

-- -- -- 22 Dicembre 2009 -- -- --

Nome e Cognome

Numero di Matricola:

Spazio riservato alla correzione

1	2	3	4	Totale
/6	/8	/6	/14	/34

Non utilizzate altri fogli. Utilizzate soltanto lo spazio sottostante. Fogli differenti non saranno presi in considerazione per la correzione. Non scrivere a matita. Per tutti gli esercizi, descrivere la complessità asintotica delle funzioni implementate

1. Si considerino due stack **S1** e **S2**, implementati con array **S1[MAX+1]** e **S2[MAX+1]**, **entrambi riempiti con interi**. Si implementi la funzione ricorsiva **void gioca (int S1[], int S2[])** che, utilizzando una libreria di funzioni di accesso agli stack (**da implementare**) prendendo in input i due stack, simuli il seguente gioco.
 - a. Se i due stack non hanno lo stesso numero di elementi, vince lo stack con più elementi.
 - b. Se invece i due stack hanno uguale numero di elementi, partendo dalla base, si confrontano ad ogni turno i numeri allo stesso livello. Se la loro somma è dispari, si elimina l'elemento di S1, se, invece è pari si elimina l'elemento di S2.
Esempio: S1 iniziale |3|2|4|10| (3 bottom dello stack) – S2 iniziale |6|2|18| (6 bottom dello stack). Vince S1. Se invece S2 è |6|2|18|4|, allora S1 finale |2|4|10|– S2 finale |6|

2. Si considerino due liste di numeri interi **Lista1** e **Lista2** implementate come liste doppiamente puntate e non circolari, utilizzando la seguente struttura

```
struct elemento {
    struct elemento *prev;
    int inf;
    struct elemento *next;}

struct elemento *Lista1,*Lista2;
```

- a. Si implementi una sola funzione **ricorsiva** (che eventualmente può richiamare sottofunzioni) che rimuova sia da Lista1 che da Lista2 tutte le occorrenze di numeri negativi.
- b. Dato un valore x, scrivere una funzione ricorsiva che, verificata l'esistenza di x in Lista1, inserisca Lista2 prima di x (in Lista1) e restituisca Lista1 così modificata. Se x non esiste in Lista1, accodare Lista2 a Lista1.

3. Sia T un albero binario, implementato con la seguente struttura a puntatori:

```
struct nodo {  
    int info1;  
    int info2;  
    struct nodo *left;  
    struct nodo *right;}
```

```
struct nodo *T;
```

- a. scrivere una funzione che verifichi che T è un albero binario di ricerca secondo la chiave (info1+ info2) per ogni nodo.
- b. Assumendo che T sia un ABR secondo la proprietà a), data una coppia di elementi (x1,x2) verificare se esiste un elemento in T che ha tale coppia usando una ricerca binaria.

4. Siano **G** e **H** due grafi orientati pesati entrambi con pesi positivi, di **n** vertici $0, 1, \dots, n-1$ e rappresentati con liste di adiacenza utilizzando la seguente struttura:

```
typedef struct graph {
    int nv;
    edge **adj; } graph;

graph *G, *H;

typedef struct edge {
    int key;
    int peso;
    struct edge *next; } edge;
```

- a. Scrivere in linguaggio C una funzione che, presi in input i due grafi **G** e **H** costruisca un terzo grafo **T** (e lo restituisca) in cui l'arco (a,b) è presente se è presente sia in **G** che in **H**, oppure in nessuno dei due. Nel primo caso prenderà come peso la somma dei pesi dei relativi archi in **G** e **H**. Nel secondo caso, prenderà peso -1
- b. Scrivere una funzione in C che verifichi che per ogni nodo di **G** e **H**, il grado incidente in **G** è uguale al grado adiacente in **H**.