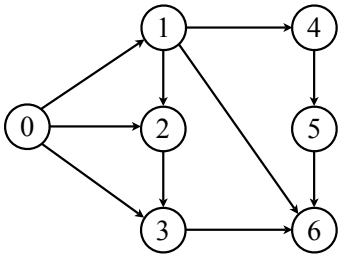
Recurso de Análise e Desenho de Algoritmos

Departamento de Informática

Universidade Nova de Lisboa

18 de Julho de 2011

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, 2 e 3 (por esta ordem). Indique:

- a ordem pela qual os vértices são tratados (passados a *TREAT* como argumento);
- o conteúdo do vector *inCounter* quando cada vértice é tratado (ou seja, sempre que o método *TREAT* está a ser executado).

2. [3 valores] Considere a função recursiva $F_X(i, j)$, onde:

- $X = (x_0, x_1, \ldots, x_n)$ é uma sequência não vazia de inteiros; e
- i e j são inteiros não negativos, inferiores ou iguais a n.

$$F_X(i,j) = \begin{cases} x_j & \text{se } i = 0 \text{ e } j \geq 0; \\ -x_i & \text{se } i \geq 1 \text{ e } j = 0; \\ \Big(F_X(i, j-1) \Big)^2 - F_X(i-1, j) & \text{se } i, j \geq 1 \text{ e } x_i \leq x_j; \\ \max\Big(F_X(i, j-1),\ F_X(i-1, j-1),\ F_X(i-1, j) \Big) & \text{se } i, j \geq 1 \text{ e } x_i > x_j. \end{cases}$$

Apresente um algoritmo, desenhado segundo a técnica da programação dinâmica, que, dada uma sequência $X = (x_0, x_1, \ldots, x_n)$ não vazia de inteiros, calcula o valor de $F_X(n, n)$. Estude (justificando) as complexidades temporal e espacial do seu algoritmo, no melhor caso e no pior caso.

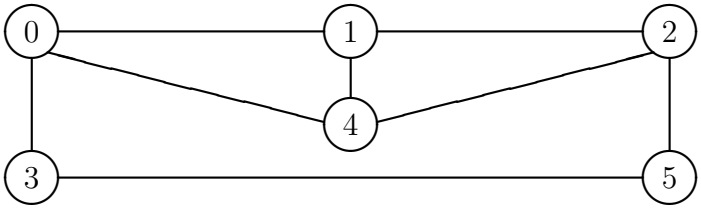
3. [4 valores] Sejam:

   - $G = (V, A)$ um grafo não orientado e
   - $C = v_1\ v_2\ \cdots\ v_m$ um caminho em G tal que $|\{v_1, v_2, \ldots, v_m\}| = m < |V|$.

   Uma *extensão Hamiltoniana de C em G* é uma permutação $w_1\ w_2\ \cdots\ w_n$ dos vértices em $V \setminus \{v_1, v_2, \ldots, v_m\}$ tal que:

   $$v_1\ v_2\ \cdots\ v_m\ w_1\ w_2\ \cdots\ w_n\ v_1 \quad \text{é um circuito de Hamilton em } G.$$

   Por exemplo, no grafo esquematizado na figura, 0 1 é uma extensão Hamiltoniana do caminho 4 2 5 3, porque 4 2 5 3 0 1 4 é um circuito de Hamilton.

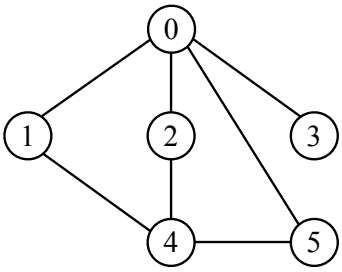


   O **Problema da Extensão Hamiltoniana de Caminho** formula-se da seguinte forma. Dados um grafo não orientado $G = (V, A)$ e um caminho $C = v_1\ v_2\ \cdots\ v_m$ em G tal que $|\{v_1, v_2, \ldots, v_m\}| = m < |V|$, existe uma extensão Hamiltoniana de C em G?

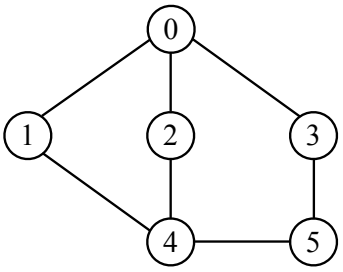
   Prove que o Problema da Extensão Hamiltoniana de Caminho é NP-completo.

4. [4 valores] Seja $G = (V, A)$ um grafo não orientado e conexo, com pelo menos dois vértices (i.e., $|V| \geq 2$). Pretende-se descobrir se G é um grafo *bipartido*. Note que o grafo é bipartido se, e só se, for possível colorir os seus vértices com duas cores, respeitando as seguintes regras:

   - cada vértice é colorido com uma (e apenas uma) das duas cores;
   - para todo o arco $(v, w) \in A$, a cor de v é diferente da cor de w.



Grafo bipartido



Grafo não bipartido

   Por exemplo, o grafo do lado esquerdo da figura é bipartido: os vértices 0 e 4 seriam coloridos com a mesma cor, e os vértices 1, 2, 3 e 5 seriam coloridos com a outra cor. No entanto, o grafo representado do lado direito não é bipartido.

   Escreva um algoritmo (em pseudo-código) que, dado um grafo não orientado, conexo e com pelo menos dois vértices, retorna *true* se o grafo é bipartido e retorna *false* se o grafo não é bipartido. Estude (justificando) a complexidade temporal do seu algoritmo, no pior caso.

5. [3 valores] Considere a classe *StackInArrayWithBackup*, das pilhas de elementos do tipo E, implementadas em vector, com *backup* automático.

```
public class StackInArrayWithBackup<E>
{
    protected E[] array;                // Memory of the stack: an array.
    protected int top;                  // Index of the element at the top of the stack.
    protected E[] arrayBackup;          // Last backup of the array.
    protected int topBackup;            // Last backup of the top.
    protected int numChanges;           // Number of changes since last backup.

    public StackInArrayWithBackup( int capacity )
    {
        array = (E[]) new Object[capacity];
        top = -1;
        arrayBackup = (E[]) new Object[capacity];
        topBackup = -1;
        numChanges = 0;
    }

    public int size( ) {  return top + 1;  }

    public void push( E element ) throws FullStackException
    {
        if ( this.size() == array.length )  throw new FullStackException();
        array[++top] = element;
        if ( ++numChanges == array.length )  this.backup();
    }

    public E pop( ) throws EmptyStackException
    {
        if ( this.size() == 0 )  throw new EmptyStackException();
        E element = array[top];
        array[top--] = null;
        if ( ++numChanges == array.length )  this.backup();
        return element;
    }

    private void backup( )
    {
        for ( int i = 0; i < array.length; i++ )
            arrayBackup[i] = array[i];
        topBackup = top;
        numChanges = 0;
    }
}
```

Considere também a função $\Phi(P)$, que atribui a cada pilha P da classe *StackInArrayWithBackup* o valor do atributo *numChanges*:

$$\Phi(P) = P.\text{numChanges}\,.$$

Prove que Φ é uma função potencial válida e calcule as complexidades amortizadas dos métodos *size*, *push* e *pop*, justificando. Nos estudos das complexidades amortizadas dos métodos *push* e *pop*, assuma que não são levantadas excepções, mas analise separadamente os casos em que a condição do segundo *if* é verdadeira e falsa.

6. [3 valores] O Nuno é um excelente dançarino e queria ganhar o concurso de dança da cidade onde mora. As regras são claras.

   - O concurso é composto por uma sequência (previamente divulgada) de provas.
   - Para cada prova, conhece-se o estilo (hip-hop, contemporânea, etc.), a música e a *pontuação máxima* que o júri pode atribuir a cada concorrente que participar na prova.
   - Há apenas cinco minutos de intervalo entre provas consecutivas, mas cada concorrente pode seleccionar as provas em que participa.
   - A pontuação de um concorrente numa prova é: ou o valor que o júri lhe atribuir, se o concorrente participar na prova; ou zero, no caso contrário.
   - A *pontuação final* de um concorrente é a soma das pontuações do concorrente em cada uma das provas.

   O Nuno sabe que ganha a pontuação máxima de qualquer prova em que participe, se tiver recuperado do cansaço que a prova anterior provocou. Por isso, calculou o *descanso necessário* de cada prova, que é o número de provas seguintes que terá de falhar se participar nessa prova.

Prova	1	2	3	4	5	6
Pontuação máxima (P)	1	2	5	4	3	1
Descanso necessário (D)	2	0	2	0	2	2

   Para exemplificar, suponha que há seis provas, cujas pontuações máximas e descansos necessários estão indicados na tabela. Se o Nuno participar na prova 3, ganha 5 pontos, mas não participará nas 2 provas seguintes (provas 4 e 5). Como o Nuno quer ter a pontuação máxima nas provas em que participa, respeitando sempre os descansos necessários, a sua maior pontuação final neste concurso seria 9 (participando nas provas 2, 4 e 5).

   Apresente **uma função recursiva** que, dadas duas sequências de inteiros,

$$P = (p_1, p_2, \ldots, p_n) \;\text{ e }\; D = (d_1, d_2, \ldots, d_n), \;\text{ para algum } n \geq 1,$$

   com as pontuações máximas (positivas) e os descansos necessários (não negativos) das provas do concurso, calcula a maior pontuação final que o Nuno pode obter. Indique claramente o que representa cada uma das variáveis que utilizar e explicite a chamada inicial.