Complessità algoritmi su strutture dati

Struttura di dato	**Ricerca**	**Complessità (caso peggiore)**
Tavola non ordinata (N elementi)	Ricerca sequenziale	O(N)
Tavola ordinata (N elementi)	Ricerca binaria	O($\log_2$N)
File non ordinato (N elementi)	*Ricerca sequenziale*	*O(N)*
File ordinato (N elementi)	*Ricerca binaria*	*O($\log_2$N)*
Lista (anche se ordinata)	Ricerca sequenziale	O(N)
Alberi binari (N nodi)	Ricerca esaustiva	O(N)
Alberi binari di ricerca	Ricerca binaria	O(h), h altezza dell'albero (vale $\log_2$N se albero bilanciato)

FONDAMENTI DI INFORMATICA II (tempo 2h) – 2013 Maggio 23

Esercizio n. 1 (punti 7)

Dato il seguente codice in linguaggio C, dove il tipo **`list`** rappresenta una lista di interi (non ordinata):

```
int p(list  A, list L)
{ int c=0;
  while (A!=NULL)
          {if  (search(A->value, L)) c++;
           A=A->next;}
 return c; }

int search(int i, list L)
{ if (L!=NULL)
         {if (i==L->value) return 1+search(i,L->next);
          else return search(i, L->next);
          }
  else return 0;  }
```

a) Si descriva cosa restituisce la funzione **`p`** e che tipo di ricerca fa la funzione **`search`**.

b) Si discuta la complessità del codice sapendo che la lista **`l1`** contiene **`M`** elementi e la lista **`l2`** contiene **`N`** elementi. Individuare la complessità della chiamata **`p(l1,l2)`** data dal numero di esecuzioni complessive del test sottolineato nella funzione **`search`**.

La funzione **p** conta gli elementi comuni alle due liste. Per ciascun elemento della prima lista, chiama la funzione **search** per cercarlo nella seconda, con ricerca sequeziale.

Se la lista **l1** contiene **M** elementi e la lista **l2** contiene **N** elementi, la complessità della chiamata **p(l1,l2)** data dal numero di esecuzioni complessive del test sottolineato nella funzione **search** è:

M x N

FONDAMENTI DI INFORMATICA – 12 Settembre 2011 - tempo 30'

Esercizio n. 1 (punti 7)

Dato il seguente codice in linguaggio C, dove il tipo **`list`** rappresenta una lista di interi, **`insord`** la funzione di inserimento ordinato in senso crescente in una lista e **`member`** la funzione di ricerca binaria in un vettore ordinato di interi:

```
#include <stdio.h>
void main(void)
{ int i, c=0;
  int v[10] = {0,1,2,3,4,5,6,7,8,9};
  list L1 = NULL;

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

  while (L1!=NULL)
     { if ( member(L1->value,v) ) c++;
       L1=L1->next; }
  printf("\n%d\n", c); }
```

a) Si indichi cosa fa questo frammento di programma;

b) Se ne indichi la complessità in termini di numero di test condizionali eseguiti dalla funzione **`member`**, ipotizzando che da input siano stati inseriti N valori distinti (da 1 a N) seguiti da 0. Si distinguano caso migliore, peggiore e medio (il vettore **`v`** è dato e di dimensione 10) motivando opportunamente.

Il programma crea una lista di N interi (N positivi). Poi conta quanti elementi di questa lista appartengono al vettore v, ed infine stampa a video tale valore.

Caso migliore: tutti gli N valori sono uguali al mediano di v, la complessità è di N volte un confronto:

$$N*1$$

Caso peggiore: tutti gli N valori sono diversi da qualsiasi elemento di v, la complessità è di N volte * $\log_2(10)$:

$$(N+1)*\log_2(10)$$

Caso medio: la ricerca binaria costa nel caso medio come nel caso peggiore:

$$N*\log_2(10)$$

FONDAMENTI DI INFORMATICA II (tempo 2h, punti 30/30) – 27 Marzo 2013

Esercizio 1 (punti 7)

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

```
#include <stdio.h>
int fun(int e, tree T)
{ if (T==NULL) return 0;
  else   if (e==T->value) return 1;
         else  if (e<T->value) return fun(e,T->left);
               else return fun(e,T->right);   }

void main(void)
{ int i, sum=0;
  tree T =NULL;
  scanf("%d",&i);
  while (i>0)  { T=ordins(i,T);
                 scanf("%d",&i); }
  for(i=0; i<10; i++)
      if ( fun(i,T) )  sum=sum+1;
  printf("%d",sum);   }
```

- Si indichi cosa fa il programma e la funzione **`fun`** (cosa viene stampato dal main?);
- Se ne indichi la complessità in termini di numero di test condizionali dell'istruzione **`if`** (test sottolineato) nella funzione **`fun`**, ipotizzando che da input siano stati dati **`M`** valori **(nessuno uguale a 0,1, ..9)** che, inseriti nell'albero **`T`** , hanno prodotto un albero perfettamente bilanciato.

Il programma crea un albero binario di ricerca di interi, leggendo i valori da input. Cerca poi con la funzione **`fun`** i primi 10 valori interi da 0 a 9, e stampa quanti di questi compaiono nell'albero.
`fun` esegue una ricerca binaria nell'albero.

Se da input sono stati dati **`M`** valori **(nessuno uguale a 0,1, ..9 – caso peggiore)** che, inseriti nell'albero **`T`** , hanno prodotto un albero perfettamente bilanciato, in ciascuno dei dieci casi la complessità è $\log_2 M$, quindi in complesso:

$$10 * \log_2 M$$

FONDAMENTI DI INFORMATICA – 19 Febbraio 2013

Dato il seguente codice in linguaggio C, dove il tipo **list** rappresenta una lista di interi, e **tree** un tipo albero binario (di ricerca). Le variabili **L** di tipo **list** e **T** di tipo **tree** siano state create leggendo da input valori che sono stati inseriti poi nelle strutture dati in modo che siano rispettivamente una lista ordinata e un albero binario di ricerca.

```
#include <stdio.h>
list crea_lista(list);
tree crea_tree(tree);
list L=NULL;
lree T=NULL;

int same(list l, tree t)
{ int trovato=0;
  tree aux=t;
  while (l!=NULL)
    { t=aux; trovato=0;
      while ((t!=NULL) && (!trovato))
         if (l->value==t->value) trovato=1;
         else if (l->value <= t->value) t=t-> left;
              else t=t->right; }
      if (trovato) l=l->next;
      else l=NULL;
    }
   return trovato; }
```

```
void main(void)
{  L=crea_lista(L);
   T=crea_tree(T);

  printf("%d", same(L,T));

}
```

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

b) Se ne indichi la complessità in termini di numero di esecuzioni del test condizionale sottolineato, ipotizzando che da input siano stati inseriti N valori in lista e M valori nell'albero e che gli N valori in lista siano tutti presenti anche nell'albero T.

Il programma crea lista L e albero T, poi con la funzione **same** in particolare controlla che ogni elemento della lista L sia in T. Questa funzione scandisce sequenzialmente L e cerca ogni elemento di L in T applicando una ricerca binaria.

La complessità in termini di numero di esecuzioni del test condizionale sottolineato, ipotizzando che da input siano stati inseriti N valori in lista e M valori nell'albero e che gli N valori in lista siano tutti presenti anche nell'albero T:

- nel caso peggiore è di N*h dove h è l'altezza dell'albero (tutti gli elementi in lista sono uguali a alcune delle foglie di T; h vale $\log_2 M$ se T è perfettamente bilanciato; se completamente sbilanciato h vale M);
- nel caso migliore è N*1 (tutti gli elementi in lista sono uguali alla radice di T)

FONDAMENTI DI INFORMATICA – 19 Febbraio 2013
Programmazione C: tempo 2 ore (+ 30 min per A+B)

Un file di tipo testo residenti.txt contiene un elenco di nominativi di residenti di un comune. Per ogni paziente, su una linea sono riportati cognome e nome (stringhe di 20 char) e il codice fiscale (stringa di 16 char). I codici fiscali sono tutti diversi tra loro.

Un secondo file, sportello.txt, di tipo testo, memorizza i residenti che si sono rivolti allo sportello di relazioni con il pubblico, per reclami vari.

Per ciascuna linea sono riportate data (nel formato gg/mm/aa), e il nominativo di un residente (cognome e nome stringhe di 20 char).

Ogni cittadino residente compare una sola volta nel primo file, ma può comparire più volte nel secondo, ma associato a date diverse tra loro.

Si scriva un programma C **strutturato in (almeno) due funzioni** dedicate rispettivamente a:

a. creare un albero binario di ricerca T, ordinato in base al campo cognome, e a parità di cognome in base al nome, che per ciascun residente del primo file (residenti.txt), memorizza cognome e nome del residente e il numero totale di reclami (ovvero il numero di volte in cui compare nel secondo sportello.txt; la funzioneA riceve il puntatore al primo file, il puntatore al secondo file, più altri parametri a scelta, e restituisce il puntatore all'albero creato;
b. accedendo a T, determinare qual è il residente che ha fatto più reclami; la funzioneB ha come parametri il puntatore all'albero, più altri parametri a scelta, e restituisce l'elemento dell'albero che ha il campo "numero totale di reclami" più alto.

c. **Per chi svolge il compito A+B (vale ulteriori 20 punti, su totale di 60 punti per A+B; tempo +30 min):** Si leggano i primi 10 elementi del file residenti.txt in un vettore V e si ordini V in base al cognome, e a parità di cognome in base al nome. La funzioneC è una procedura che ha come parametri il puntatore al file, il vettore V (più eventuali altri a scelta), il tipo restituito è `void`. Il main stampi poi a video il contenuto del vettore.

FONDAMENTI DI INFORMATICA - A+B - parteB – 24 Gennaio 2013
Programmazione C: tempo 2 ore (+ 30 min per A+B)

Un file di tipo testo pazienti.txt contiene un elenco di nominativi di pazienti di un'Azienda Ospedaliera. Per ogni paziente, su una linea sono riportati cognome e nome (stringhe di 20 char) e il codice fiscale (stringa di 16 char). I codici fiscali sono tutti diversi tra loro.

Un secondo file, reparto.txt, di tipo testo, memorizza i pazienti ricoverati in un reparto.

Per ciascuna linea sono riportate data (nel formato gg/mm/aa), e il nominativo di un paziente (cognome e nome stringhe di 20 char).

Ogni paziente compare una sola volta nel primo file, ma può comparire più volte nel secondo.

Si scriva un programma C **strutturato in (almeno) due funzioni** dedicate rispettivamente a:

a. creare una lista L in memoria centrale, ordinata in base al campo cognome, e a parità di cognome in base al nome, che per ciascun paziente del primo file (pazienti.txt), memorizza cognome e nome del paziente e il numero totale di giorni di ricovero (ovvero il numero di volte in cui compare nel secondo reparto.txt; la funzioneA riceve il puntatore al primo file, il puntatore al secondo file, più altri parametri a scelta, e restituisce il puntatore alla lista creata;

b. accedendo a L, determinare qual è il paziente che è stato ricoverato per il maggior numero di giorni; la funzioneB ha come parametri il puntatore alla lista, più altri parametri a scelta, e restituisce l'elemento della lista che ha il campo "numero totale di giorni di ricovero" più alto.

c. **Per chi svolge il compito A+B (vale ulteriori 20 punti, su totale di 60 punti per A+B; tempo +30 min):** Si inseriscano i primi 10 elementi della lista L in un vettore V e si ordini V in base al numero totale di giorni di ricovero. La funzioneC è una procedura che ha come parametri il puntatore alla lista L, il vettore V (più eventuali altri a scelta), il tipo restituito è `void`. Il main stampi poi a video il contenuto del vettore.

FONDAMENTI DI INFORMATICA – parte B –
Esercizio n. 2 (punti 20) 23 Maggio 2013

Un file di testo GITE.TXT memorizza le gite in treno di alcune classi scolastiche. Ciascuna gita contiene:

- nome accompagnatore: stringa contenente al più 20 caratteri, senza spazi
- stazione di partenza: stringa contenente al più 20 caratteri, senza spazi
- stazione di arrivo: stringa contenente al più 20 caratteri, senza spazi

Un secondo file di testo, LUNG.TXT, contiene le lunghezze chilometriche delle varie tratte. Ciascuna tratta contiene:

- stazione di partenza: stringa contenente al più 20 caratteri, senza spazi
- stazione di arrivo: stringa contenente al più 20 caratteri, senza spazi
- lunghezza della tratta (da leggere come valore intero).

Si realizzi un programma C, organizzato in almeno due funzioni, rispettivamente dedicate a:

a) costruire un albero binario di ricerca, ordinato sul campo nome accompagnatore, che riporta in ciascun nodo: nome accompagnatore, stazione di partenza, stazione di arrivo, lunghezza della tratta (intero). La funzioneA riceve i puntatori ai due file e restituisce il puntatore all'albero T;

b) previo inserimento da input di un nome accompagnatore letto nel main, accedendo a T determinare il totale dei chilometri che l'accompagnatore deve percorrere. La funzioneB riceve il nome dell'accompagnatore letto da main, il puntatore ad albero T, e restituisce un intero.

Ad esempio, se il file GITE.TXT contiene:

Rossi	Bologna	Firenze
Verdi	Padova	Venezia
Verdi	Ferrara	Padova
Bianchi	Catania	Enna
Bianchi	Enna	Palermo

e il file LUNG.TXT contiene:

Bologna	Firenze	80
Padova	Venezia	40
Ferrara	Padova	60
Catania	Enna	60
Enna	Palermo	100

per il nome “Verdi” la funzioneB restituisce un totale di chilometri pari a 100.

NOTA BENE: si consegnino i sorgenti, eseguibile e i file di uscita generati.
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.