Lezione 21
20 dicembre 1999

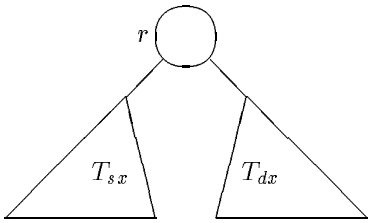
Argomenti trattati
- Alberi binari.
- Alberi di ricerca.

21.1 Alberi binari

Insieme alle liste, altre importanti strutture dati dinamiche sono gli *alberi binari*. Una struttura ad albero può essere definita ricorsivamente come:

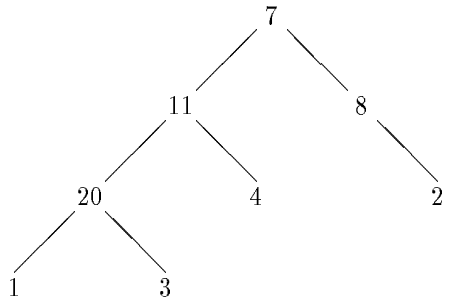
- la struttura vuota, oppure
- un nodo, detto *radice*, piú due strutture ad albero dette, rispettivamente, *sottoalbero sinistro* e *sottoalbero destro*.

In base alla definizione, un albero binario non vuoto T ha la struttura rappresentata nella seguente figura, dove r è il nodo *radice*, T_{sx} e T_{dx} sono, rispettivamente, il *sottoalbero sinistro* e il *sottoalbero destro*:



Se gli alberi T_{sx} e T_{dx} sono a loro volta non vuoti, le loro radici sono dette *figli* del nodo r.

La seguente figura mostra un albero contenente numeri interi:



Nell'albero in figura, i figli del nodo contenente il valore 11, sono i nodi contenenti 20 e 4. Il nodo contente 11 è anche detto *padre* di questi due nodi.

Una *foglia* è un nodo privo di figli. Le foglie dell'albero in figura sono i nodi contenenti i valori 1, 2, 3 e 4.

Il *livello* di un nodo di un albero T è definito ricorsivamente come segue:

- il livello della *radice* di T è 1;
- il livello dei figli di un nodo di livello i è $i + 1$.

La *profondità* di un albero è il massimo livello dei suoi nodi.

Nell'albero rappresentato nella figura precedente, il livello del nodo contenente 20 è 3, la profondità dell'albero è 4. I nodi di livello 2 sono i nodi contenenti i valori 8 e 11.

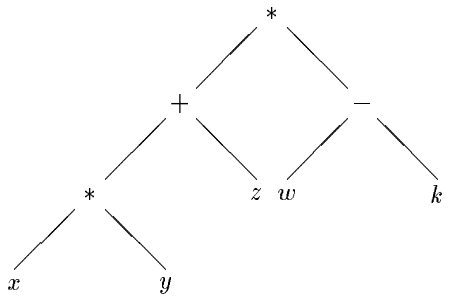
Le strutture ad albero hanno svariate applicazioni.

Ad esempio, con una struttura ad albero è possibile rappresentare organizzazioni gerarchiche, come l'indice di un libro. Un libro infatti è diviso in capitoli, ciascuno dei quali è diviso in paragrafi, che a loro volta possono essere suddivisi in sottoparagrafi. Possiamo rappresentare ciascuna di queste entità con un nodo. La radice dell'albero rappresenta il capitolo 1. Dato un nodo, rappresentiamo alla sua destra i nodi successivi dello stesso livello, e alla sua sinistra i nodi di livello inferiore. Pertanto, il figlio destro della radice corrisponderà al capitolo 2, il figlio sinistro della radice al paragrafo 1.1, il figlio destro del figlio sinistro della radice al paragrafo 1.2, e così via.

Le espressioni aritmetiche hanno una naturale rappresentazione come alberi binari. Ad esempio, l'espressione

$$(x * y + z) * (w - k)$$

può essere rappresentata dal seguente albero:



Attraversamento di alberi binari

Nel caso delle liste, abbiamo esaminato una semplice strategia di scansione che consiste nella visita di tutti i nodi di una lista dal primo all'ultimo. Per gli alberi si possono fornire differenti strategie per visitare tutti i nodi o, in altre parole, per *attraversare* un intero albero. Tali strategie possono essere esplicitate a partire dalla definizione ricorsiva di albero.

In particolare, le tre strategie fondamentali per attraversare tutti i nodi di un albero T con radice r, sottoalbero sinistro T_{sx} e sottoalbero destro T_{dx} sono:

Attraversamento in ordine anticipato o *preordine*.
Si visita prima la radice r, poi il sottalbero sinistro T_{sx} e infine il sottoalbero destro T_{dx}.

Attraversamento in ordine simmetrico o *inordine*.
Si visita prima sottalbero sinistro T_{sx}, poi la radice r e infine il sottoalbero destro T_{dx}.

Attraversamento in ordine posticipato o *postordine*.
Si visita prima il sottalbero sinistro T_{sx}, poi il sottoalbero destro T_{dx} e infine la radice r.

Ad esempio, visitando l'albero rappresentato nell'ultima figura nei tre ordini, si ottiene:

Ordine anticipato $* \; + \; * \; x \; y \; z \; - \; w \; k$

Ordine simmetrico $x \; * \; y \; + \; z \; * \; w \; - \; k$

Ordine posticipato $x \; y \; * \; z \; + \; w \; k \; - \; *$

Si noti che nella sequenza ottenuta leggendo l'espressione in ordine simmetrico (notazione *infissa)*, vi è perdita d'informazione, in quanto mancano le parentesi.

Al contrario, le sequenze ottenute mediante le visite in ordine anticipato e in ordine posticipato, corrispondono rispettivamente alla notazione *prefissa* e alla notazione *postfissa*, mediante le quali è possibile rappresentare un'espressione aritmetica *senza* bisogno di utilizzare le parentesi.

Ad esempio, un'espressione in forma postfissa può essere calcolata leggendola da sinistra a destra e sostituendo, man mano, ogni simbolo di operazione preceduto immediatamente da due valori, con il risultato.

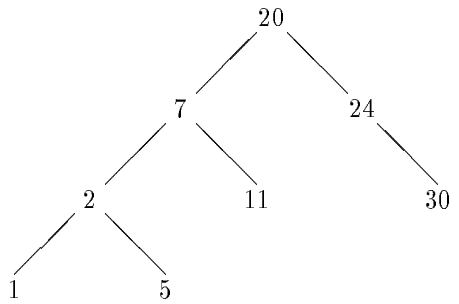
Se ad esempio x, y, z, k e w valgono rispettivamente 1, 2, 3, 4 e 5, l'espressione precedente può essere valutata nei seguenti passi:

- inizialmente l'espressione è $1 \; 2 \; * \; 3 \; + \; 5 \; 4 \; - \; *$;
- la parte $1 \; 2 \; *$ può essere valutata ottenendo $2 \; 3 \; + \; 5 \; 4 \; - \; *$;
- la parte $2 \; 3 \; +$ può essere valutata ottenendo $5 \; 5 \; 4 \; - \; *$;
- la parte $5 \; 4 \; -$ può essere valutata ottenendo $5 \; 1 \; *$;
- la parte $5 \; 1 \; *$ può essere valutata ottenendo il risultato, cioè 5.

21.2 Alberi di ricerca

Gli alberi binari sono spesso utilizzati per rappresentare insiemi di dati ordinati in base ad una chiave. In questo caso, l'albero viene costruito in modo che, per ogni suo nodo x, tutti i valori contenuti nel sottoalbero sinistro di x siano minori del valore contenuto in x, mentre tutti i valori contenuti nel sottoalbero destro di x siano maggiori (o uguali, nel caso si permettano valori ripetuti) del valore contenuto in x. Un albero che soddisfi questi vincoli viene detto *albero di ricerca*.

La seguente figura rappresenta un albero di ricerca contenente numeri interi:



Si osservi che, visitando un albero di ricerca in ordine simmetrico, si attraversano nell'ordine:

- i nodi del sottoalbero sinistro, cioè tutti i nodi i cui valori sono minori del valore della radice;
- la radice;

- i nodi del sottoalbero destro, cioè tutti i nodi i cui valori sono maggiori del valore della radice.

Pertanto, la visita in ordine simmetrico dei nodi di un albero di ricerca, produce un elenco ordinato dei valori memorizzati nell'albero.

I valori incontrati visitando in ordine simmetrico l'albero nella figura precedente, sono:

1. i valori del sottoalbero sinistro della radice;
2. il valore della radice, cioè 20;
3. i valori del sottoalbero destro della radice.

Applicando ricorsivamente ai sottoalberi lo stesso metodo di visita, si ottiene la sequenza dei valori 1 2 5 7 11 20 24 30, cioè la sequenza ordinata dei valori memorizzati nell'albero.

Supponendo di avere definito un tipo di nome `tipoalbero` per la rappresentazione degli alberi binari, la procedura di scrittura in `output` dei valori contenuti in un albero di ricerca, in ordine crescente, può essere scritta ad alto livello, basandosi sulla visita in ordine simmetrico, come segue:

```
PROCEDURE scrivi (a: tipoalbero);

BEGIN {scrivi}
   IF a <> NIL THEN
      BEGIN
         scrivi(sottoalbero sinistro di a);
         writeln(valore contenuto nella radice di a);
         scrivi(sottoalbero destro di a)
      END
END; {scrivi}
```

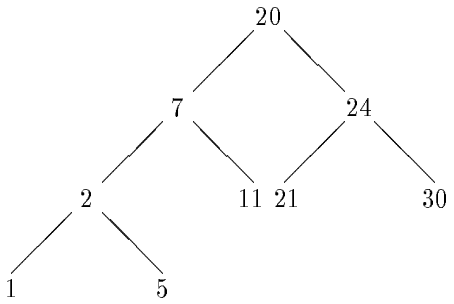
Dato un albero di ricerca, per inserire in esso un nuovo elemento usiamo il seguente metodo, basato sulla definizione ricorsiva di albero:

- se l'albero è vuoto, inseriamo l'elemento immediatamente;
- se l'albero è costituito da una radice e da due sottoalberi, confrontiamo il valore da inserire con il valore contenuto nella radice:
  - se il valore da inserire è minore del valore della radice, effettuiamo l'inserimento ricorsivamente a sinistra
  - altrimenti effettuiamo l'inserimento a destra.

Ad esempio, volendo inserire il valore 21 nell'albero raffigurato in precedenza, effettuiamo i seguenti passi, a partire dalla radice:

- confrontiamo il valore della radice, cioè 20, con il valore da inserire; poiché questo è maggiore procediamo ricorsivamente sul sottoalbero destro;
- confrontiamo il valore della radice del sottoalbero considerato, cioè 24, con il valore da inserire; poiché quest'ultimo è minore, procediamo ricorsivamente sul sottoalbero sinistro del nodo considerato (cioè il sottoalbero sinistro del nodo contenente 24);
- il sottoalbero considerato ora è vuoto; procediamo dunque all'inserimento.

Effettuato l'inserimento, l'albero diventa:



Supponendo di avere definito un tipo di nome `tipoalbero` mediante il quale rappresentare gli alberi binari, possiamo scrivere, ad alto livello, la seguente procedura di inserimento in un albero `a` di un valore `x`, basata sulla strategia ricorsiva descritta sopra:

```
PROCEDURE inserisci (VAR a: tipoalbero; x: integer);

BEGIN {inserisci}
   IF a e' vuoto THEN
      BEGIN
         crea un nuovo nodo
         inserisci nel nodo il valore x
      END
   ELSE IF x e' minore del valore contenuto nella radice di a THEN
      inserisci(sottoalbero sinistro di a, x)
   ELSE
      inserisci(sottoalbero destro di a, x)
END; {inserisci}
```

Presentiamo ora un'implementazione in Pascal degli alberi di ricerca. In particolare sviluppiamo un programma che riceve da `input` una sequenza di numeri interi, terminata da 0, e la riscrive in `output` ordinata.

Il programma sarà costituito da una fase di lettura e da una di scrittura.

Nella fase di lettura, i dati man mano letti vengono memorizzati in un albero di ricerca, inizialmente vuoto. Nella fase di scrittura, attraversando l'albero in ordine simmetrico, vengono stampati i valori contenuti nei nodi.

La definizione del tipo `tipoalbero` è del tutto analoga a quella del tipo `tipolista`. In questo caso, ogni nodo dell'albero conterrà due puntatori, rispettivamente al sottoalbero sinistro e al sottoalbero destro:

```
TYPE
   tipoalbero = ^nodoalbero;
   nodoalbero = RECORD
         info: integer;
         sx, dx: tipoalbero
      END;
```

La struttura dati fondamentale del programma è una variabile `albero` di tipo `tipoalbero`:

```
VAR
   albero: tipoalbero;
```

La procedura di lettura è organizzata secondo il solito schema; dopo avere inizializzato l'albero come vuoto, assegnando `NIL` al relativo puntatore, si effettua un ciclo del tipo:

```
leggi un numero
WHILE il numero e' diverso da zero DO
   BEGIN
      inserisci il numero nell'albero
      leggi un numero
   END;
```

Il compito di effettuare l'inserimento del numero nell'albero viene demandato ad un'altra procedura, sviluppata piú avanti, con la seguente intestazione:

```
PROCEDURE inserisci (VAR a: tipoalbero; x: integer);
```

La procedura di lettura può essere dunque scritta come:

```
PROCEDURE LeggiEOrdina (VAR a: tipoalbero);

   VAR
      x: integer;

   ...definizione della PROCEDURE inserisci...

BEGIN {LeggiEOrdina}
   a := NIL;
   readln(x);
   WHILE x <> 0 DO
      BEGIN
         inserisci(a, x);
         readln(x)
      END
END; {LeggiEOrdina}
```

La struttura della procedura `inserisci` è stata delineata precedentemente, quando abbiamo discusso dell'inserimento di un valore in un albero di ricerca. Osservando le dichiarazioni di tipo che abbiamo introdotto, è immediato passare alla seguente codifica in Pascal:

```
PROCEDURE inserisci (VAR a: tipoalbero; x: integer);

{inserisce il valore di x nell'albero di ricerca puntato da a}
{versione ricorsiva}

BEGIN {inserisci}
   IF a = NIL THEN
      BEGIN
         new(a);
         a^.info := x;
         a^.sx := NIL;
         a^.dx := NIL
      END
   ELSE IF x < a^.info THEN
      inserisci(a^.sx, x)
   ELSE
      inserisci(a^.dx, x)
END; {inserisci}
```

Anche per la procedura di scrittura è immediato passare alla codifica, rifacendoci allo schema presentato sopra:

```
PROCEDURE scrivi (a: tipoalbero);

{scrive in output il contenuto dell'albero di ricerca puntato dal parametro a}

BEGIN {scrivi}
   IF a <> NIL THEN
      BEGIN
         scrivi(a^.sx);
         writeln(a^.info);
         scrivi(a^.dx)
      END
END; {scrivi}
```

Segue il listato completo del programma:

```
PROGRAM ordinamento (input, output);

   TYPE
      tipoalbero = ^nodoalbero;
      nodoalbero = RECORD
            info: integer;
            sx, dx: tipoalbero
         END;

   VAR
      albero: tipoalbero;


   PROCEDURE LeggiEOrdina (VAR a: tipoalbero);

   {Riceve da input una sequenza di numeri interi, terminata da 0, e la memorizza}
   {nell'albero di ricerca puntato dal parametro a}

      VAR
         x: integer;

      PROCEDURE inserisci (VAR a: tipoalbero; x: integer);

      {inserisce il valore di x nell'albero di ricerca puntato da a}
      {versione ricorsiva}

      BEGIN {inserisci}
         IF a = NIL THEN
            BEGIN
               new(a);
               a^.info := x;
               a^.sx := NIL;
               a^.dx := NIL
            END
         ELSE IF x < a^.info THEN
            inserisci(a^.sx, x)
         ELSE
            inserisci(a^.dx, x)
```

```
      END; {inserisci}

   BEGIN {LeggiEOrdina}
      a := NIL;
      readln(x);
      WHILE x <> 0 DO
         BEGIN
            inserisci(a, x);
            readln(x)
         END
   END; {LeggiEOrdina}


   PROCEDURE scrivi (a: tipoalbero);

   {scrive in output il contenuto dell'albero di ricerca puntato dal parametro a}

   BEGIN {scrivi}
      IF a <> NIL THEN
         BEGIN
            scrivi(a^.sx);
            writeln(a^.info);
            scrivi(a^.dx)
         END
   END; {scrivi}


BEGIN {ordinamento}
   writeln('Inserire una sequenza di numeri interi (0 per concludere)');
   LeggiEOrdina(albero);
   writeln('La sequenza ordinata e'' ');
   scrivi(albero)
END. {ordinamento}
```

Esercizi

Nei programmi presentati negli esercizi seguenti si fa riferimento alle seguenti dichiarazioni di tipo:

```
TYPE
   tipoalbero = ^nodoalbero;
   nodoalbero = RECORD
                  info: integer;     {informazione contenuta nel nodo}
                  sx, dx: tipoalbero {puntatori ai sottoalberi sinistro e destro}
                END;
```

1. Scrivere due procedure per la stampa in `output` del contenuto di un albero binario di tipo `tipoalbero`, utilizzando la visita in ordine anticipato e la visita in ordine posticipato.

2. Si consideri la seguente espressione aritmetica:

   $$3 + 5 * (4 - 2) + 11 * 2 + (1 * 6)$$

   Disegnare l'albero binario che la rappresenta. Riscrivere poi l'espressione in notazione prefissa e in notazione postfissa.

3. Disegnare l'albero di ricerca ottenuto inserendo, uno dopo l'altro in un albero inizialmente vuoto, i numeri `15 12 11 16 10 3 99 87 14 13`.

   Si scrivano gli output prodotti visitando tale albero nei tre ordini anticipato, simmetrico e posticipato.

   Si scriva infine l'output prodotto dalle seguenti procedure, nel caso ricevano come parametro un puntatore a tale albero.

   - ```
     PROCEDURE p1 (t: tipoalbero);
     BEGIN
        IF t <> NIL THEN
           IF t^.info MOD 2 = 0 THEN
              BEGIN
                 p1(t^.sx);
                 writeln(t^.info);
                 p1(t^.dx)
              END
           ELSE
              BEGIN
                 p1(t^.dx);
                 writeln (t^.info);
                 p1(t^.sx)
              END
     END;
     ```
   - ```
     PROCEDURE p2 (t: tipoalbero);
     BEGIN
        IF t <> NIL THEN
        BEGIN
           p2(t^.dx);
           writeln(t^.info);
           p2(t^.sx)
        END
     END;
     ```

4. Scrivere una `FUNCTION` che riceva come parametri il puntatore a un albero di ricerca `a` e una variabile `x` e restituisca il puntatore a un nodo di `a` contenente il valore di `x`, se presente, oppure `NIL`.

5. Disegnare l'albero di ricerca ottenuto inserendo uno dopo l'altro in un albero inizialmente vuoto i numeri `20 11 18 16 17 3 30 4 19 25`.
   Scrivere gli output prodotti visitando tale albero nei tre ordini anticipato, simmetrico e posticipato.

6. Si dia la dichiarazione del tipo *albero binario di interi*. Si scriva una funzione ricorsiva che calcoli la somma dei valori contenuti nei nodi di un albero binario di interi.

7. Si scriva una funzione che calcoli la media dei valori contenuti nei nodi di un albero binario di interi.

8. Si scriva una funzione per il calcolo della somma dei valori contenuti nelle foglie di un albero binario di interi (si ricordi che una foglia è un nodo privo di figli).

9. Si scriva una funzione che calcoli il numero di foglie di un albero binario di interi.

10. Si scriva una funzione ricorsiva che ricevendo come parametro il puntatore ad un albero binario di interi, restituisca la profondità dell'albero.

11. Si scriva una procedura che ricevendo come parametro il puntatore alla radice di un albero binario di interi, stampi in output il numero totale di nodi, il numero di foglie, il numero di nodi con un unico figlio, il numero di nodi con due figli.

12. Scrivere una procedura che visualizzi il contenuto di un albero binario di caratteri, indicando, accanto al valore contenuto in ciascun nodo, il livello a cui si trova.

13. Scrivere una procedura che calcoli la lunghezza media dei cammini in un albero binario, cioè la media dei livelli dei nodi.

14. Si dichiari il tipo *albero ternario di interi*. Si svolgano gli esercizi precedenti per questo tipo.

15. Si scriva una procedura che trasformi un albero binario nella sua immagine speculare, cioè nell'albero ottenuto scambiando, ad ogni livello, il sottoalbero sinistro con quello destro.

16. Dimostrare che in un albero binario contenente n nodi, ci sono $n + 1$ puntatori a `NIL`. Cosa si può dire nel caso degli alberi ternari?

17. Scrivere un programma che legga un file di testo e scriva in output un elenco in ordine alfabetico di tutte le parole presenti nel file, in cui per ogni parola viene indicato il numero di volte che appare nel testo. Se ad esempio il file contiene:

```
Nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
che' la diritta via era smarrita.
Ahi quanto a dir qual era e' cosa dura
esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!
```

il programma dovrà produrre il seguente output:

```
Ahi                     1
Nel                     1
a                       1
aspra                   1
cammin                  1
che                     2
cosa                    1
del                     1
di                      1
dir                     1
diritta                 1
dura                    1
e                       3
era                     2
esta                    1
forte                   1
la                      2
mezzo                   1
mi                      1
nel                     1
nostra                  1
```

```
oscura                  1
paura                   1
pensier                 1
per                     1
qual                    1
quanto                  1
rinova                  1
ritrovai                1
selva                   2
selvaggia               1
smarrita                1
una                     1
via                     1
vita                    1
```

Si consiglia di creare un albero di ricerca in cui in ogni nodo si memorizza una parola e il relativo numero di occorrenze. Quando si incontra la parola per la prima volta, si crea il nodo con numero di occorrenze uguale a `1`, se invece la parola è già presente nell'albero, è sufficiente incrementarne il numero di occorrenze.

18. Estendere il programma precedente, in modo che, per ogni parola, oltre a scrivere in output il numero di occorrenze scriva anche i numeri delle righe in cui appare (si contino le righe progressivamente da `1`). In questo caso si può estendere l'albero costruito nell'esercizio precedente, facendo partire da ogni nodo una lista, nella quale memorizzare i numeri delle righe in cui la parola appare.

    L'output prodotto sul file di esempio dell'esercizio precedente dovrà essere:

```
Ahi             Numero occorrenze:      1
                Righe:      4
Nel             Numero occorrenze:      1
                Righe:      1
a               Numero occorrenze:      1
                Righe:      4
aspra           Numero occorrenze:      1
                Righe:      5
cammin          Numero occorrenze:      1
                Righe:      1
che             Numero occorrenze:      2
                Righe:      3     6
cosa            Numero occorrenze:      1
                Righe:      4
del             Numero occorrenze:      1
                Righe:      1
di              Numero occorrenze:      1
                Righe:      1
dir             Numero occorrenze:      1
                Righe:      4
diritta         Numero occorrenze:      1
                Righe:      3
dura            Numero occorrenze:      1
                Righe:      4
e               Numero occorrenze:      3
```

```
                Righe:     4     5     5
era             Numero occorrenze:      2
                Righe:     3     4
esta            Numero occorrenze:      1
                Righe:     5
forte           Numero occorrenze:      1
                Righe:     5
la              Numero occorrenze:      2
                Righe:     3     6
mezzo           Numero occorrenze:      1
                Righe:     1
mi              Numero occorrenze:      1
                Righe:     2
nel             Numero occorrenze:      1
                Righe:     6
nostra          Numero occorrenze:      1
                Righe:     1
oscura          Numero occorrenze:      1
                Righe:     2
paura           Numero occorrenze:      1
                Righe:     6
pensier         Numero occorrenze:      1
                Righe:     6
per             Numero occorrenze:      1
                Righe:     2
qual            Numero occorrenze:      1
                Righe:     4
quanto          Numero occorrenze:      1
                Righe:     4
rinova          Numero occorrenze:      1
                Righe:     6
ritrovai        Numero occorrenze:      1
                Righe:     2
selva           Numero occorrenze:      2
                Righe:     2     5
selvaggia       Numero occorrenze:      1
                Righe:     5
smarrita        Numero occorrenze:      1
                Righe:     3
una             Numero occorrenze:      1
                Righe:     2
via             Numero occorrenze:      1
                Righe:     3
vita            Numero occorrenze:      1
                Righe:     1
```

19. Si riscriva in forma iterativa la procedura per l'inserimento di un valore intero in un albero di ricerca.