

Aplicações de BFS e DFS

CLRS 22 Elementary Graph Algorithms

CLRS 22.2 e 22.3

Busca em largura

BFS (G, s)

```
1  para cada  $u \in G.V \setminus \{s\}$  faça
2       $u.cor \leftarrow$  branco    $u.d \leftarrow \infty$     $u.\pi \leftarrow$  nil
3   $Q \leftarrow \emptyset$     $\triangleright$  fila dos vértices descobertos
4   $s.cor \leftarrow$  cinzento    $s.d \leftarrow 0$     $s.\pi \leftarrow$  nil
5   $Q.INSIRA(s)$ 

6  enquanto  $Q \neq \emptyset$  faça
7       $u \leftarrow Q.REMOVA()$ 
8      para cada  $v \in u.Estrela$  faça
9          se  $v.cor =$  branco
10             então  $v.cor \leftarrow$  cinzento
11                  $v.d \leftarrow u.d + 1$ 
12                  $v.\pi \leftarrow u$ 
13                  $Q.INSIRA(v)$ 
14      $u.cor \leftarrow$  preto
```

Descrição

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto mas não processado
(são os vértices em Q)

Vértice preto: processado

Descrição

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto mas não processado
(são os vértices em Q)

Vértice preto: processado

BFS devolve em π uma **árvore BF** enraizada em s .

$u.\pi$: predecessor ou pai de u na árvore BF

$u.d$: distância de s a u em G

BFS descobre todos os vértices à distância k
antes de descobrir qualquer um à distância $k + 1$

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco e, ao ser descoberto, passa a ser cinzento, e depois preto.

A estrela de cada vértice descoberto é percorrida uma única vez, quando o vértice sai de Q .

Logo, com listas de adjacência, o consumo de tempo é $O(n + m)$, pois a inicialização custa $\Theta(n)$ e a soma do tamanho das listas de adjacências percorridas é $O(m)$.

O consumo de tempo de uma BFS é linear no tamanho do grafo.

Correção do algoritmo

Vamos verificar que $u.d$ é a distância de s a u e que a árvore BF contém um caminho mínimo de s a u .

Correção do algoritmo

Vamos verificar que $u.d$ é a distância de s a u e que a árvore BF contém um caminho mínimo de s a u .

Seja $\delta(s, u)$ a distância de s a u em G .

Correção do algoritmo

Vamos verificar que $u.d$ é a distância de s a u e que a árvore BF contém um caminho mínimo de s a u .

Seja $\delta(s, u)$ a distância de s a u em G .

Lema: $\delta(s, v) \leq \delta(s, u) + 1$ para toda aresta uv .

Correção do algoritmo

Vamos verificar que $u.d$ é a distância de s a u e que a árvore BF contém um caminho mínimo de s a u .

Seja $\delta(s, u)$ a distância de s a u em G .

Lema: $\delta(s, v) \leq \delta(s, u) + 1$ para toda aresta uv .

Lema: $v.d \geq \delta(s, v)$ para todo vértice ao final da BFS.

Correção do algoritmo

Vamos verificar que $u.d$ é a distância de s a u e que a árvore BF contém um caminho mínimo de s a u .

Seja $\delta(s, u)$ a distância de s a u em G .

Lema: $\delta(s, v) \leq \delta(s, u) + 1$ para toda aresta uv .

Lema: $v.d \geq \delta(s, v)$ para todo vértice ao final da BFS.

Lema: Se Q contém os vértices v_1, v_2, \dots, v_r nesta ordem, então $v_r.d \leq v_1.d + 1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \dots, r - 1$.

Correção

Seja $\delta(s, u)$ a distância de s a u em G .

Lema: $\delta(s, v) \leq \delta(s, u) + 1$ para toda aresta uv .

Lema: $v.d \geq \delta(s, v)$ para todo vértice ao final da BFS.

Lema: Se Q contém os vértices v_1, v_2, \dots, v_r nesta ordem, então $v_r.d \leq v_1.d + 1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \dots, r - 1$.

Correção

Seja $\delta(s, u)$ a distância de s a u em G .

Lema: $\delta(s, v) \leq \delta(s, u) + 1$ para toda aresta uv .

Lema: $v.d \geq \delta(s, v)$ para todo vértice ao final da BFS.

Lema: Se Q contém os vértices v_1, v_2, \dots, v_r nesta ordem, então $v_r.d \leq v_1.d + 1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \dots, r - 1$.

Corolário: Se v_i entra na fila antes de v_j , então $v_i.d \leq v_j.d$ quando v_j entra na fila.

Correção

Seja $\delta(s, u)$ a distância de s a u em G .

Lema: $\delta(s, v) \leq \delta(s, u) + 1$ para toda aresta uv .

Lema: $v.d \geq \delta(s, v)$ para todo vértice ao final da BFS.

Lema: Se Q contém os vértices v_1, v_2, \dots, v_r nesta ordem, então $v_r.d \leq v_1.d + 1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \dots, r - 1$.

Corolário: Se v_i entra na fila antes de v_j , então $v_i.d \leq v_j.d$ quando v_j entra na fila.

Teorema: *Ao final da BFS, $v.d = \delta(s, v)$ para todo $v \in V$. Ademais, para todo v alcançável de s com $v \neq s$, um caminho mais curto de s a v consiste em um caminho mais curto de s a $\pi[v]$ seguido da aresta $\pi[v].v$.*

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é alcançável a partir de um vértice v se existe caminho de v a u .

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é **alcançável** a partir de um vértice v se existe caminho de v a u .

- ▶ Num grafo dirigido, essa relação é transitiva e reflexiva.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é **alcançável** a partir de um vértice v se existe caminho de v a u .

- ▶ Num grafo dirigido, essa relação é transitiva e reflexiva.
- ▶ Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é **alcançável** a partir de um vértice v se existe caminho de v a u .

- ▶ Num grafo dirigido, essa relação é transitiva e reflexiva.
- ▶ Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é **alcançável** a partir de um vértice v se existe caminho de v a u .

- ▶ Num grafo dirigido, essa relação é transitiva e reflexiva.
- ▶ Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?

Basta achar as classes.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é **alcançável** a partir de um vértice v se existe caminho de v a u .

- ▶ Num grafo dirigido, essa relação é transitiva e reflexiva.
- ▶ Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

Como achar as componentes?

Basta achar as classes.

BFS a partir de um vértice determina sua classe.

Algumas aplicações

- Determinar componentes conexas de um grafo

Componentes conexas

Num grafo dizemos que um vértice u é **alcançável** a partir de um vértice v se existe caminho de v a u .

- ▶ Num grafo dirigido, essa relação é transitiva e reflexiva.
- ▶ Num grafo não dirigido, ela é de equivalência. Os subgrafos induzidos pelas classes são as componentes conexas do grafo.

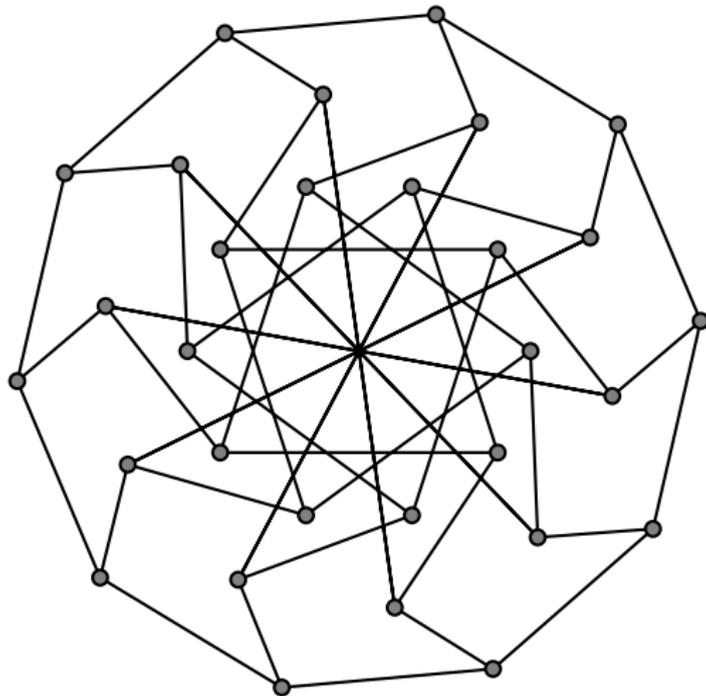
Como achar as componentes?

Basta achar as classes.

BFS a partir de um vértice determina sua classe.

- Determinar se um grafo é bipartido

Grafo conexo



Grafos bipartidos

Um grafo é **bipartido** se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

Grafos bipartidos

Um grafo é **bipartido** se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Grafos bipartidos

Um grafo é **bipartido** se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d .

Grafos bipartidos

Um grafo é **bipartido** se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d .

O algoritmo tem a mesma complexidade da BFS.
A corretude vem junto com o

Grafos bipartidos

Um grafo é **bipartido** se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d .

O algoritmo tem a mesma complexidade da BFS.

A corretude vem junto com o

Teorema

Um grafo é bipartido se e só se não tem um circuito ímpar.

Grafos bipartidos

Um grafo é **bipartido** se é possível dividir seus vértices em duas classes de tal modo que cada aresta tem uma ponta em cada classe.

ALGORITMO - Decide se G (conexo) é bipartido

Faça uma BFS. Pequena alteração: Se $v.d == u.d$, devolva NÃO.
Se terminou, a bipartição é dada pela paridade do d .

O algoritmo tem a mesma complexidade da BFS.

A corretude vem junto com o

Teorema

Um grafo é bipartido se e só se não tem um circuito ímpar.

Prova: Necessidade é fácil. Suficiência: modificar o algoritmo para devolver um circuito ímpar no caso NÃO.

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

PRÊMIOS

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

PRÊMIOS

Nota A no curso

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

PRÊMIOS

Nota A no curso

+

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

PRÊMIOS

Nota A no curso

+

Título de Doutor

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

PRÊMIOS

Nota A no curso

+

Título de Doutor

+

Grafos tripartidos

Um grafo é **tripartido** se é possível dividir seus vértices em três classes de tal modo que cada aresta tenha as pontas em classes diferentes.

Exercício

Encontrar um algoritmo polinomial para decidir se um grafo é tripartido.

PRÊMIOS

Nota A no curso

+

Título de Doutor

+

US\$10⁶

Busca em profundidade

DFS

Outra **moldura** de algoritmos
que vão fazendo seu serviço enquanto percorrem o grafo.

Busca em profundidade

DFS

Outra **moldura** de algoritmos
que vão fazendo seu serviço enquanto percorrem o grafo.

Produz uma estrutura bem mais sofisticada que a BFS.

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

DFS termina descrevendo via π uma floresta DF (Ou BP).

$u.\pi$: predecessor ou pai de u na floresta DF

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

DFS termina descrevendo via π uma floresta DF (Ou BP).

$u.\pi$: predecessor ou pai de u na floresta DF

Faz duas marcas de tempo:

$u.d$: momento da descoberta de u

$u.f$: momento da finalização de u

Busca em profundidade - elementos

Vértice branco: ainda não descoberto

Vértice cinzento: descoberto, em processamento

Vértice preto: processado

DFS termina descrevendo via π uma floresta DF (Ou BP).

$u.\pi$: predecessor ou pai de u na floresta DF

Faz duas marcas de tempo:

$u.d$: momento da descoberta de u

$u.f$: momento da finalização de u

u é branco antes de $u.d$,
cinzento entre $u.d$ e $u.f$,
preto depois de $u.f$.

Busca em profundidade

DFS (G)

- 1 **para cada** $u \in V(G)$ **faça**
- 2 $u.cor \leftarrow$ branco $u.\pi \leftarrow$ nil
- 3 tempo \leftarrow 0
- 4 **para cada** $u \in V(G)$ **faça**
- 5 **se** $u.cor =$ branco
- 6 **então** DFS-Visit(G, u)

Busca em profundidade

DFS (G)

- 1 **para cada** $u \in V(G)$ **faça**
- 2 $u.cor \leftarrow$ branco $u.\pi \leftarrow$ nil
- 3 **tempo** $\leftarrow 0$
- 4 **para cada** $u \in V(G)$ **faça**
- 5 **se** $u.cor =$ branco
- 6 **então** DFS-Visit(G, u)

DFS-Visit(G, u)

- 1 $u.cor \leftarrow$ cinzento $u.d \leftarrow$ tempo tempo \leftarrow tempo + 1
- 3 **para cada** $v \in u.Estrela$ **faça**
- 4 **se** $v.cor =$ branco
- 5 **então** $v.\pi \leftarrow u$
- 6 DFS-Visit(G, v)
- 7 $u.cor \leftarrow$ preto
- 8 $u.f \leftarrow$ tempo tempo \leftarrow tempo + 1

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco e, ao ser descoberto, passa a ser cinzento, e depois preto.

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco e, ao ser descoberto, passa a ser cinzento, e depois preto.

A lista de adjacência de cada vértice descoberto é percorrida **uma única vez**.

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco e, ao ser descoberto, passa a ser cinzento, e depois preto.

A lista de adjacência de cada vértice descoberto é percorrida uma única vez.

Logo o consumo de tempo é $O(n + m)$, onde $n = |V|$ e $m = |E|$, pois a inicialização custa $\Theta(n)$ e a soma do tamanho das listas de adjacências percorridas é $O(m)$.

Consumo de tempo

Cada vértice é descoberto uma única vez, pois é branco e, ao ser descoberto, passa a ser cinzento, e depois preto.

A lista de adjacência de cada vértice descoberto é percorrida uma única vez.

Logo o consumo de tempo é $O(n + m)$, onde $n = |V|$ e $m = |E|$, pois a inicialização custa $\Theta(n)$ e a soma do tamanho das listas de adjacências percorridas é $O(m)$.

O consumo de tempo de uma DFS é linear no tamanho do grafo.