Seconda prova *in itinere* di
Laboratorio di Algoritmi e Strutture Dati – Prof. Carlo Blundo
Anno Accademico 2004/2005– Matricole Dispari-Pari

Cognome e Nome: Numero di Matricola:

Spazio riservato alla correzione

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

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

1. Si scriva la funzione **List estrai(List L, Object o1, Object o2, Comparator C)** che restituisce la lista di tutti gli oggetti presenti in **L** compresi tra **o1** ed **o2** inclusi (per confrontare gli oggetti si deve usare il comparatore **C**). Usare solo i metodi dell'interfaccia **List** (però non è possibile utilizzare i metodi **elements()** e **positions()**). Si osservi che la funzione deve lasciare inalterato il contenuto di **L** alla fine dell'esecuzione (lo può modificare durante). Quale è la complessità del metodo proposto (giustificare la risposta).

2. Aggiungere alla classe **LinkedTree** il metodo **Iterator preorderVisit(**) che restituisce un iteratore sugli elementi dell'albero elencati secondo la visita preorder dell'albero. Quale è la complessità del metodo proposto (giustificare la risposta).

3. Implementare la seguente interfaccia

```
interface Pull {
public void add(Objec e);      //aggiunge l'elemento e al contenitore
public Object remove();        //cancella un elemento arbitrario dal contenitore
public boolean isEmpty();      //restituisce true se il contenitore è vuoto
public int size();             //restituisce il numero di elementi nel contenitore
}
```

Tutti i metodi devono avere una complessità pari a $O(1)$, giustificare la risposta.

4. Aggiungere alla classe **D** che implementa l'interfaccia **Dictionary** il metodo **void deleteAllElements(Object key)** che cancella dal dizionario tutti gli elementi che hanno chiave uguale a **key**. Se non ci sono elementi con chiave uguale a **key**, allora viene lanciata l'eccezione **NoSuchKey**. (Il problema può essere risolto anche senza sapere come è implementata **D**).

5 Descrivere le quattro versioni del problema dei cammini minimi in un grafo orientato (SSSP, SDSP, SPSP, APSP).

6. Mostrare, come un algoritmo utilizzato per risolvere il problema SSSP possa essere utilizzato per risolvere gli altri

7. Scrivere lo pseudo-codice dell'algoritmo di Dijkstra per risolvere il problema dei cammini minimi. Discutere la complessità dell'algoritmo proposto.