FONDAMENTI DI INFORMATICA II (2h) – 15 Febbr. 2008

Esercizio n. 1 (punti 7)

Dato il seguente codice in linguaggio C, dove il tipo **tree** rappresenta un albero binario di interi e **ord_ins** la funzione di inserimento per alberi binari di ricerca:

```
#include <stdio.h>
. . .
int  f(int i, tree T)
{ int n=0;
  while (T!=NULL)
     { if (i==T->value) {n++; T=T->left; }
       else
           if (i<T->value) T=T->left;
           else T=T->right; }
  return n; }

void main(void)
{ int i;
  tree T=NULL;
  scanf("%d",&i);
  do { T=ord_ins(i,T);
       scanf("%d",&i); }
  while ((i>=0) and (i<=1000));
  scanf("%d",&i);
  printf("%d", f(i,T) ); }
```

a) Si descriva cosa fa questo frammento di programma;

b) Si supponga che il ciclo **do** legga da input N interi positivi (compresi tra 0 e 1000) e successivamente -1. Si valuti la complessità come numero di esecuzioni del test sottolineato nella funzione **f**, se il valore letto successivamente all'uscita dal ciclo **do** è diverso da quelli presenti nell'albero. Si motivi la risposta, identificando i casi migliori e peggiori.

   Il programma acquisisce una sequenza di valori interi comprsi tra 0 e 1000 e li inserisce in un albero binario di ricerca.

Successivamente legge un altro intero e conta quante volte compare nell'albero tramite la funzione iterativa **`f`**.

La complessità come numero di test sottolineato nel codice dipende dall'ordine con cui i valori sono inseriti.

Caso migliore:
Primo valore inserito, il più piccolo della sequenza e si cerca poi un valore minore di tutti quelli della sequenza. Il sottoalbero sinistro della radice è vuoto (o ha il solo nodo, quello con valore -1). Il test è eseguito 1 volta (o 2 volte).
(dualmente, primo valore inserito, il più grande della sequenza e si cerca poi un valore maggiore di tutti quelli della sequenza. Il sottoalbero destro della radice è vuoto. Il test è eseguito 1 volta.)

Caso peggiore:
Sequenza ordinata di valori, per esempio in senso decrescente e si cerca poi un valore minore di tutti quelli della sequenza. Il sottoalbero sinistro della radice degenera in una lista di N elementi (N+1, se contiamo anche il nodo con valore -1). Il test è eseguito N volte (N+1, se contiamo anche il nodo con valore -1).
(dualmente, Sequenza ordinata di valori, per esempio in senso crescente e si cerca poi un valore maggiore di tutti quelli della sequenza. Il sottoalbero destra della radice degenera in una lista di N elementi. Il test è eseguito N volte.)

Caso medio:
Albero bilanciato e valore cercato non presente: si esegue il tets un numero di volte proporzionale all'altezza dell'albero ($\log_2 N$).

Esercizio n. 2 (punti 23)

Dato un file di tipo testo, PAGELLE.TXT, che contiene i voti in pagella degli alunni di una classe, dove per ciascuna riga sono riportati:

- nome, stringa di 20 caratteri;
- cognome, stringa di 20 caratteri;
- 12 voti espressi come interi positivi (1 insufficiente, 2 sufficiente, 3 buono, 4 distinto, 5 ottimo) per le 12 materie in pagella;

si chiede di scrivere un programma C organizzato in tre funzioni dedicate rispettivamente a:

a) a partire dal file PAGELLE.TXT, creare una lista L in memoria centrale che memorizzi per ciascun alunno della classe il nome e il cognome, i 12 voti (vettore di interi) e il voto medio ottenuto (reale); non si richiede che L sia ordinata su alcun campo;
b) a partire da L, produrre una seconda lista L2 ordinata in senso non crescente in base al voto medio;
c) a partire da L2, stampare in modo ricorsivo nome, cognome e voto medio degli alunni che hanno un voto medio superiore o uguale a 3.

È possibile utilizzare ***librerie C*** (ad esempio per le stringhe). Nel caso si strutturi a moduli l'applicazione qualunque ***libreria utente*** va riportata nello svolgimento.

```
/* MAIN FILE */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 12

typedef   struct {
   char nome[20];
   char cognome[20];
    int voti[N];
    float media;
} el_type;

typedef struct nodo
       {el_type  value;
           struct nodo *next;  } NODO;

typedef NODO* list;

list  crea_lista(FILE * , list );
list crea_lista2(list);
void stampa(list, int );
list  cons(el_type , list );
list ordins(el_type, list);

void main(void) {
   list L = NULL;
   list L2 = NULL;
    FILE *f;

   f = fopen("pagelle.txt", "rt");
```

```
/* PRIMA DOMANDA */
  L = crea_lista(f, L);
  fclose(f);

/* SECONDA DOMANDA */
  L2=crea_lista2(L);

/* TERZA DOMANDA */
   stampa(L2, 3);
}


list crea_lista(FILE *f, list L)
{
   el_type  EL;
   int i;
   float sum;
   while (!feof(f))
      { sum=0;
        scanf("%s%s",EL.nome, EL.cognome);
         for(i=0;i<N;i++)
             { scanf("%d",EL.voti[i]);
               sum = sum + voti[i]; }
        EL.media= sum /N;
        L=cons(EL,L);
      }
   return L;
}
```

```
list  cons(el_type e, list l)
{list t;
  t=(list)malloc(sizeof(NODO));
  t->value=e;
  t->next=l;
  return(t);
}


list crea_lista2(list L)
{ list aux=NULL;
  while(L!=NULL)
       { aux=ordins(L->value, aux);
         L=L->next; }
  return aux;
}

list   ordins(el_type el, list l)
{
 if (L==NULL) return(cons(el,l));
  else
      { if (el.media >= l->value.media)
              return(cons(el,l));
        else
return(cons(l->value,ordins(el,l->next)) );
      }
}
```

```
void stampa(list L, int m)
{
  if (L!=NULL)
       { if (L->value.media >= m)
           {printf("%s\t%s\t%f\n",
               L->value.nome,
               L->value.cognome,
               L->value.media); }
         stampa(L->next, m);
       }
}
```