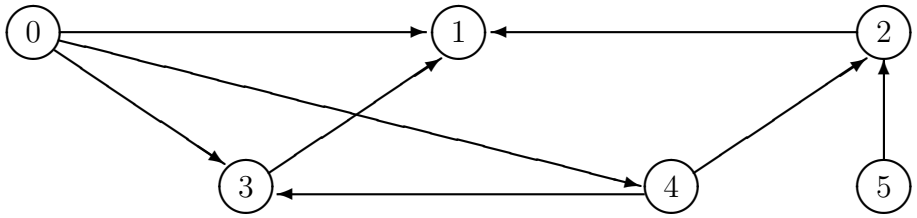
Recurso de Análise e Desenho de Algoritmos

Departamento de Informática

Universidade Nova de Lisboa

5 de Julho de 2010

1. [3 valores] Suponha que se executa o algoritmo *topologicalSort* com o grafo G esquematizado na figura. Relembre que a fila *ready* tem disciplina FIFO.



Assuma que os métodos *vertices* e *outAdjacentVertices* iteram sempre os vértices por ordem crescente. Por exemplo, $G.outAdjacentVertices(0)$ produz os vértices 1, 3 e 4 (por esta ordem). Indique:

- a ordem pela qual os vértices são tratados (passados a *TREAT* como argumento);
- o maior número de vértices presentes simultaneamente na fila (ou seja, o maior valor de *ready.size*()); e
- o conteúdo do vector *inCounter*, quando o algoritmo termina.

2. [3 valores] Considere a seguinte função recursiva $f_{X,Y,Z}(i,j)$, onde $X = (x_1, x_2, \ldots, x_m)$, $Y = (y_1, y_2, \ldots, y_n)$ e $Z = (z_1, z_2, \ldots, z_{m+n})$ são sequências não vazias de números reais, i é um inteiro entre 0 e m, e j é um inteiro entre 0 e n.

$$f_{X,Y,Z}(i,j) = \begin{cases} \text{true} & \text{se } i = 0 \text{ e } j = 0; \\ f_{X,Y,Z}(i,j-1) & \text{se } i = 0,\ j > 0 \text{ e } y_j = z_j; \\ \text{false} & \text{se } i = 0,\ j > 0 \text{ e } y_j \neq z_j; \\ f_{X,Y,Z}(i-1,j) & \text{se } i > 0,\ j = 0 \text{ e } x_i = z_i; \\ \text{false} & \text{se } i > 0,\ j = 0 \text{ e } x_i \neq z_i; \\ f_{X,Y,Z}(i-1,j) \vee f_{X,Y,Z}(i,j-1) & \text{se } i,j > 0,\ x_i = z_{i+j} \text{ e } y_j = z_{i+j}; \\ f_{X,Y,Z}(i-1,j) & \text{se } i,j > 0,\ x_i = z_{i+j} \text{ e } y_j \neq z_{i+j}; \\ f_{X,Y,Z}(i,j-1) & \text{se } i,j > 0,\ x_i \neq z_{i+j} \text{ e } y_j = z_{i+j}; \\ \text{false} & \text{se } i,j > 0,\ x_i \neq z_{i+j} \text{ e } y_j \neq z_{i+j}. \end{cases}$$

Apresente um algoritmo, desenhado segundo a técnica da programação dinâmica, que, dadas três sequências não vazias de reais, $X = (x_1, x_2, \ldots, x_m)$, $Y = (y_1, y_2, \ldots, y_n)$ e $Z = (z_1, z_2, \ldots, z_{m+n})$, calcula o valor de $f_{X,Y,Z}(m,n)$. Estude (justificando) as complexidades temporal e espacial do seu algoritmo, no melhor caso, no pior caso e no caso esperado.

3. [3 valores] Considere a classe *BinNode*, dos nós das árvores binomiais de chaves do tipo K e valores do tipo V, e a classe *BinQueue*, das filas binomiais de chaves do tipo K e valores do tipo V, ambas internas ao pacote *dataStructures*.

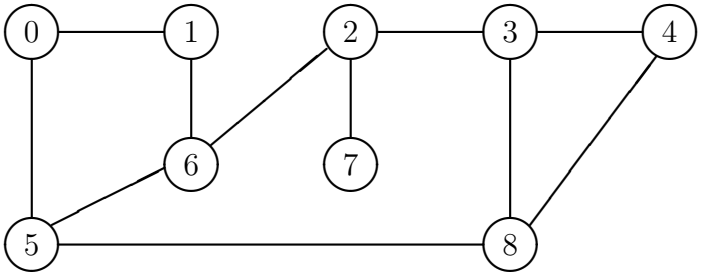
```
class BinNode<K,V>
{
    .....
    public BinNode( K key, V value );
    public Entry<K,V> getEntry( );
    public K getKey( );
    public V getValue( );
    public int getDegree( );
    public BinNode<K,V> getChild( );
    public BinNode<K,V> getLeftSibling( );
    public BinNode<K,V> getRightSibling( );
    public BinNode<K,V> getParent( );
    public void setEntry( Entry<K,V> newEntry );
    public void setEntry( K newKey, V newValue );
    public void setKey( K newKey );
    public void setValue( V newValue );
    public void setDegree( int newDegree );
    public void setChild( BinNode<K,V> newChild );
    public void setLeftSibling( BinNode<K,V> newLeftSibling );
    public void setRightSibling( BinNode<K,V> newRightSibling );
    public void setParent( BinNode<K,V> newParent );
}

class BinQueue<K extends Comparable<K>, V>
{
    .....
    // Decomposes the specified binomial tree, whose degree is k > 0,
    // into two independent binomial trees of degree k − 1,
    // and returns them in the array (whose length is two).
    private BinNode<K,V>[] decompose( BinNode<K,V> tree )
        throws InvalidDegreeException;
}
```

O método *decompose* é privado e não depende dos atributos da classe. Recebe uma árvore binomial de grau k e, se k for positivo, decompõe a árvore em duas árvores binomiais de grau $k-1$, retornando-as no vector. A árvore inicial deixa de existir porque não se criam nós; apenas se alteram ligações. Em qualquer uma das três árvores binomiais referidas, o irmão esquerdo da raiz é a raiz e o irmão direito da raiz é *null*.

Implemente o método *decompose* e calcule a complexidade temporal do seu algoritmo, no melhor caso e no pior caso, justificando.

4. [4 valores] Considere um grafo não orientado (e não pesado). Dados dois vértices v e w, entre os quais existe, pelo menos, um caminho, a *distância mínima* entre v e w é o comprimento dos menores caminhos entre v e w.

   Por exemplo, no grafo representado na figura, a distância mínima entre 4 e 5 é dois, porque o caminho 4 8 5 tem comprimento dois e nenhum caminho entre 4 e 5 tem comprimento inferior a dois.

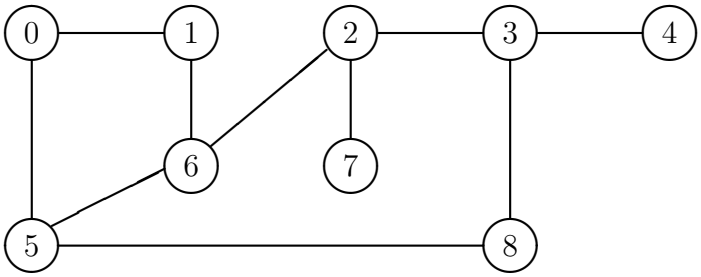


   Dados um grafo não orientado, um vértice v e um inteiro positivo d, pretende-se encontrar todos os vértices cuja distância mínima a v é d. No grafo do exemplo, os vértices cuja distância mínima a 6 é dois são 0, 3, 7 e 8.

   Escreva um algoritmo (em pseudo-código) que, dados um grafo não orientado, um vértice v e um inteiro positivo d, retorna um iterador com todos os vértices do grafo cuja distância mínima a v é d. A ordem dos vértices no iterador é arbitrária. Para simplificar, pode assumir que há, pelo menos, um vértice nas condições pedidas. Estude (justificando) a complexidade temporal do seu algoritmo, no pior caso.

5. [4 valores] Sejam G um grafo não orientado e k um inteiro positivo. Diz-se que G *tem um caminho mais longo que* k se, e só se, existir um caminho simples em G cujo comprimento é superior a k.

   Por exemplo, o grafo esquematizado na figura tem um caminho mais longo que cinco, porque 5 0 1 6 2 3 4 é um caminho simples com comprimento seis.



   O **Problema do Caminho Mais Longo** formula-se da seguinte forma.
   Dados um grafo G não orientado e um inteiro positivo k, G tem um caminho mais longo que k?

   Prove que o Problema do Caminho Mais Longo é NP-completo.

6. [3 valores] O João quer conhecer melhor Portugal, tendo decidido visitar várias cidades, viajando sempre de comboio. Ele já definiu a sequência de viagens que vai realizar e, para cada uma, já sabe a data e o preço do bilhete de comboio. Recentemente, descobriu que a CP vende um cartão de desconto, que custa 240 euros e que dá 50% de desconto no preço de todos os bilhetes de comboio comprados durante um ano, a partir da data da aquisição do cartão. O objectivo é ajudar o João a determinar o custo mínimo da totalidade das viagens, sabendo que se podem comprar os cartões de desconto da CP.

   Para exemplificar, considere a sequência de viagens da tabela seguinte. O custo mínimo da totalidade das viagens seria 1 250 euros, adquirindo dois cartões de desconto: o primeiro a 11 de Agosto de 2011 e o segundo a 11 de Agosto de 2014.

Viagem	**Data**	**Preço (em euros)**
1	11 / Jul / 2010	20
2	11 / Ago / 2011	200
3	11 / Jul / 2012	300
4	11 / Set / 2012	100
5	11 / Ago / 2014	200
6	11 / Dez / 2014	600

Tabela com a data e o preço de cada viagem.

   Apresente **uma função recursiva** que, dados:

   - o número (positivo) n de viagens a realizar;
   - o custo d, em euros, do cartão de desconto (que, no exemplo, é 240);
   - a sequência d_1, d_2, ..., d_n das datas das viagens, ordenada cronologicamente; e
   - a sequência p_1, p_2, ..., p_n dos preços das respectivas viagens, em euros,

   calcula o custo mínimo da totalidade das viagens, assumindo que cada cartão de desconto é válido por um ano e que o desconto é 50%. Indique claramente o que representa cada uma das variáveis que utilizar e explicite a chamada inicial.