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

– – – 30 GENNAIO 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
/10	/12	/12	/34

Non utilizzate altri fogli. Utilizzate soltanto lo spazio sottostante e il retro dei fogli. 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. Gli altri devono svolgere anche gli esercizi 4 e 5.

1. Si consideri uno Stack **S**, implementato con array **S[MAX]**. Si implementi la funzione ricorsiva **void raddoppia Stack(int S[MAX])** che raddoppia le occorrenze di ogni elemento nello stack lasciando invariato l'ordine degli elementi. Si ricorda che lo stack è una struttura dati che permette l'accesso ai suoi dati solo dal top.
   Esempio: stack iniziale 1|3|3|6|2 -- stack finale 1|1|3|3|3|3|6|6|2|2

2. Si considerino due liste di numeri interi **Lista1 e Lista2** implementate entrambe come lista doppiamente puntata non circolare implementata utilizzando la seguente struttura

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

struct elemento *Lista_1,*Lista_2;
```

   a. Si implementi la funzione **void togli_pari_dispari(struct elemento *L, int a)** che elimina dalla lista **Lista1** tutte i numeri dispari (passando 1 al posto di a) e tutti i numeri pari dalla lista **Lista2 (**passando 0 al posto di a**)** senza usare strutture dati aggiuntive e senza cambiare l'ordine degli elementi.
   b. si implementi la funzione ricorsiva **interleaving** che prende in input ***Lista1** e ***Lista2** modificate e restituisce l'interleaving della 2 liste.
   Esempio, sia **Lista1** uguale a 1→2→3→2, **Lista2** uguale a 1→2→1→2→4→2→5, dopo **togli_pari_dispari, Lista1** è uguale a 2→2, **Lista2** è uguale a 1→1→5, interleaving restituisce 2→1→2→1→5

3. Siano **G** e **H** due grafi 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 una funzione che preso in input i due grafi **G e H**, restituisce **G** con i pesi dei suoi archi diminuiti dei pesi degli archi corrispondenti in H. Se un arco ottiene peso negativo, l'arco deve essere rimosso. Descrivere la complessità della funzione implementata.

Gli studenti che non hanno frequentato il corso e non hanno consegnato i progetti devono risolvere anche i seguenti esercizi aggiuntivi:

Spazio riservato alla correzione

1	2	3	4	5	Totale
/6	/8	/8	/8	/-5	/30

4. Dati due alberi binari di ricerca T1 e T2 implementati con la seguente struttura a puntatori:

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

struct nodo *T1,*T2;
```

   implementare una funzione in linguaggio C che rimuova dall'albero tutti i nodi con valore **inforadice** dispari. Poi si costruisca un nuovo albero **T** in cui **inforadice** è data dalla somma dei rispettivi valori **inforadice** nei nodi corrispondenti.
   Rispondere inoltre alla seguente domanda motivando la risposta: L'albero **T** ottenuto è ancora un albero binario di ricerca?

5. Dopo una breve descrizione sulla rappresentazione di un grafo tramite matrice di adiacenza e liste di adiacenza, descrivere due scenari in cui si evince il vantaggio di usare l'una o l'altra rappresentazione, in termini di complessità asintotica delle operazioni.