Laboratorio di Algoritmi e Strutture Dati I Docenti: Murano / Cutugno

-- -- -- 11 SETTEMBRE 2006 -- -- --

Laurea in Informatica Università degli Studi di Napoli "Federico II"

Nome e Cognome Numero di Matricola:

Spazio riservato alla correzione

1	2	3	Totale
/12	/4	/14	/30

Non utilizzate altri fogli. Utilizzate soltanto lo spazio sottostante. Fogli differenti non saranno presi in considerazione per la correzione. Non scrivere a matita

Gli studenti che hanno frequentato e consegnato i progetti devono svolgere solamente gli esercizi 1, 2 e 3. Tutti gli altri devono anche svolgere anche l'esercizio 4.

1. Si considerino due Stack **S1** e **S2**, implementati con array **S1[MAX] e S2[MAX],** e riempiti con interi **da 1 a 9**. Si implementi la funzione ricorsiva **void gioco, che indipendentemente dalla implementazione della struttura dati Stack,** prendendo in input i due Stack, effettui un gioco nel seguente modo:
   Ad ogni turno del gioco si considera **la somma modulo 10** dei valori al top dei due Stack. Se tale somma è minore di 5, vince il primo Stack, altrimenti vince il secondo. Ad ogni iterazione, si rimuove il top dallo Stack perdente. Perde lo Stack che finisce per primo i suoi valori. La funzione termina indicando lo Stack vincente, che ritorna come all'inizio del gioco, mentre quello perdente risulta vuoto. Si ricordi che lo Stack è una struttura dati che permette l'accesso solo al top. Scrivere tutte le funzioni utilizzate. Non utilizzare altre strutture dati.
   Esempio: Sia S1=|4|7|9| e S2 = |2|9|. Risposta: S2 vince.

2. Si considerino due liste di numeri interi **Lista1 e Lista2 "ordinate in modo crescente"** implementate come liste singolarmente puntate e non circolari, utilizzando la seguente struttura

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

struct elemento *Lista1,*Lista;
```

Si implementi la funzione ricorsiva **merge** che ricorsivamente prendendo in input le due liste, inserisce nella prima lista, mantenendo l'ordine crescente della lista, i valori pari della seconda, rimuovendoli.

Esempio, sia **Lista1** uguale a 1→10 e **Lista2** uguale a 2→4→7. Dopo l'esecuzione di merge, **Lista1** sarà uguale a **1→2→4→10,** e **Lista2** sarà uguale a 7.

3. Siano **G** e **H** due grafi non orientati pesati entrambi con pesi positivi, di **n** vertici 0, 1,…, 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;
```

scrivere in linguaggio C tre funzioni che in successione, presi in input i due grafi **G e H**,

a. Sostituiscano tutti gli archi in G e H in modo che ogni arco (a, b) in G abbia peso a+b in G, e a*b in H
b. Per ogni arco (a, b) in G e H, rimuovano l'arco in G se la somma dei due pesi è dispari e da H in caso contrario.
c. Preso **m** come input, controllino se esiste un sottografo di G, di m nodi, fortemente connesso
d. Studiare la complessità delle funzioni implementate.

Gli studenti che non hanno consegnato il progetto devono risolvere il seguente esercizio aggiuntivo:

Spazio riservato alla correzione

1	2	3	4	Totale
/10	/4	/10	/6	/30

4. Dato un albero binario di ricerca T implementato con la seguente struttura a puntatori:

```
struct nodo {
        int inforadice;
        struct nodo *left;
        struct nodo *right;}

struct nodo *T;
```

implementare una funzione in linguaggio C ricorsiva che rimuova tutti ogni nodo con valore negativo dall'albero T e il suo sottoalbero destro. Dunque, per ogni nodo rimosso, il suo sottoalbero sinistro diventa sottoalbero sinistro di suo padre.
Descrivere la complessità della funzione implementata.

5. Scrivere in linguaggio C la funzione **reverse** che preso in input un grafo orientato pesato cambi il verso di tutti i suoi archi e se ne discuta la complessità. Si utilizzi come struttura dati quella dell'esercizio precedente, opportunamente adattata.