Esercizi di preparazione alla prova scritta

ARGOMENTI: Istruzione dominante, complessità

ESERCIZIO n° 1

Si analizzi il seguente frammento di codice C:

```
#include <stdio.h>
#define N 10
int V[N]={1,2,3,4,5,6,7,8,9,10};
. . .

main()
{ FILE *f;
  int i,j;
  f=fopen("MIO.TXT","rt");
  while (!feof(f))
      { fscanf(f,"%d", &j);
        if (member(j,V))
             printf("Trovato \t %d\n",j);
      }
  fclose(f);
}
```

Supponendo che la funzione **member** realizzi una ricerca esaustiva, si individui l'istruzione dominante nel **main**;

si indichi la complessità di questo codice, nel caso peggiore, in termini della dimensione **N** del vettore di interi e del numero **M** di valori letti dal file MIO.TXT.

SOLUZIONE:

```
main()
{ FILE *f;
  int i,j;
  f=fopen("MIO.TXT","rt");
  while (!feof(f))
      { fscanf(f,"%d", &j);
        if (member(j,V))
             /* istruzione dominante */

             printf("Trovato \t %d\n",j);
      }
  fclose(f);
}
```

Nel caso peggiore, la chiamata di **member** (ricerca esaustiva in un vettore di **N** componenti) ha un costo O(**N**) e viene invocata **M** volte, quindi la complessità risulta O(**M*N**).

Nel caso migliore, O(**M**).

Nel caso medio, O(**M*N/2**).

ESERCIZIO n° 2

Dato il seguente frammento di codice:

```
#include <stdio.h>
#define N ...
int X= ...;

int f(int a, int b)
{   if (!(a||b)) return 1;
    else   return f(a-2,b-1) + ++X;
}

main()
{   printf("%d\n", f(2*N,N)); }
```

Individuare l'istruzione dominante nella funzione **f**;

Valutare la complessità della computazione che si innesca attraverso la chiamata ad **f** da parte della funzione **main**, in funzione di N.

SOLUZIONE:

L'istruzione dominante è il test dell'**if** (o la chiamata ricorsiva).

La funzione, chiamata con parametri **2*N** ed **N**, raggiunge la condizione di terminazione dopo **N** chiamate ricorsive (alla **N**+1-esima). Quindi la complessità è O(**N**).

ESERCIZIO n° 3

Dato il seguente codice C, in cui le liste L1 ed L2 sono liste ordinate in senso crescente di interi:

```
...

void main(void)
{...
 list  L1, L2, L;

 ... /* creazione liste L1 ed L2 ordinate */


L=L1;

while (L!=NULL)
  {if (L2!=NULL)
       while ((isequal(head(L),head(L2)))&&
              (L2!=NULL))
             { printf("%d",head(L2));
               L2=tail(L2);
             }
   if ((L2!=NULL)&&(L!=NULL))
      while ((isless(head(L),head(L2)))&&
             (L!=NULL))
            L=tail(L);
  }
...
```

Valutarne la complessità nel caso in cui le due liste abbiano gli stessi elementi (siano identiche), in termini della loro dimensione **N**.

SOLUZIONE:

In questo caso, essendo i test dei cicli **while** più innestati le istruzioni dominanti (**isless** ed **isequal**), per ciascun elemento di **L** si eseguono due volte sia il test chiamando **isequal** sia il test chiamando **isless**.

In totale **2*N (-1)** chiamate (test).

ESERCIZIO n° 4

Dato il seguente frammento di codice:

```
#include <stdio.h>
#define N 10
#define M 100

void main(void)
{
 int Tab[M];
 int V[N];
 int i,j, trovato;
 leggi(V);
 leggi2(Tab);

 for (i=0;i<N;i++)
    { j=0;
      trovato=0;
      while ((j<M)&& !trovato)
        { if (V[i]==Tab[j])
              {printf("Trovato %d", i);
               trovato=1;}
          j++;
         }
    }
}
```

Individuare l'istruzione dominante nella funzione **main**;

Valutarne il numero di esecuzioni in funzione di **N** ed **M** nel caso migliore e nel caso peggiore.

SOLUZIONE:

```
#include <stdio.h>
#define N 10
#define M 100
void main(void)
{int Tab[M];
 int V[N];
 int i,j, trovato;
 leggi(V);
 leggi2(Tab);
 for (I=0;i<N;i++)
    { j=0;
      trovato=0;
      while ((j<M)&& !trovato)
        { if (V[i]==Tab[j]) /*DOMINANTE */
              {printf("Trovato %d", i);
               trovato=1;}
          j++;  }
    }
}
```

Il caso migliore si ha quando il primo elemento del vettore **Tab** è uguale a tutti gli elementi del vettore **V**: in questo caso il test dell'**if** è eseguito una sola volta, per ciascun elemento di **V**, ovvero **N** volte.

Il caso peggiore si ha quando nessun elemento del vettore **Tab** è uguale ad alcun elemento del vettore **V**: in questo caso il test dell'**if** è eseguito **M** volte, per ciascun elemento di **V**, ovvero **N*M** volte in totale.

ESERCIZIO n° 5

Dato il seguente codice in linguaggio C, dove il tipo **list** rappresenta una lista di interi, **ins_ord** la funzione di inserimento ordinato e **member** la funzione (ricorsiva) di ricerca su liste:

```
#include <stdio.h>
. . .
void main(void)
{ int i,sum=0;
  list L1=NULL;
  list L2=NULL;

  scanf("%d",&i);
  do { L1=ins_ord(i,L1);
       scanf("%d",&i); }
  while (i>0);

  scanf("%d",&i);
  do { L2=ins_ord(i,L2);
       scanf("%d",&i); }
  while (i>0);

  do { if (member(L1->value,L2))
          sum++;
     L1=L1->next;
    }
  while (L1!=NULL);
  printf(''%d Trovati'',sum);
}
```

Si indichi cosa fa questo frammento di programma;

Supponendo che da ingresso siano dati i valori:

N N-1 … 2 1 0

10 9 … 2 1 0

si indichi il numero di chiamate della funzione **member** eseguite e il numero (totale) di confronti eseguiti all'interno di tale funzione.

SOLUZIONE:

Il programma legge da input due sequenze di valori positivi e crea, con due cicli do, due liste. Successivamente, con il terzo ciclo do, conta quanti elementi della prima lista appartengono alla seconda.

Per i primi **N-11** valori della lista L1, si analizza tutta la lista L2. Per gli ultimi dieci elementi della prima lista, invece, si controllano sono il primo, il primo e il secondo, il primo e il secondo e il terzo, etc etc elementi di L2.

Il numero di chiamate alla **member** risulta quindi:

(N-11)*11 + 1*1 + 1*2 + ... + 1*11 =

(N-11)*11 + 11*12/2

ESERCIZIO n° 6

Dato il seguente codice in linguaggio C, dove il tipo **tree** rappresenta un albero binario di caratteri e **ins_ord** la funzione di inserimento ordinato (che inserisce elementi minori o uguali alla radice nel sotto-albero sinistro, elementi maggiori nel sotto-albero destro) in un albero binario di ricerca:

```
#include <stdio.h>
. . .
void main(void)
{ char c;
  int tot=0;
  tree T=NULL;
  do  { scanf("%c",&c);
        T=ins_ord(c,T);  }
  while (c>'A');
  scanf("%c",&c);
  while (T!=NULL)
       { if (c==T->value)
              { tot++;
                T=T->left; }
         else  if (c<T->value) T=T->left;
               else T=T->right;  }
  printf(''%d'',tot);
}
```

Si indichi cosa fa questo frammento di programma;

Se ne indichi la complessità in termini di numero di test condizionali eseguiti dalla istruzione **while** (test sottolineato), ipotizzando che da input siano stati inseriti gli N caratteri (N dispari): **acacacac..acacacA**

e successivamente il carattere: **b**

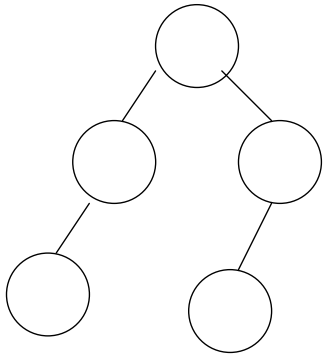
SOLUZIONE:

Il programma legge da input, con un ciclo do, una sequenza di caratteri che inserisce in un albero binario di ricerca. Successivamente, letto un carattere da input, con il secondo ciclo do conta quanti elementi dell'albero sono uguali a tale carattere.

Se da input sono stati inseriti gli N caratteri (N dispari):

acacacac..acacacA

l'albero ha di fatto due soli cammini dalla radice a due foglie (ciascuno di lunghezza (N-1)/2) e risulta del tipo:



Leggendo successivamente il carattere **b** (non presente nell'albero) si paga un costo proporzionale all'altezza dell'albero, ovvero pari a (N-1)/2.

ESERCIZIO n° 7

Il seguente programma C è compilato correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione e qual è il numero di invocazioni della procedura **Proc** (si motivi opportunamente la risposta data)?

```
#include <stdio.h>
#include <stdlib.h>
#define L 20
void Proc(char [], int);
int N=L;
main ()
{ char *s; int i;
 s = (char *) malloc(L);
 s = "Fondamenti di Infor";
 Proc(s,L);
 N--; N+2;
 printf("Dopo Proc N vale: %d\n",N);
  for (i=L-2; i>0; i=i-10)
            printf("%d\n", *(s+i));
}

void  Proc(char y[], int DIM)
{ int i, N;
  N++; i=1;
  printf("Qui N vale: %d\n", N);
  while (i<DIM) { y[i]=y[0]; i=i+5; }
```

SOLUZIONE:

Il programma è corretto sintatticamente e stampa:

Qui N vale: 1

Dopo l'invocazione di Proc N vale: 19

114

116

Infatti la stringa s viene inizializzata a "Fondamenti di Infor", terminatore compreso. Successivamente viene invocata la procedura Proc(), alla quale si passa per riferimento la stringa (vettore di char). In particolare, Proc pone uguale a 'F' i caratteri della stringa di posizione 1, 6, 11, e 16. Infine, il programma principale stampa a video i caratteri s[18] e s[8], che non sono stati modificati da Proc() e valgono ancora rispettivamente 'r' e 't'; di tali caratteri viene stampato il codice ASCII corrispondente (formato %d nella printf()).

ESERCIZIO n° 8

Scritto del 16 febbraio 2004

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

```
#include <stdio.h>
. . .

tree member(int i, tree T)
{ if (T==NULL) return NULL;
  else  if (i==T->value) return T;
        else if (i<T->value) return(member(i,T->left));
             else return(member(i,T->right));
}

void main(void)
{ int i;
  tree T1=NULL;
  scanf("%d",&i);
  do { T1=ord _ins(i,T1);
       scanf("%d",&i); }
  while (i>0);
  scanf("%d",&i);
  inorder(member(i,T1));
}
```

a) Si indichi cosa fa questo frammento di programma;
b) Si supponga che il ciclo **do** legga da input M valori interi positivi e a seguire 0, crescenti come valore (per es., 2, 4, 6, 8 se M=4). Si valuti la complessità come numero di confronti eseguiti (test sottolineato nella funzione **member**), se il valore da cercare non è presente nell'albero (per es., 10).

Soluzione: (feb 2004)

a) Il programma legge da input una sequenza di interi con un ciclo **do**), terminata da un valore <=0 e inserisce i valori letti in un albero binario di ricerca. Successivamente legge un intero **i**, lo cerca tramite la **member** nell'albero creato e stampa il (sotto)albero avente **i** nella radice, tramite visita simmetrica.

b) Supponendo che il ciclo **do** legga da input M valori interi positivi e a seguire 0, crescenti come valore (per es., 2, 4, 6, 8 se M=4), l'albero viene creato tutto sbilanciato a destra (solo la radice ha un nodo figlio sinistro) e il numero di confronti eseguiti (test sottolineato nella funzione **member**) risulta in questo caso pari a M.

ESERCIZIO n° 9

Scritto del 18 febbraio 2005

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 member(int  i, tree T)
{ if (T==NULL) return 0;
  else  if (i==T->value)
          return (1 + member(i,T->left)
                    + member(i,T->right));
        else
           if (i<T->value)
                return member(i,T->left);
           else return member(i,T->right); }

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<=9));
  scanf("%d",&i);
  printf("%d",member(i,T)); }
```

- Si indichi cosa fa questo frammento di programma;
- Si supponga che il ciclo **do** legga da input N+1 interi (N valori compresi tra 1 e 9, e poi il valore 10). Si valuti la complessità come numero di attivazioni della funzione **member**, se il valore letto successivamente all'uscita dal ciclo **do** non è presente nell'albero.

Soluzione: (feb 2005)

c) Il programma legge da input una sequenza di interi con un ciclo **do**), terminata da un valore non compreso tra 0 e 9 e inserisce i valori letti in un albero binario di ricerca. Successivamente legge un intero **i**, lo cerca tramite la **member** nell'albero creato. La **member** restituisce il numero di nodi dell'albero che hanno un valore uguale ad **i**.

d) Supponendo che il ciclo **do** legga da input N+1 interi (N valori compresi tra 1 e 9, e poi il valore 10, la complessità come numero di attivazioni della funzione **member** dipende dai valori inseriti.

- In particolare, se gli N valori sono dati in ordine crescente/decrescente, l'albero risulta completamente sbilanciato (il sottoalbero sinistro della radice degenera in una lista) e il numero di attivazioni della funzione **member** (nel caso peggiore in cui si cerchi un valore <0) è pari a N+1.
- Se invece gli N valori dati in ingresso portano ad avere un albero perfettamente bilanciato, il numero di attivazioni della funzione **member** è pari all'altezza del'albero più uno ($\log_2(N)+1$).

ESERCIZIO n° 10

Scritto del 17 febbraio 2006

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


```
#include <stdio.h>
...
int uguali(int  i, tree root)
{ if (root==NULL) return 0;
  else  if (i==root->value)
          return (1 + uguali(i,root->left));
        else if (i>root->value)
          return uguali(i,root->right);
        else return uguali(i,root->left); }

void main(void)
{ int i;
  tree T=NULL;
  scanf("%d",&i);
  do { T=ord_ins(i,T);
       scanf("%d",&i); }
  while ((i>=1) and (i<=9));
  scanf("%d",&i);
  printf("%d",uguali(i,T)); }
```

- Si indichi cosa fa questo frammento di programma;
- Si supponga che il ciclo **do** legga da input N+1 interi (N interi compresi tra 1 e 9, e poi il valore 0). Si valuti la complessità (nel caso migliore e nel caso peggiore) come numero di test di uguaglianza (test sottolineato nella funzione **uguali)**, se l'intero letto successivamente all'uscita dal ciclo **do** è maggiore di tutti quelli presenti nell'albero.

Soluzione: (feb 2006)

Il programma legge da input una sequenza di interi compresi tra 1 e 9 terminata dal valore 0 e inserisce i valori letti in un albero binario di ricerca. Successivamente legge un intero **i**, conta quanti dell'albero che hanno un valore uguale ad **i**.

Letti da input N+1 interi (N interi compresi tra 1 e 9, e poi il valore 0), la complessità (nel caso migliore e nel caso peggiore) come numero di test di uguaglianza (test sottolineato nella funzione **uguali)**, se l'intero letto successivamente all'uscita dal ciclo **do** è maggiore di tutti quelli presenti nell'albero, è:

- Caso migliore, albero completamente sbilanciato a sinistra e valori inseriti in senso decrescente, 1 confronto

- Caso peggiore, albero completamente sbilanciato a destra e valori inseriti in senso crescente, N confronto

ESERCIZIO n° 11 Scritto del 16 febbraio 2007

Dato il seguente codice in linguaggio C, dove il tipo **list** rappresenta una lista di interi e **insord** la funzione di inserimento ordinato in senso non decrescente in una lista:

```
#include <stdio.h>
#define N 10

int ricerca(int e, int V[], int i, int j)
{ int m=(i+j)/2;
  int trovato=0;
  while((i<=j)&&(!trovato)) /*(istr. dom.)*/
     { if (e==V[m]) trovato=1;
       else if (e<V[m])  j=m-1;
            else  i=m+1;
       m=(i+j)/2; }
  return trovato;
}

void main(void)
{ int i, t=0;
  int V[N]= {1,2,3,4,5,6,7,8,9,10};
  list L1 =NULL;

  scanf("%d",&i);
  do  { L1=insord(i,L1);
        scanf("%d",&i);   }
  while (i>0);

  scanf("%d",&i);
  while ( (L1!=NULL))
      {if (i==L1->value)
            if (ricerca(i,V,0,N))  t++;
       L1=L1->next; }
  printf("%d",t);
}
```

c) Si indichi cosa fa questo frammento di programma e la funzione **ricerca** in particolare;

d) Se ne indichi la complessità in termini di numero di test condizionali dell'istruzione **while** (test sottolineato) nella

funzione **`ricerca`**, ipotizzando che da input siano stati inseriti **`M`** valori, in funzione di **`M`** ed **`N`**. Discutere il caso migliore e il caso peggiore.

Soluzione:

a) Nel main si leggono da tastiera alcuni valori interi (maggiori di 0 che è il terminatore della sequenza da input) e li si inserisce in ordine non decrescente nella lista L1. Si acquisice poi un ulteriore valore che viene ricercato dapprima nella lista, confrontandolo con il valore corrente della lista (che viene man mano fatta "scorrere") (ricerca esaustiva). All'interno di questo ciclo viene richiamata la funzione ricerca che svolge la ricerca binaria dell'elemento i all'interno del vettore V[]; viene restituito al main il risultato della ricerca (trovato o non trovato), se trovato il main incrementa il valore di t. Perciò il compito di questo programma è contare quante volte (t) l'elemento i letto da tastiera è presente sia nella lista L1, sia nel vettore V[]. Al termine si stampa a video il valore di t.

b) Il caso migliore si ha se l'elemento i è il rpimo della lista L1 e il termine mediano del vettore (ossia 5), in questo caso si farà una sola iterazione del ciclo while. Perciò la complessità in questo caso sarà O(M,N) = 1; nel caso sia l'ultimo elemento di L1, la complessità del caso migliore sarà uguale a O(M,N) = M.
Il caso peggiore si ha invece quando l'elemento i (presente nella lista L1, in ultima posizione) non è presente nel vettore (o si ritrova all'ultima iterazione). In tal caso per ogni iterazione della funzione ricerca il vettore verrà dimezzato, cosicché si avrà una complessità del O(M, N) = $M*\log_2 N$ se i è uguale ad ogni elemento della lista L1 e non compare in V, oppure O(M, N) = $\log_2 N$ se i compare una sola volta in L1 e non compare in V.