Prima prova *in itinere* di
Laboratorio di Algoritmi e Strutture Dati – Prof. Carlo Blundo
Anno Accademico 2005/2006 – Matricole Dispari-Pari

Cognome e Nome: Numero di Matricola:

Spazio riservato alla correzione

1	2	3	4	5	6	7	totale
/15	/15	/12	/8	/8	/12	/20	/90

Non usare altri fogli, usare solo lo spazio sottostante. Fogli differenti da questo non saranno presi in considerazione per la correzione.

1. Sia G=(V,E) un grafo non orientato e sia $s \in V$ il nodo sorgente su cui è invocata la procedura BFS(G,s). Si indichi con $\delta(s,v)$ il minimo numero di archi in un cammino da s a v, se c'è (∞ se non ci sono cammini). Provare che al termine di BFS, $d[v] \geq \delta(s,v)\ \forall\ v \in V$

2. Si supponga che un grafo G abbia la seguente lista delle adiacenze
   1 – (2,3,4)
   2 – (1,3.4)
   3 – (1,2,4)
   4 – (1,2,3,6)
   5 – (6,7,8)
   6 – (4,5,7)
   7 – (5,6,8)
   8 – (5,7)

   a. Disegnare il grafo
   b. Fornire l'albero BFS ottenuto applicando la visita BSF a partire dal vertice 1
   c. Fornire la foresta ottenuta applicando la visita DFS a partire dal vertice 1 (per ogni vertice indicare il valore del campo d ed il campo f)

3. Implementare il TDA **Queue** usando come variabile d'istanza della classe una coda a doppio ingresso (**DlinkedDeque**). Cambia la complessità di tempo dei vari metodi implementati rispetto all'implementazione **ArrayQueue**? Giustificare la risposta.

4 Aggiungere alla classe **LinkedList** (implementa l'interfaccia **List**) il metodo **void reset()** che cancella la lista su cui è invocata. Non è sufficiente settare header e trailer a null, ma tutti i nodi della lista devono essere cancellati.

5. Aggiungere alla classe **ArrayVector** (implementa l'interfaccia **Vector**) il metodo
   boolean belongsTo(Object obj, rank r1, rank r2)
   che restituisce **true** se l'oggetto **obj** è un elemento del **Vector** su cui è invocato avente un rango compreso tra **r1** ed **r2** inclusi. Quale è la complessità di tempo del metodo proposto? Giustificare la risposta.

6. Aggiungere alla classe **NodeSequence** (implementa l'interfaccia **Sequence**) il metodo
   void removeDuplicates()

che elimina tutti gli elementi duplicati presenti nella **Sequence** su chi è invocato. Quale è la complessità di tempo del metodo proposto? Giustificare la risposta.

7. Uno **SmartStack** è un contenitore che memorizza oggetti di tipo **Item** definiti come segue:

```
public class Item {
        private int numItem;
        private String itemName;

        public Item(String str)  {itemName=str; numItem=1;}
        public upNumber()      { numItem++; }
        public downNumber()  { numItem--; }
        public getNumber()     { return numItem; }
        public getItem()         { return itemName; }
}
```

Implementare l'interfaccia

```
public interface SmartStack {
        public int size();
        public void smartPush (String element);
        public String smartPop() throws SmartStackEmptyException;
}
```

tenendo presente il seguente funzionamento dei metodi

smartPush (String element) – se lo **SmartStack** contiene un oggetto **Item** il cui **itemName** è uguale alla stringa **element**, allora per quello oggetto viene incrementato di uno il valore della variabile **numItem**. Altrimenti, viene inserito al top dello **SmartStack** un nuovo oggetto **Item** contenente la stringa **element** con **numItem** settato ad 1.

String smartPop() – restituisce la stringa **itemName** contenuta nell'oggetto **Item** al top dello **SmartStack**. Il valore **numItem** corrispondente viene decrementato di uno. Se tale valore diventa nullo, allora rimuove l'oggetto **Item** dal top dello **SmartStack**.

size() – restituisce il numero degli elementi presenti nello **SmartStack** contando ogni elemento **numItem** volte.