Appello del 5 luglio 2004
Laboratorio di Algoritmi e Strutture Dati
Anno Accademico 2003/2003

Cognome e Nome: Docente:
Numero di Matricola:

Spazio riservato alla correzione

1	2	3	4	5	6	totale
/15	/15	/15	/15	/15	/15	/90

1. Aggiungere alla classe **NodeSequence** (implementa l'interfaccia **Sequenze** usando una lista doppiamente lincata) il metodo **ObjectIterator elementsUpTo**(int **n**), che ricevuto in input un intero **n** restituisce in output un iteratore sugli elementi che hanno rango minore od uguale ad **n**. Il metodo deve lanciare l'eccezione **notEnoughElements** se non ci sono **n** elementi nella sequenza.

2. Aggiungere alla classe **LinkedTree** (implementa l'interfaccia **Tree** usando come nodo la classe **TreeNode**) il metodo **int numNodes(TreeNode a, TreeNode b)** che restituisca in output il numero di nodi nella path dal nodo **a** al nodo **b**. Il metodo deve restituire –1 se il nodo **a** non è antenato del nodo **b** e viceversa . Dire quale sia la complessità del metodo proposto nel caso peggiore, motivando la risposta. Una bozza della classe **TreeNode** è la seguente:

```
public class TreeNode implements Position {
  private Object element;  private TreeNode parent;  private NodeList children;
 …..
}
```

3. Provare il seguente teorema. Se si fa riferimento ad altri lemmi e/o corollari è necessario enunciarli.
   Teorema: Se f è un flusso in una rete di flusso G con sorgente s e destinazione t, allora le seguenti condizioni sono equivalenti:
   1. f è un massimo flusso in G
   2. G_f non contiene augmenting path
   3. |f|= c(S,T) per un taglio (S,T)

4. Visita in profondità
   a. Scrivere lo pseudocodice dell'algoritmo per la visita in profondità di un grafo (DFS). Commentare l'uso delle strutture dati utilizzate. [8 punti]

   b. Indicare, giustificando la risposta, la complessità dell'algoritmo DFS. [7 punti]

5. Aggiungere alla classe **UnsortedSequencePriorityQueue** (implementa l'interfaccia **PriorityQueue** usando un'istanza della classe **NodeSequence**) il metodo
   ObjectIterator getInterval(Object k_1**, Object** k_2**)**
   due chiavi k_1 e k_2, con k_1 minore di k_2, restituisca un iteratore sugli elementi di Q che hanno chiave k tale che $k_1 \leq k \leq k_2$.
   a. L'elemento di Q a priorità minima è quello che ha come chiave la stringa più corta
   b. Si supponga che **Comp** sia il comparatore utilizzato per confrontare le chiavi nella coda a priorità

6. Scrivere una funzione Java che ricevuta in input una stringa restituisce true se la stringa contiene un'espressione parentesizzata in maniera corretta (tutte le parentesi sono bilanciate). Ad esempio le parentesi nelle seguenti espressioni sono bilanciate
    a. Rd(3e)d(d(i)d)
    b. C(f(f(f(g(t))d)iu)d)dfe

   Ma le parentesi nelle seguenti espressioni non sono bilanciate
    c. Rwe(fd)(fd(ffs)c
    d. Fdr(((d)(d)d(f(id(i))f)(d))

   Suggerimento, utilizzare uno Stack S inserendo in S solo le parentesi aperte.