Corso di Laurea in INFORMATICA

Algoritmi e Strutture Dati

MODULO N. 3

Esercizi su Liste

Ricerca in una lista ordinata.

Fusione di liste.

Ordinamento di liste.

Questi lucidi sono stati preparati da Oronzo Altamura, Maria Francesca Costabile e Floriana Esposito, Università degli Studi di Bari, per uso didattico. Essi contengono materiale originale di proprietà dell'Università degli Studi di Bari e/o figure di proprietà di altri autori, società e organizzazioni di cui e' riportato il riferimento. Tutto o parte del materiale può essere fotocopiato per uso personale o didattico ma non può essere distribuito per uso commerciale. Qualunque altro uso richiede una specifica autorizzazione da parte dell'Università degli Studi di Bari e degli altri autori coinvolti.

PROBLEMA

RICERCA IN UNA LISTA LINEARE ORDINATA

PROGETTARE E IMPLEMENTARE UN ALGORITMO PER RICERCARE IN UNA LISTA LINEARE E ORDINATA PER CHIAVE ALFABETICA

UNA LISTA LINEARE ORDINATA E' UN PARTICOLARE TIPO DI LISTA IN CUI L'ORDINE SEQUENZIALE DEGLI ELEMENTI E' LEGATO AD UNA RELAZIONE D'ORDINE (£) DEFINITA SUGLI ELEMENTI.

PERTANTO SE

$L = a_1\ a_2 \ldots a_n$

ALLORA

$a_1 £ a_2 £ \ldots £ a_n$

QUANDO SI DEVONO EFFETTUARE OPERAZIONI DI INSERIMENTO E DI CANCELLAZIONE DI ELEMENTI IN UNA LISTA LINEARE ORDINATA E' NECESSARIO STABILIRE PRIMA LA POSIZIONE IN CUI FA FATTA LA VARIAZIONE.

CONSIDERIAMO UNA LISTA DI NOMI MESSI IN ORDINE ALFABETICO NELLA QUALE INTENDIAMO INSERIRE UN NUOVO NOME

AD ESEMPIO

POS.	NOME	
1.	ANNA	NOME DA INSERIRE O DA CANCELLARE
2.	ANTONIO	
3.	CARLO	← DARIO
4.	DAVIDE	
5.	ELENA	
6.	FILIPPO	
7.	GIULIO	

PER PRIMA COSA E' NECESSARIO TROVARE LA POSIZIONE IN CUI IL NUOVO ELEMENTO VA INSERITO.

LA STRUTTURA SEQUENZIALE DELLA LISTA IMPONE CHE SI EFFETTUI UNA SCANSIONE SIA CHE SI VOGLIA INSERIRE UN NOME, SIA CHE SI VOGLIA CANCELLARLO. E' DUNQUE NECESSARIO

SCANDIRE LA LISTA FINCHE'
NON SI TROVI UN NOME CHE SEGUE IL NOME DATO IN ORDINE ALFABETICO (INSERIMENTO)
OPPURE
NON SI TROVI UN NOME EGUALE AL NOME DATO (CANCELLAZIONE)

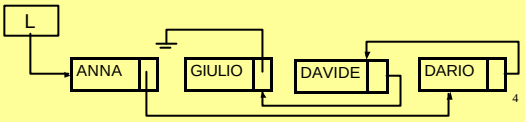
LA PARTE CENTRALE DELL'ALGORITMO DI RICERCA SARA'

MENTRE IL NOME CERCATO SEGUE NELL'ORDINE ALFABETICO IL NOME CORRENTE DI LISTA ESEGUI

(a) PASSA AL NOME SUCCESSIVO NELLA LISTA

QUESTO ALGORITMO NON TIENE CONTO DEL CASO IN CUI IL NOME CERCATO SEGUE L'ULTIMO NOME NELL'ELENCO.

IN UNA REALIZZAZIONE DELLA LISTA ORDINATA ATTRAVERSO RAPPRESENTAZIONE COLLEGATA, AD OGNI NOME SARA' ASSOCIATO UN CAMPO CHE FA RIFERIMENTO ALLA LOCAZIONE DEL SUCCESSIVO ELEMENTO IN ORDINE ALFABETICO DELLA LISTA. QUESTO SIGNIFICA CHE NON C'E' PIU' UNA CORRISPONDENZA OBBLIGATORIA TRA L'ORDINE FISICO CON CUI I NOMI SONO MEMORIZZATI E L'ORDINE LOGICO (CORRISPONDENTE QUI A QUELLO ALFABETICO).



NEL CONDURRE UNA RICERCA IN UNA LISTA L VA SEGUITO L'ORDINE LOGICO E NON QUELLO FISICO. SE LA STRUTTURA E' REALIZZATA CON PUNTATORI

```
cella = RECORD
        nome: tiponome;
        successivo: ^cella;
END;
lista = ^cella;
posizione = ^cella;
```

ALLORA USEREMO L'INFORMAZIONE SULLA CONNESSIONE LOGICA DATA DA successivo PER PROCEDERE LUNGO LA LISTA.

SE IL NODO corrente PUNTA ALL'ELEMENTO CORRENTE ALLORA BASTERA' EFFETTUARE L'ASSEGNAZIONE

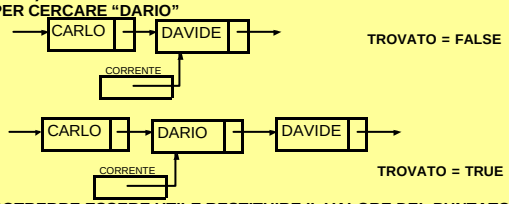
corrente : = SUCCLISTA (corrente, L)

PER SCORRERE LA CATENA FINCHE' NON SI TROVI LA POSIZIONE IN CUI IL NOME VA INSERITO (IN CASO DI INSERIMENTO) O E' PRESENTE (IN CASO DI RICERCA).

ALLA PROCEDURA CHAMANTE POSSIAMO RESTITUIRE UNA VARIABILE BOOLEANA TROVATO CHE AIUTI A SAPERE SE L'ELEMENTO RICERCATO C'E' O NON C'E'.

INOLTRE, PUO' ESSERE UTILE RESTITUIRE IL VALORE DEL PUNTATORE corrente CHE PUNTERA' AL PRIMO ELEMENTO DELLA LISTA CON NOME IN ORDINE ALFABETICO "SUPERIORE" AL NOME CERCATO (CASO TROVATO = FALSE) O ALL'ELEMENTO COL MEDESIMO NOME DI QUELLO CERCATO (NEL CASO IN CUI TROVATO = TRUE).

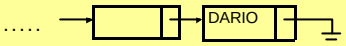
PER CERCARE "DARIO"



POTREBBE ESSERE UTILE RESTITUIRE IL VALORE DEL PUNTATORE MEMORIZZATO NELL'ELEMENTO CONTENENTE IL NOME CHE CERCHIAMO. IN TAL CASO VA MODIFICATO IL MECCANISMO DI AGGIORNAMENTO DEI PUNTATORI.

precedente : = corrente
corrente : = SUCCLISTA (corrente, L)

COSA SUCCEDE SE L'ELEMENTO CERCATO E' L'ULTIMO ELEMENTO NELLA LISTA?



CI ATTENDIAMO CHE CI VENGA SEGNALATO CHE QUELLO E' L'ULTIMO ELEMENTO E CHE LA LISTA E' FINITA.

DUNQUE, LA RICERCA TERMINA QUANDO SI TROVA LA POSIZIONE DEL NOME CERCATO O SI INCONTRA FINELISTA(L)=T (EQUIVALENTE, NELLA REALIZZAZIONE CON PUNTATORI, AL FATTO CHE IL PUNTATORE ASSOCIATO A "DARIO" ABBIA VALORE nil)

NEL CASO IN CUI IL NOME CERCATO SIA IL PRIMO ELEMENTO NELL'ELENCO VA RESTITUITA LA POSIZIONE ATTRAVERSO PRIMOLISTA(L)

7

ALGORITMO

1. DEFINISCI IL NOME CERCATO E L'INIZIO LISTA
2. INIZIALIZZA *PRECEDENTE* E *CORRENTE* A INIZIO LISTA
3. PONI *TROVATO* A FALSO
4. MENTRE LA RICERCA CONTINUA E NON E' FINITA LA LISTA ESEGUI:
   - a) SE IL NOME CERCATO PRECEDE ALFABETICAMENTE IL NOME CORRENTE ALLORA
     - a. 1) FA IN MODO DI INTERROMPERE LA RICERCA
   - ALTRIMENTI
   - a'. 1) PONI *PRECEDENTE* A *CORRENTE*
   - a'. 2) AGGIORNA *CORRENTE* PUNTANDO ALL'ELEMENTO SUCCESSIVO
1. STABILISCI SE IL NOME CERCATO E' *TROVATO*
2. RESTITUISCI *PRECEDENTE*, *CORRENTE* E *TROVATO*

COME SEMPRE NE DIAMO UNA REALIZZAZIONE SENZA E CON ASTRAZIONE DATI

8

```
Const namesize = 20;
Type nameformat = packed array [ 1..namesize ] of char;
     listpointer = ^listnode;
     listnode = record
                  listname : nameformat;
                  next : listpointer;
                end;
Procedure listsearch (searchname : nameformat; var current, previous :
     listpointer; listhead : listpointer; var found : boolean );
Var notfound:boolean ;        {se vero indica il prossimo nodo da esaminare }
Begin                         {ricerca nella lista ordinata la presenza di searchname}
        {asserisci; listhead punta alla testa della lista}
     previous := nil; notfound := true; current := listhead ;
     found := false;
     {invariante: searchname appare alfabeticamente dopo name nel nodo
     precedente se previous node <> nil }
     while notfound and (current <> nil) do
          with current^ do
               if searchname <= listname then
                    notfound := false
               else
               begin {move pointers to next node}
                    previous := current;
                    current := next;
               end;

          {asserisci: searchname corrisponde a o capita prima del nodo corrente se esiste}
          if current <> nil then
               if searchname = current^.listname then found := true
               else found := false
end
```

9

```
Const dimnome = 20;
Type     tiponome = packed array [1..dimnome] of char ;
         p_cella = ^cella
         cella = RECORD
                   nome : tiponome;
                   successivo : p_cella;
                 END;
         lista : p_cella;
         posizione : p_cella
PROCEDURE CERCA_IN_LISTA (cercato: tiponome; var corrente, precedente: posizione;
         L : lista; var trovato: boolean );
Var      continua : boolean ;
         nome_corr : tiponome;
BEGIN
         corrente := PRIMOLISTA (L);
         precendente := corrente;      continua := TRUE;
         trovato := FALSE;
         WHILE    continua  AND NOT FINELISTA (corrente, L) DO
         BEGIN
                  nome_corr := LEGGILISTA(corrente, L);
                  IF cercato <= nome_corr THEN
                                    continua := FALSE
                  ELSE
                  BEGIN
                           precedente := corrente;
                           corrente := SUCCLISTA(corrente, L);
                  END;
                  IF NOT FINELISTA (corrente, L) THEN
                                    TROVATO : = (cercato = nome_corr);
         END;
END;
```

10

FUSIONE DI LISTE ORDINATE

DATE DUE LISTE ORDINATE, PRODURRE UNA TERZA LISTA OTTENUTA DALLA FUSIONE DELLE PRIME DUE E CHE RISPETTA LO STESSO ORDINAMENTO

LE LISTE SONO SEQUENZE DI ELEMENTI, PER CUI L'ACCESSO ALL'ULTIMO ELEMENTO PUO' SOLO AVVENIRE MEDIANTE SCANSIONE SEQUENZIALE.

QUESTO VINCOLO RENDE INEFFICIENTE L'APPLICAZIONE ALLE LISTE DELL'ALGORITMO REALIZZATO PER DUE VETTORI ORDINATI. SI DEVE QUINDI RINUNCIARE AL CONFRONTO DEGLI ULTIMI DUE ELEMENTI PER STABILIRE SUBITO QUALE SEQUENZA VENGA ESAURITA PER PRIMA, E DI CONSEGUENZA PERDE SENSO ANCHE IL CONFRONTO FRA L'ULTIMO ELEMENTO DELLA PRIMA SEQUENZA AD ESAURIRSI E IL PRIMO ELEMENTO DELL'ALTRA.

IN ALTRI TERMINI L'ALGORITMO DI FUSIONE FRA LISTE E' SOSTANZIALMENTE FONDATO SUL SEGUQNTE CICLO:

MENTRE NON E' FINITA LISTA1 E NON E' FINITA LISTA2

CONFRONTA I DUE ELEMENTI CORRENTI DI LISTA1 E LISTA2

MEMORIZZA L'ELEMENTO MINORE IN LISTA3 ED AGGIORNA L'ELEMENTO CORRENTE DELLA LISTA INTERESSATA

11

```
Procedure fusione (var Lista1, Lista2, Lista3 : Lista);
Var
         p1, p2, p3: posizione;
         elem1, elem2: tipoelem ;
Begin
         crealista (Lista3);
         p1 := primolista(Lista1);
         p2 := primolista(Lista2);
         p3 := primolista(Lista3);
         while not finelista (p1, Lista1) and not finelista (p2, Lista2) do
         begin
                  elem1 := leggilista(p1, Lista1);
                  elem2 := leggilista(p2, Lista2);
                  if elem1 < elem2 then
                  begin
                           inslista (elem1, p3, Lista3);
                           p1 := succlista (p1, Lista1);
                  end
                  else
                  begin
                           inslista (elem2, p3, Lista3);
                           p2 := succlista (p2, Lista2);
                  end;
                  p3:=succlista (p3, Lista3);
         end;
end.
```

12

```
        while not finelista (p1, Lista1) do
        begin
                inslista (leggilista(p1, Lista1), p3, Lista3);
                p1 := succlista (p1, Lista1);
                p3 := succlista (p3, Lista3);
        end
        while not finelista (p2, Lista2) do
        begin
                inslista (leggilista(p2, Lista2), p3, Lista3);
                p2 := succlista (p2, Lista2);
                p3 := succlista (p3, Lista3);
        end;
end.
```


ORDINAMENTO DI UNA LISTA

DATA UNA LISTA, ORDINARE GLI ELEMENTI IN ORDINE CRESCENTE RISPETTO AD UNA RELAZIONE D'ORDINE £.

PER ORDINARE UNA LISTA NON SI POSSONO UTILIZZARE GLI ALGORITMI PROPOSTI PER GLI ARRAY IN QUANTO BASATI TUTTI SULL'ACCESSO DIRETTO AGLI ELEMENTI.

UTILIZZEREMO A TALE SCOPO IL METODO NOTO COME ORDINAMENTO NATURALE (O NATURAL MERGE SORT), GENERALMENTE UTILIZZATO PER ORDINARE I FILE.

DATA UNA LISTA $L = a_1\ a_2\ \ldots\ an$, DIREMO CHE UNA SOTTOSEQUENZA

$a_i\ a_{i+1}\ \ldots\ a_k$

COSTITUISCE UNA CATENA (O RUN) SE ACCADE CHE:

$a_{i-1} > a_i$

a_j £ a_{j+1} per ogni j = i, i+1, … , k -1

$a_k > a_{k+1}$

UNA LISTA E' QUINDI UNA SEQUENZA DI CATENE.

ESEMPIO

L = 82 | 16 | 14 15 84 | 25 77 | 13 75 | 4

L E' OTTENUTA DALLA CONCATENAZIONE DI SEI CATENE

UNA PROPRIETA' INTERESSANTE E' CHE LA FUSIONE DI DUE SEQUENZE DI N CATENE PRODUCE UNA SINGOLA SEQUENZA DI N CATENE.

COSI', FONDENDO LE CATENE CHE COMPONGONO UNA LISTA SI OTTIENE UNA NUOVA LISTA CON UN NUMERO DIMEZZATO DI CATENE.

RIPETENDO QUESTA OPERAZIONE AD OGNI PASSO, DOPO AL PIU' $LOG_2\ N$ PASSI LA LISTA SARA' COMPLETAMENTE ORDINATA.

ESEMPIO

L = 82 | 16 | 14 15 84 | 25 77 | 13 72 | 4

1° PASSO: FONDIAMO CATENE CONSECUTIVE:

L = 16 82 | 14 15 25 77 84 | 4 13 72

2° PASSO:

L = 14 15 16 25 77 82 84 | 4 13 72

3° PASSO:

L = 4 13 14 15 16 25 72 77 82 84

MA COME DETERMINARE LE SEQUENZE DI CATENE DA FONDERE?

SI PUO' PENSARE AD UNA FASE DI DISTRIBUZIONE IN CUI LE CATENE DI L SONO DISTRIBUITE ALTERNATIVAMENTE A DUE LISTE AUSILIARIE A E B. AL TERMINE DI QUESTA FASE A E B CONTERRANNO OGNUNA N/2 CATENE ORIGINARIE DI L.

ESEMPIO

A : 82 | 14 15 84 | 13 72

B : 16 | 25 77 | 4

SI NOTI CHE IL NUMERO DI CATENE EFFETTIVE IN A E B PUO' ANCHE ESSERE MINORE DI N/2, POICHE' DOPO LA DISTRIBUZIONE DUE O PIU' CATENE ORIGINARIE IN L FORMANO UN'UNICA CATENA IN A (O B).

AD ESEMPIO, B HA EFFETTIVAMENTE DUE CATENE:

B : 16 25 77 | 4

FONDENDO LE CATENE DISTRIBUITE SI PUO' CREARE UNA NUOVA LISTA L IL CUI GRADO DI ORDINAMENTO E' MAGGIORE.

L'ALGORITMO DI ORDINAMENTO ALTERNA FASI DI DISTRIBUZIONE A FASI DI FUSIONE, FINO A QUANDO SI OTTIENE UNA LISTA CON UN'UNICA CATENA


```
Procedure ordinamento Naturale ( var L : lista);
var     A, B : lista;
        numero_catene : integer;
begin
        repeat
                crealista(A);
                crealista(B);
                distribuisci (L, A, B);
                numero_catene := 0;
                crealista (L);
                merge (A, B, L, numero_catene);
        until numero_catene = 1;
end;
```

LA FASE DI DISTRIBUZIONE SI BASA SUL RICHIAMO DELLA PROCEDURA copiaCatena.

```
Procedure distribuisci (L: lista; var A,B : lista);
Var     pl, pa, pb : posizione;
Begin
        pl := primolista (L); pa := primolista (A);
        pb := primolista (B);
        repeat
                copiaCatena(pl, L, pa, A);
                if not finelista(pl. L) then
                        copiaCatena(pl, L, pb, B);
        until finelista(pl, L);
end;
```

MENTRE LA FASE DI FUSIONE SI BASA SUL RICHIAMO DELLA PROCEDURA fondiCatena. QUESTA PRODUCE UNA SINGOLA CATENA FUSA IN **L** A PARTIRE DA UNA CATENA DELLA LISTA **A** ED UNA DELLA LISTA **B**. E' IN QUESTA PROCEDURA CHE SI SFRUTTA LA RELAZIONE D'ORDINE SUGLI ELEMENTI (VEDI CONFRONTO <).

19

```
Procedure merge (A,B : lista; var L : lista; var numero_catene : integer );
Var
        pa, pb, pl : posizione;
Begin
        pa := primolista (A);
        pb := primolista (B);
        pl := primolista (L);
        while not finelista (pa, A) and not finelista (pb, B) do
        begin
                fondiCatena(pa, A, pb, B, pl, L);
                numero_catene := numero_catene +1;
        end;
        while not finelista (pa, A) do
        begin
                copiaCatena (pa, A, pl, L);
                numero_catene := numero_catene +1;
        end;
        while not finelista (pb, B) do
        begin
                copiaCatena(pb, B, pl, L);
                numero_catene := numero_catene + 1;
        end;
end;
```

20

```
Procedure fondiCatena (  var    pa: posizione;   A : lista;
                         var    pb : posizione;  B : lista;
                         var    pl : posizione; var L : lista; );
Var
        finecatena : boolean;
Begin
        repeat
                if leggilista (pa, A) < leggilista (pb, B) then
                begin
                        copia (pa, A, pl, L, finecatena);
                        if finecatena then
                                copiaCatena (pb, B, pl, L);
                end
                else
                begin
                        copia (pb, B, pl, L, finecatena);
                        if finecatena then
                                copiacatena(pa, A, pl, L);
                end;
        until finecatena;
end;
```

21

```
Procedure copiaCatena ( var     pa : posizione;   A : lista;
                        var     pb : posizione;   B : lista );
var
        finecatena : boolean;
Begin
        repeat
                copia (pa, A, pb, B, finecatena);
        until finecatena;
end;
```

INFINE LA PROCEDURA COPIA SARA' DATA DA:

```
Procedure copia (   var   pa : posizione;   A : lista;
                    var   pl : posizione; var L : lista;
                          var   finecatena : boolean );
Var elemento : tipoelem;
Begin
        elemento := leggilista (pa, A);
        inslista (elemento, pl, L);
        pa := succlista (pa, A);
        pl := succlista (pl, L);
        if finelista (pa, A) then finecatena := true
        else finecatena := elemento > leggilista (pa, A);
end;
```

22

SONO POSSIBILI DIVERSI INNESTI DELLE SEI PROCEDURE VISTE SOPRA. SEGUENDO I PRINCIPI DELLA PROGRAMMAZIONE MODULARE SCRIVEREMO LE PROCEDURE NELL'ORDINE DI COMPLESSITA' DEL CODICE, OVVERO:

```
Procedure copia (       var     pa : posizione;   A : lista;
                        var     pl : posizione; var L : lista;
                        var     finecatena : boolean );
var elemento : tipoelem;
Begin
 - - - - - -
End;


Procedure copiaCatena (var pa: posizione; . . . . . );
var finecatena : boolean;
Begin
.........
end;
```

23

```
Procedure fondiCatena (var .....);
var finecatena : boolean;
Begin
....
end;

Procedure merge ( A, B : lista, .... );
Var pa, pb, pl : posizione;
Begin
.......
end;

Procedure distribuisci( .... );
.....
Procedure ordinamentoNaturale ( ... );
....
```

24