Exercícios de Análise e Desenho de Algoritmos

Aulas Práticas de 23 e 24 de Maio de 2016

Departamento de Informática da FCT NOVA

Exercício 1

Qualquer jogo que pode ser jogado por uma só pessoa é um *solitaire*.

A classe *Solitaire* permite implementar uma paciência com um número ilimitado de jogos, jogados consecutivamente sem qualquer interrupção. Em cada jogo (*game*):

- cria-se uma permutação arbitrária dos números $0, 1, \ldots, n-1$, onde n é múltiplo de 4;
- o jogador tem exatamente $n/4$ jogadas (*guesses*);
- em cada jogada, o jogador tenta adivinhar o número que está numa posição da sequência, ganhando um ponto se acertar.

Considere a função $\Phi(S)$, que atribui a cada objeto S da classe *Solitaire* o quádruplo do valor do atributo *numGuesses*:

$$\Phi(S) = 4 \times S.\text{numGuesses}\,.$$

Prove que Φ é uma função potencial válida e calcule as complexidades amortizadas dos métodos *getScore* e *guess*, justificando. No estudo da complexidade amortizada do método *guess*, assuma que não é levantada a exceção, mas analise separadamente os casos em que a condição do segundo *if*: é verdadeira; é falsa. Assuma que o método *nextInt* da classe *Random* tem complexidade temporal constante.

```
import java.util.Random;

public class Solitaire {

    private Random generator;
    private int[] sequence;          // Memory of the pseudorandom sequence.
    private int guessesPerGame;      // Number of guesses in each game.
    private int numGuesses;          // Number of guesses in the current game.
    private int score;               // Number of right guesses in the current game.

    public Solitaire( int numberOfGuessesPerGame ) {
        generator = new Random();
        guessesPerGame = numberOfGuessesPerGame;
        sequence = new int[guessesPerGame * 4];
        for ( int i = 0;  i < sequence.length;  i++ )
            sequence[i] = i;
        this.permute();
        numGuesses = 0;
        score = 0;
    }
```

```
public boolean guess( int pos, int value ) throws InvalidPositionException {
    if ( pos < 0  ||  pos >= sequence.length )
        throw new InvalidPositionException();
    if ( numGuesses == guessesPerGame ) {
        this.permute();
        numGuesses = 0;
        score = 0;
    }
    numGuesses++;
    boolean goodGuess = sequence[pos] == value;
    if ( goodGuess )
        score++;
    return goodGuess;
}

public int getScore( ) {
    return score;
}

private void permute( ) {
    for ( int i = 0;   i < sequence.length − 1;   i++ ) {
        int pos = i + generator.nextInt(sequence.length − i);
        // Swap elements at positions i and pos.
        int aux = sequence[i];
        sequence[i] = sequence[pos];
        sequence[pos] = aux;
    }
}

}
```

Exercício 2

Considere 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.

```
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;
    }

}
```

Exercício 3

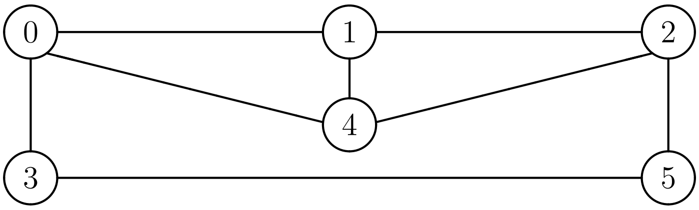
Sejam:

- $G = (V, A)$ um grafo não orientado;
- $C = v_1\ v_2\ \cdots\ v_m$ 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.

Exercício 4

Sejam:

- D um conjunto (*o domínio*);
- P um subconjunto de D (chamado uma *pré-seleção*);
- $\mathcal{C}$ uma coleção de subconjuntos de D;
- n um inteiro positivo.

Um **conjunto de representantes de $\mathcal{C}$, de cardinalidade n, que respeita a pré-seleção** P é um conjunto $R \subseteq D$ que verifica:

$$(\forall X \in \mathcal{C})\ X \cap R \neq \emptyset, \qquad |R| = n \qquad \text{e} \qquad R \subseteq P.$$

Por exemplo, se:

- o domínio for $D' = \{1, 2, 3, 4, 5, 6\}$;
- a pré-seleção for $P' = \{1, 3, 5\}$; e
- $\mathcal{C}' = \{\{1, 2, 3\},\ \{2, 4, 5\},\ \{3, 4\},\ \{2, 3, 4, 6\}\}$,

$R' = \{3, 5\}$ é um conjunto de representantes de $\mathcal{C}'$, de cardinalidade 2, que respeita P'.

O **Problema dos Representantes Pré-seleccionados** formula-se da seguinte forma. Dados um conjunto D, um subconjunto P de D, uma coleção $\mathcal{C}$ de subconjuntos de D e um inteiro positivo n, existe um conjunto de representantes de $\mathcal{C}$, de cardinalidade inferior ou igual a n, que respeita P?

Prove que o Problema dos Representantes Pré-seleccionados é NP-completo.