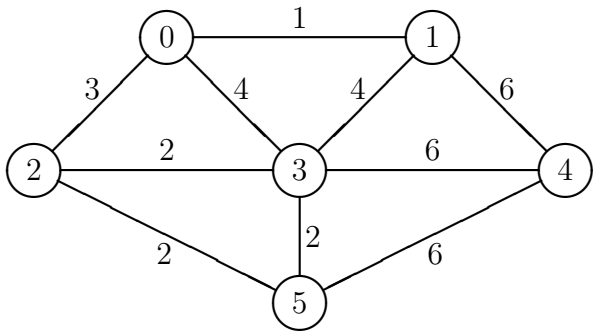
2º Teste de Análise e Desenho de Algoritmos

Departamento de Informática

Universidade Nova de Lisboa

8 de Junho de 2015

1. Suponha que se executa o algoritmo de Kruskal com o grafo esquematizado na figura.



Indique:

(a) [2 valores] uma ordem pela qual os arcos podem ser inseridos no resultado (i.e., no vetor *mst*);

(b) [1 valor] o número total de vezes que o método *removeMin* é executado; e

(c) [1 valor] o custo da árvore encontrada.

```
public class MultiStack<E> {

    private Deque<Pair<E,Integer>> stack;
    private int currentSize;

    public MultiStack( ) {
        stack = new LinkedList<Pair<E,Integer>>();
        currentSize = 0;
    }

    public int size( ) {
        return currentSize;
    }

    public void multiPush( E element, int n ) throws RuntimeException {
        if ( n <= 0 )
            throw new RuntimeException();

        boolean done = false;
        if ( !stack.isEmpty() ) {
            Pair<E,Integer> top = stack.getLast();
            if ( top.getFirst().equals(element) ) {
                top.setSecond( top.getSecond() + n );
                done = true;
            }
        }
        if ( !done )
            stack.addLast( new Pair<E,Integer>(element, n) );
        currentSize += n;
    }

    public void multiPop( int n ) throws RuntimeException {
        if ( n <= 0 )
            throw new RuntimeException();

        int toDelete = n;
        while ( !stack.isEmpty() && toDelete > 0 ) {
            Pair<E,Integer> top = stack.getLast();
            int topNumElems = top.getSecond();
            if ( toDelete >= topNumElems ) {
                stack.removeLast();
                toDelete -= topNumElems;
            }
            else {
                top.setSecond( topNumElems - toDelete );
                toDelete = 0;
            }
        }
        currentSize -= n - toDelete;
    }

}
```

2. [6 valores] A página anterior contém a classe *MultiStack*, de filas com disciplina LIFO, de elementos do tipo E. A operação *multiPush*(e, n) empilha o elemento e n vezes. A operação *multiPop*(n) desempilha: n elementos, se existirem pelo menos n elementos na pilha; todos os elementos que estiverem na pilha, no caso contrário.

   Considere a função $\Phi(M)$, que atribui a cada objeto M da classe *MultiStack* o número de pares guardados no atributo *stack*:

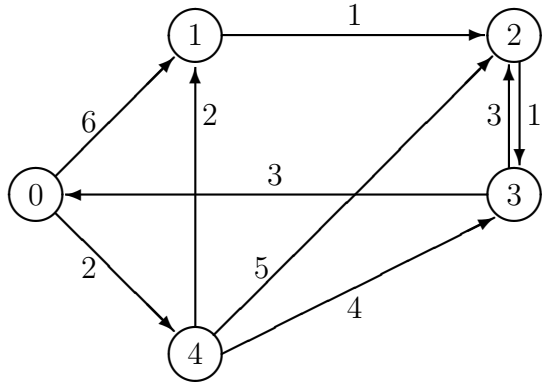
   $$\Phi(M) = M.\text{stack.size}() \,.$$

   Prove que Φ é uma função potencial válida e calcule as complexidades amortizadas dos métodos *size*, *multiPush* e *multiPop*, justificando. No estudo da complexidade amortizada do método *multiPush*, assuma que não é levantada a exceção, mas analise separadamente os casos em que a variável *done*: passa a *true*; permanece a *false*. No estudo da complexidade amortizada do método *multiPop*, assuma que não é levantada a exceção, mas analise separadamente os casos em que a condição do *if* no corpo do ciclo: é sempre verdadeira; é falsa alguma vez. Assuma que o método *equals* da classe dos elementos do tipo E e os métodos usados das classes *LinkedList* e *Pair* têm complexidade constante.

3. [6 valores] Dados:

   - um grafo G orientado e pesado, cujos arcos têm pesos inteiros positivos,
   - dois vértices, o e d, e
   - um número inteiro positivo, *maxValue*,

   pretende-se descobrir se existe algum caminho de o para d em G cujo comprimento pesado não excede *maxValue*.



Nó o	**Nó** d	*maxValue*	**Resultado**
0	2	6	*true*
1	4	5	*false*

   Para exemplificar, no grafo representado na figura, há caminho de 0 para 2 cujo comprimento pesado não excede 6 (por exemplo, 0 4 1 2, cujo comprimento pesado é 5), mas não há caminho de 1 para 4 cujo comprimento pesado seja inferior ou igual a 5. A tabela indica o resultado que se pretende obter nestes dois casos.

   Apresente uma função booleana (em pseudo-código) que recebe:

   - um grafo G orientado e pesado, cujos arcos têm pesos inteiros positivos,
   - dois vértices, o e d, e
   - um número inteiro positivo, *maxValue*,

   e retorna *true* se, e só se, existir algum caminho de o para d em G cujo comprimento pesado não excede *maxValue*. Estude (justificando) a complexidade temporal do seu algoritmo, no pior caso.

4. [4 valores] Vai realizar-se, pelo segundo ano consecutivo, um encontro de trocas de livros no Rossio. Cada participante leva um livro e tenta encontrar alguém que leve um livro que lhe interesse. O ano passado, quando ambos tinham interesse no livro que o outro tinha levado, a troca de livros foi efetuada. Ou seja, se o participante A tinha interesse no livro que o participante B tinha levado e o participante B tinha interesse no livro que o participante A tinha levado, A e B trocaram os livros entre si. Quem não encontrou um parceiro interessado na troca, saiu (bastante frustado) com o livro que tinha levado.

   Este ano, a organização vai estender as trocas a grupos de participantes. Por exemplo, se o participante A tiver interesse no livro que o participante B leva, B tiver interesse no livro que C leva, C tiver interesse no livro que D leva e D tiver interesse no livro que A leva, os quatro participantes poderão trocar os seus livros (circularmente). Os grupos poderão envolver um número arbitrário de participantes (desde que seja superior ou igual a dois). Mas, tal como no ano passado, ninguém dará o seu livro sem receber um que lhe interesse em troca. O livro dado numa troca tem de ser o que é levado para o encontro. Portanto, cada participante pode estar envolvido, no máximo, numa troca.

   Este ano, as trocas serão estipuladas pela organização. Durante a tarde, os participantes mostrarão os livros que levarem uns aos outros e a organização registará os seus interesses, na forma "o participante X tem interesse no livro que o participante Y trouxe". Cada participante poderá ter interesse em zero, um ou mais livros. Ao princípio da noite, a organização decidirá que trocas se irão efetuar (respeitando as regras descritas anteriormente).

   Quando se souber o número total de participantes e todos os seus interesses, a organização quer descobrir, o mais rapidamente possível, se há alguma forma de realizar as trocas que garanta que todos os participantes saem com um livro diferente daquele que levaram. Por exemplo, se aparecerem 5 participantes e forem registados os 7 seguintes interesses:

   $$(1,2),\ (2,3),\ (3,1),\ (4,5),\ (5,4),\ (1,3)\ \text{ e }\ (1,4),$$

   onde (X, Y) representa que o participante X tem interesse no livro que o participante Y levou, é possível efetuar trocas de forma a que todos os participantes saiam com um livro diferente daquele que levaram. Basta constituir dois grupos de trocas: um grupo com os participantes 1, 2 e 3 e o outro com os participantes 4 e 5.

   Apresente uma função booleana (em pseudo-código) que recebe os dois seguintes argumentos.

   - O número (inteiro positivo) P de participantes no encontro.
     Os participantes são identificados pelos números $1, 2, \ldots, P$.
   - A matriz I (com duas colunas) com os interesses dos participantes. Cada linha r de I regista um interesse e significa que o participante $I[r][0]$ tem interesse no livro que o participante $I[r][1]$ levou. Garante-se que $I[r][0] \neq I[r][1]$, porque ninguém pode ter interesse no livro que leva, e que todas as linhas da matriz são distintas.

   A função deve retornar *true* se, e só se, houver alguma forma de realizar as trocas que garanta que todos os participantes saem com um livro diferente daquele que levaram. **O corpo da sua função booleana deve construir um grafo e chamar um ou vários algoritmos de grafos estudados, como se eles estivessem numa biblioteca**, mesmo que esses algoritmos retornem resultados que não interessam para resolver este problema e, consequentemente, sejam menos eficientes do que poderiam ser para este caso. Em vez de programar a construção do grafo, pode indicar claramente que grafo construiria, usando o exemplo para ilustrar a sua construção, e que estruturas de dados usaria para o implementar.