Grafos

Grafos

Orientados ou **não orientados**

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

Grafo $G = (V, E)$

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

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

Vértices e arcos

Se $G = (V, E)$ e $(u, v) \in E$

- O nó v diz-se adjacente ao nó u
- Os nós u e v são vizinhos
- Se G é 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
- Se G é 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 $v_i \in V$

$$v_0\, v_1\, \ldots\, v_k \quad (k \geq 0)$$

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

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

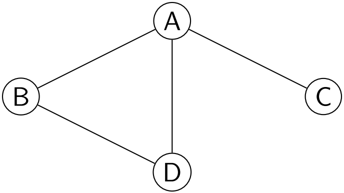
O caminho v_0 é o caminho de comprimento 0, de v_0 para v_0

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

Exemplos de grafos

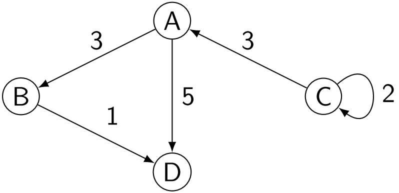
Grafo não orientado e não pesado

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



Grafo orientado pesado

$$G = (\{A, B, C, D\}, \{(A, B, 3), (B, D, 1), (A, D, 5), (C, A, 3), (C, C, 2)\})$$



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 $v_1, v_2, \ldots, v_k$ são distintos

Um grafo é acíclico se não contém qualquer ciclo simples

Representação / Implementação

Listas de adjacências

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

Matriz de adjacências

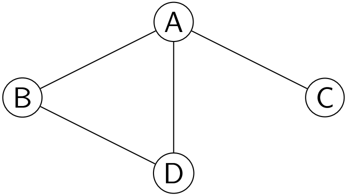
- Grafos densos $\quad (|E| = O(V^2))$
- Permite verificar rapidamente se $(u, v) \in E$
- Complexidade espacial $\; O(V^2)$

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

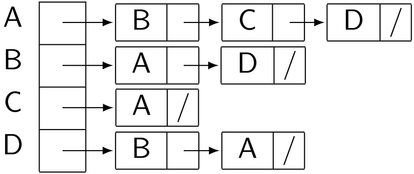
Representação / Implementação

Grafo não orientado e não pesado

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



Listas de adjacências



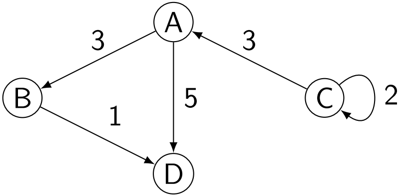
Matriz de adjacências

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0

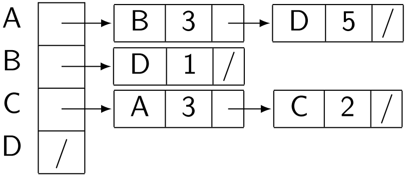
Representação / Implementação

Grafo orientado pesado

Grafo $G = (\{A, B, C, D\}, \{(A, B, 3), (B, D, 1), (A, D, 5), (C, A, 3), (C, C, 2)\})$



Listas de adjacências



Matriz de adjacências

	A	B	C	D
A	0	3	0	5
B	0	0	0	1
C	3	0	2	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)

```
for each vertex u in G.V - {s} do
    u.color <- WHITE
    u.d <- INFINITY
    u.p <- NIL
s.color <- GREY
s.d <- 0
s.p <- NIL
Q <- EMPTY                             // queue
ENQUEUE(Q, s)
while Q != EMPTY do
    u <- DEQUEUE(Q)                    // explore next vertex
    for each vertex v in G.adj[u] do
        if v.color = WHITE then
            v.color <- GREY
            v.d <- u.d + 1
            v.p <- u
            ENQUEUE(Q, v)
    u.color <- 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 <- WHITE
3      u.d <- INFINITY
4      u.p <- NIL
```

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

```
5  s.color <- GREY
6  s.d <- 0
7  s.p <- NIL
8  Q <- 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

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

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)

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

DFS-VISIT(G, u)

```
time <- time + 1                  // white vertex u has just
u.d <- time                       // been discovered
u.color <- GREY
for each vertex v in G.adj[u] do // explore edge (u, v)
    if v.color = WHITE then
        v.p <- u
        DFS-VISIT(G, v)
u.color <- BLACK                  // blacken u; it is finished
time <- time + 1
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	

Análise da complexidade temporal de DFS

O ciclo das linhas 1–3 [DFS] é executado $|V|$ vezes

DFS-VISIT é chamada para cada um dos $|V|$ vértices

Para cada vértice u (e considerando a implementação através de listas de adjacências), o ciclo das linhas 4–7 [DFS-VISIT] é executado

$$|\,\texttt{G.adj}[u]\,| \quad \text{vezes}$$

Tendo todas as operações custo constante, considerando todas as chamadas a DFS-VISIT, DFS corre em tempo

$$O(V + \sum_{u \in V} |\,\texttt{G.adj}[u]\,|) = O(V + E)$$