Lezione 24
13–14 gennaio 2000

Argomenti trattati
- Esempi su alberi di ricerca e code.

24.1 Esempio: costruzione di un cross reference

Costruiamo un programma che legga un testo da input e scriva in output un elenco in ordine alfabetico di tutte le parole presenti, indicando, per ciascuna, il numero di volte che appare nel testo.

Il programma sarà costituito da due fasi principali. Nella prima fase viene letto il file di `input`, isolando le parole e memorizzandole, man mano che vengono lette, in un'opportuna struttura, insieme al rispettivo numero di occorrenze. Nella seconda fase viene prodotto l'output richiesto, visualizzando il contenuto della struttura utilizzata.

Rappresentiamo ogni parola con un `PACKED ARRAY` di 15 caratteri (il valore 15 verrà definito mediante una costante, in modo da rendere facilmente modificabile il programma). Le posizioni della parola non utilizzate verranno riempite con spazi bianchi. Eventuali parole costituite da piú di 15 simboli verranno troncate al quindicesimo simbolo. Per la rappresentazione delle parole utilizziamo dunque il seguente tipo `stringa`:

```
CONST
   lung = 15; {lunghezza massima delle parole trattate}

TYPE
   stringa = PACKED ARRAY[1..lung] OF char;
```

Dobbiamo scegliere ora la struttura dati in cui memorizzare le parole lette nel testo e i contatori associati ad esse. Su tale struttura dovremo essere in grado di effettuare le seguenti operazioni:

- *inserimento* di una nuova parola, quando viene incontrata per la prima volta;
- *ricerca* di una parola e conseguente *aggiornamento* del contatore ad essa associata;
- *visualizzazione* in ordine alfabetico delle parole e dei relativi contatori.

Confrontiamo ora alcune possibili scelte della struttura dati.

- *Pila*, realizzata mediante un array.
  L'operazione di *inserimento* è estremamente semplice; per la *ricerca* di un elemento è necessario attraversare l'intera struttura; per la *visualizzazione* è necessario riordinare il contenuto della struttura. Inoltre questa struttura è di *dimensione fissa*.

- *Array ordinato.*
  L'*inserimento* è un'operazione dispendiosa (per inserire un elemento in un array ordinato occorre spostare tutti gli elementi successivi al punto in cui va effettuato l'inserimento); la *ricerca* può essere realizzata efficientemente utilizzando l'algoritmo di ricerca dicotomica; per la *visualizzazione* è sufficiente scorrere tutti gli elementi contenuti nell'array, che è già ordinato. Anche in questo caso la *dimensione* della struttura è *fissa*.

- *Pila*, realizzata mediante una lista.
  Come nel caso della pila realizzata mediante un array, l'operazione di *inserimento* risulta semplice, mentre quella di *ricerca* può richiedere l'attraversamento dell'intera struttura; per la *visualizzazione* è necessario riordinare il contenuto della struttura, operazione che per le liste risulta piú complessa che per gli array. Essendo realizzata mediante puntatori, la *dimensione* della struttura può essere variata dinamicamente in base alle esigenze che si hanno durante l'esecuzione del programma.

- *Lista ordinata.*
  Le operazioni di *inserimento* e di *ricerca* possono richiedere l'attraversamento dell'intera struttura. In particolare, poiché l'accesso a questa struttura è sequenziale, non è possibile realizzare ricerche dicotomiche. L'operazione di *visualizzazione* è semplice in quanto la struttura è già ordinata. La *dimensione* della struttura può essere variata dinamicamente.

- *Albero di ricerca.*
  Le operazioni di *inserimento* e di *ricerca* richiedono *in media* poco tempo (se l'albero è "abbastanza" bilanciato). La *visualizzazione* viene realizzata facilmente utilizzando la visita in ordine simmetrico. Inoltre, la *dimensione* della struttura varia dinamicamente, creando i nodi man mano che servono.

In base alle precedenti osservazioni, la struttura piú idonea appare l'albero di ricerca. La scelta di una struttura dinamica ha inoltre il vantaggio di non fissare, all'interno del programma, limiti sul numero massimo di parole trattate.

Introduciamo dunque le definizioni e le dichiarazioni necessarie per definire l'albero. Osserviamo che in ogni nodo deve essere memorizzata una parola, insieme al relativo contatore, oltre che, naturalmente, i puntatori ai sottoalberi sinistro e destro.

```
TYPE
   tipoalbero = ^nodoalbero;
   nodoalbero = RECORD
         parola: stringa; {parola}
         occorr: integer; {numero di occorrenze}
         sx, dx: tipoalbero
      END;

VAR
   a: tipoalbero;
```

Le due procedure che realizzano le due fasi fondamentali del programma avranno le seguenti intestazioni:

- `PROCEDURE lettura (VAR albero:  tipoalbero)`
- `PROCEDURE scrittura (albero:  tipoalbero)`

Il codice del programma principale è semplicemente:

```
BEGIN {occorrenze}
   lettura(a);
   scrittura(a)
END. {occorrenze}
```

La procedura `scrittura` effettua una visita in ordine simmetrico all'albero, mostrando i valori dei campi `parola` e `occorr` di ciascun nodo. Il codice della procedura è riportato successivamente, insieme al testo completo del programma.

Analizziamo piú in dettaglio la costruzione della procedura di lettura. Compito di questa procedura è quello di leggere il file di `input`, individuando le parole che lo costituiscono. Ogni volta che viene individuata una parola, occorre vedere se sia già presente nell'albero. In caso affermativo, se ne incrementa il contatore, altrimenti la si inserisce con contatore uguale a uno. Per svolgere questo compito definiamo dunque una procedura di *ricerca con inserimento*, che chiameremo semplicemente `inserimento`, con la seguente intestazione:

```
PROCEDURE inserimento (VAR a: tipoalbero; p: stringa);
```

Sviluppiamo subito questa procedura e, successivamente, quella di lettura.

La procedura deve ricercare nell'albero `a` un nodo contenente la parola `p`. Se tale nodo non viene trovato allora deve essere creato un nuovo nodo in cui si inserisce `p`.

Per scrivere la procedura ci basiamo sulla definizione ricorsiva di albero. Ricordiamo che un albero è *vuoto* oppure è un nodo, detto *radice*, a cui sono associati due sottoalberi, detti *sottoalbero sinistro* e *sottoalbero destro*. Poiché l'albero viene ordinato secondo l'ordine alfabetico delle parole, tutte le parole del sottoalbero sinistro precedono alfabeticamente la parola contenuta nella radice, mentre tutte le parole del sottoalbero destro seguono alfabeticamente quella contenuta nella radice.

Se l'*albero è vuoto* non può contenere la parola `p`. Pertanto creiamo un nuovo nodo, in cui memorizziamo `p`. Se l'albero *non è vuoto* confrontiamo invece la parola contenuta nella radice con `p`. Se coincidono la ricerca termina, incrementando il contatore memorizzato nella radice (campo `occorr`), altrimenti si procede ricorsivamente sul sottoalbero sinistro, se `p` precede alfabeticamente la parola contenuta nella radice, o sul sottoalbero destro, nel caso rimanente.

La procedura è dunque strutturata come segue:

```
IF l'albero a e' vuoto THEN
   crea un nodo in cui memorizzare p con contatore uguale a 1
ELSE IF p coincide con la parola memorizzata nella radice di a THEN
   incrementane il contatore
ELSE IF p precede alfabeticamente la parola memorizzata nella radice di a THEN
   procedi ricorsivamente sul sottoalbero sinistro
ELSE
   procedi ricorsivamento sul sottoalbero destro
```

Il confronto alfabetico tra due `PACKED ARRAY` di caratteri può essere effettuato con l'operatore `<`. Segue il codice di una procedura basata sullo schema precedente:

```
PROCEDURE inserimento (VAR a: tipoalbero; p: stringa);

{ricerca la parola p nell'albero a; se e' presente ne incrementa il contatore}
{altrimenti la inserisce con contatore uguale a 1}

BEGIN {inserimento}
   IF a = NIL THEN {albero vuoto: la parola non c'e'}
      BEGIN
```

```
         new(a);
         a^.parola := p;
         a^.occorr := 1;
         a^.sx := NIL;
         a^.dx := NIL
      END
   ELSE {albero non vuoto}
      IF p = a^.parola THEN {la parola p e' memorizzata nella radice}
         a^.occorr := a^.occorr + 1
      ELSE IF p < a^.parola THEN {ricerca ricorsiva nei sottoalberi}
         inserimento(a^.sx, p)
      ELSE
         inserimento(a^.dx, p)
END; {inserimento}
```

Sviluppiano ora la procedura `lettura`. Compito di questa procedura è quello di leggere il testo presente in `input`, individuare le parole che lo costituiscono e inserirle man mano nell'albero, servendosi della procedura `inserimento` che abbiamo appena codificato.

Sono possibili diverse realizzazioni di questa procedura, molto differenti tra loro. La soluzione che presentiamo è basata sullo schema per l'esame di file di testo che abbiamo presentato nella Lezione 11. Questo schema utilizza un ciclo piú esterno, in cui un file di testo è visto come una sequenza di righe:

```
WHILE l'input non e' finito DO
BEGIN
   esamina la riga corrente
   spostati sulla riga successiva
END
```

L'esame della riga corrente viene a sua volta realizzato con un ciclo in cui viene elaborato ciascun carattere:

```
WHILE la riga non e' finita DO
BEGIN
   leggi il prossimo carattere
   elabora il carattere letto
END
```

Questo schema deve essere adattato al problema che stiamo considerando. In particolare, il nostro obiettivo è quello di individuare, all'interno della sequenza di caratteri letti, le parole.

Come convenzione decidiamo che una parola è una sequenza di lettere maiuscole e minuscole. Tutto ciò che non è una lettera viene considerato come un delimitatore di fine parola. Ad esempio, secondo questa convenzione, nella sequenza di caratteri

```
abc abba, (ab23cd)ef
```

sono contenute le parole `abc`, `abba`, `ab`, `cd` ed `ef`. Chiaramente, si può sviluppare il programma scegliendo convenzioni differenti.

Leggendo iterativamente i caratteri in `input` possiamo procedere come segue.

- Quando *viene letta una lettera*, la si memorizza all'interno di una variabile `word` di tipo `stringa`, in cui si costruisce man mano la parola. Nell'esempio precedente, leggendo la prima `a`, si andrà a memorizzarla in `word[1]`, leggendo poi la `b`, la si memorizzerà in `word[2]`, e così via sino alla fine della parola. Introduciamo anche un indice `pos`, contenente un valore nel subrange `0..lung` per indicare in che posizione di `word` finisce la parola letta. Ad esempio, dopo avere letto la lettera `b`, `pos` dovrà contenere 2.

In pratica, se `ch` è la variabile contenente il carattere appena letto, occorrerà effettuare le seguenti operazioni:

```
IF ch contiene una lettera THEN
   BEGIN
      incrementa pos
      inserisci ch nella posizione pos dell'array word
   END
```

Queste operazioni possono essere codificate come:

```
IF ch IN ['A'..'Z', 'a'..'z'] THEN
   BEGIN
      pos := pos + 1;
      word[pos] := ch
   END
```

Se la parola è troppo lunga, cioè se `pos` supera il valore della costante `lung`, si ha un errore in esecuzione. Abbiamo detto che le parole troppo lunghe vengono troncate, ignorando i caratteri dopo la lunghezza massima consentita. Pertanto è sufficiente introdurre un ulteriore controllo:

```
IF ch IN ['A'..'Z', 'a'..'z'] THEN
   IF pos < lung THEN
      BEGIN
          pos := pos + 1;
          word[pos] := ch
      END
```

- Quando il carattere letto *non è una lettera*, se il carattere precedente era una lettera si è raggiunta la fine di una parola. Ad esempio, nella sequenza di caratteri `abc abba, (ab23cd)ef`, il primo spazio indica la fine della parola `abc`, che andrà dunque inserita nell'albero, mentre il secondo spazio non indica la fine nessuna parola (infatti la parola `abba` è delimitata dalla successiva virgola).

  Pertanto, le operazioni da effettuare sono:

```
IF ch non contiene una lettera THEN
   IF il carattere precedente era una lettera THEN
      inserisci la parola letta nell'albero
```

  Prima di inserire la parola nell'albero, riempiamo le posizioni successive a `pos` dell'array `word` con degli spazi bianchi, usando il ciclo:

```
FOR pos := pos + 1 TO lung DO
   word[pos] := ' '
```

  Per l'inserimento richiamiamo la procedura precedentemente sviluppata, scrivendo

```
inserimento(albero, word)
```

Dopo l'inserimento è necessario predisporre la variabile `word` alla lettura di un'altra parola. Questo può essere fatto semplicemente riportando a 0 il valore di `pos`, indicando così che `word` contiene la parola vuota:

```
pos := 0
```

Per ricordare se il carattere precedente era una lettera usiamo una variabile `preclettera` di tipo `boolean`, inizializzata a `false`, il cui valore viene posto a `true` dopo la lettura di una lettera e viene riportato a `false` dopo l'inserimento di una parola nel testo.

In base a queste osservazioni, la fase di elaborazione del carattere letto può essere scritta come:

```
IF ch IN ['A'..'Z', 'a'..'z'] THEN {il carattere e' una lettera}
   BEGIN
      preclettera := true;
      IF pos < lung THEN
         BEGIN {aggiungi il carattere letto a word}
            pos := pos + 1;
            word[pos] := ch
         END
   END
ELSE IF preclettera THEN {e' stata raggiunta la fine di una parola}
   BEGIN
      {si riempiono le posizioni di word dopo la fine della parola di spazi}
      FOR pos := pos + 1 TO lung DO
         word[pos] := ' ';

      {si inserisce la parola nell'albero}
      inserimento(albero, word);

      {si predispone la variabile pos per la lettura di una nuova parola}
      pos := 0;
      preclettera := false
   END
```

Osserviamo che la fine di una parola può essere delimitata anche dall'*end–of–line*. Quando la riga finisce con un carattere alfabetico, bisogna dunque memorizzare l'ultima parola letta nell'albero. Per fare questo possiamo riscrivere il blocco di istruzioni che inizia con il test `IF preclettera...` alla fine dell'esame di ciascuna riga (cioè dopo il ciclo piú interno della procedura).

La procedura va inoltre completata con le opportune inizializzazioni:

- l'albero va inizializzato come vuoto;
- la variabile `pos` va inizializzata a 0, per indicare che `word` contiene la parola vuota;
- la variabile `preclettera` va inizializzata a `NIL`.

Il codice della procedura, sviluppata secondo questo schema è:

```
PROCEDURE lettura (VAR albero: tipoalbero);

{restituisce tramite il parametro un albero di ricerca contenente le parole}
{presenti in input con l'indicazione del numero di occorrenze di ciascuna}
```

```
   VAR
      ch: char;             {per leggere i caratteri da input}
      word: stringa;        {per leggere le parole da input}
      pos: 0..lung;         {indica quanti caratteri significativi contiene word}
      preclettera: boolean; {per ricordare se il carattere precedente era una lettera}


   ...PROCEDURE inserimento...


BEGIN {lettura}
   albero := NIL;
   pos := 0;
   preclettera := false;

   WHILE NOT eof DO
      BEGIN
         WHILE NOT eoln DO
            BEGIN
               read(ch);

               IF ch IN ['A'..'Z', 'a'..'z'] THEN {il carattere e' una lettera}
                  BEGIN
                     preclettera := true;
                     IF pos < lung THEN
                        BEGIN {aggiungi il carattere letto a word}
                           pos := pos + 1;
                           word[pos] := ch
                        END
                  END
               ELSE IF preclettera THEN {e' stata raggiunta la fine di una parola}
                  BEGIN
                     {si riempiono le posizioni di word dopo la fine della parola di spazi}
                     FOR pos := pos + 1 TO lung DO
                        word[pos] := ' ';

                     {si inserisce la parola nell'albero}
                     inserimento(albero, word);

                     {si predispone la variabile pos per la lettura di una nuova parola}
                     pos := 0;
                     preclettera := false
                  END
            END; {while not eoln...}

         readln;

         IF preclettera THEN {se la riga letta terminava con una parola...}
            BEGIN
               {si riempiono le posizioni di word dopo la fine della parola di spazi}
               FOR pos := pos + 1 TO lung DO
                  word[pos] := ' ';
```

```
            {si inserisce la parola nell'albero}
            inserimento(albero, word);

            {si predispone la variabile pos per la lettura di una nuova parola}
            pos := 0;
            preclettera := false
         END

   END {while not eof...}

END; {lettura}
```

Riportiamo qui di seguito il testo completo del programma:

```
PROGRAM occorrenze (input, output);

{visualizza in ordine alfabetico le parole presenti in input indicando, per ciascuna di esse,}
{il numero di occorrenze}

   CONST
      lung = 15; {lunghezza massima delle parole trattate}

   TYPE
      stringa = PACKED ARRAY[1..lung] OF char;

      tipoalbero = ^nodoalbero;
      nodoalbero = RECORD
            parola: stringa; {parola}
            occorr: integer; {numero di occorrenze}
            sx, dx: tipoalbero
         END;

   VAR
      a: tipoalbero;


   PROCEDURE lettura (VAR albero: tipoalbero);

   {restituisce tramite il parametro un albero di ricerca contenente le parole}
   {presenti in input con l'indicazione del numero di occorrenze di ciascuna}

      VAR
         ch: char;             {per leggere i caratteri da input}
         word: stringa;        {per leggere le parole da input}
         pos: 0..lung;         {indica quanti caratteri significativi contiene word}
         preclettera: boolean; {per ricordare se il carattere precedente era una lettera}


      PROCEDURE inserimento (VAR a: tipoalbero; p: stringa);

      {ricerca la parola p nell'albero a; se e' presente ne incrementa il contatore}
      {altrimenti la inserisce con contatore uguale a 1}
```

```
   BEGIN {inserimento}
      IF a = NIL THEN {albero vuoto: la parola non c'e'}
         BEGIN
            new(a);
            a^.parola := p;
            a^.occorr := 1;
            a^.sx := NIL;
            a^.dx := NIL
         END
      ELSE {albero non vuoto}
         IF p = a^.parola THEN {la parola p e' memorizzata nella radice}
            a^.occorr := a^.occorr + 1
         ELSE IF p < a^.parola THEN {ricerca ricorsiva nei sottoalberi}
            inserimento(a^.sx, p)
         ELSE
            inserimento(a^.dx, p)
   END; {inserimento}


BEGIN {lettura}
   albero := NIL;
   pos := 0;
   preclettera := false;

   WHILE NOT eof DO
      BEGIN
         WHILE NOT eoln DO
            BEGIN
               read(ch);

               IF ch IN ['A'..'Z', 'a'..'z'] THEN {il carattere e' una lettera}
                  BEGIN
                     preclettera := true;
                     IF pos < lung THEN
                        BEGIN {aggiungi il carattere letto a word}
                           pos := pos + 1;
                           word[pos] := ch
                        END
                  END
               ELSE IF preclettera THEN {e' stata raggiunta la fine di una parola}
                  BEGIN
                     {si riempiono le posizioni di word dopo la fine della parola di spazi}
                     FOR pos := pos + 1 TO lung DO
                        word[pos] := ' ';

                     {si inserisce la parola nell'albero}
                     inserimento(albero, word);

                     {si predispone la variabile pos per la lettura di una nuova parola}
                     pos := 0;
                     preclettera := false
```

```
                END
            END; {while not eoln...}

         readln;

         IF preclettera THEN {se la riga letta terminava con una parola...}
            BEGIN
               {si riempiono le posizioni di word dopo la fine della parola di spazi}
               FOR pos := pos + 1 TO lung DO
                  word[pos] := ' ';

               {si inserisce la parola nell'albero}
               inserimento(albero, word);

               {si predispone la variabile pos per la lettura di una nuova parola}
               pos := 0;
               preclettera := false
            END

      END {while not eof...}

   END; {lettura}


   PROCEDURE scrittura (albero: tipoalbero);

   {visualizza in output il contenuto di albero}

   BEGIN {scrittura}
      IF albero <> NIL THEN
         BEGIN
            scrittura(albero^.sx);
            writeln(albero^.parola, albero^.occorr);
            scrittura(albero^.dx)
         END
   END; {scrittura}


BEGIN {occorrenze}
   lettura(a);
   scrittura(a)
END. {occorrenze}
```

Vogliamo ora estendere il programma precedente in modo che, per ogni parola, vengano indicati anche i numeri delle righe in cui essa appare. Ad esempio, se una parola appare nella prima, nella quinta e nell'ottava riga del testo, in `output` dovrà essere indicato l'elenco `1 5 8`.

Per realizzare quanto richiesto, occorre estendere la struttura dati precedente. In particolare, ogni nodo dell'albero dovrà contenere, oltre alla parola e al numero di occorrenze, l'elenco dei numeri delle righe in cui la parola appare. Osserviamo che la lunghezza di questo elenco varia da parola a parola: alcune parole possono apparire una sola volta nel testo, altre possono essere presenti molte volte. Memorizzando l'elenco in una struttura statica, come un array, si rischierebbe di sprecare spazio nel caso di parole che appaiono poche volte e di non avere spazio sufficiente per

le parole molto frequenti. Utilizziamo quindi, per ciascun elenco, una struttura dinamica. Le operazioni che dobbiamo effettuare sull'elenco sono:

- inserimento di un nuovo valore nell'elenco: quando viene incontrata una parola occorre inserire nell'elenco ad essa associata il numero della riga in cui la parola è stata trovata;
- visualizzazione dell'elenco nella fase di scrittura.

Osserviamo che poiché il testo viene letto sequenzialmente a partire dalla prima riga, i numeri delle righe in cui una parola appare formano un elenco *crescente*. È opportuno che l'elenco venga stampato in questo stesso ordine.

Possiamo dunque rappresentare ciascun elenco con una lista, in cui gli inserimenti vengono effettuati *alla fine* della lista. Come abbiamo visto, in questo caso, per agevolare le operazioni di inserimento conviene far uso di due puntatori, uno al primo, l'altro all'ultimo elemento della lista. In altre parole, rappresentiamo ogni elenco con una coda.

La struttura dati è dunque costituita da un *albero* di ricerca, ordinato lessicograficamente. In ogni nodo di tale albero sono memorizzate le seguenti informazioni:

- una *parola*, nel campo `parola`;
- il numero di occorrenze nel testo di tale parola, nel campo `occorr`;
- i puntatori ai sottoalberi sinistro e destro, nei campi `sx` e `dx`;
- l'*elenco* dei numeri delle righe in cui appare la parola; tale elenco è memorizzato in una coda, i cui puntatori al primo e all'ultimo elemento sono memorizzati nel campo `elenco`.

Per realizzare questa struttura facciamo uso delle seguenti definizioni di costanti, tipi e variabili:

```
CONST
   lung = 15; {lunghezza massima delle parole trattate}

TYPE
   stringa = PACKED ARRAY[1..lung] OF char;

   tipolista = ^nodolista;
   nodolista = RECORD
         nriga: integer;
         pros: tipolista
      END;

   tipocoda = RECORD
         primo, ultimo: tipolista
      END;

   tipoalbero = ^nodoalbero;
   nodoalbero = RECORD
         parola: stringa;    {parola}
         occorr: integer;    {numero di occorrenze}
         sx, dx: tipoalbero; {puntatori ai sottoalberi}
         elenco: tipocoda    {elenco dei numeri delle righe in cui appare la parola}
      END;

VAR
   a: tipoalbero;
```

La struttura del programma principale resta immutata.

La procedura `lettura` deve ora contare le righe man man esse vengono lette. Per fare ciò si introduce una variabile `riga`, di tipo `integer`, inizializzata a 1, e incrementata prima di cominciare a leggere una nuova riga.

La procedura `inserimento` riceverà ora tre parametri: il puntatore alla radice dell'albero, la parola da inserire, il numero della riga in cui è stata trovata la parola. Pertanto la nuova intestazione sarà:

```
PROCEDURE inserimento (VAR a: tipoalbero; p: stringa; r: integer);
```

Naturalmente, le chiamate all'interno della procedura `lettura` vanno modificate, passando come terzo parametro il numero della riga in esame. Le procedura `lettura`, con queste modifiche, è riportata nel testo completo del programma.

Analizziamo in dettaglio la procedura `inserimento`. Lo schema è del tutto analogo a quello utilizzato nel programma precedente. Nel caso di albero vuoto, quando si crea il nodo in cui si memorizza la parola, occorre creare anche un nodo della coda, in cui si memorizza il numero della riga in cui la parola è stata incontrata. Se l'albero non è vuoto e la parola viene trovata nella radice, oltre a incrementare il contatore associato alla parola, occorre inserire alla fine della coda associata alla parola il numero della riga in cui la parola è stata trovata:

```
IF l'albero a e' vuoto THEN
   BEGIN
      crea un nodo in cui memorizzare p con contatore uguale a 1
      crea un elemento contenente il numero r della riga
      collega l'elemento creato al nodo dell'albero creato
   END
ELSE IF p coincide con la parola memorizzata nella radice di a THEN
   BEGIN
      incrementane il contatore
      crea un elemento contenente il numero r della riga
      inserisci l'elemento nella coda associata alla radice
   END
ELSE IF p precede alfabeticamente la parola memorizzata nella radice di a THEN
   procedi ricorsivamente sul sottoalbero sinistro
ELSE
   procedi ricorsivamento sul sottoalbero destro
```

Per eseguire le nuove operazioni è utile introdurre un puntatore ausiliario `n` di tipo `tipolista`.

Analizziamo in dettaglio i due casi che dobbiamo considerare:

- *Albero vuoto*
  La creazione del primo nodo dell'albero avviene come prima:

  ```
  new(a);
  a^.parola := p;
  a^.occorr := 1;
  a^.sx := NIL;
  a^.dx := NIL
  ```

  La creazione del primo elemento della coda avviene utilizzando il puntatore `n`:

  ```
  new(n);
  n^.nriga := r;
  n^.pros := NIL
  ```

L'elemento puntato da `n` deve essere ora inserito, come unico elemento, nella coda associata al nodo puntato da `a`. I puntatori al primo e all'ultimo elemento di tale coda si trovano nel campo `elenco` del nodo. È sufficiente quindi scrivere:

```
a^.elenco.primo := n;
a^.elenco.ultimo := n
```

- *`p` coincide con la parola memorizzata nella radice.*
  In questo caso, dopo avere incrementato il campo `occorr` mediante l'istruzione

  ```
  a^.occorr := a^.occorr + 1
  ```

  e avere creato il nuovo elemento utilizzando, come nel caso precedente, il puntatore `n`:

  ```
  new(n);
  n^.nriga := r;
  n^.pros := NIL
  ```

  occorre collegare il nodo puntato `n` alla coda preesistente. Come abbiamo visto, ciò avviene mediante i seguenti passi:

  – si collega il nuovo nodo *dopo* l'ultimo elemento della coda; in questo caso il puntatore all'ultimo elemento si trova in `a^.elenco.ultimo`; per collegare il nuovo nodo dopo tale elemento occorre modificarne il campo `pros` facendolo puntare al nuovo nodo:

  ```
  a^.elenco.ultimo^.pros := n
  ```

  – si aggiorna il puntatore all'ultimo elemento, facendolo puntare al nuovo nodo:

  ```
  a^.elenco.ultimo := n
  ```

La nuova versione della procedura `inserimento` è:

```
PROCEDURE inserimento (VAR a: tipoalbero; p: stringa; r: integer);

{ricerca la parola p nell'albero a; se e' presente ne incrementa il contatore}
{altrimenti la inserisce con contatore uguale a 1; in ogni caso memorizza in numero}
{r della riga in cui e' stata trovata la parola}

   VAR
      n: tipolista;

BEGIN {inserimento}
   IF a = NIL THEN {albero vuoto: la parola non c'e'}
      BEGIN
         {creazione del nodo dell'albero}
         new(a);
         a^.parola := p;
         a^.occorr := 1;
         a^.sx := NIL;
         a^.dx := NIL;

         {creazione del primo nodo della coda}
         new(n);
         n^.nriga := r;
```

```
         n^.pros := NIL;

         {collegamento del nodo della coda al nodo dell'albero}
         a^.elenco.primo := n;
         a^.elenco.ultimo := n
      END
   ELSE {albero non vuoto}
      IF p = a^.parola THEN {la parola p e' memorizzata nella radice}
         BEGIN
            {incremento del numero di occorrenze}
            a^.occorr := a^.occorr + 1;

            {creazione di un nuovo nodo per l'elenco delle righe}
            new(n);
            n^.nriga := r;
            n^.pros := NIL;

            {inserimento del nodo n nella coda del nodo a}
            a^.elenco.ultimo^.pros := n;
            a^.elenco.ultimo := n
         END
      ELSE IF p < a^.parola THEN {ricerca ricorsiva nei sottoalberi}
         inserimento(a^.sx, p, r)
      ELSE
         inserimento(a^.dx, p, r)
END; {inserimento}
```

La procedura `scrittura` dovrà visualizzare per ogni parola i numeri delle righe in cui è stata incontrata. Inseriamo dunque, dopo la stampa del campo `parola` e del campo `occorr` di un nodo, un ciclo in cui viene attraversata la coda, stampandone i campi `nriga`:

```
PROCEDURE scrittura (albero: tipoalbero);

{visualizza in output il contenuto di albero}

   VAR
      n: tipolista;

BEGIN {scrittura}
   IF albero <> NIL THEN
      BEGIN
         {visualizzazione del sottoalbero sinistro}
         scrittura(albero^.sx);

         {visualizzazione della parola e del numero di occorrenze}
         writeln(albero^.parola, 'Numero di occorrenze:', albero^.occorr : 6);

         {visualizzazione dei numeri delle righe in cui appare la parola}
         write(' ' : lung, 'Righe:');
         n := albero^.elenco.primo;
         WHILE n <> NIL DO
            BEGIN
               write(n^.nriga : 6);
```

```
            n := n^.pros
         END;
      writeln;

      {visualizzazione del sottoalbero destro}
      scrittura(albero^.dx)
   END
END; {scrittura}
```

Segue il testo completo del programma

```
PROGRAM occorrenze2 (input, output);

{visualizza in ordine alfabetico le parole presenti in input indicando, per ciascuna di esse,}
{il numero di occorrenze e i numeri delle righe in cui appare}

   CONST
      lung = 15; {lunghezza massima delle parole trattate}

   TYPE
      stringa = PACKED ARRAY[1..lung] OF char;

      tipolista = ^nodolista;
      nodolista = RECORD
            nriga: integer;
            pros: tipolista
         END;

      tipocoda = RECORD
            primo, ultimo: tipolista
         END;

      tipoalbero = ^nodoalbero;
      nodoalbero = RECORD
            parola: stringa;    {parola}
            occorr: integer;    {numero di occorrenze}
            sx, dx: tipoalbero; {puntatori ai sottoalberi}
            elenco: tipocoda    {elenco dei numeri delle righe in cui appare la parola}
         END;

   VAR
      a: tipoalbero;


   PROCEDURE lettura (VAR albero: tipoalbero);

   {restituisce tramite il parametro un albero di ricerca contenente le parole}
   {presenti in input con l'indicazione del numero di occorrenze di ciascuna e dei}
   {numeri delle righe in cui appare}

      VAR
         ch: char;             {per leggere i caratteri da input}
         word: stringa;        {per leggere le parole da input}
```

```
   pos: 0..lung;          {indica quanti caratteri significativi contiene word}
   preclettera: boolean; {per ricordare se il carattere precedente era una lettera}
   riga: integer;         {contatore delle righe}


PROCEDURE inserimento (VAR a: tipoalbero; p: stringa; r: integer);

{ricerca la parola p nell'albero a; se e' presente ne incrementa il contatore}
{altrimenti la inserisce con contatore uguale a 1; in ogni caso memorizza in numero}
{r della riga in cui e' stata trovata la parola}

   VAR
      n: tipolista;

BEGIN {inserimento}
   IF a = NIL THEN {albero vuoto: la parola non c'e'}
      BEGIN
         {creazione del nodo dell'albero}
         new(a);
         a^.parola := p;
         a^.occorr := 1;
         a^.sx := NIL;
         a^.dx := NIL;

         {creazione del primo nodo della coda}
         new(n);
         n^.nriga := r;
         n^.pros := NIL;

         {collegamento del nodo della coda al nodo dell'albero}
         a^.elenco.primo := n;
         a^.elenco.ultimo := n
      END
   ELSE {albero non vuoto}
      IF p = a^.parola THEN {la parola p e' memorizzata nella radice}
         BEGIN
            {incremento del numero di occorrenze}
            a^.occorr := a^.occorr + 1;

            {creazione di un nuovo nodo per l'elenco delle righe}
            new(n);
            n^.nriga := r;
            n^.pros := NIL;

            {inserimento del nodo n nella coda del nodo a}
            a^.elenco.ultimo^.pros := n;
            a^.elenco.ultimo := n
         END
      ELSE IF p < a^.parola THEN {ricerca ricorsiva nei sottoalberi}
         inserimento(a^.sx, p, r)
      ELSE
         inserimento(a^.dx, p, r)
```

```
   END; {inserimento}


BEGIN {lettura}
   albero := NIL;
   pos := 0;
   preclettera := false;
   riga := 1;

   WHILE NOT eof DO
      BEGIN
         WHILE NOT eoln DO
            BEGIN
               read(ch);

               IF ch IN ['A'..'Z', 'a'..'z'] THEN {il carattere e' una lettera}
                  BEGIN
                     preclettera := true;
                     IF pos < lung THEN
                        BEGIN {aggiungi il carattere letto a word}
                           pos := pos + 1;
                           word[pos] := ch
                        END
                  END
               ELSE IF preclettera THEN {e' stata raggiunta la fine di una parola}
                  BEGIN
                     {si riempiono le posizioni di word dopo la fine della parola di spazi}
                     FOR pos := pos + 1 TO lung DO
                        word[pos] := ' ';

                     {si inserisce la parola nell'albero}
                     inserimento(albero, word, riga);

                     {si predispone la variabile pos per la lettura di una nuova parola}
                     pos := 0;
                     preclettera := false
                  END
            END; {while not eoln...}

         readln;

         IF preclettera THEN {se la riga letta terminava con una parola...}
            BEGIN
               {si riempiono le posizioni di word dopo la fine della parola di spazi}
               FOR pos := pos + 1 TO lung DO
                  word[pos] := ' ';

               {si inserisce la parola nell'albero}
               inserimento(albero, word, riga);

               {si predispone la variabile pos per la lettura di una nuova parola}
               pos := 0;
```

```
                preclettera := false
             END;

          riga := riga + 1

       END {while not eof...}

   END; {lettura}


   PROCEDURE scrittura (albero: tipoalbero);

   {visualizza in output il contenuto di albero}

      VAR
         n: tipolista;

   BEGIN {scrittura}
      IF albero <> NIL THEN
         BEGIN
            {visualizzazione del sottoalbero sinistro}
            scrittura(albero^.sx);

            {visualizzazione della parola e del numero di occorrenze}
            writeln(albero^.parola, 'Numero di occorrenze:', albero^.occorr : 6);

            {visualizzazione dei numeri delle righe in cui appare la parola}
            write(' ' : lung, 'Righe:');
            n := albero^.elenco.primo;
            WHILE n <> NIL DO
               BEGIN
                  write(n^.nriga : 6);
                  n := n^.pros
               END;
            writeln;

            {visualizzazione del sottoalbero destro}
            scrittura(albero^.dx)
         END
   END; {scrittura}


BEGIN {occorrenze}
   lettura(a);
   scrittura(a)
END. {occorrenze}
```