Laurea In Ingegneria dell'Informazione

Esercitazioni Guidate di Tecniche della Programmazione

Note introduttive: Seguire le raccomandazioni date nelle precedenti EG …

12. Esercitazione autoguidata 12 – alberi binari

12.1. Costruzione di un albero (directory ALBERI1)

Scrivere un programma che

- riceve il nome di un file contenente la rappresentazione parentetica di un albero binario di interi;
- ne costruisce l'albero binario in memoria centrale, mediante `struct` e puntatori;
- stampa l'albero, ottenendo sul video la medesima rappresentazione parentetica di partenza;
- dealloca poi l'albero.

(Nei file `A1.TXT`, `A2.TXT` , `A3.TXT` ci sono le rappresentazioni parentetiche di tre alberi. Provare con queste il programma).

Suggerimenti seguono …

Suggerimento 1: (forma parentetica)

ecco la forma parentetica per l'albero che ha radice 12 e sottoalberi sin e des vuoti:

```
( 12()())
```

ecco invece la forma parentetica per l'albero che ha

- radice 12
- sottoalbero sin con radice 24 e sottoalberi sin e des vuoti
- sottoalbero des con radice 36 e sottoalberi sin e des vuoti

```
( 12( 24()())( 36()()))
```

ecco invece la forma parentetica per l'albero che ha

- radice 12
- sottoalbero sin con radice 36 e sottoalberi sin e des vuoti
- sottoalbero des con
    - radice 24
    - sottoalbero sin avente
        - radice 48
        - sottoalbero sin avente
            - radice 64 e sottoalberi sin e des vuoti
        - sottoalbero des vuoto
    - sottoalbero des vuoto

```
( 12( 36()())( 24( 48( 64()())()())()))
```

Suggerimento 2:

```
/* definizione dei tipi di dato */

typedef int TipoElemAlbero;          /* albero di interi */

struct StructAlbero {
  TipoElemAlbero info;
  struct StructAlbero *des,
                      *sin;
};
typedef struct StructAlbero TipoNodoAlbero;
typedef TipoNodoAlbero * TipoAlbero;
```

******* Il prossimo suggerimento riguarda, se serve, la definizione della funzione di lettura di un albero da file;
questo adesso è un po' prematuro (bisogna pensare a delineare la main() che useremo … ma può essere utile per fare mente locale sul processo di costruzione dell'albero che useremo.

Suggerimento 3:

per costruire l'albero si usa la chiamata alla funzione

```
TipoAlbero leggiAlberoDaFile(char *nomeFile);
```

questa funzione deve

- aprire il file di nome `nomeFile` e associarlo ad una opportuna variabile FILE *
- leggere (per consumarla dall'input) la `'('` di inizio albero
- leggere un altro carattere
    - se questo carattere è `')'`abbiamo incontrato i caratteri `"()"` di un albero vuoto e quindi viene restituito NULL
    - altrimenti (e se la forma parentetica è corretta, cosa che assumiamo sia vera) abbiamo un vero albero, con l'informazione da assegnare alla radice pronta per essere letta dal file, per cui dobbiamo
        - allocare un nodo
        - leggere e assegnare al campo info l'informazione dal file
        - assegnare al campo sin il sottoalbero sinistro (che è pronto per essere letto dal file in questo momento)
        - assegnare al campo `des` il sottoalbero destro (che sarà pronto per essere letto dal file appena finita la lettura del sinistro)
        - leggere (per consumarla dall'input) la ')' che termina questo albero

******* se si vuole si può fare un abbozzo di questa funzione, da raffinare quando avremo stabilito la forma della funzione main()

******* I prossimi due suggerimenti riguardano:

- la dichiarazione delle funzioni da usare nella `main()`
- la `main()`

Suggerimento 4

```
/* prototipi delle funzioni usate */

void leggiElemAlberoDaFile(FILE *fin, TipoElemAlbero *pelem);
void stampaElemAlbero(TipoElemAlbero elem);
TipoAlbero leggiAlberoDaFile(FILE *fin);
TipoAlbero leggiSottoAlberoDaFile(FILE *fin);
void stampaAlbero(TipoAlbero alb);
void deallocaAlbero(TipoAlbero *pAlb);
```

Suggerimento 5:

questa potrebbe essere la funzione main() ...

```
int main() {
 TipoAlbero albero;
 char nomeFile[20];   /* per il nome del file */

printf("\n - nome del file con la rappr. par.: ");
 scanf("%s", nomeFile);

 albero = leggiAlberoDaFile(nomeFile);

 printf("\n - ecco l'albero: ");
 stampaAlbero(albero);

 printf("\n - adesso dealloco l'albero: ");
 deallocaAlbero(&albero);

printf("\nFINE\n");
return 0;
}
```

******* A questo punto bisogna affrontare e risolvere la funzione di lettura albero da file
I prossimi 4 suggerimenti riguardano in particolare

- la lettura di un oggetto di tipo TipoElemALBERO, eseguita nella funzione leggiAlberoDaFile();
  questa lettura può essere fatta "ad hoc" con una scanf, oppure attraverso l'uso di una funzione apposita leggiElemAlberoDaFile; quest'ultima soluzione permette di scrivere la funzione di lettura dell'albero in modo più generale e riutilizzabile (infatti se dovessimo gestire alberi di altre cose che interi, dovremmo cambiare la funzione leggiElem (caratteristica di TipoElem) e non quella di lettura dell'albero

- la stesura della funzione leggiAlberoDaFile()

- la gestione della funzione leggiSottoAlberoDaFile(), importante per l'algoritmo applicato in leggiAlberoDaFile().

Suggerimento 6:

Per leggere un'informazione di tipo `TipoElemAlbero`, invece di usare `scanf("%d"...)`, che andrebbe bene solo per alberi di interi, usiamo una chiamata alla funzione

```
leggiElemAlberoDaFile(FILE * f, TipoElemAlbero *el).
```

In questo modo, quando dovremo usare la funzione di costruzione dell'albero per alberi con elementi diversi (ad esempio, caratteri), la funzione di costruzione rimarra' intatta mentre dovremo aggiornare solo la funzione

`leggiElemAlberoDaFile()`

(in altre parole, non sarà necessario mettere le mani nella funzione di costruzione, che è più complicata, mentre basterà ritoccare una funzione meno complicata).

******* Su come è fatta questa funzione ritorneremo in un suggerimento verso la fine della sezione; adesso torniamo alle funzioni più complesse …

Suggerimento 7:

provare a completare questo (corposo) brano della funzione `leggiAlberoDaFile()`

```
TipoAlbero leggiAlberoDaFile(char *nomeDelFile){
  TipoAlbero res;
  TipoElemAlbero el;
  char ccc;
  FILE *f;

  f = fopen(nomeDelFile, "r");     /* apertura file */

  fscanf(f, "%c", &ccc);           /* legge la parentesi aperta */
  fscanf(f, "%c", &ccc);           /* legge carattere successivo */

  if (ccc == ')')
    return NULL;              /* ha letto (): albero vuoto */
  else {
    /* ccc non era ')', quindi c'e' da leggere l'informazione della radice */
        leggiElemAlberoDaFile(f, &el);
    /* e poi creare il nodo radice e avviare la costruzione
        dei due sottoalberi (prima sinistro e poi destro) */
  ...
```

Suggerimento segue

Suggerimento 8:

Nella funzione `leggiAlberoDaFile()`, per costruire un sottoalbero, si usa la chiamata alla funzione

```
TipoAlbero leggiSottoAlberoDaFile(FILE *fin);
```

questa funzione

- presume che il file fin metta a disposizione i dati relativi ad un (sotto)albero
- (inoltre presume che il cursore di lettura del file sia posizionato proprio sulla parentesi iniziale del (sotto)albero) e
- legge tali dati, dalla '(' di inizio albero alla ')' di fine albero

Per far ciò, ovviamente, bisogna fare qualcosa di molto simile a quanto fatto nella funzione `leggiAlberoDaFile()`:

- leggere dal file `fin` la '(' di inizio albero
- leggere dal file `fin` un altro carattere
  - se questo carattere e' ')' abbiamo incontrato i caratteri "()" di un albero vuoto e quindi viene restituito NULL (il sottoalbero letto è vuoto!)
  - altrimenti abbiamo un vero albero, con l'informazione da assegnare alla radice pronta per essere letta dal file, per cui dobbiamo
    - allocare un nodo
    - leggere dal file `fin` e assegnare al campo info l'informazione dal file
    - costruire il sottoalbero sinistro (sul cui inizio è posizionato il cursore di lettura del file) e assegnarlo al campo sin
    - ne frattempo il cursore di input è transitato sul sottoalbero sinistro e quindi ora è posizionato sul sottoalbero destro, per cui dobbiamo costruire il sottoalbero destro e assegnarlo al campo `des`
    - leggere dal file `fin` la ')' che termina questo albero

Suggerimento segue

Suggerimento 9:

provare a completare questo (corposo) brano della funzione `leggiSottoAlberoDaFile()`

```
TipoAlbero leggiSottoAlberoDaFile(FILE * fin){
  TipoAlbero res;
  TipoElemAlbero el;
  char ccc;

  fscanf(fin, "%c", &ccc);       /* legge la parentesi aperta */
  fscanf(fin, "%c", &ccc);       /* legge carattere successivo */

  if (ccc == ')')
    return NULL;              /* ha letto (): albero vuoto */
  else {
    /* ccc non era ')', quindi c'e' da leggere l'informazione della radice */
    leggiElemAlberoDaFile(fin, &el);
   /* ora si crea il nodo radice e si avvia la costruzione dei due sottoalberi
(prima sinistro e poi destro) */
...
```

***** Torniamo sulle funzioni accessorie: lettura da file e stampa di elementi dell'albero.**

Ora bisogna programmare la funzione di lettura da file di un elemento da inserire in un albero
e quella di stampa dell'elemento medesimo

nonché poi la funzione che produce la stampa dell'albero in forma parentetica (usando la funzione di stampa-elemento)

La prima funzione viene chiamata dagli algoritmi di costruzione albero: è importante che essa sia una funzione "generica" (che incapsula in sé stessa le necessità e le caratteristiche di gestione/uso particolari del TipoElem in questione);

la seconda viene usata nella stampa dell'albero l'albero in forma parentetica.

Provare a progettare e implementare tali funzioni; seguono suggerimenti in proposito.

Suggerimento 10:

`leggiElemAlberoDaFile()` e **`stampaElemAlbero()`**sono funzioni programmate ad hoc per il tipo dell'informazione contenuta nei nodi dell'albero (`TipoElemAlbero`, che in questo esercizio è `int`).

La prima riceve una variabile **`FILE*`** corrispondente ad un file aperto in lettura e l'indirizzo di una variabile di tipo **`TipoElemAlbero`**, che riempie con la lettura fatta dal file;

la seconda riceve un parametro di tipo **`TipoElemAlbero`** e lo stampa.

Se l'albero è di interi, si tratta di volgari `scanf` e `printf` con formato di conversione **"%d"**!

***** ora si tratta di completare la stampa di un sottoalbero, in forma parentetica: qual e` l'algoritmo?**
Provare a scriverlo e poi guardare il suggerimento 11

Suggerimento 11:

La stampa di un (sotto)albero, resa in forma parentetica prevede la

- stampa della ‘(‘
- stampa dell’informazione collegata alla radice
- stampa del sottoalbero sin
- stampa del sottoalbero des
- stampa della ‘)’ di fine (sotto)albero;

Però, se il (sotto)albero è vuoto bisogna stampare solo “( )”.

12.2. Costruzione di un albero di caratteri (directory ALBERI2)

Scrivere un programma che

- riceve il nome di un file contenente la rappresentazione parentetica di un albero binario di CARATTERI;
- ne costruisce l'albero binario in memoria centrale, mediante `struct` e puntatori;
- stampa l'albero tre volte, limitando le stampe alle sole informazioni contenute nei nodi (niente parentesi):
  - la prima volta attuando una *strategia di **visita*** **in preordine**;
  - la seconda volta attuando una *strategia di **visita*** **in postordine**,
  - la terza volta attuando una *strategia di **visita*** **simmetrica**;
- poi dealloca l'albero e saluta.

Nei file `ACHAR1.TXT`, `ACHAR2.TXT` ci sono le rappresentazioni parentetiche di tre alberi. Provare con queste il programma.

Suggerimenti seguono

Suggerimento 1 di 3:

Se l'albero è il seguente
`( A( B()())( C()()))`
l'output del programma sarà (potrebbe essere)

```
- nome del file con la rappr. par.: ACHAR1.TXT

 - stampa in preordine:  A B C
 - stampa in postordine: B C A
 - stampa in simmetrica: B A C
 - adesso dealloco l'albero:
FINE
```

Se l'albero è il seguente

(D(B(A()())(C()()))(F(E()())()))
l'output del programma sarà (potrebbe essere)

```
- nome del file con la rappr. par.: ACHAR2.TXT

 -  stampa in preordine: D B A C F E
 - stampa in postordine: A C B E F D
 - stampa in simmetrica: A B C D E F
 - adesso dealloco l'albero:
FINE
```

***** ora bisogna specificare le dichiarazioni di tipi di dati …**
Provare a scrivere e poi guardare il suggerimento 2

Suggerimento 2 di 3:

Nella definizione dei tipi di dato c'è un'unica modifica da fare!

```
typedef char TipoElemAlbero;          /* albero di caratteri */
```

il resto non cambia rispetto ad ALBERI1.C (esercizio precedente).

*** ora scrivere i prototipi delle funzioni che si ritiene si dovranno usare …
E poi guardare il suggerimento 3

Suggerimento 3 di 3:

Ecco i prototipi delle funzioni usate nella proposta di soluzione di `alberi2.c`:

- queste funzioni vanno cambiate rispetto all'esercizio precedente (dato che adesso si lavora con alberi di caratteri e non più di interi):

```
void leggiElemAlberoDaFile(FILE *fin, TipoElemAlbero *pelem);
void stampaElemAlbero(TipoElemAlbero elem);
```

richiameranno al loro interno funzioni adatte ai char, invece che agli int, dato che ora **`TipoElemAlbero`** e` char

- queste altre rimangono intatte (evviva: considerando lo sforzo fatto per produrle, è un'ottima cosa il non doverci rimettere mano);
  i cambiamenti dovuti al cambio di tipo di informazione nei nodi si riflettono solo sulle funzioni "ad hoc" di lettura e stampa.

```
TipoAlbero leggiAlberoDaFile(char *nomeFile);
TipoAlbero leggiSottoAlberoDaFile(FILE *fin);
void deallocaAlbero(TipoAlbero *pAlb);
```

- queste sono poi le funzioni da definite ex novo, relative alla stampa secondo le varie strategie

```
void stampaAlberoPreordine(TipoAlbero alb);
void stampaAlberoPostordine(TipoAlbero alb);
void stampaAlberoSimmetrica(TipoAlbero alb);
```

12.3. Esercizio da inventare (nessuna soluzione proposta)

Ripercorrendo gli esercizi svolti durante la seconda lezione sugli alberi, sperimentare in un unico programma l'uso degli algoritmi visti a proposito di

- ricerca di elementi in un albero binario generico;
- calcolo del numero di occorrenze di un elemento;
- calcolo del massimo elemento contenuto in un albero;

Per sviluppare questa soluzione è bene usare (estendere) uno dei due programmi visti sopra … ad esempio quello tra le soluzioni distribuite in directory pubblica, **albero1.c**

Salvate una nuova versione di alberi1 e usate quella …

Forse per il terzo punto c'è bisogno di un suggerimento? Che segue …

Suggerimento 1

Intanto magari semplifichiamo le cose dicendo che gli elementi dell'albero sono interi.

Poi, bisogna decidere come utilizzare la funzione nella **`main`** …

Ad esempio con

```
printf("\n\n\n - adesso calcolo il massimo con maxAlb2:");
printf("\n ...");
printf("\n ...");
printf("\n ...");
massimo = maxAlb2(albero);
printf("\n - massimo = %d", massimo);
```

La funzione sarà

```
int maxAlb2(TipoAlbero albero)
```

E calcolerà il massimo valore presente nell'albero confrontando la radice con il massimo valore presente nel sottoalbero sinistro e con il massimo valore presente nel sottoalbero …

Poi, la struttura stessa (intrinsecamente ricorsiva) dell'albero ci suggerisce come scrivere l'algoritmo di questa funzione

Come?

Pensarci, scrivere almeno uno schema di algoritmo e poi guardare il suggerimento che segue

Suggerimento 2

Sfruttiamo la struttura dell'albero …

- se l’albero è vuoto restituiamo … ?
- se l’albero non è vuoto, allora ha
    - una radice
    - un SIN
    - un DES

  che cosa restituiamo in questo caso?

  Il massimo sarà o nel SIN (il massimo del sottoalbero SIN) ,
  o sarà nel DES (il massimo del sottoalbero DES),
  oppure sarà la radice

  ma quale sarà in effetti?

  Calcoliamo il massimo con una specie di applicazione dell’algoritmo di massimo parziale …

Farlo … almeno provare a farlo … poi c’è il suggerimento 3

Suggerimento 3

Alg

1) se vuoto `return 0`

2) nel caso “non vuoto” … per cui eseguiamo tre passi di uso di una variabile maxParz … inizializzandola con l’unica informazione che abbiamo direttamente disponile – cioe` la radice) e calcolando i massimi di SIN e DES
   a. maxParz = radice
   b. maxSin = massimo del sottoalbero sinistro … come calcolarlo?
   c. maxDes = massimo del sottoalbero sinistro … come calcolarlo?

3) se maxParz perde con maxSin, allora maxParz = maxSin;
   e poi, se maxParz perde con maxDes allora maxParz = …

0) servono maxSin, maxDes, maxParz; alb e’ l’albero (parametro formale)

Scrivere l’intestazione della funzione, le variabili locali e il passo 1; confrontare con il suggerimento che segue

Suggerimento 4

```
int maxAlb2(TipoAlbero albero) {

  int maxParz, maxSin, maxDes;

  if (albero == NULL) return 0;

…

return maxParz;
}
```

Ora aggiungere il passo 2 e poi vedere il suggerimento successivo

Suggerimento 5

```
int maxAlb2(TipoAlbero albero) {
  int maxParz, maxSin, maxDes;

  if (albero == NULL) return 0;

  maxParz = albero->info;
  maxSin = maxAlb2(albero->sin);
  maxDes = maxAlb2(albero->des);

…
return maxParz;
}
```

Ora completare il codice e poi vedere il suggerimento seguente

Suggerimento 6

```
    if (maxParz < maxSin) maxParz=maxSin;

    if (maxParz < maxDes) maxParz=maxDes;

  return maxParz;
  }
```

Qualche suggerimento per visualizzare l'esecuzione della funzione (e debuggarla in caso …) segue

Suggerimento 7

Provare ad usare le printf suggerite qui sotto, per visualizzare come viene eseguito il compito di trovare il massimo nell'albero.

```
int maxAlb2(TipoAlbero albero) {
  int maxParz, maxSin, maxDes;

  if (albero == NULL) return 0;

/* per debugging ... */
printf("\n - analizzo sottoalbero %d", albero->info);
/**/

  maxParz = albero->info;
  maxSin = maxAlb2(albero->sin);
  maxDes = maxAlb2(albero->des);

  if (maxParz < maxSin) maxParz=maxSin;

  if (maxParz < maxDes) maxParz=maxDes;

/* per debugging ... */
printf("\n ---- analizzato sottoalbero %d", albero->info);
/**/

return maxParz;
}
```

12.4. maxAlb1() con effetto collaterale

Scrivere la funzione maxAlb() che, ricevendo un parametro di output mx ed un albero di interi, assegna al parametro di output il valore massimo tra quelli presenti nell'albero.

maxAlb1() è una funzione in cui il parametro di output è il "massimo parziale";
questo massimo parziale viene dato alla funzione insiele al sottoalbero da visitare alla ricerca di ulteriori massimi …

Se l'albero è vuoto, non c'è nulla da fare; altrimenti, si fa combattere la radice con il parametro di output (modificandolo se questo è più piccolo della radice) e poi si chiama ancora maxAlb1, con il medesimo parametro di output e con una volta il sottoalbero SIN e una il DES.

Provare a scrivere l'algoritmo …

Suggerimento 1

0) parametri: **`mx`** (indirizzo della locazione in cui viene gestito il massimo parziale), e **`alb`**

1) con alb = NULL non si deve fare niente, quindi qui si controlla solo se alb <> NULL
   se lo è …
   1.1) se radice > massimo parziale, allora si modifica il massimo parziale
   1.2) si chiama maxAlb con parametri: il massimo parziale (parametro di output) e il sottoalbero SIN
   1.3) si chiama maxAlb con parametri: il massimo parziale (parametro di output) e il sottoalbero DES

Provare l'algoritmo su un albero di esempio preso da quelli dati in questa esercitazione

Disegnare una sola locazione per **`mx`**

è sempre questa che viene passata alle chiamate di `maxAlb()`

e annotare la locazione di mx mentre si esegue l'algoritmo, cioè metterci dentro i valori che via via assume.

Poi scrivere l'intestazione della funzione `maxAlb( …)`

Suggerimento 2

```
maxAlb(int *mx, TipoAlbero albero)
```

Qual è il tipo del risultato?

Suggerimento 3

```
void maxAlb1(int *mx, TipoAlbero albero)
```

E adesso scrivere il codice della funzione

Suggerimento 4

```
void maxAlb1(int *mx, TipoAlbero albero)
{
  if (albero != NULL) {
    if (albero->info > *mx)

…

return;
}
```

Completare il codice

Suggerimento 5

```
void maxAlb1(int *mx, TipoAlbero albero)
{
  if (albero != NULL) {
    if (albero->info > *mx)
       *mx = albero->info;
   maxAlb1(mx, albero->sin);
   maxAlb1(mx, albero->des);
printf("\n - analizzato sottoalbero %d", albero->info);
  }
return;
}
```

Un problema interessante è “come chiamare questa funzione nella main” …

Scrivere il codice corrispondente, immaginando di inserirlo in una nuova versione del programma **alberi1**

Suggerimento 6

Assumiamo che nella main ci sia una variabile intera
`massimo`
Questo è evidentemente il parametro di output in cui vogliamo che alla fine sia contenuto il massimo dell'albero …

Il problema interessante è nel fatto che per chiamare la funzione dovremo passare l'indirizzo di `massimo`, e questo dovrà essere inizializzato (per permettere di confrontarlo nel codice della funzione.

Come si inizializza il massimo parziale, di solito?
Assegnandogli il primo valore della sequenza di valori da analizzare …

Solo dopo si potrà effettuare una chiamata come, ad esempio,
```
maxAlb1(&massimo, albero->sin);
```

Lo so, è complicato …

Idealmente, ora provate a scrivere il codice (l'istruzione) per inizializzare `massimo` …

E nel caso sbirciate il suggerimento successivo

Suggerimento 7

```
massimo = albero->info;
```

e poi?

Quale codice inserire dopo di questo, nella `main()`?

Suggerimento 8

Ricordare che siamo in alberi1.c … La variabile albero è quella della `main`(), corrispondente all'albero costruito con lettura da un file di input.

```
massimo = albero->info;
maxAlb1(&massimo, albero->sin);
maxAlb1(&massimo, albero->des);
```

Va bene … sperimentate il tutto

Quando ci saranno problemi (per il codice che inserite in alberi1.c) guardate il prossimo suggerimento ☺

Suggerimento 9

```
if (albero) {
   printf("\n\n\n - adesso calcolo il massimo con maxAlb:");
   printf("\n ...");
   printf("\n ...");
   printf("\n ...");
   massimo = albero->info;
   maxAlb1(&massimo, albero->sin);
   maxAlb1(&massimo, albero->des);
   printf("\n - massimo = %d", massimo);
 }
```

Ovviamente l'albero deve essere non vuoto: se è vuoto non c'è nulla da calcolare …

12.5. Alberi di ricerca (nessuna soluzione proposta)

Scrivere un programma capace di costruire un albero di interi a partire dalla sua rappresentazione parentetica, verificando **poi** che si tratti di un albero di ricerca e permettendo successivamente di eseguire una serie di ricerche di elementi nell'albero, fatte usando l'algoritmo piu' conveniente.

La verifica che l'albero sia di ricerca viene effettuata applicando la definizione:

1) sappiamo che se il massimo degli elementi del sottoalbero sinistro è maggiore della radice, l'albero NON è di ricerca;
2) sappiamo anche che se il minimo degli elementi del sottoalbero destro è minore della radice, l'albero NON è di ricerca (infatti in tal caso non è vero che tutti gli elementi del sottoalbero destro sono più grandi della radice);
3) e abbiamo riflettuto sul fatto che se il massimo del sottoalbero destro e' piu' piccolo della radice e il minimo del sottoalbero sinistro è più grande della radice, allora l'albero complessivo "potrebbe" essere di ricerca … a patto che i sottoalberi sinistro e destro lo siano.

Quindi la verifica deve procedere ricorsivamente su tutti i possibili sottoalberi.

Algoritmo per diRicerca(ALBERO) segue

diRicerca(ALBERO)

0) consideriamo le due parti (sinistra e destra) dell'albero e usiamo due variabili logiche okSin e okDes … ad esempio dato ALBERO (con RADICE, SIN, DES)
   a. okSin vale 1 se la parte sinistra di ALBERO segue le regole che le competono (massimo del sottoalbero sinistro piu' piccolo di RADICE, e sottoalbero sinistro e' di ricerca).
   b. okDes vale 1 se la parte destra di ALBERO segue le regole che le competono (minimo di DES piu' grande di RADICE, e DES e' di ricerca.

1) okSin = okDes = 0 (non possiamo dire che le due parti siano ok)

2) Se ALBERO e' vuoto → **sì, è di ricerca**

3) Se ALBERO e' non vuoto → ha RAD, SIN, DES

   a. Se SIN e' vuoto la parte sinistra non puo' violare le regole: okSin = 1
      else
      okSin = 1 se … vedi punto 0).a

   b. Se DES e' vuoto come sopra … okDes = 1
      else
      okDes = 1 se … vedi punto 0).b

4) L'albero e' di ricerca se sia okSin che okDes sono uguali a 1.

Per questa funzione non proponiamo una soluzione programmativa …

Si presume che un sottoalbero vuoto sia di ricerca (non viola nessuna delle regole stabilite per essere di ricerca).

Quindi un albero singleton è di ricerca anche lui …

Sottoproblemi significativi sono quelli corrispondenti al calcolo del massimo e minimo per un sottoalbero.

Una volta scritta la funzione diRicerca(), realizzare le funzioni maxAlb() e minAlb(), che dato un albero restituiscono, rispettivamente, il massimo e il minimo dei valori contenuti nell'albero.

Un suggerimento su maxAlb() segue (qui sviluppiamo una soluzione lievemente diversa da quelle precedenti maxAlb1 e maxAlb2)

Suggerimento
Abbiamo visto che un potenziale problema è nel calcolo del massimo (minimo) di un albero: si può procedere, a tale riguardo, in modi diversi;

comunque ecco un modo per calcolare il massimo tra i valori di un albero (supponiamo che le informazioni nei nodi siano tali da poter usare l'operatore '>')

Algoritmo per MAXALB (ALBERO): calcolo del massimo in un albero non vuoto usando ancora una volta il vecchio e collaudato algoritmo con il massimo parziale.

Provare a scrivere l'algoritmo che calcola il massimo in un albero non vuoto, appoggiandosi su una variabile che rappresenta il massimo parziale …

Suggerimento
Tenere presente che in un albero non vuoto la struttura mette a disposizione un elemento (RADICE) e due sottoalberi.
La RADICE e' il massimo parziale, inizialmente … poi bisogna vedere se e come cambiare questo massimo parziale…

Suggerimento sull'algoritmo completo segue
maxAlb()

0) serve maxParz … lo chiamiamo maxPerAlbero (maxPerA), e m per contenere i valori di massimo calcolati sui sottoalberi
1) inizialmente il massimo parziale è la radice di ALBERO (ricordiamoci che partiamo dall'assunto che ALBERO sia non vuoto, sennò un massimo non può essere calcolato in generale)
    a. maxParzA = radice
2) se SIN e non vuoto,
    a. se MAX(SIN) > maxPerA allora riassegnare maxPerA
3) se DES e non vuoto,
    a. se MAX(DES) > maxPerA allora riassegnare maxPerA
4) a questo punto maxPerA e' stato assegnato (almeno all'inizio, alla radice, ed eventualmente anche dopo se almeno uno dei sottoalberi di ALBERO era non vuoto ed è risultato avere un massimo maggiore di maxPerA).
    a. quindi maxPerA contiene un valore, che restituiamo.

Prima di buttarsi sul codice …
provare questo algoritmo su alberi diversi (ad esempio sugli alberi usati durante le lezioni

il codice suggerito segue …

```
int maxAlb (TipoAlb a) {
   int m,
       maxPerA = a->info;

   if ( a->sin ) {
      m = maxAlb( a->sin );
      if( m > maxPerA)
         maxPerA = m;
   }
   if ( a->des ) {
        m = maxAlb( a->des );
        if ( m > maxPerA )
             maxPerA = m;
   }
return maxPerA;
}
```

Per completare la funzione diRicerca() manca solo la definizione di minAlb()
Che ☺ è lasciata per esercizio.