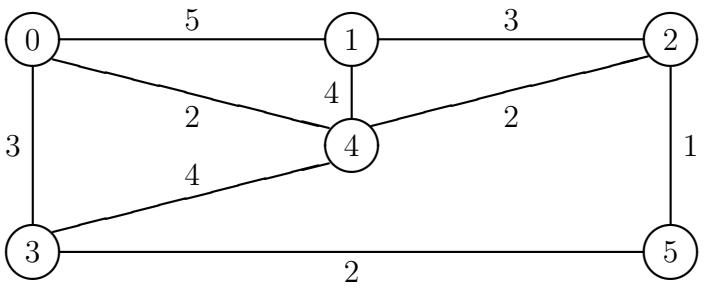
Exame de Análise e Desenho de Algoritmos

Departamento de Informática
Universidade Nova de Lisboa
27 de Junho de 2011

1. [3 valores] Suponha que se executa o algoritmo de Prim com o grafo G esquematizado na figura.



Assumindo que a origem é o vértice 0 (ou seja, que o método $G.aVertex()$ retorna 0), indique:

- a ordem pela qual os vértices são seleccionados;
- o número total de vezes que o método *decreaseKey* é executado; e
- a árvore mínima de cobertura encontrada (representando-a como quiser).

2. [3 valores] Considere a seguinte função recursiva $C(i, j)$, onde i e j são números inteiros tais que $0 \leq j \leq i$.

$$C(i,j) = \begin{cases} 1 & \text{se } i \geq 0 \text{ e } j = 0; \\ 1 & \text{se } i \geq 1 \text{ e } j = i; \\ C(i-1,j) + C(i-1,j-1) & \text{se } i,j \geq 1 \text{ e } j < i. \end{cases}$$

Apresente um algoritmo, desenhado segundo a técnica da programação dinâmica, que, dados dois números inteiros m e n tais que $0 \leq n \leq m$, calcula o valor de $C(m, n)$. Estude (justificando) as complexidades temporal e espacial do seu algoritmo, no pior caso.

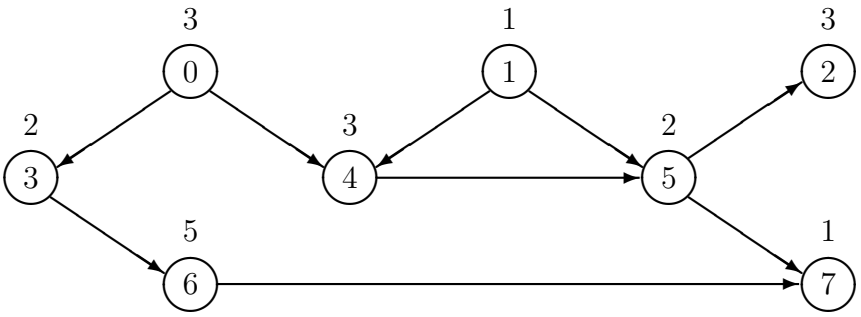
(Continue, porque o exame tem mais quatro perguntas.)

3. [3 valores] Considere a seguinte hierarquia de interfaces e classes.

```
public interface UnionFindWithSizes extends UnionFind
{
    // Returns the number of sets in the partition.
    int size( );

    // Returns the number of elements in the set that contains
    // the specified element.
    int size( int element ) throws InvalidElementException;
}

public interface UnionFind
{
    // Returns the representative of the set that contains the specified element.
    int find( int element ) throws InvalidElementException;

    // Removes the two distinct sets S1 and S2 whose representatives are
    // the specified elements, and inserts the set S1 ∪ S2.
    // The representative of the new set S1 ∪ S2 can be any of its members.
    void union( int representative1, int representative2 ) throws
        InvalidElementException, NotRepresentativeException
        EqualSetsException;
}

public class UnionFindInArray implements UnionFind
{
    // The partition is a forest implemented in an array.
    protected int[] partition;

    // Definition of the range of valid elements.
    protected String validRangeMsg;

    // Creates the partition {{0}, {1}, ..., {domainSize − 1}}.
    public UnionFindInArray( int domainSize ) { ··· }

    . . . . . .
}

public class UnionFindWithSizesInArray extends UnionFindInArray
    implements UnionFindWithSizes
{
    . . . . . .
}
```

Implemente a classe *UnionFindWithSizesInArray* e calcule as complexidades temporais dos métodos *find*, *union*, *size/0* e *size/1*, no melhor caso e no pior caso, justificando. Se recorrer aos métodos da classe *UnionFindInArray*, identifique as versões que seleccionaria (união por dimensão, união por altura, etc.). **Sugestão:** Use o mecanismo de herança do Java e minimize o número de linhas de código da nova classe.

4. [4 valores] Considere um grafo orientado, pesado e acíclico, onde cada vértice representa uma tarefa de um projecto. 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. O peso (positivo) de um vértice é a duração (em dias) da tarefa. Pretende-se descobrir a *duração mínima* (em dias) do projecto, assumindo que qualquer tarefa se pode realizar logo que todas as tarefas que a precedem tiverem terminado.



Por exemplo, a duração mínima do projecto esquematizado na figura é 11. Para uma melhor compreensão do problema, indica-se o primeiro e o último dia em que cada tarefa decorreria.

Tarefa	0	1	2	3	4	5	6	7
Primeiro dia	1	1	9	4	4	7	6	11
Último dia	3	1	11	5	6	8	10	11

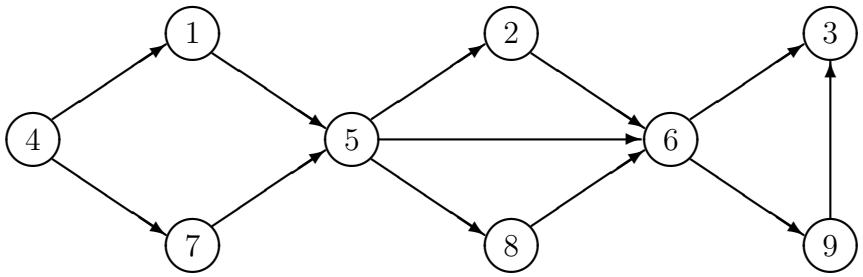
Escreva um algoritmo (em pseudo-código) que, dados um grafo orientado e acíclico, com a informação sobre as tarefas do projecto, e um vector *weights* de inteiros positivos tal que $weights[v]$ tem a duração da tarefa v, calcula a duração mínima do projecto. Estude (justificando) a complexidade temporal do seu algoritmo, no pior caso.

(Continue, porque o exame tem mais duas perguntas.)

5. [4 valores] Sejam $G = (V, A)$ um grafo orientado e n um inteiro positivo. Uma *cobertura de caminhos de G de cardinalidade n* é um conjunto $V' \subseteq V$ tal que:

$$|V'| = n \quad \text{e} \quad \text{todo o caminho de comprimento 2 em } G \text{ tem um vértice de } V'.$$

Por exemplo, $\{5, 6\}$ é uma cobertura de caminhos de cardinalidade 2 do grafo esquematizado na figura.



O **Problema da Cobertura de Caminhos** formula-se da seguinte forma.
Dados um grafo orientado G e um inteiro positivo n, existe uma cobertura de caminhos de G de cardinalidade inferior ou igual a n?

Prove que o Problema da Cobertura de Caminhos é NP-completo.

6. [3 valores] O jogo *Sequint*, jogado por dois adversários, começa com uma sequência (não vazia) com um número par de inteiros positivos, todos distintos. Em cada jogada, um dos jogadores retira, ou o número mais à esquerda, ou o número mais à direita, reduzindo o comprimento da sequência em uma unidade. Os jogadores jogam alternadamente, enquanto a sequência não for vazia. Ganha o jogador cuja soma dos números retirados for maior. Se as duas somas forem iguais, há empate.

Imagine que joga o Sequint com uma criança, que retira sempre o número maior (dos dois possíveis). Por ser uma criança, o seu adversário é sempre o primeiro a jogar. Para exemplificar, suponha que a sequência inicial era $(12, 3, 5, 1, 30, 10)$. Uma sequência de jogadas possível (que lhe daria a victória) seria 12, 3, 10, 30, 5, 1.

Apresente **uma função recursiva** que, dada uma sequência inicial do Sequint,

$$(x_1, x_2, \ldots, x_n), \quad \text{onde } n \text{ é um número par positivo,}$$

calcula a maior soma que poderia obter se jogasse com a criança. Indique claramente o que representa cada uma das variáveis que utilizar e explicite a chamada inicial.