

CLRS 22 Elementary Graph Algorithms

CLRS 22.1 e 22.2

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.
- ▶ 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: vários modelos usados em IA,
o autômato dos subconjuntos associado a um autômato não determinístico.
Queremos algoritmos polinomiais na descrição; alguns raros existem, em geral se usam heurísticas, com erro.

Neste curso, só grafos grandes e algoritmos exatos.

A classe Grafo

Um grafo tem dois conjuntos...

A classe **Conjunto**

A classe Conjunto

Alguns métodos:

Pertence: dado um elemento, pertence ao conjunto?

AjunteElemento

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?

São úteis:

Inserir e **Remover** arestas dadas as pontas

Testar se dois vértices são adjacentes.

Estrela: dado um vértice, o 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)

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:

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 $O(n + m)$.

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 $O(n^2)$.

Principais diferenças

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

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.

Inicialização

BFS (G, s)

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

6  $s.cor \leftarrow$  cinzento
7  $s.d \leftarrow 0$ 
8  $s.\pi \leftarrow$  nil
9  $Q.INSIRA(s)$ 
```

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    $Q \leftarrow \emptyset$    ▷ 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

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.