

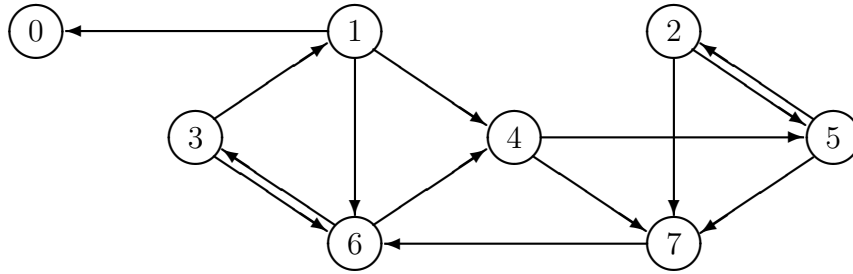
1º Teste de Análise e Desenho de Algoritmos

Departamento de Informática

Universidade Nova de Lisboa

26 de Abril de 2014

1. Suponha que se executa o algoritmo *bfsTraversal* (que percorre os vértices de um grafo em largura) com o grafo *G* esquematizado na figura.



Assuma que os métodos *vertices* e *outAdjacentVertices* iteram sempre os vértices por ordem crescente. Por exemplo, *G.outAdjacentVertices(1)* produz os vértices 0, 4 e 6 (por esta ordem). Indique:

- (a) [2 valores] a ordem pela qual os vértices são percorridos (passados a *TREAT* como argumento);
 - (b) [1 valor] o número de vezes que o método *bfsExplore* é executado;
 - (c) [1 valor] o maior número de vértices presentes simultaneamente na fila (ou seja, o maior valor de *waiting.size()*); e
 - (d) [1 valor] os vértices que se encontram na fila numa situação de maior ocupação (i.e., num caso em que *waiting.size()* é máximo).
2. [4 valores] O vetor *uf.partition* contém o estado da partição *uf* (i.e., o estado da instância *uf* da classe *UnionFindInArray*). Assuma que se adotou união por nível e representante com compressão do caminho.

uf.partition:

-3	0	0	2	-3	4	4	6	4
0	1	2	3	4	5	6	7	8

Indique o estado da partição (ou seja, o conteúdo do vetor *uf.partition*) após a execução de **cada uma** das seguintes instruções, pela ordem indicada.

```
int r1 = uf.find(3);
int r2 = uf.find(7);
int r3 = uf.find(8);
uf.union(0, 4);
```

3. Considere o algoritmo *noName*, que recebe um grafo não orientado.

```
int noName( UndiGraph graph )
{
    Iterator<Edge> allEdges = graph.edges();
    UnionFind vertPartition = new UnionFindInArray( graph.numVertices() );
    int xpto = graph.numVertices();
    while ( allEdges.hasNext() && xpto > 1 )
    {
        Edge edge = allEdges.next();
        Vertex[] endPoints = edge.endVertices();
        int rep1 = vertPartition.find( endPoints[0] );
        int rep2 = vertPartition.find( endPoints[1] );
        if ( rep1 != rep2 )
        {
            xpto--;
            vertPartition.union(rep1, rep2);
        }
    }
    return xpto;
}
```

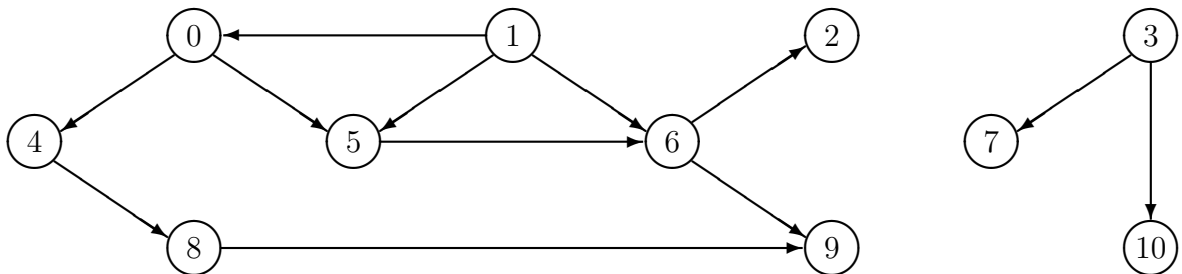
- (a) [1 valor] Apresente um grafo G' com 5 vértices tal que $noName(G')$ retorna 3.
- (b) [1 valor] O que é que o algoritmo *noName* calcula? Seja conciso e utilize a nomenclatura de grafos na sua resposta.
- (c) [1 valor] Que nome se dá aos grafos G tais que $noName(G)$ retorna 1?
- (d) Assuma que os seguintes métodos têm complexidade constante:
- os métodos *edges* e *numVertices* do grafo;
 - os métodos *hasNext* e *next* do iterador;
 - o método *endVertices* do arco.

Em relação à partição, considere que se adotou união por tamanho e representante sem efeitos laterais.

- (i) [1 valor] Qual é a complexidade temporal de *noName*, no pior caso? Justifique e indique uma situação em que o pior caso ocorre.
- (ii) [1 valor] Qual é a complexidade temporal de *noName*, no melhor caso? Justifique e indique uma situação em que o melhor caso ocorre.

No cálculo da complexidade temporal de *noName*, tanto no pior caso, como no melhor, assuma que o número de arcos do grafo é muito superior ao número de vértices.

4. [6 valores] Considere um grafo orientado e acíclico, onde cada vértice representa uma tarefa de um projeto. A existência de um arco do vértice v para o vértice w indica que a tarefa v terá de estar concluída antes de se iniciar a tarefa w . Assuma que todas as tarefas demoram exatamente um dia e que qualquer tarefa se realiza logo que todas as tarefas que a precedem tiverem terminado. Pretende-se descobrir o número máximo de tarefas realizadas num (só) dia.



Por exemplo, no caso do grafo representado acima, as tarefas do projeto seriam realizadas nos seguintes dias.

Dia	Tarefas Realizadas no Dia
1	1 e 3
2	0, 7 e 10
3	4 e 5
4	6 e 8
5	2 e 9

Portanto, o número máximo de tarefas realizadas num (só) dia é 3.

Escreva um algoritmo (em pseudo-código) que recebe um grafo orientado e acíclico e retorna o número máximo de tarefas realizadas num (só) dia. Estude (justificando) a complexidade temporal do seu algoritmo.