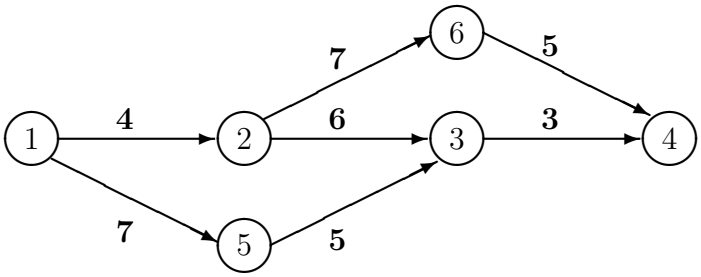
Exame de Análise e Desenho de Algoritmos

Departamento de Informática
Universidade Nova de Lisboa
1 de Julho de 2008

1. [3 valores] Suponha que se executa o algoritmo de Edmonds-Karp com o grafo G esquematizado na figura, a fonte 1 e o dreno 4.



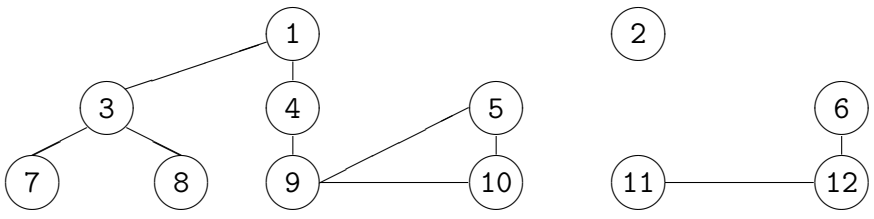
Assuma que o método *outIncidentEdges* itera sempre os vértices por ordem crescente. Por exemplo, $G.outIncidentEdges(1)$ produz os vértices 2 e 5 (por esta ordem). Indique:

- a sequência de caminhos encontrados da fonte para o dreno e, para cada um desses caminhos, o valor do incremento;
- o valor do fluxo máximo e o fluxo de cada arco de G, quando o algoritmo termina.

2. [4 valores] Pretende-se colorir os vértices de um grafo $G = (V, A)$ não orientado, de forma a satisfazer a seguinte restrição, para quaisquer $v, w \in V$:

v e w são coloridos com a mesma cor se, e só se, existe um caminho de v para w em G.

Por exemplo, no grafo representado na figura, seria necessário usar três cores: uma cor para os vértices `1`, `3`, `4`, `5`, `7`, `8`, `9` e `10`; outra cor para o vértice `2`; e a terceira cor para os vértices `6`, `11` e `12`. Oito é o número máximo de vértices que são coloridos com a mesma cor.

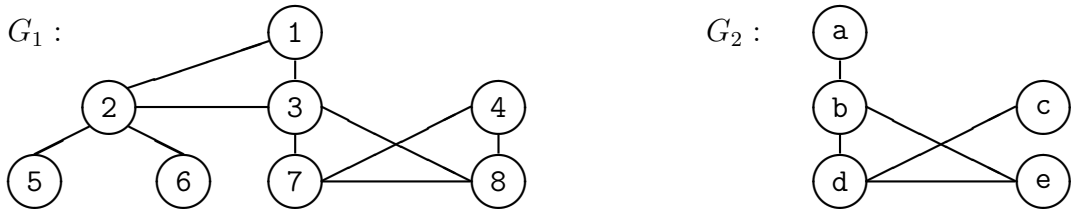


Escreva um algoritmo (em pseudo-código) que, dado um grafo não orientado, calcula o número de cores necessárias para colorir o grafo e o número máximo de vértices que são coloridos com a mesma cor. Estude (justificando) a complexidade temporal do seu algoritmo, no pior caso.

3. [4 valores] Sejam $G = (V, A)$ e $G^\star = (V^\star, A^\star)$ dois grafos não orientados. Diz-se que G *tem um subgrafo isomorfo a* $G^\star$ se, e só se, existir uma função injectiva $f : V^\star \longrightarrow V$ tal que:

$$(\forall x, y \in V^\star)\ (x, y) \in A^\star \implies (f(x), f(y)) \in A.$$

Em relação aos grafos representados na figura, é fácil verificar que G_1 tem um subgrafo isomorfo a G_2. Basta considerar que os vértices `a`, `b`, `c`, `d` e `e` correspondem aos vértices `1`, `3`, `4`, `7` e `8`, respectivamente.



O **Problema do Subgrafo Isomorfo** formula-se da seguinte forma.
Dados dois grafos não orientados, G e $G^\star$, G tem um subgrafo isomorfo a $G^\star$?

Prove que o Problema do Subgrafo Isomorfo é NP-completo.

4. [3 valores] Considere a seguinte função recursiva $f(i, j)$, onde i e j são inteiros positivos tais que $i < j$, e g é uma função inteira definida para quaisquer três inteiros.

$$f(i,j) = \begin{cases} 0, & \text{se } j - i = 1 \text{ ou } j - i = 2; \\ \min\limits_{i<k<j} \left(f(i,k) + f(k,j) + g(i,k,j)\right), & \text{se } j - i > 2. \end{cases}$$

Apresente um algoritmo, desenhado segundo a técnica da programação dinâmica, que, dados dois inteiros positivos m e n tais que $m < n$, calcula o valor da função $f(m, n)$. Assuma que a função g está implementada. Estude (justificando) a complexidade temporal e espacial do seu algoritmo, no pior caso, considerando que a complexidade temporal e espacial de g é sempre constante.

5. [3 valores] Suponha que lhe dão uma sequência não vazia de números em binário e dois inteiros não negativos, U e Z, que indicam o número de uns e de zeros de que dispõe. O objectivo é descobrir o maior número de elementos da sequência que podem ser construídos com esses U uns e Z zeros.

Por exemplo, se a sequência fosse $(11, 1010, 100)$ e só pudesse usar $U = 3$ uns e $Z = 4$ zeros, poderia formar, no máximo, dois elementos da sequência: ou 11 e 100; ou 1010 e 100.

Apresente **uma função recursiva** que, dados uma sequência s (de comprimento $n \geq 1$) de números em binário e dois inteiros não negativos, U e Z, calcula o maior número de elementos de s que podem ser construídos usando, no máximo, U uns e Z zeros. Indique claramente o que representa cada uma das variáveis que utilizar e explicite a chamada inicial.

6. [3 valores] O método *deepestEntries*, da classe interna *BinQueue*, retorna um vector com todas as entradas da fila binomial que se encontram à profundidade máxima. Por outras palavras, se 4 for a maior profundidade de um nó da fila, o vector conterá apenas todas as entradas à profundidade 4.

```
class BinNode<K,V>
{
    public BinNode( K key, V value );
    public EntryClass<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( EntryClass<K,V> newEntry );
    public void setEntry( K newKey, V newValue );
    public void setValue( V newValue );
    public void setDegree( int newDegree );
    public void incrementDegree( );
    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>
{
    // (Pointer to) the first binomial tree.
    protected BinNode<K,V> head;

    .....

    // Returns an array with the entries stored in
    // the deepest nodes of the binomial queue,
    // if the binomial queue is not empty;
    // otherwise, returns null.
    protected Entry<K,V>[] deepestEntries( );
}
```

Implemente o método *deepestEntries* (na classe interna *BinQueue*) e calcule a complexidade temporal do seu algoritmo, no melhor caso, no pior caso e no caso esperado, justificando.