008AA – ALGORITMICA E LABORATORIO / A0006 – ALGORITMICA
Appello dell'11 settembre 2009

Cognome Nome: **N. Matricola:** **Corso: A B**

Esercizio 1. *(6+6 punti)* Un intervallo $[a, b]$ di interi rappresenta l'insieme $\{a, a+1, a+2, \ldots, b\}$, dove $a \leq b$. Per esempio, $[15, 17]$ rappresenta gli interi in $\{15, 16, 17\}$. L'unione tra intervalli può essere vista come l'unione tra i corrispondenti insiemi di interi. Per esempio, l'unione di $[8, 9], [10, 10]$, $[15, 17]$, $[16, 19]$ e $[18, 18]$ rappresenta l'insieme $\{8, 9, 10, 15, 16, 17, 18, 19\}$. Vale la proprietà che l'unione di intervalli può essere sempre rappresentata in modo univoco come una lista ordinata L di intervalli *disgiunti e non adiacenti*, in quanto gli intervalli adiacenti del tipo $[a, b]$ e $[b+1, c]$ possono essere sempre visti come un unico intervallo $[a, c]$. Per esempio, l'unione degli intervalli $[8, 9], [10, 10]$, $[15, 17]$, $[16, 19]$ e $[18, 18]$ è rappresentato da $L = [8, 10], [15, 19]$.

Dati in ingresso $2n$ interi $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$, interpretati come n intervalli $[a_0, b_0]$, $[a_1, b_1]$, $\ldots$, $[a_{n-1}, b_{n-1}]$, si vuole stampare la lista L che rappresenta la loro unione.

(a) Progettare un algoritmo che risolve il problema in tempo $O(n^2)$.

(b) Progettare un algoritmo che risolve il problema in tempo $O(n \log n)$ secondo il seguente schema:

(b.1) ordina gli intervalli in base al valore dei loro estremi sinistri a_i (ipotizzando che siano tutti distinti);

(b.2) scandisce gli intervalli secondo l'ordine sopra, e li fonde utilizzando una struttura di dati "pila".

Soluzione (a): Passo 0: $L = [a_0, b_0]$.

Passo $i > 0$ per elaborare $[a_i, b_i]$ in $O(n)$ tempo. Sia $L = [c_0, d_0], \ldots, [c_{k-1}, d_{k-1}]$ la lista attualmente in uso:

- Se $[a_i, b_i]$ è contenuto in un intervallo di L, salta al passo $i + 1$.
- Altrimenti:
  - Elimina da L tutti gli intervalli $[c_j, d_j]$ tali che $[c_j, d_j] \subseteq [a_i, b_i]$ ottenendo la lista L'.
  - Trova, se esiste, l'unico intervallo $[c', d']$ in L' che interseca $[a_i, b_i]$ oppure è adiacente ad esso, dove $c' < a_i$; altrimenti, poni $c' = a_i$.
  - Trova, se esiste, l'unico intervallo $[c'', d'']$ in L' che interseca $[a_i, b_i]$ oppure è adiacente ad esso, dove $b_i < d''$; altrimenti, poni $d'' = b_i$.
  - Aggiorna $L = L' \cup [c', d'']$.

Passo finale. Restituisci $L = [c_0, d_0], \ldots, [c_{k-1}, d_{k-1}]$, la lista ottenuta dopo il passo $i = n - 1$.

Soluzione (b): Sia $[a_0, b_0]$, $[a_1, b_1]$, $\ldots$, $[a_{n-1}, b_{n-1}]$ la sequenza di intervalli ordinati risultanti dal passo (b.1). Inizialmente, metti $[a_0, b_0]$ nella pila.

Passo $i > 0$. Sia $[a, b]$ l'intervallo in cima alla pila e $[a_i, b_i]$ l'intervallo corrente:

- Se $a_i \leq b+1$, fai pop di $[a, b]$ e push di $[a, \max\{b, b_i\}]$. (Notare che deve valere $a < a_i$ per il punto (b.1).)
- Se $a_i > b + 1$, fai push di $[a_i, b_i]$.

Stampa il contenuto della pila alla fine dell'ultimo passo.

Cognome Nome: **N.Matr:**

Esercizio 2. *(6+6 punti)* Si consideri un albero binario di ricerca, di altezza h, che memorizza un insieme di chiavi (una chiave per nodo), alcune delle quali possono essere uguali.

(a) Scrivere il codice per l'operazione di inserimento di una chiave k. Nel caso trovi nel nodo corrente una chiave che è uguale a k, l'operazione prosegue nel figlio sinistro del nodo. Il tempo di esecuzione deve essere $O(h)$.

(b) Scrivere il codice per l'operazione $conta(k)$, che restituisce il numero di occorrenze della chiave k nell'albero costruito mediante l'inserimento delle chiavi, come descritto nel punto (a). Il tempo di esecuzione deve essere $O(h)$.

Soluzione (a): Assumiamo che il campo chiave di un nodo u sia u.`chiave`.

```
inserisci( u, k )
   if ( u == NULL ) {
      u = NuovoNodo();
      u.chiave = k;
      u.sx = u.dx = NULL;
   } else if ( k <= u.chiave ) {
      u.sx = Inserisci( u.sx, k );
   } else {
      u.dx = Inserisci( u.dx, k );
   }
   return u;
}
```

Soluzione (b): Adottiamo una variante dello schema di ricerca in un albero binario di ricerca con radice r e sfruttiamo la proprietà dell'operazione di inserimento (a) che le chiavi uguali devo giacere tutte lungo alcuni nodi di uno stesso cammino radice-foglia. Abbiamo che $conta(k)$ restituisce il risultato della chiamata a `count(r,k)`.

```
count( u, k )
   if ( u == NULL ) return 0;
   if ( k < u.chiave ) return count( u.sx, k );
   if ( k > u.chiave ) return count( u.dx, k );
   return 1 + count( u.sx, k );  // k non puo' essere a destra
```

Cognome Nome: **N.Matr:**

Esercizio 3. *(1+2+1+2 punti)* Si consideri un grafo diretto G, rappresentato mediante le seguenti liste di adiacenza:

```
 1: 2, 9, 11
 2: 3, 6, 10
 3: 5
 4: 1, 3, 5
 5: 8, 12
 6: 2, 3, 12
 7: 1, 4, 12
 8: (vuota)
 9: 7, 10
10: 2, 8, 11
11: (vuota)
12: (vuota)
```

(a) Disegnare G sul foglio.

(b) Indicare l'ordine con cui i vertici di G sono scoperti dalle visite BFS e DFS, partendo dal vertice 1.

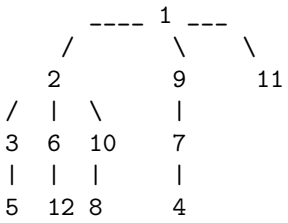
(c) Disegnare gli alberi BFS e DFS sul foglio.

(d) Per ogni arco di G, indicare la classificazione indotta dalla DFS (in avanti, all'indietro, trasversale, ...).

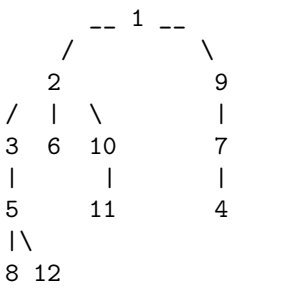
Soluzione (a): ovvia, serve per i punti successivi

Soluzione (b):
BFS: 1, 2, 9, 11, 3, 6,10, 7, 5, 12, 8, 4.
DFS: 1, 2, 3, 5, 8, 12, 6, 10, 11, 9, 7, 4.

Soluzione (c):
Albero BFS



Albero DFS



Soluzione (d):
Archi in avanti: (1,11)
Archi all'indietro: (4,1), (6,2), (7,1), (10,2)
Archi trasversali: (4,3), (4,5), (6,3), (6,12), (7,12), (9,10), (10,8)
I rimanenti archi appartengono all'albero DFS (vedi sopra).