

Grafos

Grafos

Orientados ou **não orientados**

Pesados (ou **etiquetados**) ou **não pesados** (**não etiquetados**)

$$G = (V, E)$$

V – conjunto dos **nós** (ou **vértices**)

$E \subseteq V^2$ – conjunto dos **arcos** (ou **arestas**)

$w : E \rightarrow \mathbb{R}$ – **peso** (ou **etiqueta**) de um arco

Vértices e arcos

Se $(u, v) \in E$

- ▶ O nó v diz-se **adjacente** ao nó u
- ▶ Num **grafo orientado**
 - ▶ O nó u é a **origem** do arco (u, v)
 - ▶ O nó v é o **destino** do arco (u, v)
 - ▶ O nó u é um **predecessor** (ou **antecessor**) do nó v
 - ▶ O nó v é um **sucessor** do nó u
- ▶ Num **grafo não orientado**
 - ▶ Os nós u e v são as **extremidades** do arco (u, v)
 - ▶ Os arcos (u, v) e (v, u) são o **mesmo** arco
 - ▶ Logo, o nó u também é **adjacente** ao nó v

O **grau** do nó u é o **número** de arcos $(u, v) \in E$

Caminhos

Um **caminho** num grafo $G = (V, E)$ qualquer é uma **sequência não vazia** de vértices $v_i \in V$

$$v_0 v_1 \dots v_k \quad (k \geq 0)$$

tal que $(v_i, v_{i+1}) \in E$, para $i < k$

O **comprimento** do caminho $v_0 v_1 \dots v_k$ é k , o número de arestas que contém

O caminho v_0 é o caminho de comprimento 0, de v_0 para v_0

Um caminho é **simples** se $v_i \neq v_j$ quando $i \neq j$

Ciclos

Um **ciclo**, num **grafo orientado**, é um caminho em que

$$v_0 = v_k \quad \text{e} \quad k > 0$$

Num **grafo não orientado**, um caminho forma um **ciclo** se

$$v_0 = v_k \quad \text{e} \quad k \geq 3$$

Um **ciclo** é **simples** se v_1, v_2, \dots, v_k são **distintos**

Um grafo é **acíclico** se não contém qualquer **ciclo simples**

Representação / Implementação

Listas de adjacências

- ▶ Grafos esparsos ($|E| \ll |V|^2$)
- ▶ Permite descobrir rapidamente os vértices adjacentes a um vértice
- ▶ Complexidade espacial $O(V + E)$

Matriz de adjacências

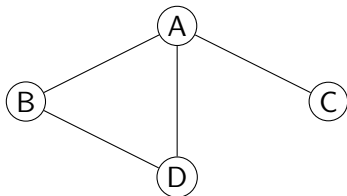
- ▶ Grafos densos ($|E| = O(V^2)$)
- ▶ Permite verificar rapidamente se $(u, v) \in E$
- ▶ Complexidade espacial $O(V^2)$

Na notação O , V e E significam, respectivamente, $|V|$ e $|E|$

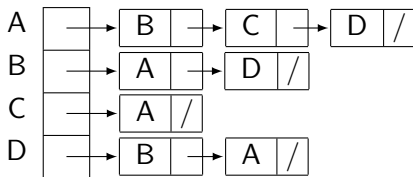
Representação / Implementação

Grafo não orientado e não pesado

Grafo $G = (\{A, B, C, D\}, \{(A, B), (B, D), (A, D), (C, A)\})$



Listas de adjacências



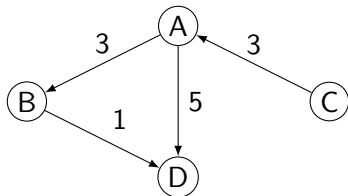
Matriz de adjacências

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 1 | 0 | 0 | 1 |
| C | 1 | 0 | 0 | 0 |
| D | 1 | 1 | 0 | 0 |

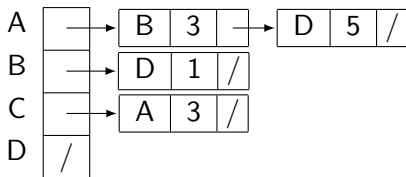
Representação / Implementação

Grafo orientado pesado

Grafo $G = (\{A, B, C, D\}, \{(A, B, 3), (B, D, 1), (A, D, 5), (C, A, 3)\})$



Listas de adjacências



Matriz de adjacências

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 3 | 0 | 5 |
| B | 0 | 0 | 0 | 1 |
| C | 3 | 0 | 0 | 0 |
| D | 0 | 0 | 0 | 0 |

Percursos básicos em grafos

Percurso em largura

Nós são tratados por ordem crescente de distância ao nó em que o percurso se inicia

Percurso em profundidade

Nós são tratados pela ordem por que são encontrados

Percurso em largura (a partir do vértice s)

BFS(G, s)

```
1  for each vertex  $u$  in  $G.V - \{s\}$  do
2       $u.color \leftarrow WHITE$ 
3       $u.d \leftarrow INFINITY$ 
4       $u.p \leftarrow NIL$ 
5   $s.color \leftarrow GREY$ 
6   $s.d \leftarrow 0$ 
7   $s.p \leftarrow NIL$ 
8   $Q \leftarrow EMPTY$                                 // queue
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq EMPTY$  do
11      $u \leftarrow DEQUEUE(Q)$                         // explore next vertex
12     for each vertex  $v$  in  $G.adj[u]$  do
13         if  $v.color = WHITE$  then
14              $v.color \leftarrow GREY$ 
15              $v.d \leftarrow u.d + 1$ 
16              $v.p \leftarrow u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color \leftarrow BLACK$                         //  $u$  has been explored
```

Percurso em largura

Breadth-first search

Descobre um **caminho mais curto** de um vértice **s** a qualquer outro vértice

Calcula o seu **comprimento** (linhas 3, 6 e 15)

Constrói a **árvore da pesquisa em largura** (linhas 4, 7 e 16), que permite reconstruir o caminho identificado

Atributos dos vértices

| | | |
|--------------|--|--------------------------------|
| color | WHITE | não descoberto |
| | GREY | descoberto, mas não processado |
| | BLACK | processado |
| d | distância a s | |
| p | antecessor do nó no caminho a partir de s | |

Análise da complexidade temporal de BFS (1)

Grafo implementado através de **listas de adjacências**

BFS(G, s)

```
1  for each vertex  $u$  in  $G.V - \{s\}$  do
2       $u.color \leftarrow WHITE$ 
3       $u.d \leftarrow INFINITY$ 
4       $u.p \leftarrow NIL$ 
```

- **Ciclo das linhas 1–4** é executado $|V| - 1$ vezes

```
5   $s.color \leftarrow GREY$ 
6   $s.d \leftarrow 0$ 
7   $s.p \leftarrow NIL$ 
8   $Q \leftarrow EMPTY$                                 // queue
9   $ENQUEUE(Q, s)$ 
```

- **Linhas 5–9** com custo constante

Análise da complexidade temporal de BFS (2)

- Ciclo das linhas 10–18 é executado $|V|$ vezes, no pior caso

```
10 while Q != EMPTY do
11     u <- DEQUEUE(Q)
12     for each vertex v in G.adj[u] do
13         if v.color = WHITE then
14             v.color <- GREY
15             v.d <- u.d + 1
16             v.p <- u
17             ENQUEUE(Q, v)
18     u.color <- BLACK
```

- Mas o ciclo das linhas 12–17 é executado, no pior caso

$$\sum_{v \in V} |G.adj[v]| = |E| \text{ (orientado)} \text{ ou } 2|E| \text{ (não orientado)} \text{ vezes}$$

porque cada vértice só entra na fila **uma** vez

Análise da complexidade temporal de BFS (3)

Considerando que todas as operações, incluindo ENQUEUE e DEQUEUE, têm custo $O(1)$

- ▶ O ciclo das linhas 1–4 tem custo $O(V)$
- ▶ Conjuntamente, os ciclos das linhas 10–18 e 12–17 têm custo $O(E)$

Logo, a complexidade temporal de BFS é $O(V + E)$

Análise da complexidade temporal de BFS (4)

Grafo implementado através da matriz de adjacências

Na linha 12, é necessário percorrer uma linha da matriz, com $|V|$ elementos

Como o ciclo das linhas 10–18 é executado $|V|$ vezes, no pior caso, o custo combinado dos dois ciclos é $O(V^2)$

- ▶ Correspondente a aceder a todas as posições de uma matriz $|V| \times |V|$

Neste caso, a complexidade temporal de BFS será $O(V^2)$

Percurso em profundidade

DFS(G)

```
1 for each vertex u in G.V do
2     u.color <- WHITE
3     u.p <- NIL
4 time <- 0                      // global variable
5 for each vertex u in G.V do
6     if u.color = WHITE then
7         DFS-VISIT(G, u)
```

DFS-VISIT(G, u)

```
1 time <- time + 1              // white vertex u has just
2 u.d <- time                    // been discovered
3 u.color <- GREY
4 for each vertex v in G.adj[u] do // explore edge (u, v)
5     if v.color = WHITE then
6         v.p <- u
7         DFS-VISIT(G, v)
8 u.color <- BLACK               // blacken u; it is finished
9 time <- time + 1
10 u.f <- time                   // record u's finishing time
```


Percurso em profundidade

Depth-first search

Constrói a **floresta da pesquisa em profundidade** (linhas 3 [DFS] e 6 [DFS-VISIT])

Atributos dos vértices

| | | |
|--------------|--|-------------------------------|
| color | WHITE | não descoberto |
| | GREY | descoberto e em processamento |
| | BLACK | processado |
| d | instante em que foi descoberto | |
| f | instante em que terminou de ser processado | |
| p | antecessor do nó num caminho que o contém | |