0	1	∞	∞	∞
pai(v)	null	a	null	null	null

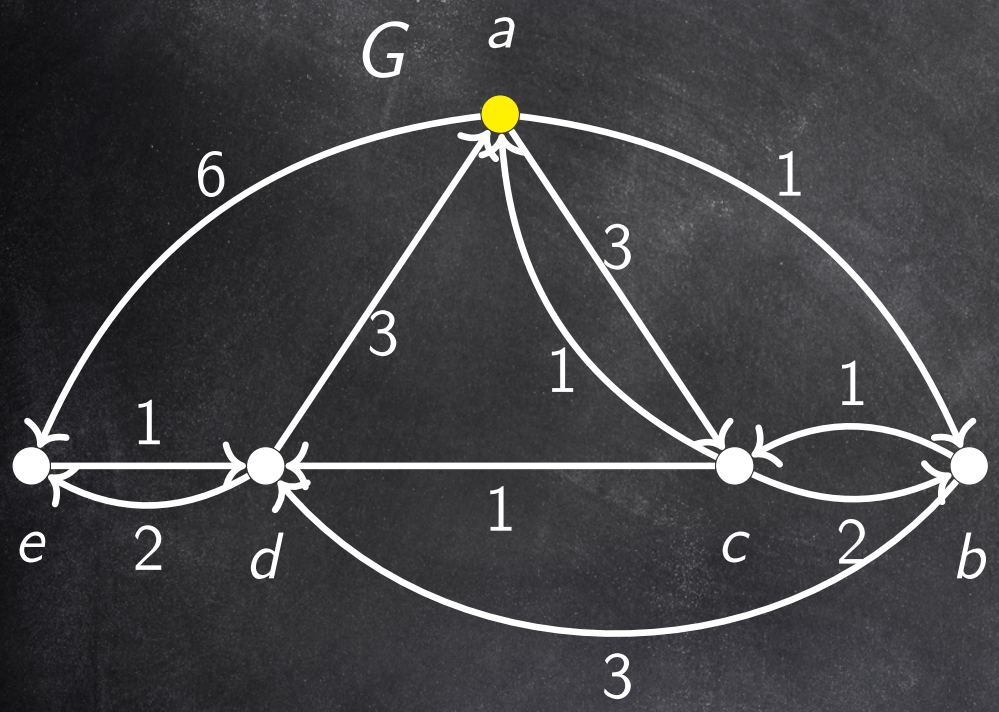
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(1, b)]$



v	a	b	c	d	e
L(v)	0	1	∞	∞	∞
pai(v)	null	a	null	null	null

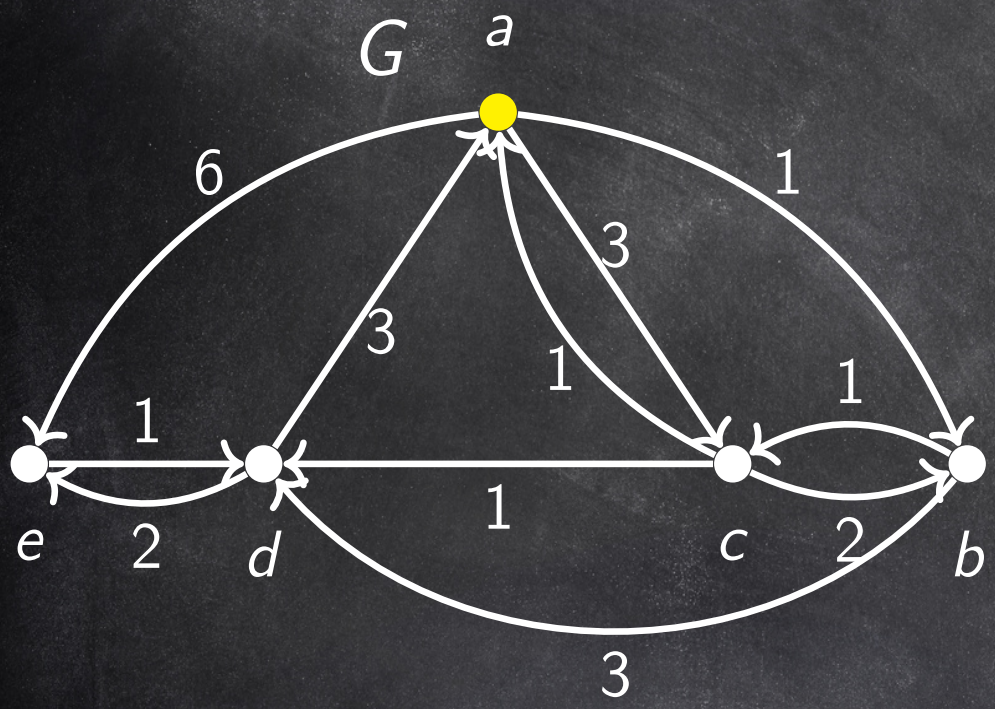
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(1, b), (3, c)]$



v	a	b	c	d	e
L(v)	0	1	3	∞	∞
pai(v)	null	a	a	null	null

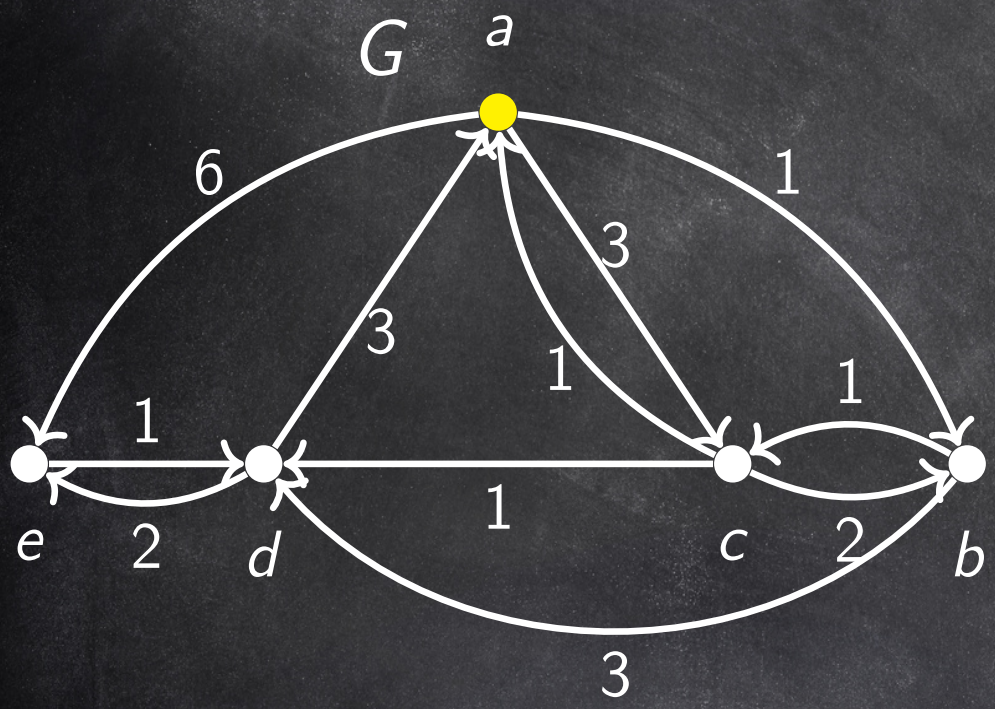
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(1, b), (3, c)]$



v	a	b	c	d	e
L(v)	0	1	3	∞	∞
pai(v)	null	a	a	null	null

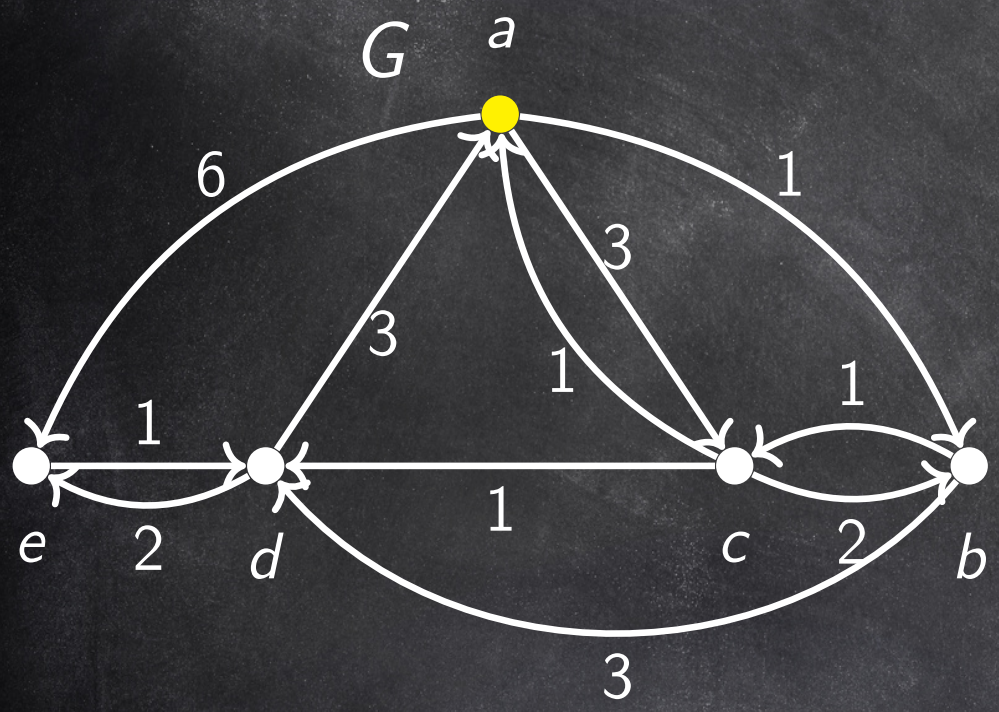
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(1, b), (3, c), (6, e)]$



v	a	b	c	d	e
L(v)	0	1	3	∞	6
pai(v)	null	a	a	null	a

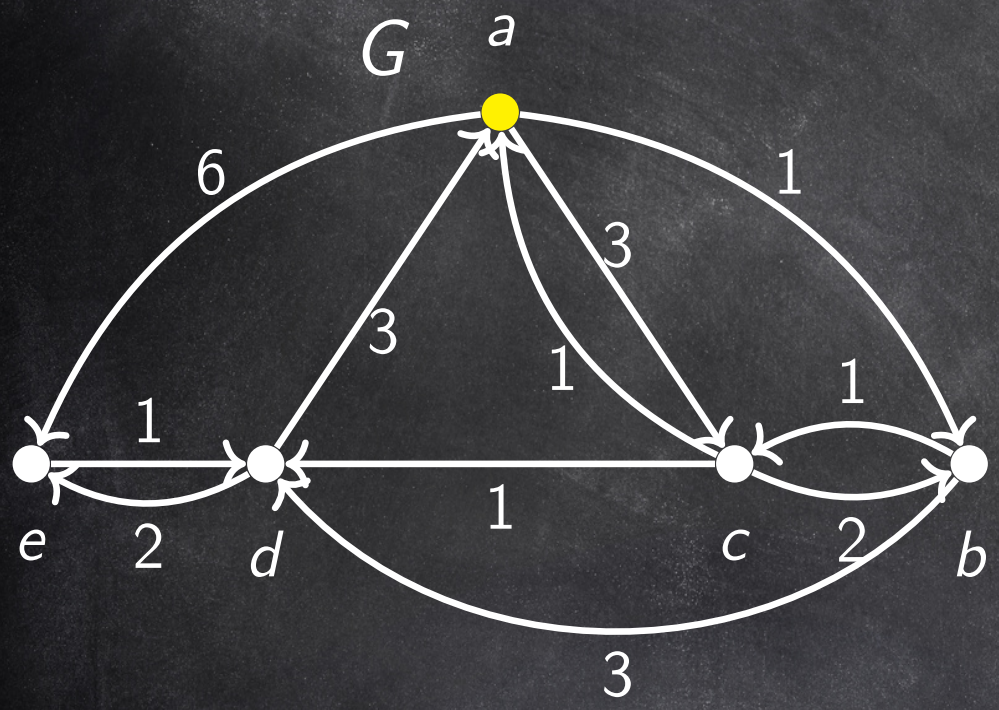
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(1, b), (3, c), (6, e)]$



v	a	b	c	d	e
L(v)	0	1	3	∞	6
pai(v)	null	a	a	null	a

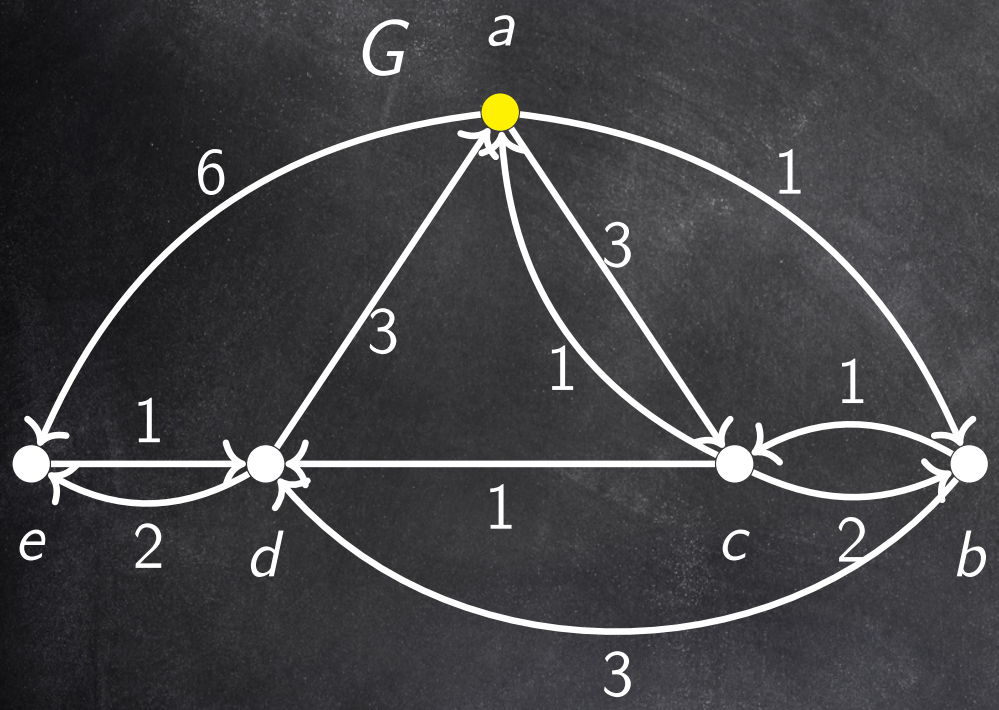
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(3, c), (6, e)]$



v	a	b	c	d	e
L(v)	0	1	3	∞	6
pai(v)	null	a	a	null	a

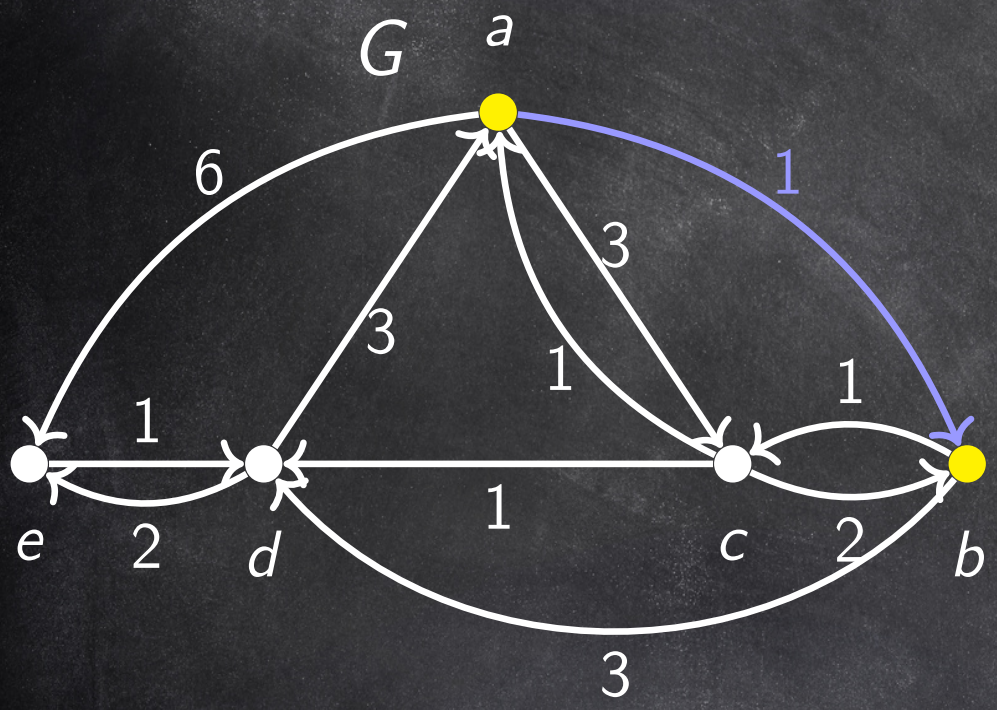
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(3, c), (6, e)]$



v	a	*b*	*c*	d	*e*
L(v)	0	1	3	∞	6
pai(v)	null	*a*	*a*	null	*a*

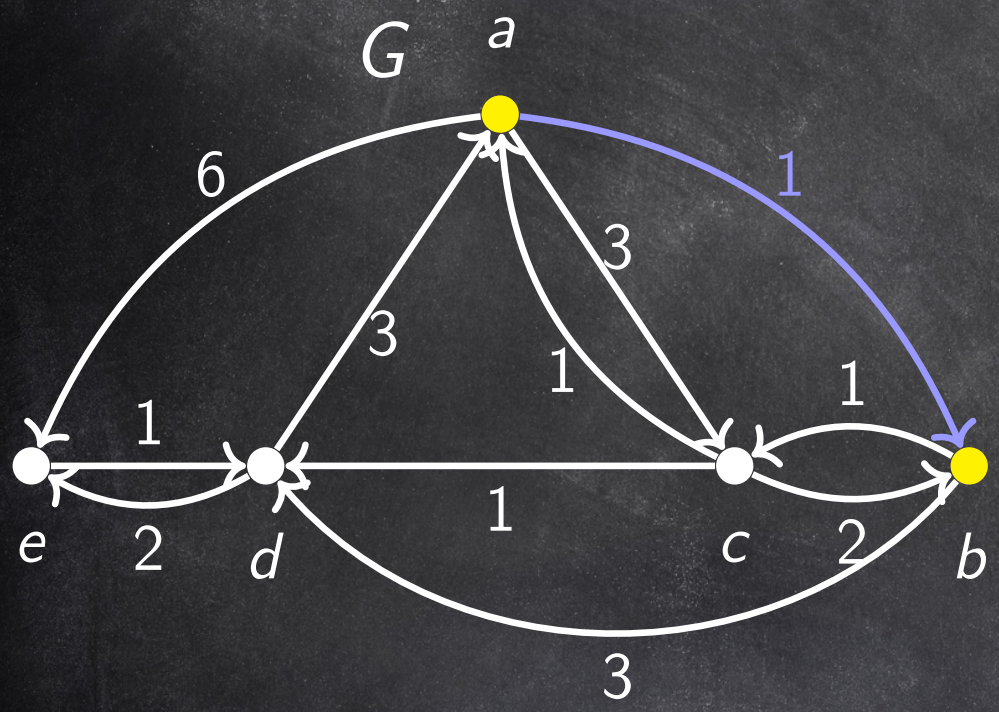
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(2, c), (3, c), (6, e)]$



v	a	*b*	*c*	d	*e*
L(v)	0	1	2	∞	6
pai(v)	null	*a*	*b*	null	*a*

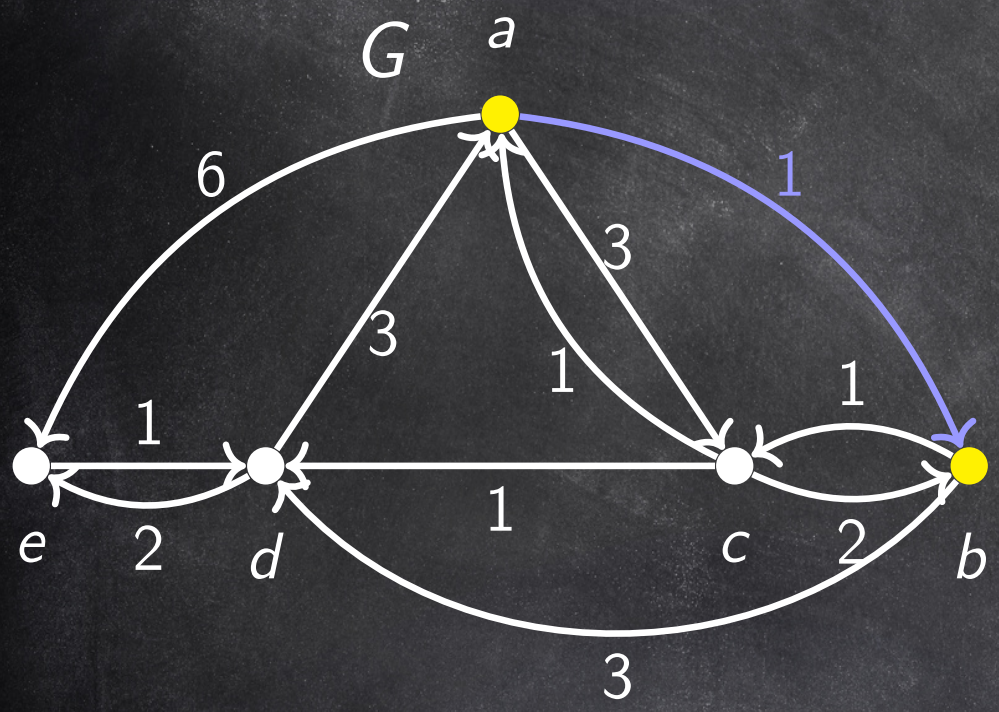
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(2, c), (4, d), (3, c), (6, e)]$



v	a	b	c	d	e
L(v)	0	1	2	4	6
pai(v)	null	a	b	b	a

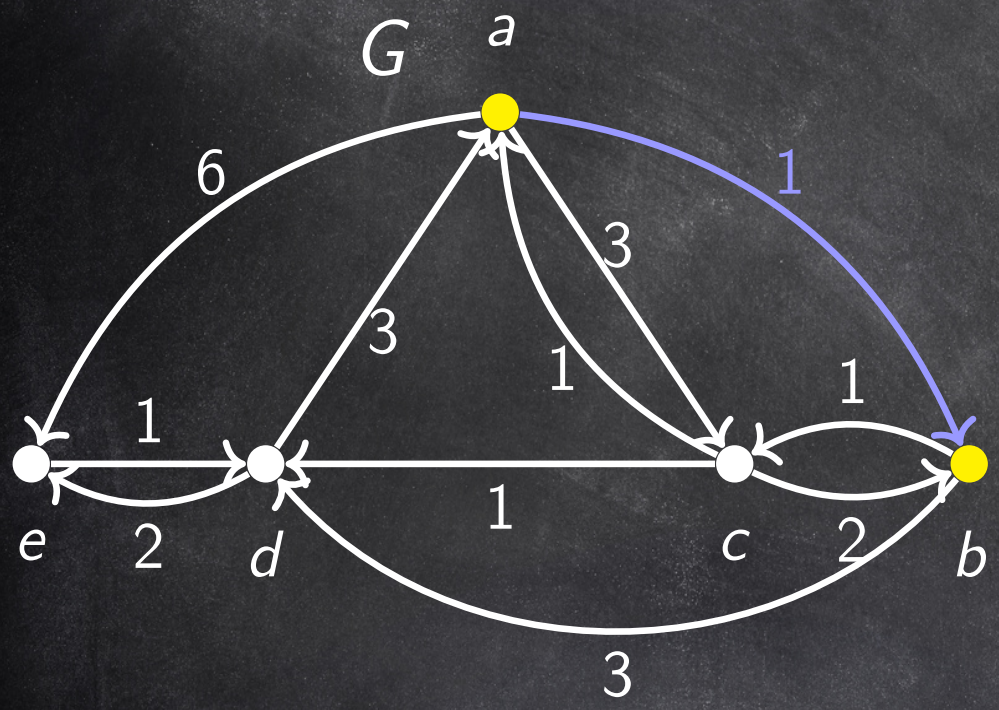
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(2, c), (4, d), (3, c), (6, e)]$



v	a	b	c	d	e
L(v)	0	1	2	4	6
pai(v)	null	a	b	b	a

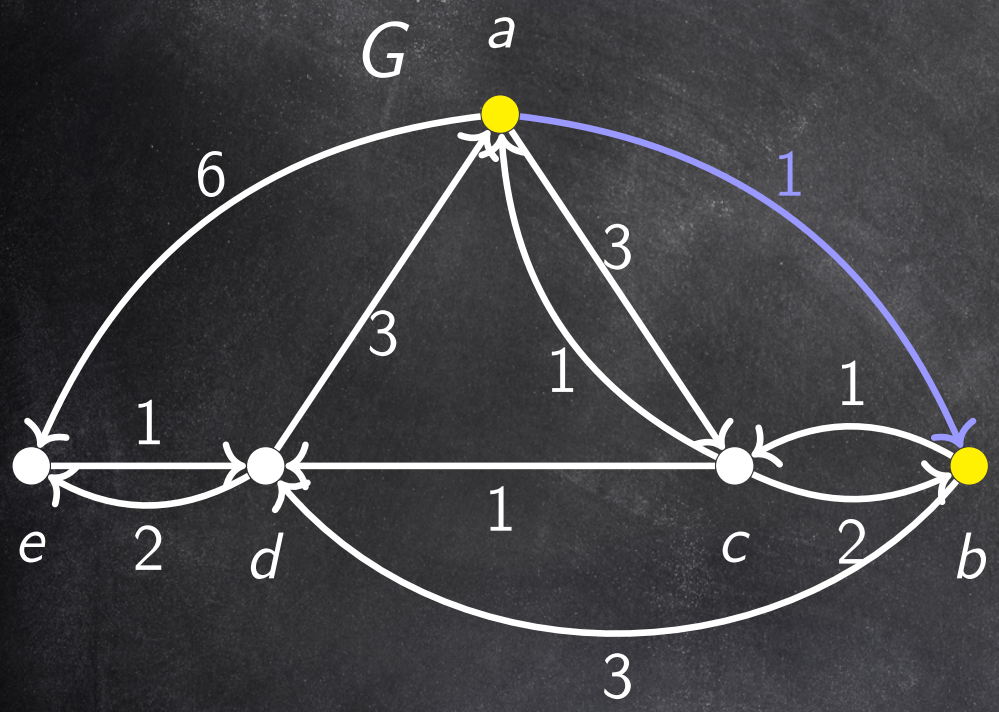
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(2, c), (4, d), (3, c), (6, e)]$



v	a	b	c	d	e
L(v)	0	1	2	4	6
pai(v)	null	a	b	b	a

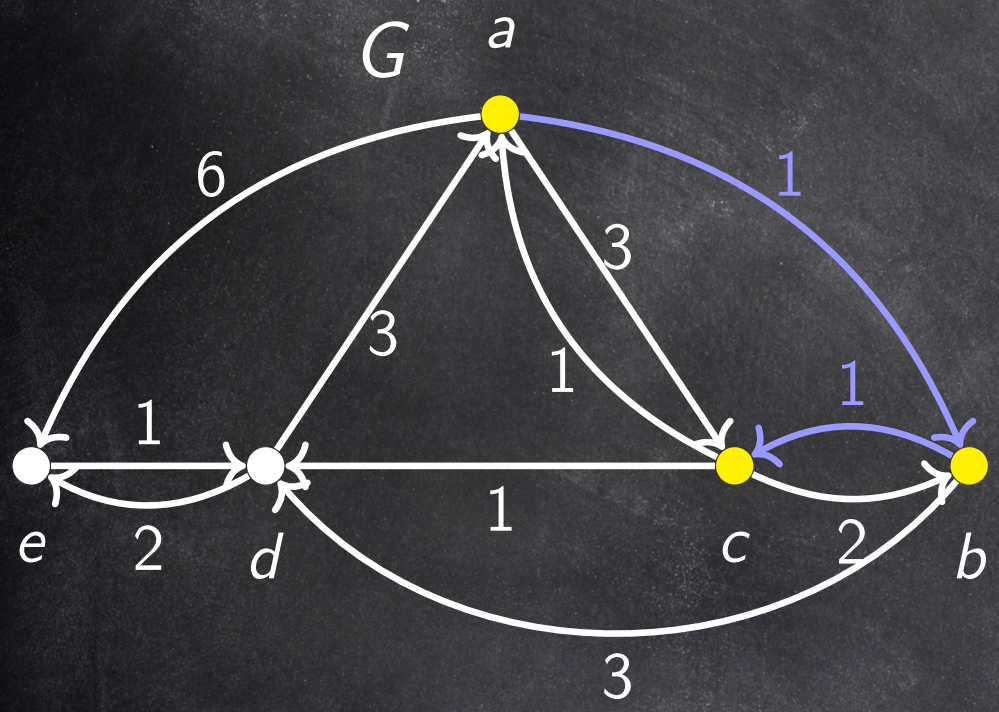
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(3, c), (6, e), (4, d)]$



v	a	b	c	d	e
L(v)	0	1	2	4	6
pai(v)	null	a	b	b	a

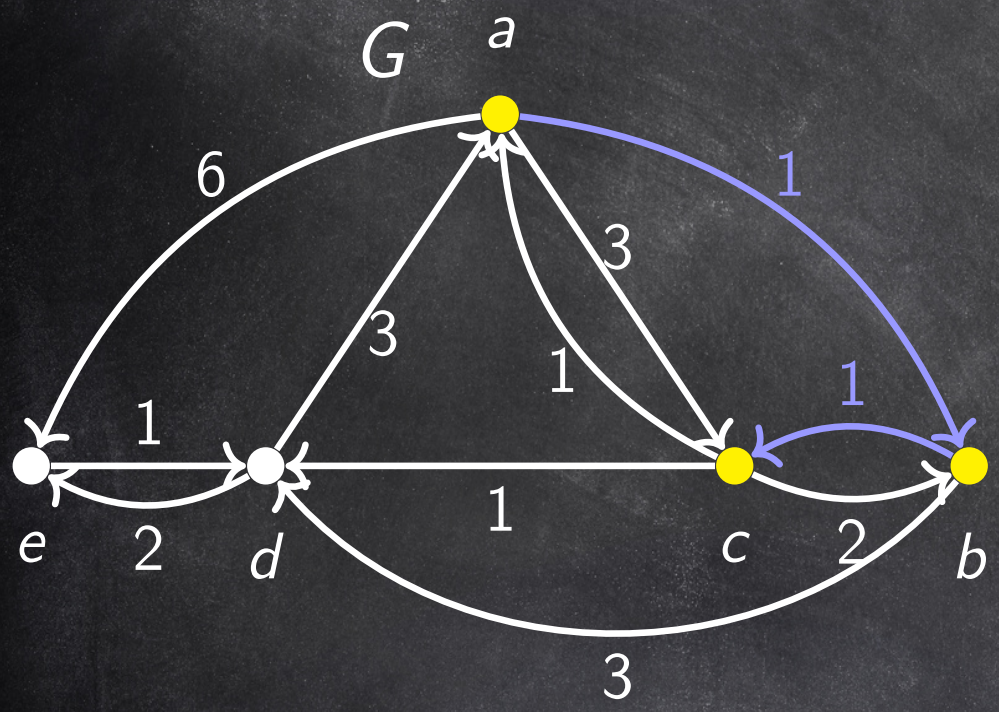
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(3, c), (3, d), (6, e), (4, d)]$



v	a	b	c	d	e
L(v)	0	1	2	3	6
pai(v)	null	a	b	c	a

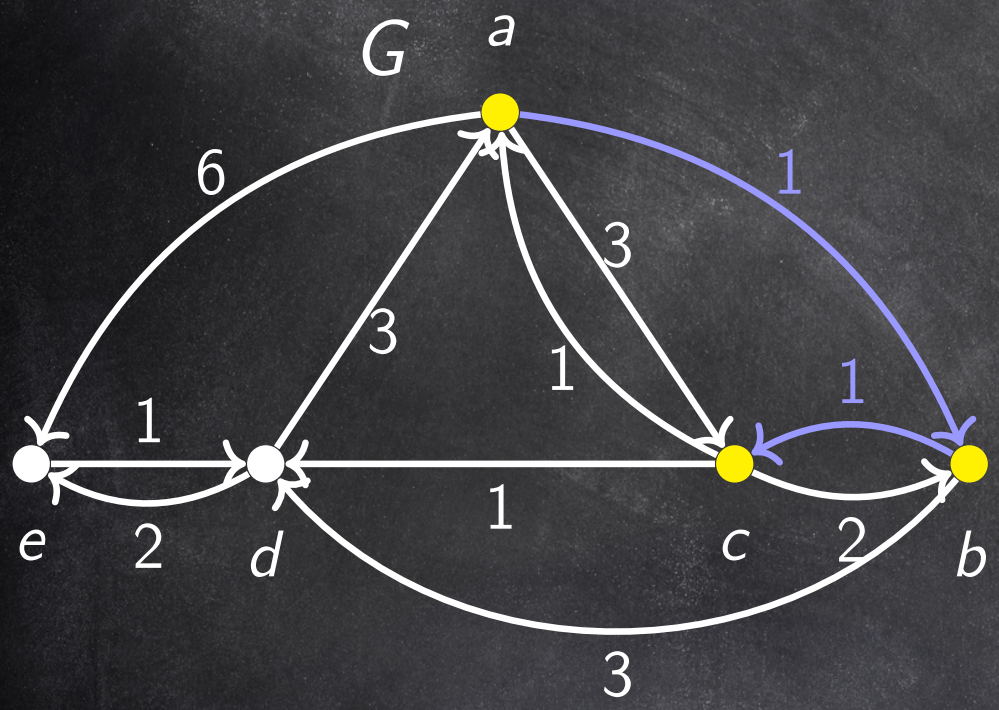
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(3, c), (3, d), (6, e), (4, d)]$



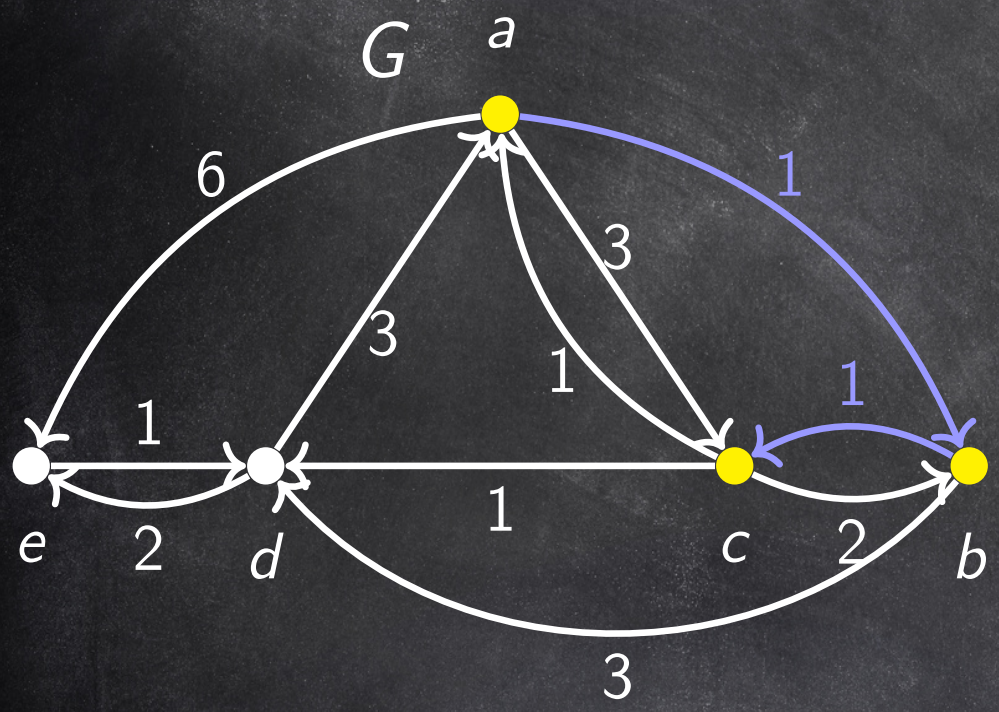
v	a	b	c	d	e
L(v)	0	1	2	3	6
pai(v)	null	a	b	c	a

Luís Felipe

30/04/26

Exemplo

Raiz: a



Heap:
$[(3, d), (4, d), (6, e)]$.
$(3, c)$ saiu da heap e tupla descartada, pois $dist_v = 3$, mas $L[c] = 2$.
Próximo a sair é $(3, d)$.

v	a	b	c	d	e
L(v)	0	1	2	3	6
pai(v)	null	a	b	c	a

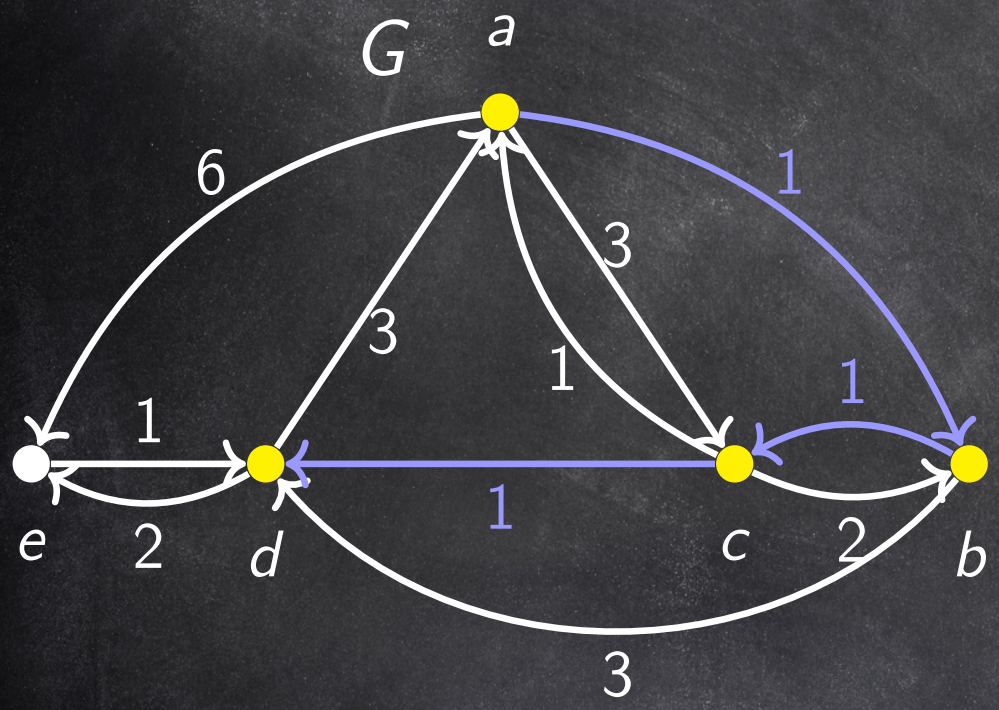
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(4, d), (6, e), (5, e)]$



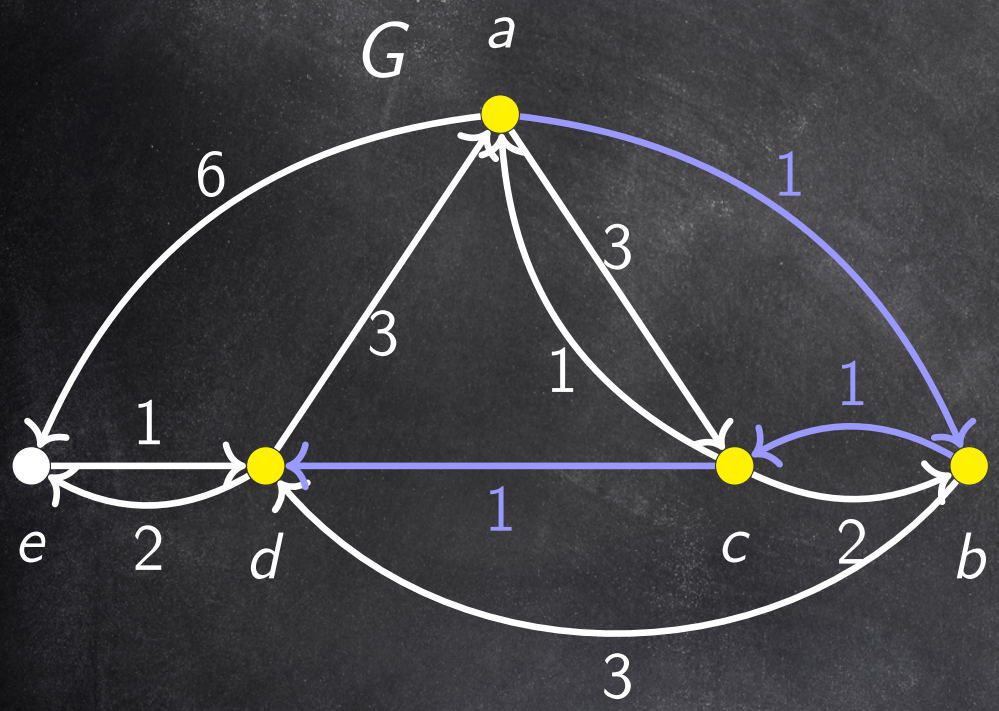
v	a	b	c	d	e
L(v)	0	1	2	3	6
pai(v)	null	a	b	c	a

Luís Felipe

30/04/26

Exemplo

Raiz: a



Heap:
$[(5, e), (6, e)]$.
$(4, d)$ saiu da heap e a tupla descartada, pois $dist_v = 4$, mas $L[d] = 3$.
Próximo a sair é $(5, e)$.

v	a	b	c	d	e
L(v)	0	1	2	3	5
pai(v)	null	a	b	c	d

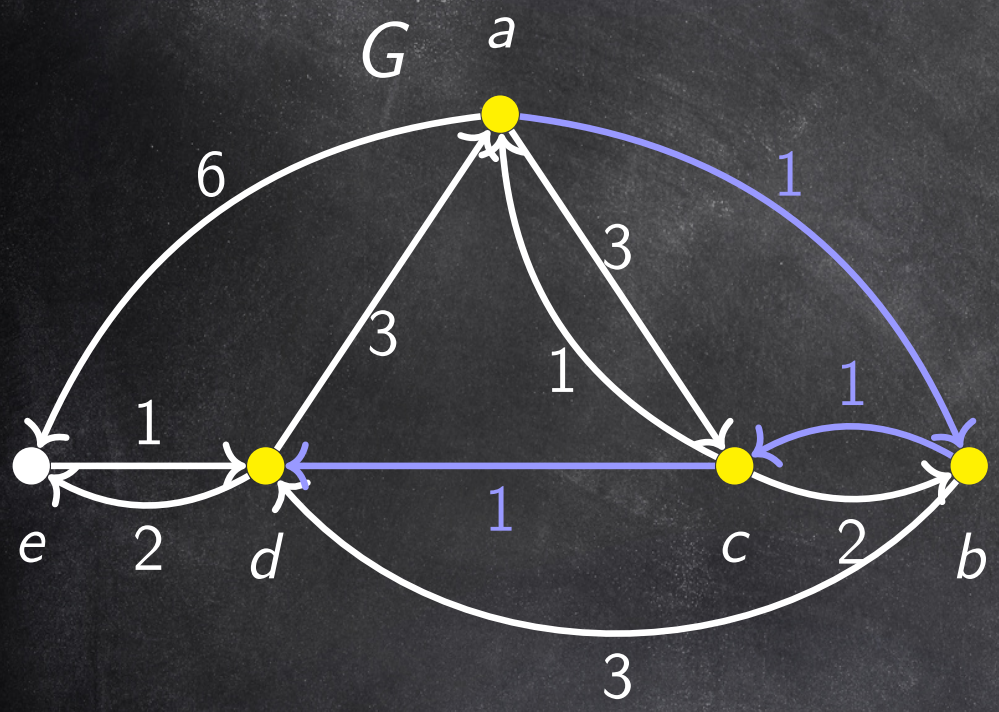
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
$[(6, e)]$



v	a	b	c	d	e
L(v)	0	1	2	3	5
pai(v)	null	a	b	c	d

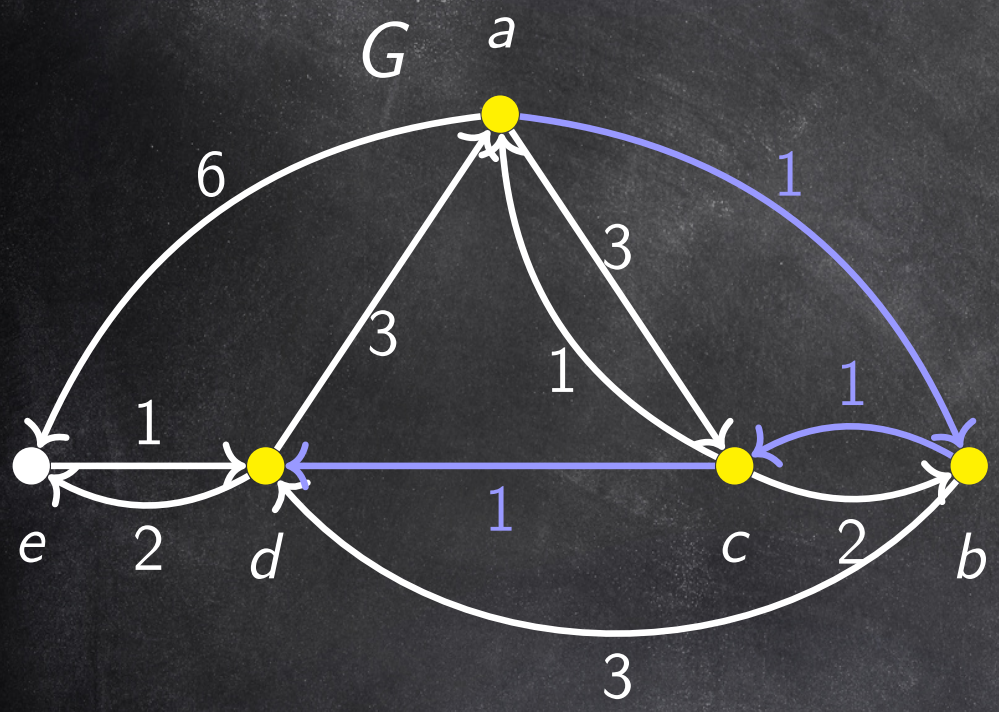
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
[ ]



v	a	b	c	d	e
L(v)	0	1	2	3	5
pai(v)	null	a	b	c	d

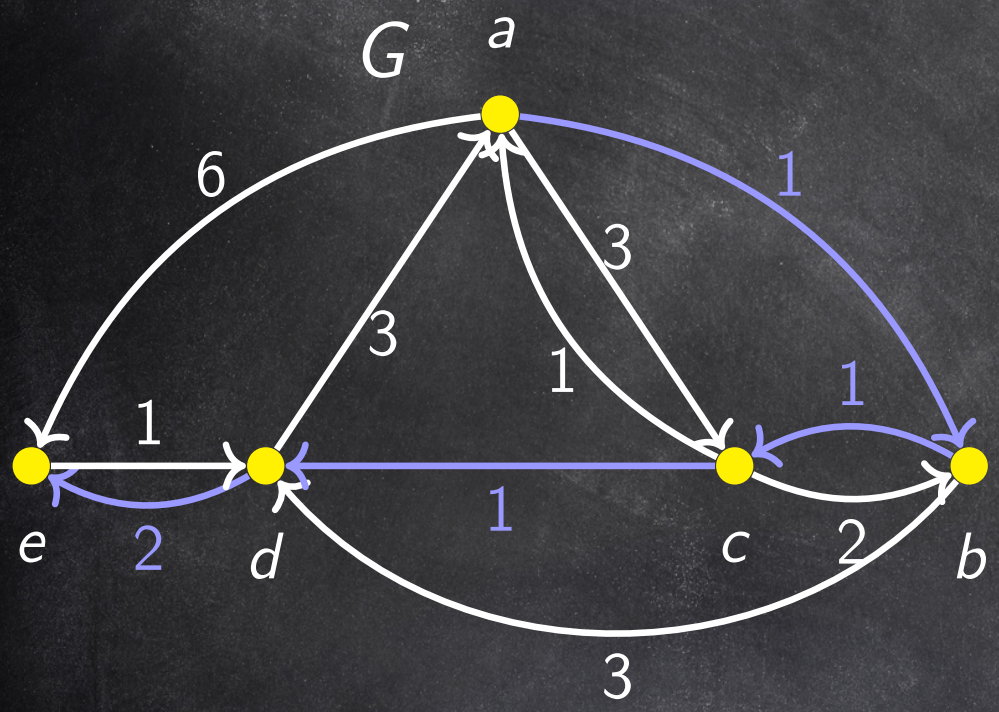
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
[ ]



v	a	b	c	d	e
L(v)	0	1	2	3	5
pai(v)	null	a	b	c	d

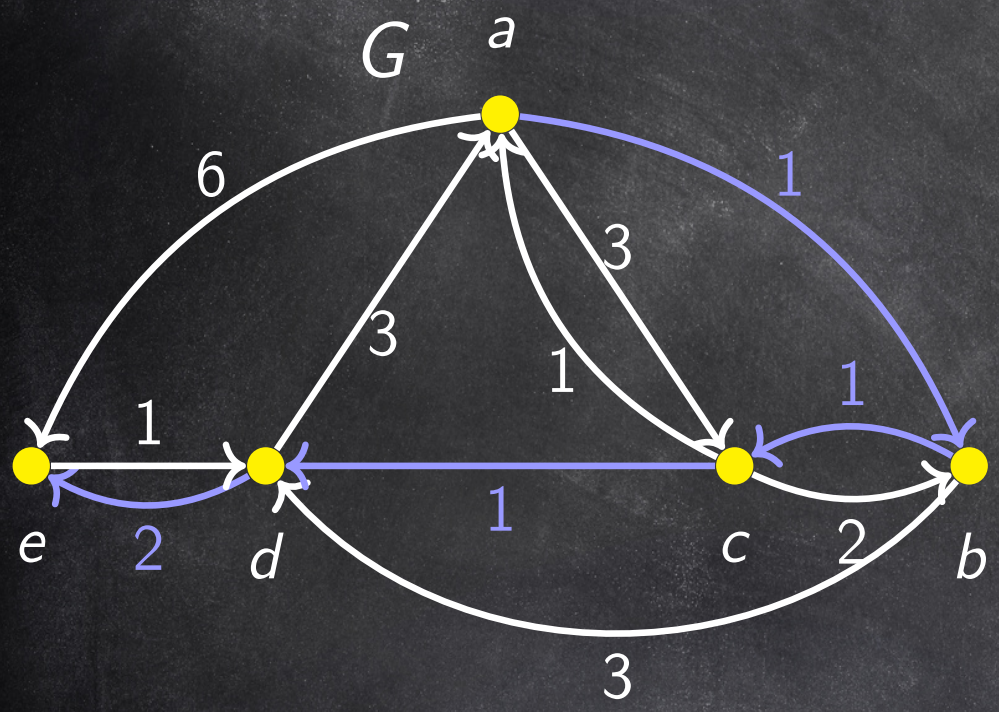
Luís Felipe

30/04/26

Exemplo

Raiz: a

Heap:
[ ]



v	a	b	c	d	e
L(v)	0	1	2	3	5
pai(v)	null	a	b	c	d

Luís Felipe

30/04/26

Complexidade do Algoritmo de Dijkstra

- A complexidade é dominada pelas seguintes operações:

Luís Felipe

30/04/26

Complexidade do Algoritmo de Dijkstra

- A complexidade é dominada pelas seguintes operações:
  - n remoções da fila de prioridades: $O(n \log n)$
  - n inserções: $O(n \log n)$
  - até m atualizações de prioridade: $O(m \log n)$

Luís Felipe

30/04/26

Complexidade do Algoritmo de Dijkstra

- A complexidade é dominada pelas seguintes operações:
  - n remoções da fila de prioridades: $O(n \log n)$
  - n inserções: $O(n \log n)$
  - até m atualizações de prioridade: $O(m \log n)$
- Logo:

$$O((n + m) \log n)$$

Luís Felipe

30/04/26

Complexidade do Algoritmo de Dijkstra

- A complexidade é dominada pelas seguintes operações:
  - n remoções da fila de prioridades: $O(n \log n)$
  - n inserções: $O(n \log n)$
  - até m atualizações de prioridade: $O(m \log n)$
- Logo:

$$O((n+m)\log n)$$

- Casos particulares:
  - grafo esparso ($m = O(n)$): $O(n \log n)$
  - grafo denso ($m = O(n^2)$): $O(n^2 \log n)$

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Hipótese: Para todo $j < i$, $L(v_j) = dist(r, v_j)$.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Hipótese: Para todo $j < i$, $L(v_j) = dist(r, v_j)$.

Passo: Seja v_i o próximo vértice removido. Considere um caminho mínimo P de r até v_i. Seja u o último vértice de P já processado e w o próximo vértice no caminho.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Hipótese: Para todo $j < i$, $L(v_j) = dist(r, v_j)$.

Passo: Seja v_i o próximo vértice removido. Considere um caminho mínimo P de r até v_i. Seja u o último vértice de P já processado e w o próximo vértice no caminho.

Pela hipótese, $L(u) = dist(r, u)$. Ao processar u, o algoritmo tenta melhorar $L(w)$ usando a aresta (u, w), isto é,
$L(w) \leftarrow \min\{L(w), L(u) + W(u, w)\}$, logo $L(w) \leq dist(r, w)$.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Hipótese: Para todo $j < i$, $L(v_j) = dist(r, v_j)$.

Passo: Seja v_i o próximo vértice removido. Considere um caminho mínimo P de r até v_i. Seja u o último vértice de P já processado e w o próximo vértice no caminho.

Pela hipótese, $L(u) = dist(r, u)$. Ao processar u, o algoritmo tenta melhorar $L(w)$ usando a aresta (u, w), isto é,
$L(w) \leftarrow \min\{L(w), L(u) + W(u, w)\}$, logo $L(w) \leq dist(r, w)$.

Como v_i tem o menor rótulo entre os não processados, $L(v_i) \leq L(w)$, então $L(v_i) \leq dist(r, v_i)$.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Hipótese: Para todo $j < i$, $L(v_j) = dist(r, v_j)$.

Passo: Seja v_i o próximo vértice removido. Considere um caminho mínimo P de r até v_i. Seja u o último vértice de P já processado e w o próximo vértice no caminho.

Pela hipótese, $L(u) = dist(r, u)$. Ao processar u, o algoritmo tenta melhorar $L(w)$ usando a aresta (u, w), isto é,
$L(w) \leftarrow \min\{L(w), L(u) + W(u, w)\}$, logo $L(w) \leq dist(r, w)$.

Como v_i tem o menor rótulo entre os não processados, $L(v_i) \leq L(w)$, então $L(v_i) \leq dist(r, v_i)$.

Por outro lado, sempre vale $L(v_i) \geq dist(r, v_i)$, pois $L(v_i)$ é sempre o custo de um caminho de r até v_i. Portanto, $L(v_i) = dist(r, v_i)$.

Luís Felipe

30/04/26

Corretude do Algoritmo de Dijkstra

Objetivo: Mostrar que, ao final do algoritmo, $L(v) = dist(r, v)$ para todo vértice v.

Prova por indução:

Seja $v_1, v_2, \ldots, v_n$ a ordem de remoção da fila de prioridades.

Base: $v_1 = r$. Temos $L(r) = 0 = dist(r, r)$.

Hipótese: Para todo $j < i$, $L(v_j) = dist(r, v_j)$.

Passo: Seja v_i o próximo vértice removido. Considere um caminho mínimo P de r até v_i. Seja u o último vértice de P já processado e w o próximo vértice no caminho.

Pela hipótese, $L(u) = dist(r, u)$. Ao processar u, o algoritmo tenta melhorar $L(w)$ usando a aresta (u, w), isto é,
$L(w) \leftarrow \min\{L(w), L(u) + W(u, w)\}$, logo $L(w) \leq dist(r, w)$.

Como v_i tem o menor rótulo entre os não processados, $L(v_i) \leq L(w)$, então $L(v_i) \leq dist(r, v_i)$.

Por outro lado, sempre vale $L(v_i) \geq dist(r, v_i)$, pois $L(v_i)$ é sempre o custo de um caminho de r até v_i. Portanto, $L(v_i) = dist(r, v_i)$.

Obs.: Consideramos somente quando os pesos são não negativos.

Luís Felipe

30/04/26

Como obter os caminhos mínimos?

- Os ponteiros $pai(v)$ definem uma árvore T.

Luís Felipe

30/04/26

Como obter os caminhos mínimos?

- Os ponteiros $pai(v)$ definem uma árvore T.
- Essa árvore é chamada de árvore de caminhos mínimos.

Luís Felipe

30/04/26

Como obter os caminhos mínimos?

- Os ponteiros $pai(v)$ definem uma árvore T.
- Essa árvore é chamada de árvore de caminhos mínimos.
- Para obter o caminho mínimo da raiz r até um vértice v:
  - basta seguir os ponteiros *pai* de v até r.

Luís Felipe

30/04/26

Como obter os caminhos mínimos?

- Os ponteiros $pai(v)$ definem uma árvore T.
- Essa árvore é chamada de árvore de caminhos mínimos.
- Para obter o caminho mínimo da raiz r até um vértice v:
  - basta seguir os ponteiros *pai* de v até r.
- O custo desse caminho é:

$$L(v)$$

Luís Felipe

30/04/26

Como obter os caminhos mínimos?

- Os ponteiros $pai(v)$ definem uma árvore T.
- Essa árvore é chamada de árvore de caminhos mínimos.
- Para obter o caminho mínimo da raiz r até um vértice v:
  - basta seguir os ponteiros *pai* de v até r.
- O custo desse caminho é:

$$L(v)$$

- Ou seja, o valor calculado pelo algoritmo.

Luís Felipe

30/04/26

Grafos não direcionados

- Como aplicar Dijkstra em grafos não direcionados?

Luís Felipe

30/04/26

Grafos não direcionados

- Como aplicar Dijkstra em grafos não direcionados?
- Transformamos o grafo G em um dígrafo H:

Luís Felipe

30/04/26

Grafos não direcionados

- Como aplicar Dijkstra em grafos não direcionados?
- Transformamos o grafo G em um dígrafo H:
  - para cada aresta xy de peso p em G, criar as arestas (x, y) e (y, x) em H com peso p;

Luís Felipe

30/04/26

Grafos não direcionados

- Como aplicar Dijkstra em grafos não direcionados?
- Transformamos o grafo G em um dígrafo H:
  - para cada aresta xy de peso p em G, criar as arestas (x, y) e (y, x) em H com peso p;
  - se não houver aresta xy, considerar peso ∞.

Luís Felipe

30/04/26

Grafos não direcionados

- Como aplicar Dijkstra em grafos não direcionados?
- Transformamos o grafo G em um dígrafo H:
  - para cada aresta xy de peso p em G, criar as arestas (x, y) e (y, x) em H com peso p;
  - se não houver aresta xy, considerar peso ∞.
- Então basta executar Dijkstra em H.

Luís Felipe

30/04/26

Exercício

Mostre que o algoritmo de Dijkstra se comporta como uma busca em largura quando todos os pesos são iguais a 1.

Luís Felipe

30/04/26

Exercício

Mostre que o algoritmo de Dijkstra se comporta como uma busca em largura quando todos os pesos são iguais a 1.

Dica:

- compare os valores $L(v)$ com os níveis da BFS;

Luís Felipe

30/04/26

Exercício

Mostre que o algoritmo de Dijkstra se comporta como uma busca em largura quando todos os pesos são iguais a 1.

Dica:

- compare os valores $L(v)$ com os níveis da BFS;
- observe a ordem de remoção da fila de prioridades.

Luís Felipe

30/04/26

Exercício

Mostre que o algoritmo de Dijkstra se comporta como uma busca em largura quando todos os pesos são iguais a 1.

Dica:

- compare os valores $L(v)$ com os níveis da BFS;
- observe a ordem de remoção da fila de prioridades.

Conclusão esperada:

- a árvore de caminhos mínimos é uma árvore de largura;
- $L(v)$ corresponde à distância em número de arestas.

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

aumenta a distância em exatamente 1;

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

  aumenta a distância em exatamente 1;
- logo, $L(v)$ representa o número de arestas no caminho da raiz até v.

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

  aumenta a distância em exatamente 1;
- logo, $L(v)$ representa o número de arestas no caminho da raiz até v.

Comportamento da fila de prioridades:

- vértices são removidos em ordem crescente de $L(v)$;

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

  aumenta a distância em exatamente 1;
- logo, $L(v)$ representa o número de arestas no caminho da raiz até v.

Comportamento da fila de prioridades:

- vértices são removidos em ordem crescente de $L(v)$;
- ou seja, primeiro todos com $L = 0$, depois $L = 1$, depois $L = 2$, etc.

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

  aumenta a distância em exatamente 1;
- logo, $L(v)$ representa o número de arestas no caminho da raiz até v.

Comportamento da fila de prioridades:

- vértices são removidos em ordem crescente de $L(v)$;
- ou seja, primeiro todos com $L = 0$, depois $L = 1$, depois $L = 2$, etc.

Conclusão:

- o algoritmo processa os vértices por níveis;

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

  aumenta a distância em exatamente 1;
- logo, $L(v)$ representa o número de arestas no caminho da raiz até v.

Comportamento da fila de prioridades:

- vértices são removidos em ordem crescente de $L(v)$;
- ou seja, primeiro todos com $L = 0$, depois $L = 1$, depois $L = 2$, etc.

Conclusão:

- o algoritmo processa os vértices por níveis;
- a árvore obtida é uma árvore de largura;

Luís Felipe

30/04/26

Prova: Dijkstra com pesos 1 = BFS

Suponha que todas as arestas de G têm peso 1.

Observação:

- toda atualização do tipo

$$L(w) \leftarrow L(v) + 1$$

  aumenta a distância em exatamente 1;
- logo, $L(v)$ representa o número de arestas no caminho da raiz até v.

Comportamento da fila de prioridades:

- vértices são removidos em ordem crescente de $L(v)$;
- ou seja, primeiro todos com $L = 0$, depois $L = 1$, depois $L = 2$, etc.

Conclusão:

- o algoritmo processa os vértices por níveis;
- a árvore obtida é uma árvore de largura;
- portanto, Dijkstra coincide com a BFS.

Pesos negativos: o que dá errado?

- O algoritmo assume que, ao escolher v com menor $L(v)$:
  nenhum caminho futuro pode melhorar $L(v)$

Pesos negativos: o que dá errado?

- O algoritmo assume que, ao escolher v com menor $L(v)$:
  nenhum caminho futuro pode melhorar $L(v)$
- Isso é verdadeiro apenas se os pesos são não negativos.

Pesos negativos: o que dá errado?

- O algoritmo assume que, ao escolher v com menor $L(v)$:

  nenhum caminho futuro pode melhorar $L(v)$

- Isso é verdadeiro apenas se os pesos são não negativos.
- Com pesos negativos:
  - um caminho posterior pode reduzir o custo de um vértice já processado;

Pesos negativos: o que dá errado?

- O algoritmo assume que, ao escolher v com menor $L(v)$:

  nenhum caminho futuro pode melhorar $L(v)$

- Isso é verdadeiro apenas se os pesos são não negativos.
- Com pesos negativos:
  - um caminho posterior pode reduzir o custo de um vértice já processado;
- Portanto:

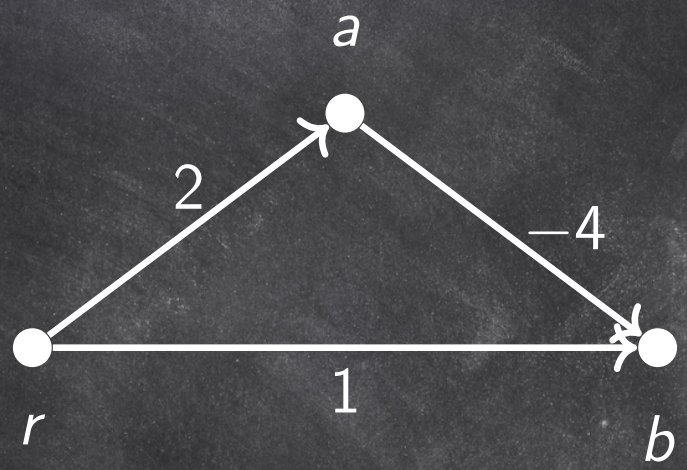
  o algoritmo pode produzir resultados incorretos

Luís Felipe

30/04/26

Contraexemplo com pesos negativos

Considere o grafo:

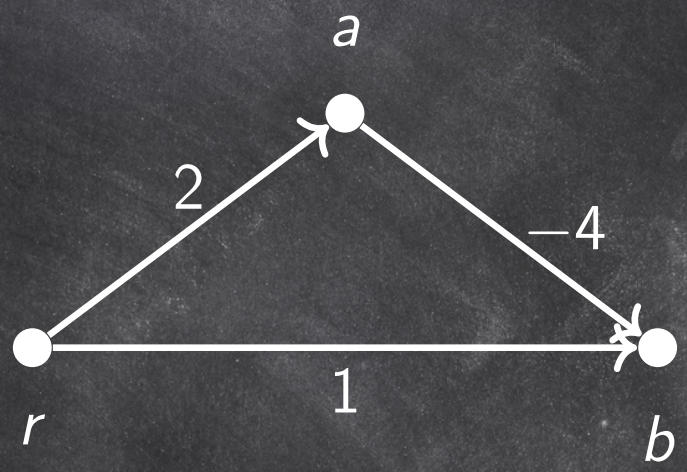


Luís Felipe

30/04/26

Contraexemplo com pesos negativos

Considere o grafo:



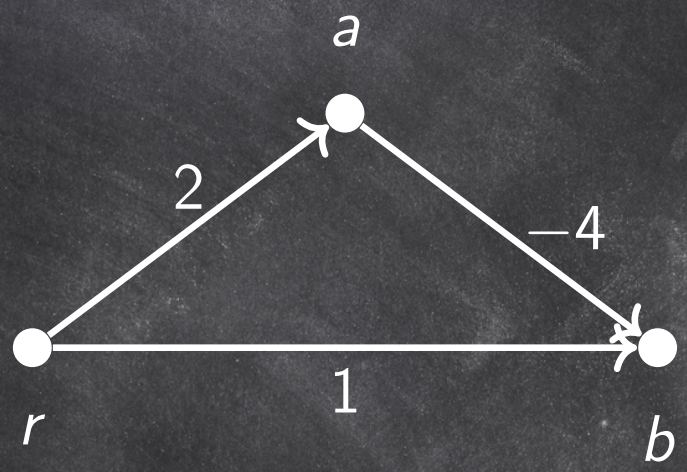
- Caminho direto: $r \to b$ tem custo 1

Luís Felipe

30/04/26

Contraexemplo com pesos negativos

Considere o grafo:



- Caminho direto: $r \to b$ tem custo 1
- Caminho alternativo:

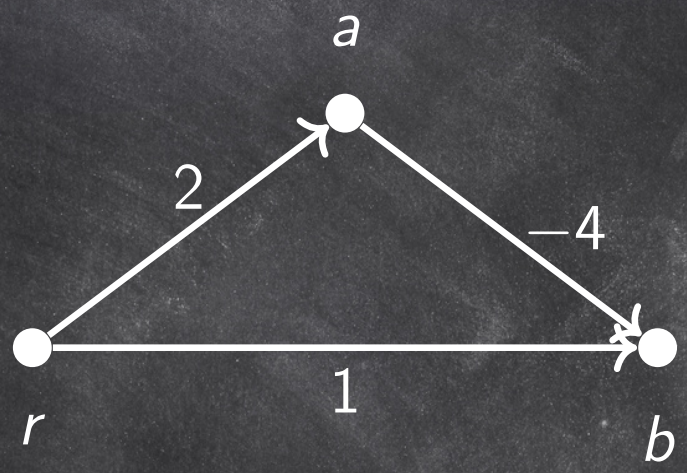
$$r \to a \to b = 2 + (-4) = -2$$

Luís Felipe

30/04/26

Contraexemplo com pesos negativos

Considere o grafo:



- Caminho direto: $r \to b$ tem custo 1
- Caminho alternativo:

$$r \to a \to b = 2 + (-4) = -2$$

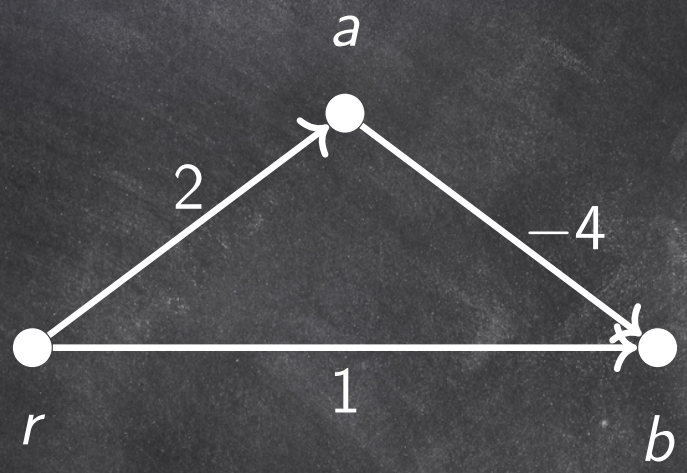
- O menor caminho é via a

Luís Felipe

30/04/26

Contraexemplo com pesos negativos

Considere o grafo:



- Caminho direto: $r \to b$ tem custo 1
- Caminho alternativo:

$$r \to a \to b = 2 + (-4) = -2$$

- O menor caminho é via a

Problema:

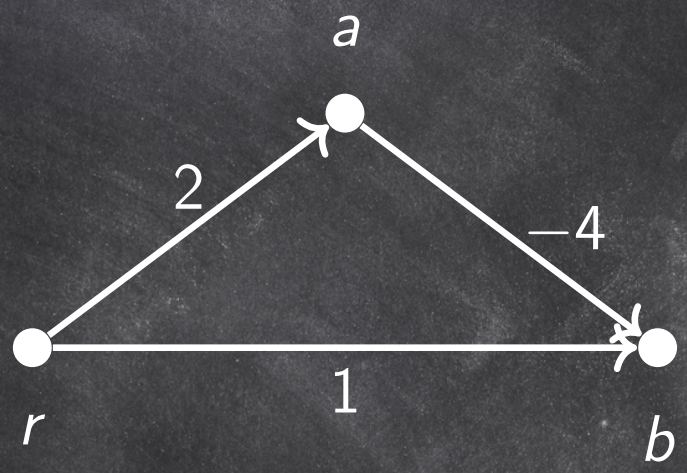
- Dijkstra pode escolher b antes de explorar a;

Luís Felipe

30/04/26

Contraexemplo com pesos negativos

Considere o grafo:



- Caminho direto: $r \to b$ tem custo 1
- Caminho alternativo:

$$r \to a \to b = 2 + (-4) = -2$$

- O menor caminho é via a

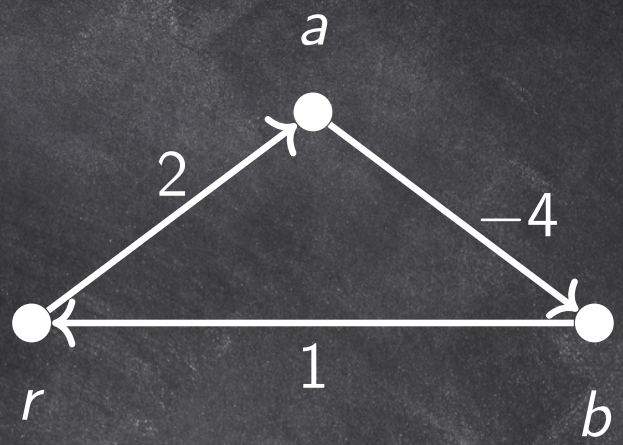
Problema:

- Dijkstra pode escolher b antes de explorar a;
- e fixar incorretamente o valor $L(b) = 1$.

Luís Felipe

30/04/26

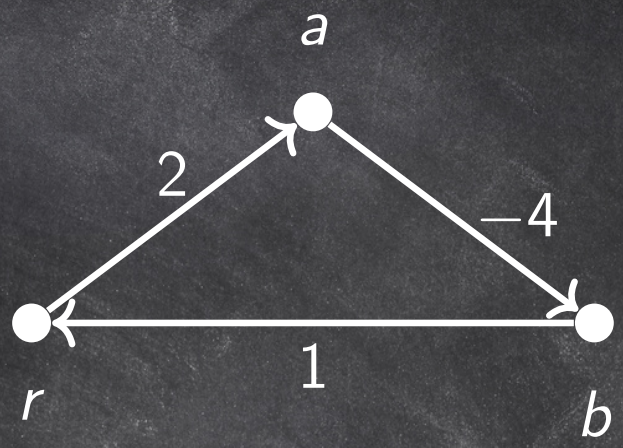
Ciclos negativos



Luís Felipe

30/04/26

Ciclos negativos



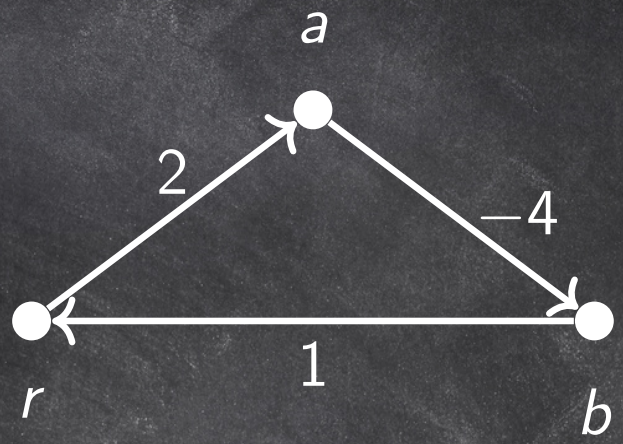
- Soma do ciclo:

$$2 + (-4) + 1 = -1$$

Luís Felipe

30/04/26

Ciclos negativos



- Soma do ciclo:

$$2 + (-4) + 1 = -1$$

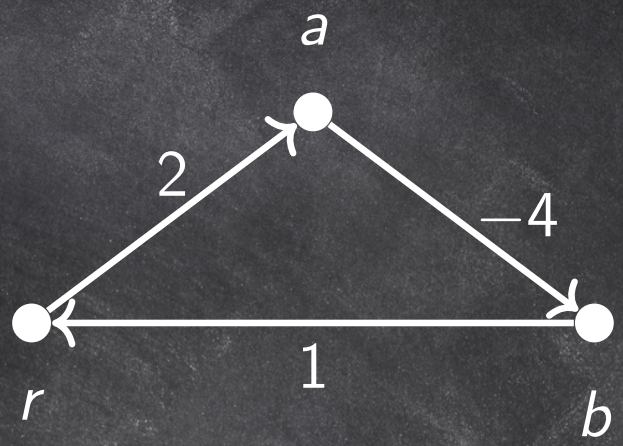
- Podemos percorrer o ciclo indefinidamente:

$$r \to a \to b \to r \to a \to b \to r \ldots$$

Luís Felipe

30/04/26

Ciclos negativos



- Soma do ciclo:

$$2 + (-4) + 1 = -1$$

- Podemos percorrer o ciclo indefinidamente:

$$r \to a \to b \to r \to a \to b \to r \ldots$$

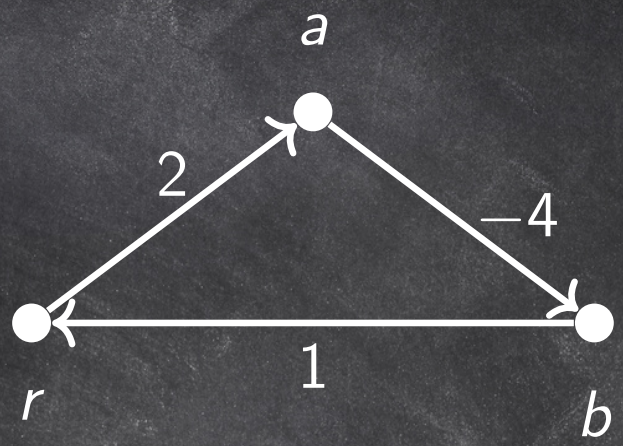
- O custo do caminho diminui a cada volta:

$$\Rightarrow \text{custo} \ \to -\infty$$

Luís Felipe

30/04/26

Ciclos negativos



- Soma do ciclo:

$$2 + (-4) + 1 = -1$$

- Podemos percorrer o ciclo indefinidamente:

$$r \to a \to b \to r \to a \to b \to r \ldots$$

- O custo do caminho diminui a cada volta:

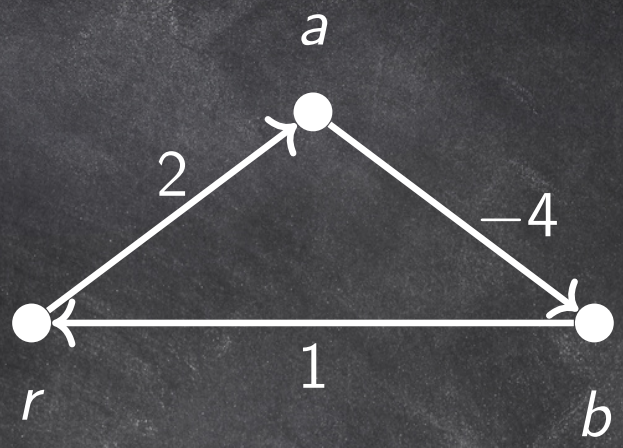
$$\Rightarrow \text{custo} \ \to -\infty$$

- Portanto, não existe caminho mínimo bem definido.

Luís Felipe

30/04/26

Ciclos negativos



- Soma do ciclo:

$$2 + (-4) + 1 = -1$$

- Podemos percorrer o ciclo indefinidamente:

$$r \to a \to b \to r \to a \to b \to r \ldots$$

- O custo do caminho diminui a cada volta:

$$\Rightarrow \text{custo} \ \to -\infty$$

- Portanto, não existe caminho mínimo bem definido.
- Logo, o problema de caminhos mínimos deixa de fazer sentido nesse caso.

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;
  - com ciclos negativos, o problema pode nem ter solução.

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;
  - com ciclos negativos, o problema pode nem ter solução.

Pergunta natural:

Como resolver caminhos mínimos com pesos negativos?

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;
  - com ciclos negativos, o problema pode nem ter solução.

Pergunta natural:

Como resolver caminhos mínimos com pesos negativos?

Resposta:

Algoritmo de Bellman-Ford

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;
  - com ciclos negativos, o problema pode nem ter solução.

Pergunta natural:

Como resolver caminhos mínimos com pesos negativos?

Resposta:

Algoritmo de Bellman-Ford

- Funciona com pesos negativos;

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;
  - com ciclos negativos, o problema pode nem ter solução.

Pergunta natural:

Como resolver caminhos mínimos com pesos negativos?

Resposta:

Algoritmo de Bellman-Ford

- Funciona com pesos negativos;
- Detecta ciclos negativos;

Luís Felipe

30/04/26

E se houver pesos negativos?

- Vimos que:
  - Dijkstra funciona apenas para pesos não negativos;
  - com pesos negativos, ele pode produzir resultados incorretos;
  - com ciclos negativos, o problema pode nem ter solução.

Pergunta natural:

Como resolver caminhos mínimos com pesos negativos?

Resposta:

Algoritmo de Bellman-Ford

- Funciona com pesos negativos;
- Detecta ciclos negativos;
- Veremos na próxima aula.