MC558 — Análise de Algoritmos II

Cid C. de Souza   Cândida N. da Silva   Orlando Lee

2 de abril de 2023

Antes de mais nada...

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- O que vocês tem em mãos é uma versão modificada preparada para atender a meus gostos.
- Nunca é demais enfatizar que o material é apenas um guia e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial o CLR ou CLRS).

Orlando Lee

Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram direta ou indiretamente com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Uma lista destes "colaboradores" (em ordem alfabética) é dada abaixo:
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

Buscas em grafos

Notação

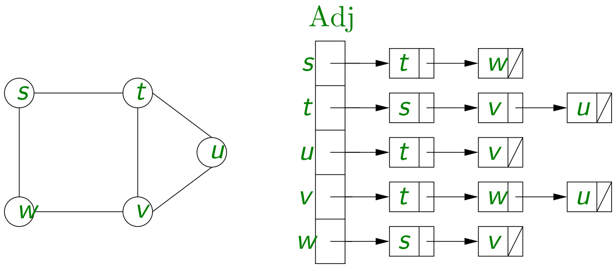
- Para um grafo G (orientado ou não) denotamos por $V[G]$ seu conjunto de vértices e por $E[G]$ seu conjunto de arestas.
- Para denotar complexidades nas expressões com O ou Θ usaremos V e E em vez de $|V[G]|$ ou $|E[G]|$. Por exemplo, $\Theta(V+E)$ ou $O(V^2)$.
- Usaremos a notação $g(v)$ ($g^{-}(v)$, $g^{+}(v)$) para denotar o grau (grau de entrada, grau de saída) de v.

Listas de adjacência

- Seja $G = (V, E)$ um grafo simples (orientado ou não).
- CLRS usa (u, v) para denotar uma aresta (orientada ou não).
- A representação de G por uma *lista de adjacências* consiste no seguinte.

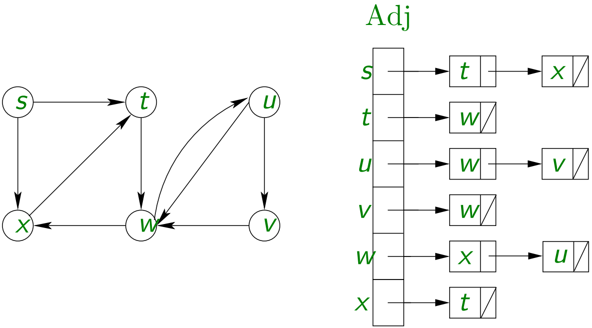
  Para cada vértice u, temos uma lista ligada $\mathrm{Adj}[u]$ dos vértices *adjacentes* a u, ou seja, v aparece em $\mathrm{Adj}[u]$ se (u, v) é uma aresta de G. Os vértices podem estar em qualquer ordem em uma lista.

Listas de adjacência



Tamanho: $|V| + \sum_{v \in V} g(v) = |V| + 2|E| = O(V + E)$.

Lista de adjacências



Tamanho: $|V| + \sum_{v \in V} g^+(v) = |V| + |E| = O(V + E)$.

Exemplo

Recebe um grafo G (na forma de listas de adjacências) e imprime as listas.

ImprimeVizinhos(G)

para cada $u \in V[G]$ **faça**

  imprime u seguido de ":"

  para cada $v \in \mathrm{Adj}[u]$ **faça**

    imprime v

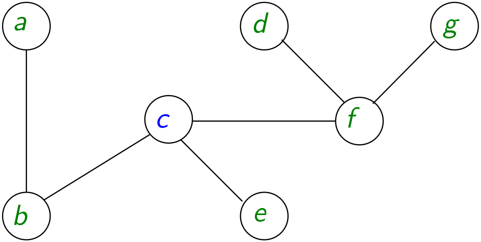
Qual é a complexidade do algoritmo ImprimeVizinhos? $O(V + E)$

Representação de árvores (orientadas)

- Neste curso, veremos vários algoritmos que usam árvores para representar a solução de algum problema.
- Mostraremos agora a representação usada por esses algoritmos.

Representação de árvores

Uma *árvore enraizada* é uma árvore com um vértice especial chamado *raiz*.

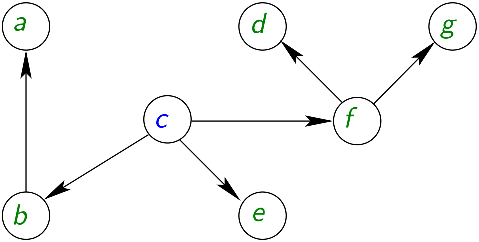


raiz c

Representação de árvores

Uma *árvore orientada com raiz* r é um grafo orientado **acíclico** $T = (V, E)$ tal que:

1. $g^{-}(r) = 0$,
2. $g^{-}(v) = 1$ para $v \in V - \{r\}$.

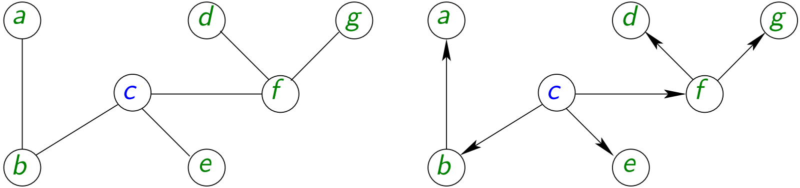


raiz c

Vetor π ($\pi[v]$ é pai de v):

v	a	b	c	d	e	f	g
$\pi[v]$	b	c	N	f	c	c	f

N é um símbolo usado para indicar não existência.



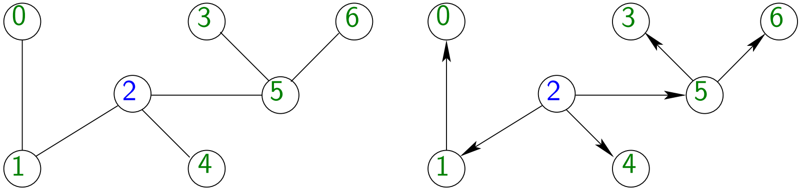
raiz c

Representação de árvores em C

Vetor de predecessores π:

v	0	1	2	3	4	5	6
$\pi[v]$	1	2	N	5	2	2	5

N é uma abreviatura de NIL, um símbolo usado para indicar não existência.



raiz 2

- Os algoritmos de busca que veremos constroem uma árvore (ou floresta) que é um subgrafo do grafo de entrada.
- Usualmente supomos que há um vértice s que será a raiz da árvore.
- A representação desta árvore permite determinar facilmente o caminho que vai da raiz s a um vértice v.
  O caminho (inverso) de s a v na árvore é:

  $$v, \pi[v], \pi[\pi[v]], \pi[\pi[\pi[v]]], \ldots, s.$$

Como obter os caminhos na árvore

Imprime o caminho de s a v na árvore de raiz s representada por $\pi[\ ]$.

```
PRINT-PATH(π, s, v)
    se v = s então
        imprima s
    senão
        PRINT-PATH(π, s, π[v])
        imprima v.
```

Complexidade: $O(\text{comprimento do caminho}) = O(V)$.

Busca em grafos

- Grafos são estruturas mais complicadas do que listas, vetores e árvores (binárias).
  Precisamos de métodos para explorar/percorrer um grafo (orientado ou não-orientado).
- Busca em largura (breadth-first search – BFS)
  Busca em profundidade (depth-first search – DFS)
- Pode-se obter várias informações sobre a estrutura do grafo que podem ser úteis para projetar algoritmos eficientes para determinados problemas.

Busca em largura

- Dizemos que um vértice v é **alcançável** a partir de um vértice s em um grafo G se existe um caminho de s a v em G.
- Definição: a distância de s a v é o comprimento de um caminho mais curto de s a v.
  Denotamos este valor por $\mathrm{dist}(s, v)$.
- Se v **não é alcançável** a partir de s, então $\mathrm{dist}(s, v) = \infty$ (*distância infinita*).

- Busca em largura recebe um grafo $G = (V, E)$ e um vértice especificado s chamado fonte (*source*).
- Percorre todos os vértices alcançáveis a partir de s em ordem de distância deste. Vértices a mesma distância podem ser percorridos em qualquer ordem.
- Constrói uma Árvore de Busca em Largura com raiz s. Cada caminho de s a um vértice v nesta árvore corresponde a um caminho mais curto de s a v.

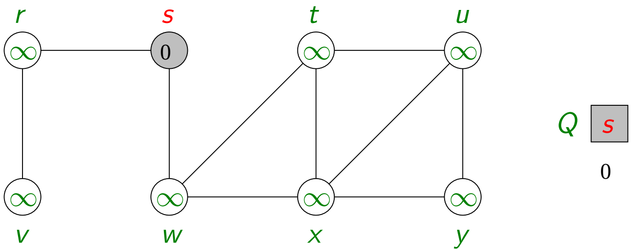
Busca em largura

- Inicialmente a Árvore de Busca em Largura contém apenas o vértice fonte s.
- Para cada vizinho v de s, o vértice v e a aresta (s, v) são acrescentadas à árvore.
- O processo é repetido para os vizinhos dos vizinhos de s e assim por diante, até que todos os vértices atingíveis por s sejam inseridos na árvore.
- Este processo é implementado através de uma fila Q munida das operações Enqueue e Dequeue.

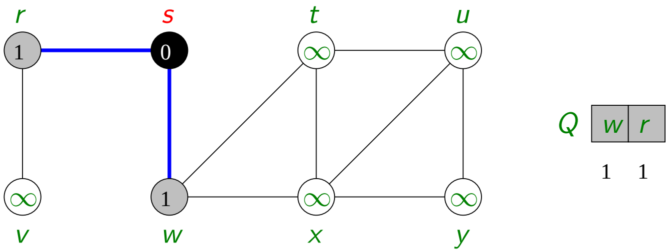
Busca em largura

- Busca em largura atribui cores a cada vértice: branco, cinza e preto.
- Cor branca $=$ "não visitado".
  Inicialmente todos os vértices são brancos.
- Cor cinza $=$ "visitado pela primeira vez" (na fila).
- Cor Preta $=$ "teve seus vizinhos visitados".
- As cores não são necessárias para o funcionamento do algoritmo, mas facilitam sua análise.

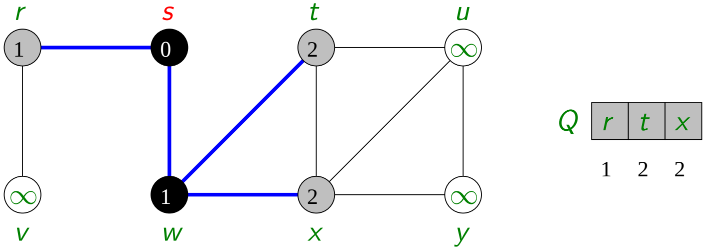
Exemplo (CLRS)



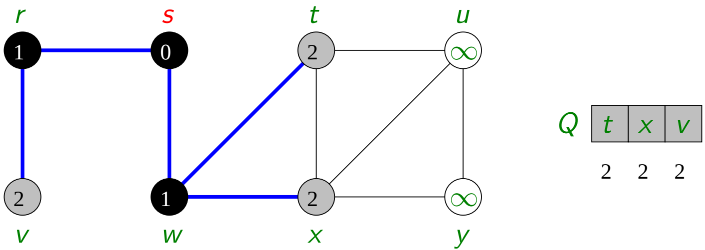
Exemplo (CLRS)



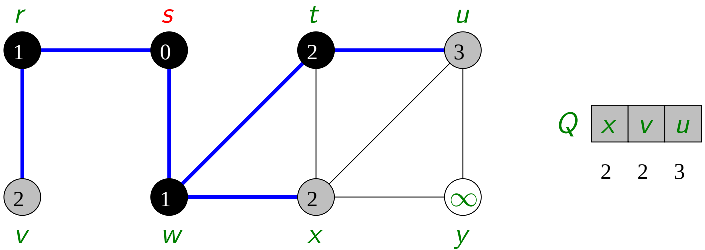
Exemplo (CLRS)



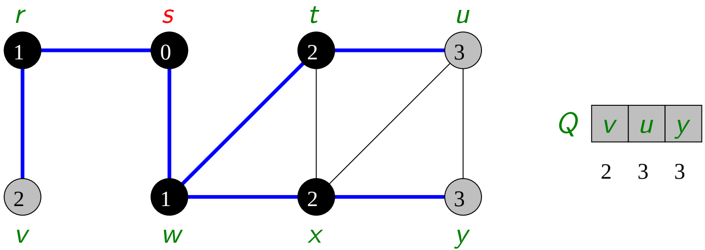
Exemplo (CLRS)



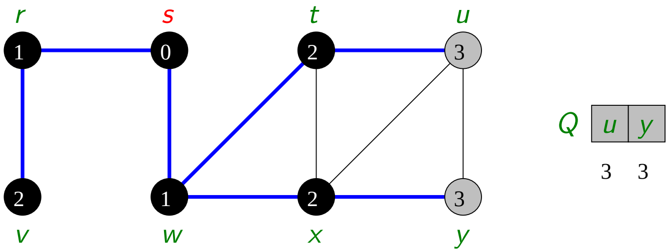
Exemplo (CLRS)



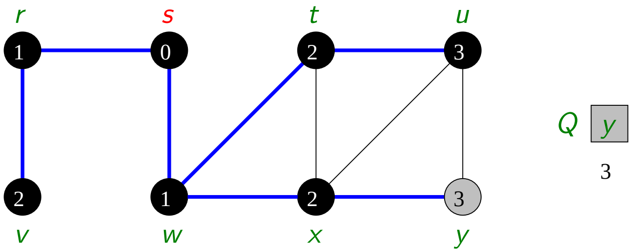
Exemplo (CLRS)



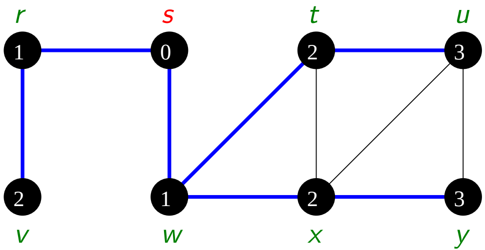
Exemplo (CLRS)



Exemplo (CLRS)



Exemplo (CLRS)



Q $\emptyset$

Representação da árvore e das distâncias

- A raiz da Árvore de Busca em Largura é s.
- Representamos a árvore através de um vetor $\pi[\ ]$.
- Uma variável $d[v]$ é usada para armazenar a distância de s a v (que será determinada durante a busca).

Recebe um grafo G (na forma de listas de adjacências) e um vértice $s \in V[G]$ e devolve
(i) para cada vértice v, a distância de s a v em G e
(ii) uma Árvore de Busca em Largura.

$\mathrm{BFS}(G, s)$

```
▷ Inicialização
para cada  u ∈ V[G] − {s} faça
    cor[u] ← branco
    d[u] ← ∞
    π[u] ← NIL
cor[s] ← cinza
d[s] ← 0
π[s] ← NIL
```

Busca em largura

```
Q ← ∅
ENQUEUE(Q, s)
enquanto Q ≠ ∅ faça
    u ← DEQUEUE(Q)
    para cada v ∈ Adj[u] faça
        se cor[v] = branco então
            cor[v] ← cinza
            d[v] ← d[u] + 1
            π[v] ← u
            ENQUEUE(Q, v)
    cor[u] ← preto
```

Consumo de tempo

Método de análise agregado.

- A inicialização consome tempo $\Theta(V)$.
- Depois que um vértice deixa de ser branco, ele não volta a ser branco novamente. Assim, cada vértice é inserido na fila Q no máximo uma vez. Cada operação sobre a fila consome tempo $\Theta(1)$ resultando em um total de $O(V)$.
- Em uma lista de adjacência, cada vértice é percorrido apenas uma vez. A soma dos comprimentos das listas é $\Theta(E)$. Assim, o tempo gasto para percorrer as listas é $O(E)$.

Conclusão:

A complexidade de tempo de BFS é $O(V + E)$.

Agora falta mostrar que BFS funciona.

Corretude

Lembre-se que $\mathrm{dist}(s, v)$ a distância de s a v.

Precisamos mostrar que:

- $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V[G]$.
- A função predecessor $\pi[\,]$ define uma Árvore de Busca em Largura com raiz s.

Alguns lemas

Note que $d[v]$ e $\pi[v]$ nunca mudam após v ser inserido na fila.

Lema 1. Se $d[v] < \infty$ então v pertence à árvore T induzida por $\pi[]$ e o caminho de s a v em T tem comprimento $d[v]$.

Prova:

Indução no número de operações $\textsc{Enqueue}$.

Alguns lemas

Base: quando s é inserido na fila temos $d[s] = 0$ e s é a raiz da árvore T.

Passo de indução: v é descoberto enquanto a busca é feita em u (percorrendo $\mathrm{Adj}[u]$). Então por HI existe um caminho de s a u em T com comprimento $d[u]$.

Como tomamos $d[v] = d[u] + 1$ e $\pi[v] = u$, o resultado segue. ■

Alguns lemas

$d[v]$ é uma estimativa superior de $\mathrm{dist}(s, v)$.

Corolário 1. Durante a execução do algoritmo vale o seguinte invariante

$$d[v] \geq \mathrm{dist}(s, v) \text{ para todo } v \in V[G].$$

■

Alguns lemas

Lema 2. Suponha que $\langle v_1, v_2, \ldots, v_r \rangle$ seja a configuração da fila Q na linha 10 em uma iteração qualquer.

Então

$$d[v_r] \leq d[v_1] + 1$$

e

$$d[v_i] \leq d[v_{i+1}] \text{ para } i = 1, 2, \ldots, r - 1.$$

Em outras palavras, os vértices são inseridos na fila em ordem não-decrescente dos valores $d[\ ]$ e há no máximo dois valores de $d[\ ]$ para vértices na fila.

Prova do Lema 2

Prova: Indução no número de operações ENQUEUE e DEQUEUE.

Base: $Q = \{s\}$. O lema vale trivialmente.

Passo de indução:

Caso 1: v_1 é removido de Q.

$\langle v_1, v_2, \ldots, v_r \rangle \quad \Rightarrow \quad \langle v_2, \ldots, v_r \rangle$

Agora v_2 é o primeiro vértice de Q. Então

$$d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1.$$

As outras desigualdades se mantêm.

Prova do Lema 2

Passo de indução:

Caso 2: $v = v_{r+1}$ é inserido em Q.

$\langle v_1, v_2, \ldots, v_r \rangle \quad \Rightarrow \quad \langle v_1, v_2, \ldots, v_r, v_{r+1} = v \rangle$

Suponha que a busca é feita em u neste momento. Logo $d[u] \leq d[v_1]$. Então

$$d[v_{r+1}] = d[v] = d[u] + 1 \leq d[v_1] + 1.$$

Pela HI segue que $d[v_r] \leq d[u] + 1$. Logo

$$d[v_r] \leq d[u] + 1 = d[v] = d[v_{r+1}].$$

As outras desigualdades se mantêm. ■

Árvore

Teorema. Seja G um grafo e $s \in V[G]$.

Então após a execução de BFS,

$$d[v] = \mathrm{dist}(s, v) \text{ para todo } v \in V[G]$$

e

$$\pi[\,] \text{ define uma Árvore de Busca em Largura.}$$

Prova: Basta provar que $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V[G]$.

Note que se $\mathrm{dist}(s, v) = \infty$ então $d[v] = \infty$ pelo Corolário 1.

Então vamos considerar o caso em que $\mathrm{dist}(s, v) < \infty$.

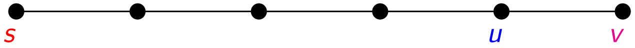
Corretude

Vamos provar por indução em $\mathrm{dist}(s, v)$ que $d[v] = \mathrm{dist}(s, v)$.

Base: Se $\mathrm{dist}(s, v) = 0$ então $v = s$ e $d[s] = 0$.

Hipótese de indução: Suponha então que $d[u] = \mathrm{dist}(s, u)$ para todo vértice u com $\mathrm{dist}(s, u) < k$.

Seja v um vértice com $\mathrm{dist}(s, v) = k$. Considere um caminho mínimo de s a v em G e chame de u o vértice que antecede v neste caminho.



Note que $\mathrm{dist}(s, u) = k - 1$ (Por quê?).

Corretude

```
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10   enquanto  Q ≠ ∅ faça
11      u ← DEQUEUE(Q)
12      para cada v ∈ Adj[u] faça
13         se  cor[v] = branco então
14            cor[v] ← cinza
15            d[v] ← d[u] + 1
16            π[v] ← u
17            ENQUEUE(Q, v)
18      cor[u] ← preto
```

Considere o instante em que u foi removido da fila Q (linha 11 de BFS). Neste instante, v é branco, cinza ou preto.

Corretude

- se v é branco, então a linha 15 faz com $d[v] = d[u] + 1 = (k - 1) + 1 = k$.

- se v é cinza, então v foi visitado antes por algum vértice w (logo, $v \in \mathrm{Adj}[w]$) e $d[v] = d[w] + 1$. Pelo Lema 2, $d[w] \leq d[u] = k - 1$ e segue que $d[v] = k$.

- se v é preto, então v já passou pela fila Q e pelo Lema 2, $d[v] \leq d[u] = k - 1$. Mas por outro lado, pelo Corolário 1, $d[v] \geq \mathrm{dist}(s, v) = k$, o que é uma contradição. Este caso não ocorre.

Portanto, $d[v] = \mathrm{dist}(s, v)$. ■

Exercício

Já vimos o seguinte resultado.

Teorema. Um grafo G é bipartido se, e somente se, não contém um ciclo ímpar.

Exercício. Projete um algoritmo linear baseado em BFS que dado um grafo G representado por listas de adjacências, devolve

- uma bipartição de G, ou
- um ciclo ímpar de G.