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

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

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 $O(V^2)$

- Corresponde 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

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

Ordenação topológica

Seja $G = (V, E)$ um grafo orientado acíclico (DAG, de *directed acyclic graph*)

Ordem topológica

Se existe um arco de u para v, u está antes de v na ordem dos vértices

$$(u, v) \in E \;\Rightarrow\; u < v$$

TOPOLOGICAL-SORT(G)

1. Aplicar DFS(G)
2. Inserir cada vértice à cabeça de uma lista, quando termina o seu processamento
3. Devolver a lista, que contém os vértices por (alguma) ordem topológica

Ordenação topológica

Adaptação de DFS

$G = (V, E)$ – grafo orientado acíclico (DAG)

TOPOLOGICAL-SORT(G)

```
for each vertex u in G.V do
    u.color <- WHITE
L <- EMPTY                           // lista, global
for each vertex u in G.V do
    if u.color = WHITE then
        DFS-VISIT'(G, u)
return L
```

DFS-VISIT'(G, u)

```
u.color <- GREY
for each vertex v in G.adj[u] do
    if v.color = WHITE then
        DFS-VISIT'(G, v)
u.color <- BLACK
LIST-INSERT-HEAD(L, u)
```

Ordenação topológica

Outro algoritmo

TOPOLOGICAL-SORT'(G)

```
for each vertex u in G.V do
    u.i <- 0
for each edge (u,v) in G.E do
    v.i <- v.i + 1                 // arcos com destino v
L <- EMPTY                         // lista
S <- EMPTY                         // conjunto
for each vertex u in G.V do
    if u.i = 0 then
        SET-INSERT(S, u)
while S != EMPTY do
    u <- SET-DELETE(S)             // retira um nó de S
    for each vertex v in G.adj[u] do
        v.i <- v.i - 1
        if v.i = 0 then
            SET-INSERT(S, v)
    LIST-INSERT-TAIL(L, u)
return L
```