Laboratorio di Algorimi e Strutture Dati - prova di laboratorio
Tema d'esame del 21 gennaio 2021
Docente: Violetta Lonati

1.

Considerate la seguente funzione, che deve restituire il prodotto dei suoi due argomenti

```
int multiply(int x, int y) {
  if ( AAAAA ) {
    BBBBB;
  }
  else {
    return multiply(x - 1, y) + y;
  }
}
```

Cosa si deve scrivere al posto di AAAAA e di BBBBB?

- ✓ **AAAAA x == 0**
  BBBBB return 0
- **AAAAA x == 0**
  BBBBB return 1
- **AAAAA x == 1**
  BBBBB return 1
- **AAAAA x ==1**
  BBBBB return 0
- **AAAAA x == y**
  BBBBB return x*y
- **AAAAA x ==1**
  BBBBB return y

2.

Considerate il seguente programma:

```c
#include <stdio.h>
#define print(x) printf("%d ", x)
int x;
void Q(int z) {
    z += x;
    print(z);
}

void P(int *y) {
    int x = *y + 2;
    Q(x);
    *y = x - 1;
    print(x);
}

int main(void) {
    x = 5;
    P(&x);
    print(x);
    return 0;
}
```

Quale è l'output del programma? **Risposta:** 12 7 6

3.

Abbiamo i puntatori al primo e all'ultimo elemento di una lista concatenata semplice (con solo puntatore a next). Quali delle seguenti operazioni hanno tempo di esecuzione che dipende dalla lunghezza della lista?

- [ ] **Cancellazione del primo elemento**
- [ ] ✓ **Cancellazione dell'ultimo elemento**
- [ ] **Inserimento all'inizio della lista**
- [ ] **Inserimento alla fine della lista**
- [ ] ✓ **Ricerca di un elemento**

4.

a) In una lista doppiamente concatenata, il numero di puntatori sui quali si interviene per un'operazione di inserimento è:

- **4**
- **0**
- **2**
- **1**
- ✓ **nessuna delle altre risposte**

b) Giustificate brevemente la risposta.

Risposta:

Vedere risposte lunghe in fondo al documento

5.

Considerate queste porzione di codice.

```
struct item {
   int data;
   struct item *next;
};

int f(struct item *p) {
   return (
          (p == NULL) ||
          (p->next == NULL) ||
          (( p->data <= p->next->data) && f(p->next))
   );
}
```

Sia p l'indirizzo del primo elemento di una lista, la funzione f(p) restituisce 1 se e solo se

- **La lista è ordinata in maniera decrescente**
- **La lista è ordinata in maniera crescente**
- **La lista, a partire dal secondo elemento, è ordinata in maniera non decrescente**

- La lista è ordinata in maniera non crescente
- ✓ **La lista è ordinata in maniera non decrescente**
- **La lista, a partire dal secondo elemento, è ordinata in maniera crescente**
- **Nessuna delle altre risposte**

6.

a) Questa funzione dovrebbe incrementare di 1 il valore di ogni nodo di un albero binario, ma non è corretta.
BitNode è un tipo che rappresenta un nodo di albero binario: si tratta di un puntatore a una struttura con 2 membri left e right che puntano rispettivamente ai figli sinistro e destro, e un membro val di tipo intero.

```
void f(BitNode root) {
   if (root != NULL) {
       root -> val++;
       if (root -> left != NULL) {
          root -> left -> val++;
          f(root -> left -> left);
       }
       if (root -> right != NULL) {
          root-> right-> val++;
          f(root -> right -> right);
       }
   }
}
```

Spiegate cosa fa invece la funzione, poi individuate la causa di questo errore e descrivetela.

Risposta:

Vedere risposte lunghe in fondo al documento

b) Correggete la funzione.

Risposta:

Vedere risposte lunghe in fondo al documento

7.

Considerate la seguente funzione, che riceve un vettore A di n interi e un vettore B di m interi.

```
int f (int A[], int B[], int n, int m) {
   for ( int i= 0; i < n; i++) {
      int found = 0;
      for (int j = 0; j < m; j++)
         if (A[i] == B[j])
            found = 1;
      if (!found) return 0;
   }
   return 1;
}
```

a) Sia A = {2,4,5}. Selezionate tutti e soli i B per cui la funzione restituisce 1.

- ☐ ✔ **B = {5,4,2}**
- ☐ **B = {4,2}**
- ☐ ✔ **B = {2,4,5,6}**
- ☐ **B = {}**
- ☐ ✔ **B = {5,2,4,2}**

b) Cosa fa la funzione f?

- ✓ **Decide se ogni valore di A si trova anche in B**
- **Decide se ogni valore di B si trova anche in A**
- **Decide se A e B sono uguali**
- **Decide se A e B hanno almeno un elemento in comune**
- **Decide se A e B hanno lo stesso numero di elementi**
- **Decide se A e B hanno i primi due elementi in comune**
- **Decide se A e B contengono gli stessi valori**

c) Sia N la lunghezza di A e M la lunghezza di B. Quale è il tempo di esecuzione nel caso pessimo, in funzione di N e M?

- ✓ **O(NM)**
- **O(N+M)**
- **O(N^2)**
- **O(M^2)**
- **O(N log N)**

d) Riprogettate la funzione f in modo che sia asintoticamente più veloce. Potete usare strutture dati di supporto.

Potete spiegare la vostra soluzione a parole (siate brevi), con pseudocodice, con un disegno, oppure potete scriverla in C. Indicate il tempo di esecuzione; più la funzione è veloce, meglio è.

Risposta:

Vedere risposte lunghe in fondo al documento

Risposte alle domande aperte

4b) Dipende da dove si inserisce l'elemento.

Assumiamo di avere già creato un nuovo nodo col valore da inserire nella lista, e con i due puntatori prev e next a NULL.

- Se inseriamo il nodo in testa dovremo modificare 3 puntatori: la testa della lista (che deve puntare al nuovo nodo), il next del nuovo nodo (che deve puntare a quello che era il nodo in testa alla lista), il prev di quello che era il primo nodo e diventerà il secondo. NB: non c'è bisogno di modificare il prev del nodo nuovo, che deve rimanere a NULL

- Simmetricamente, se inseriamo il nodo in coda dovremo modificare 3 puntatori: il puntatore alla fine della lista (che deve puntare al nuovo nodo), il prev del nuovo nodo (che deve puntare a quello che era l'ultimo nodo della lista), il next nel nodo che era l'ultimo nodo e diventerà il penultimo. NB: non c'è bisogno di modificare il next del nodo nuovo, che deve rimanere NULL

- Se inseriremo il nodo in una posizione intermedia (caso che potrebbe aversi in caso di lista ordinata) dovremo cambiare 4 puntatori: i due puntatori next e prev del nuovo nodo (in modo che puntino rispettivamente ai due nodi subito dopo e subito prima al punto di inserimento), il puntatore next del nodo subito prima del punto di inserimento, il puntatore prev del nodo subito prima del punto di inserimento.

NOTA 1: era necessario considerare i vari casi possibili, o specificare che si faceva riferimento a una precisata variante delle liste o dell'operazione di inserimento. Chi ha considerato solo uno dei casi possibili senza discutere questa scelta ha ricevuto un punteggio parziale.

NOTA 2: è stata considerata corretta anche la risposta di chi ha scelto 4 e ha specificato nella risposta di aver contato un quarto puntatore NULL anche nel caso dell'inserimento in testa (prev del nodo nuovo) o di inserimento in coda (next del nuovo nodo).

6a) La funzione incrementa il valore del nodo puntato da root, poi verifica per ciascun figlio di root se si tratta del nodo nullo; in caso contrario ne incrementa il valore poi richiama ricorsivamente se stessa su un solo figlio. In questo modo dunque non si visitano tutti i nodi; ad esempio non si visita il sottoalbero che ha per radice il figlio destro del figlio sinistro di root.

Inoltre, non c'è nessun motivo per "scendere" al livello dei "nipoti" di root; basterebbe invocare la funzione sui due figli. In questo caso non ci sarebbe nemmeno bisogno di verificare che tali nodi siano non nulli, poiché questo controllo viene gestito all'inizio della chiamata ricorsiva stessa.

6b)

```
void f( BitNode root ) {
        if ( root != NULL ){
                root -> val++;
                f(root -> left);
                f(root -> right);
        }
}
```

7d) Si potrebbe costruire una hashtable contenente gli elementi di B e, per ogni elemento di A, verificare la presenza di tale elemento in B tramite la hastable. La costruzione ha tempo O(M), la verifica della presenza di un elemento ha tempo O(1) e va ripetuta tante volte quanti sono gli elementi di A, per un tempo complessivo O(N). Quindi complessivamente la soluzione ha costo O(N+M).

NOTA: una variante meno efficiente della precedente, ma comunque considerata in maniera positiva (ma non a punteggio pieno) è quella di costruire un albero binario di ricerca invece della tabella di hash, con un costo di O(MlogM) per la costruzione dell'albero e un costo di O(logM) per la ricerca, da ripetere N volte, ottenendo dunque un costo complessivo di O((M+N) logM).

Simile valutazione è stata data alle soluzioni che prevedevano l'ordinamento del vettore di B e la ricerca dicotomica di ogni elemento di A, per un costo di O(M logM) per l'ordinamento e un costo di O(logM) per la ricerca, da ripetere N volte, ottendendo quindi un costo complessivo di O((M+N) logM).

NB: Entrambe queste soluzioni sfruttano il fatto che gli elementi di B costituiscono un insieme ordinato e si basano sulla scelta di una struttura dati (albero binario di ricerca o array ordinato che sia) che rappresenta tale insieme ordinato. In particolare con queste strutture si possono implementare efficientemente, oltre che la ricerca, anche altre funzioni come: ricerca del minimo, del massimo, del successore; stampa in ordine; ecc. Le tabelle di hash invece sono strutture adatte a svolgere efficientemente la ricerca di elementi (più efficientemente che non con alberi binari di ricerca o vettori ordinati), ma non a rappresentare relazioni d'ordine.

Nel caso di questo esercizio, anche se gli elementi di A e B sono ordinabili, si deve affrontare semplicemente un problema di ricerca (la funzione nel testo avrebbe senso anche nel caso di elementi diversi da int, non da un insieme ordinato) e non è necessario rappresentare la relazione d'ordine. Per questo una soluzione che usa un tabella di hash è preferibile.