Grafos e BFS

CLRS 22 Elementary Graph Algorithms

CLRS 22.1 e 22.2

Grafos

Grafo: par (V, E)

onde V é um conjunto (finito) de vértices,

E é um conjunto de arestas,

e uma função que associa a cada aresta um par de vértices (suas pontas).

Convenção:

- $n = |V|$
- $m = |E|$

Grafos

Grafo: par (V, E)
onde V é um conjunto (finito) de vértices,
E é um conjunto de arestas,
e uma função que associa a cada aresta
um par de vértices (suas pontas).

Convenção:

- $n = |V|$
- $m = |E|$

Se a cada aresta é associado um par *ordenado* de vértices,
o grafo é dirigido, senão é não dirigido.

Grafos

Grafo: par (V, E)
onde V é um conjunto (finito) de vértices,
E é um conjunto de arestas,
e uma função que associa a cada aresta
um par de vértices (suas pontas).

Convenção:

- $n = |V|$
- $m = |E|$

Se a cada aresta é associado um par *ordenado* de vértices,
o grafo é dirigido, senão é não dirigido.

Quando arestas distintas têm pontas distintas,
é possível identificar as arestas com o seu par de pontas.

Grandes × Gigantes

Essa terminologia não é padrão.

- Um grafo é só grande se ele pode ser armazenado em memória $O(n+m)$, é estático, ou tem modificações controladas, e algoritmos polinomiais em n e m são aceitáveis.

Grandes × Gigantes

Essa terminologia não é padrão.

- Um grafo é só grande se ele pode ser armazenado em memória $O(n+m)$, é estático, ou tem modificações controladas, e algoritmos polinomiais em n e m são aceitáveis.
- Um grafo é gigante se algoritmos polinomiais em n e m não são aceitáveis. Os principais tipos são:

Grandes × Gigantes

Essa terminologia não é padrão.

- Um grafo é só grande se ele pode ser armazenado em memória $O(n+m)$, é estático, ou tem modificações controladas, e algoritmos polinomiais em n e m são aceitáveis.
- Um grafo é gigante se algoritmos polinomiais em n e m não são aceitáveis. Os principais tipos são:
  - Grafos altamente dinâmicos, em que a presença de um vértice ou aresta é incerta, e cujo tamanho impede até buscas lineares.
    Ex: vários grafos associados à Internet, ou ao cérebro.
    Em geral, requerem algoritmos probabilísticos, sublineares, com erro.

Grandes × Gigantes

Essa terminologia não é padrão.

- Um grafo é só grande se ele pode ser armazenado em memória $O(n+m)$, é estático, ou tem modificações controladas, e algoritmos polinomiais em n e m são aceitáveis.
- Um grafo é gigante se algoritmos polinomiais em n e m não são aceitáveis. Os principais tipos são:
  - Grafos altamente dinâmicos, em que a presença de um vértice ou aresta é incerta, e cujo tamanho impede até buscas lineares.
    Ex: vários grafos associados à Internet, ou ao cérebro.
    Em geral, requerem algoritmos probabilísticos, sublineares, com erro.
  - Grafos descritos implicitamente, cujo tamanho é pelo menos exponencial na descrição.
    Ex: o autômato dos subconjuntos associado a um autômato não determinístico; vários modelos usados em IA.
    Queremos algoritmos polinomiais na descrição;
    alguns raros existem, em geral se usam heurísticas, com erro.

Grandes × Gigantes

Essa terminologia não é padrão.

- Um grafo é só grande se ele pode ser armazenado em memória $O(n+m)$, é estático, ou tem modificações controladas, e algoritmos polinomiais em n e m são aceitáveis.
- Um grafo é gigante se algoritmos polinomiais em n e m não são aceitáveis. Os principais tipos são:
  - Grafos altamente dinâmicos ...
  - Grafos descritos implicitamente ...

Nesta disciplina, só grafos grandes e algoritmos exatos.

A classe Grafo

Um grafo tem dois conjuntos...

A classe Grafo

Um grafo tem dois conjuntos...

A classe **Conjunto**

A classe Conjunto

Alguns métodos:

Pertence: dado um elemento, ele pertence ao conjunto?

A classe Conjunto

Alguns métodos:

- Pertence: dado um elemento, ele pertence ao conjunto?
- JunteElemento

A classe Conjunto

Alguns métodos:

- Pertence: dado um elemento, ele pertence ao conjunto?
- JunteElemento
- RemovaElemento

A classe Conjunto

Alguns métodos:

- Pertence: dado um elemento, ele pertence ao conjunto?
- JunteElemento
- RemovaElemento
- Iterador: para todo elemento do conjunto...

A classe Conjunto

Alguns métodos:

Pertence: dado um elemento, ele pertence ao conjunto?

JunteElemento

RemovaElemento

Iterador: para todo elemento do conjunto...

E os elementos?

A classe Conjunto

Alguns métodos:

Pertence: dado um elemento, ele pertence ao conjunto?

JunteElemento

RemovaElemento

Iterador: para todo elemento do conjunto...

E os elementos?

Em princípio, deve ser fácil adicionar propriedades.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

Inserir e Remover arestas dadas as pontas.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

- Inserir e Remover arestas dadas as pontas.
- Testar se dois vértices são adjacentes.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

- Inserir e Remover arestas dadas as pontas.
- Testar se dois vértices são adjacentes.
- Estrela: dado um vértice, conjunto de arestas incidentes a ele.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

Inserir e Remover arestas dadas as pontas.

Testar se dois vértices são adjacentes.

Estrela: dado um vértice, conjunto de arestas incidentes a ele.

- Para grafo dirigido, os arcos saindo do vértice.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

Inserir e Remover arestas dadas as pontas.

Testar se dois vértices são adjacentes.

Estrela: dado um vértice, conjunto de arestas incidentes a ele.

- Para grafo dirigido, os arcos saindo do vértice.
- Para grafo ou digrafo *simples*,
  pode ser o conjunto de *vizinhos* do vértice.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

Inserir e Remover arestas dadas as pontas.

Testar se dois vértices são adjacentes.

Estrela: dado um vértice, conjunto de arestas incidentes a ele.

- Para grafo dirigido, os arcos saindo do vértice.
- Para grafo ou digrafo *simples*,
  pode ser o conjunto de *vizinhos* do vértice.

A classe Grafo

Agora sim: dois conjuntos!
Que outros métodos e atributos?

São úteis:

Inserir e Remover arestas dadas as pontas.

Testar se dois vértices são adjacentes.

Estrela: dado um vértice, conjunto de arestas incidentes a ele.

- Para grafo dirigido, os arcos saindo do vértice.
- Para grafo ou digrafo *simples*,
  pode ser o conjunto de *vizinhos* do vértice.

Esquema típico de percurso de um grafo:

```
para todo vértice v
    para toda aresta na estrela de v
```

Atributos de vértices e arestas

Suponhamos que se queira dar uma cor a cada vértice.

Objetos:

Atributo v.cor (acesso direto, não vamos exagerar na POO)

Atributos de vértices e arestas

Suponhamos que se queira dar uma cor a cada vértice.

Objetos:
Atributo v.cor (acesso direto, não vamos exagerar na POO)

Vértices numerados:
Vetor paralelo: cor[v]

Atributos de vértices e arestas

Suponhamos que se queira dar uma cor a cada vértice.

Objetos:
Atributo v.cor (acesso direto, não vamos exagerar na POO)

Vértices numerados:
Vetor paralelo: cor[v]

Atributos de vértices e arestas

Suponhamos que se queira dar uma cor a cada vértice.

Objetos:
Atributo v.cor (acesso direto, não vamos exagerar na POO)

Vértices numerados:
Vetor paralelo: cor$[v]$

Daqui para a frente, usaremos a notação de atributos.

Representação de grafos

Grafo $G = (V, E)$.

Listas de adjacências:

Representação de grafos

Grafo $G = (V, E)$.

Listas de adjacências:

n listas, uma para cada vértice v de G:

v.adj: lista dos vértices que são adjacentes a v
(ou seja, vértices u tais que (u, v) ou $\{u, v\} \in E$)

Representação de grafos

Grafo $G = (V, E)$.

Listas de adjacências:

n listas, uma para cada vértice v de G:

v.adj: lista dos vértices que são adjacentes a v
(ou seja, vértices u tais que (u, v) ou $\{u, v\} \in E$)

Se G é dirigido, então m entradas, senão $2m$ entradas.

O tamanho da representação é $\Theta(n + m)$.

Representação de grafos

Grafo $G = (V, E)$.

Listas de adjacências:

n listas, uma para cada vértice v de G:

v.adj: lista dos vértices que são adjacentes a v
(ou seja, vértices u tais que (u, v) ou $\{u, v\} \in E$)

Se G é dirigido, então m entradas, senão $2m$ entradas.

O tamanho da representação é $\Theta(n + m)$.

Matriz de adjacências:

Representação de grafos

Grafo $G = (V, E)$.

Listas de adjacências:

n listas, uma para cada vértice v de G:

v.adj: lista dos vértices que são adjacentes a v
(ou seja, vértices u tais que (u, v) ou $\{u, v\} \in E$)

Se G é dirigido, então m entradas, senão $2m$ entradas.

O tamanho da representação é $\Theta(n + m)$.

Matriz de adjacências:

Matriz binária A de dimensão $n \times n$ onde
$A[u][v] = 1$ se e somente se (u, v) ou $\{u, v\} \in E$.

O tamanho da representação é $\Theta(n^2)$.

Principais diferenças

- Percorrer uma estrela:

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$
  - Matriz: $\mathrm{O}(1)$

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$
  - Matriz: $\mathrm{O}(1)$

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$
  - Matriz: $\mathrm{O}(1)$

Esparsidade

G é k-esparso se $m/n \leq k$.

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$
  - Matriz: $\mathrm{O}(1)$

Esparsidade

G é k-esparso se $m/n \leq k$.

- Neste caso, $m \leq kn \ll n^2$

Principais diferenças

- Percorrer uma estrela:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$ — percurso completo $\mathrm{O}(m)$
  - Matriz: $\mathrm{O}(n)$ — percurso completo $\mathrm{O}(n^2)$
- Testar adjacência:
  - Listas: $\mathrm{O}(\text{tamanho da estrela})$
  - Matriz: $\mathrm{O}(1)$

Esparsidade

G é k-esparso se $m/n \leq k$.

- Neste caso, $m \leq kn \ll n^2$
- e com a representação com listas, o percurso completo é mais eficiente que com matriz de adjacência (com n grande).

Busca em largura

É uma moldura de algoritmos que vão fazendo seu serviço enquanto percorrem o grafo.

Busca em largura

É uma moldura de algoritmos que vão fazendo seu serviço enquanto percorrem o grafo.

Inicialização

BFS (G, s)

```
1  para cada u ∈ G.V \ {s} faça
2      u.cor ← branco
3      u.d ← ∞
4      u.π ← nil
5  Q ← ∅          ▷ fila dos vértices descobertos
6  s.cor ← cinzento
7  s.d ← 0
8  s.π ← nil
9  Q.INSIRA(s)
```

Busca em largura

```
BFS (G, s)
 1  para cada u ∈ G.V \ {s} faça
 2        u.cor ← branco    u.d ← ∞    u.π ← nil
 3  Q ← ∅                 ▷ fila dos vértices descobertos
 4  s.cor ← cinzento    s.d ← 0    s.π ← nil
 5  Q.INSIRA(s)
```

Busca em largura

```
BFS (G, s)
 1  para cada u ∈ G.V \ {s} faça
 2       u.cor ← branco    u.d ← ∞    u.π ← nil
 3  Q ← ∅          ▷ fila dos vértices descobertos
 4  s.cor ← cinzento    s.d ← 0    s.π ← nil
 5  Q.INSIRA(s)

 6  enquanto Q ≠ ∅ faça
 7       u ← Q.REMOVA()
 8       para cada v ∈ u.Estrela faça
 9            se v.cor = branco
10                 então v.cor ← cinzento
11                       v.d ← u.d + 1
12                       v.π ← u
13                       Q.INSIRA(v)
14       u.cor ← 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

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.

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.

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 $O(m)$.

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 $v_1, v_2, \ldots, v_r$ nesta ordem,
então $v_r.d \leq v_1.d+1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \ldots, r-1$.

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 $v_1, v_2, \ldots, v_r$ nesta ordem, então $v_r.d \leq v_1.d+1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \ldots, r-1$.

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 $v_1, v_2, \ldots, v_r$ nesta ordem, então $v_r.d \leq v_1.d+1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \ldots, r-1$.

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 $v_1, v_2, \ldots, v_r$ nesta ordem, então $v_r.d \leq v_1.d+1$ e $v_i.d \leq v_{i+1}.d$ para $i = 1, \ldots, r-1$.

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 $\pi[v].v$.*