3. Strutture lineari di dati: Liste

Il concetto di sequenza. Le Liste: specifiche e realizzazioni attraverso rappresentazioni sequenziali e collegate.

Algoritmi e Strutture Dati + Lab
A.A. 21/22

Informatica
Università degli Studi di Bari “Aldo Moro”

Nicola Di Mauro

Strutture lineari

- Consideriamo strutture di dati che si sviluppano in una dimensione e possono essere considerate come sequenze di oggetti cioè
  - le strutture lineari di dati
- In esse è importante l'esistenza di una relazione d'ordine tra gli oggetti che ci aiuta ad individuarli e a selezionarli.
- Per distinguere le diverse strutture è rilevante considerare
  - a) i modi di individuare la posizione in cui operare nella sequenza (modi di accedere)
  - b) i modi di agire nella posizione individuata

Strutture lineari /2

- Modi di accedere
  - accesso diretto
  - accesso per scansione
  - accesso agli estremi
- Il vettore (array) è il tipico esempio di accesso diretto
  - consente di accedere al singolo componente mediante il nome e il meccanismo dell'indice
- La lista è un esempio di accesso per scansione
  - consente l'accesso all'elemento generico solo dopo aver scandito gli elementi che lo precedono
- Pila e coda sono esempi di accesso agli estremi
  - pila (lifo last in first out)
  - coda (fifo first in first out)

Strutture lineari /3

- Relativamente ai modi di agire nelle posizioni possiamo avere i seguenti tipi di operazione
  - lettura del valore di un componente (ispezione)
  - aggiornamento del valore di un componente (cambio di valore)
  - inserimento di un nuovo componente (scrittura)
  - rimozione di un componente (cancellazione)

Strutture lineari /3

- Sono possibili diverse combinazioni tra i modi di accedere e i modi d'operare
  - L'unico problema riguarda la combinazione tra accesso diretto a tutta la struttura e possibilità di inserimento e rimozione di componenti
- Generalmente queste operazioni alterano la numerazione dei componenti e ciò è in contrasto col principio che i componenti siano individuati dal numero d'ordine della loro posizione
- Tutte le strutture lineari, tranne il vettore, sono dotate di operatori di inserimento e rimozione di componenti.

Liste

- Una lista è una sequenza finita, anche vuota, di elementi dello stesso tipo
- La differenza col concetto di “insieme” è che mentre in un insieme un elemento non può comparire più di una volta, nella lista uno stesso elemento può comparire più volte, in posizioni diverse
- Gli elementi della lista, cui sono associate delle informazioni, sono definiti atomi o nodi
- Indichiamo la lista con la notazione
  - l = < a1, a2, … , an > n >= 0
- A ciascun elemento di una lista viene associata una posizione
  - pos(i)
- e un valore
  - a(i)

Accesso

- Si può accedere direttamente solo al primo elemento della sequenza
  - per accedere al generico elemento occorre scandire sequenzialmente gli elementi della lista che lo precedono
- Le operazioni
  - È possibile aggiungere (inserire) e togliere (cancellare) elementi; poiché la lista è a dimensione variabile queste operazioni che alterano la dimensione sono fondamentali (nel vettore non sono concesse)
- La lista è dunque una struttura dati dinamica

Lunghezza e sottoliste

- La lunghezza di una lista è il numero dei suoi elementi. Se il numero di elementi è zero la lista si dice vuota
- La lunghezza conta le posizioni, non i simboli distinti, così un simbolo che compare k volte contribuisce con k unità alla lunghezza della lista
- Se l = < a1, a2, … , an > è una lista allora per ogni i e j tali che 1 <= i <= j <= n
  - < ai, … , aj > è' una sottolista di l, ottenuta partendo dalla posizione i e prendendo tutti gli elementi fino alla posizione j
- La lista vuota < > è sottolista di qualsiasi lista

Specifica sintattica

- Tipi: lista, posizione, boolean, tipoelem
- Operatori:
  - crealista : ( ) → lista
  - listavuota : (lista) → boolean
  - leggilista : (posizione, lista) → tipoelem
  - scrivilista : (tipoelem,posizione,lista) → lista
  - primolista : (lista) → posizione
  - finelista : (posizione, lista) → boolean
  - succlista : (posizione, lista) → posizione
  - predlista : (posizione, lista) → posizione
  - inslista : (tipoelem,posizione,lista) → lista
  - canclista : (posizione, lista) → lista

Specifica semantica

- Tipi:
  - Lista: insieme delle sequenze l = < a1, a2, … , an >, n>=0, di elementi di tipo tipoelem dove l'elemento i-esimo ha valore ai e posizione pos(i)
  - boolean: insieme dei valori di verità
- **Operatori**
- crealista = l'
  - POST: l' = <> (sequenza vuota)
- listavuota(l)= b
  - POST: b = true se l= < >
    b = false altrimenti
- leggilista(p,l) = a
  - PRE: p = pos(i) 1 <= i <= n
  - POST: a = a(i)

Specifica semantica /2

- scrivilista(a,p,l) = l'
  - PRE: p = pos(i) 1 <= i <= n
  - POST: l' = < a1, a2, … , ai-1, a, ai+1, … , an >
- primolista(l) = p
  - POST: p = pos(1)
- finelista(p,l) = b
  - PRE: p = pos(i) 1 <= i <= n+1
  - POST: b = true se p = pos(n+1)
b= false altrimenti
- succlista(p,l) = q
  - PRE: p = pos(i) 1 <= i <= n
  - POST: q = pos(i+1)

Specifica semantica /3

- predlista(p,l) = q
  - PRE: p = pos(i) 2 <= i <=n
  - POST: q = pos(i-1)
- inslista(a,p,l) = l'
  - PRE: p = pos(i) 1 <= i <= n+1
  - POST: l' = < a1, a2, … , ai-1, a, ai , ai+1, … , an >, se 1 <= i <= n
    l' = <a1, a2, … ,an, a > , se i = n+1
    (e quindi l' = < a > se i = 1 e l = < >)
- canclista(p,l) = l'
  - PRE: p = pos(i) 1 <= i <= n
  - POST: l' = < a1, a2, … , ai-1, ai+1, … , an >

Specifica semantica /4

- Dalla specifica semantica emerge che per accedere a un elemento occorre conoscerne la posizione
- Questo nelle liste è possibile solo per il primo elemento. L'unico operatore che dà per risultato la posizione (senza altre informazioni) è primolista.
- Per gli altri la posizione si ottiene conoscendo a priori la posizione dell'elemento precedente (o seguente) e applicando l'operazione succlista (o predlista).
- Dunque, per accedere ad un generico elemento occorre
  - scandire la lista a partire dal primo elemento
- L'operatore listavuota è ridondante visto che può essere sostituito da finelista (primolista (l) , l)
- Nota: in alcuni testi si trova una diversa specifica di primolista(l) che restituisce l'elemento primo piuttosto che la posizione.

Eliminazione di duplicati

- Un esempio di come sia possibile, avendo a disposizione gli operatori definiti nella algebra per il trattamento di una lista, risolvere un generico problema, evitando di entrare nei dettagli relativi a come la lista si e' implementata, è dato di seguito.
- Supponiamo di utilizzare come linguaggio algoritmico di riferimento il pascal e supponiamo di disporre, in pascal, degli operatori sul tipo lista.
- Ci poniamo il seguente problema
  - eliminazione di duplicati:
  - Data una lista l, i cui elementi siano interi, eliminare da l gli elementi che sono duplicati.

Eliminazione di duplicati /2

```
epurazione(lista l)
    p = primolista(l)
    while not finelista(p, l) do
        q = succlista(p, l)
        while not finelista(q, l) do
            if leggilista(p,l) = = leggilista(q,l) then
    canclista(q,l)
                    q = succlista(q, l)
     p = succlista(p,l);
```

- Come è evidente si è potuto fare riferimento alla struttura lista senza entrare nei dettagli realizzativi
- Per realizzare una lista distinguiamo due rappresentazioni fondamentali: rappresentazione sequenziale e rappresentazione collegata

Realizzazione sequenziale con vettore

- Una lista può essere rappresentata usando un vettore (array monodimensionale).
- Poiché il numero di elementi che compongono la lista può variare si utilizza una variabile **primo** per il valore dell'indice della componente del vettore in cui è memorizzato il primo elemento della lista.
- Si utilizza un'altra variabile lunghezza per indicare il numero di elementi di cui e' composta la lista rappresentata.

Realizzazione sequenziale con vettore /2

LISTA

(4 5 1 21 45)

Questa rappresentazione consente di realizzare molto semplicemente alcune delle operazioni definite per la lista.

Il vero problema riguarda l'inserzione e la rimozione di componenti

1	4
2	5
3	1
4	21
5	45
6	78
7	12
8	1
9	-5
10	0
11	-2
12	61

PRIMO

1

LUNGHEZZA

5

Realizzazione sequenziale con vettore /2

LISTA

(4 5 1 **11** 21 45)

La banale inserzione in terza posizione di un nuovo elemento causa lo spostamento verso il basso del quarto e del quinto elemento

1	4
2	5
3	1
4	11
5	21
6	45
7	12
8	1
9	-5
10	0
11	-2
12	61

PRIMO

1

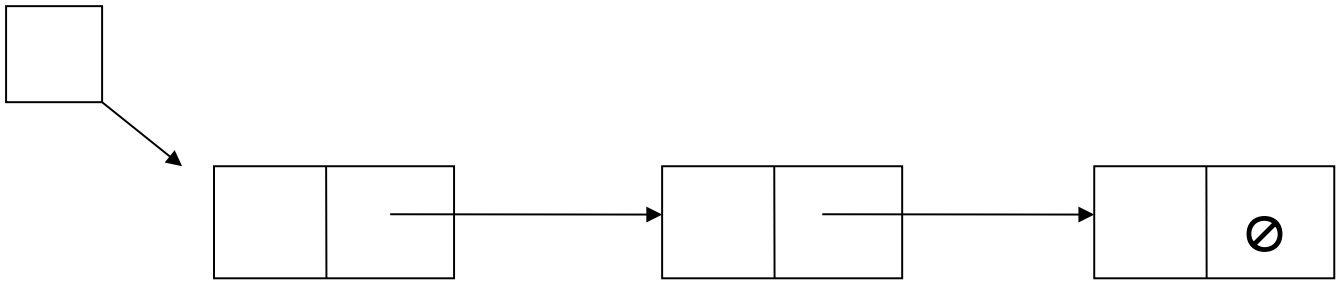
LUNGHEZZA

6

La rappresentazione collegata

- L'idea fondamentale della rappresentazione collegata di una lista è quella di memorizzare i suoi elementi associando ad ognuno di essi una particolare informazione (riferimento) che permetta di individuare la locazione in cui è memorizzato l'elemento successivo.
- Per visualizzare tale rappresentazione si usa una notazione grafica in cui
  - gli elementi sono rappresentati mediante nodi
  - i riferimenti mediante archi che collegano nodi

La rappresentazione collegata /2



- Si può notare che si usa un riferimento al primo elemento della lista (riferimento iniziale) e un simbolo speciale ⊘ come riferimento associato all'ultimo nodo (nel caso la lista sia vuota, tale simbolo compare direttamente nel riferimento iniziale).

Realizzazione con cursori

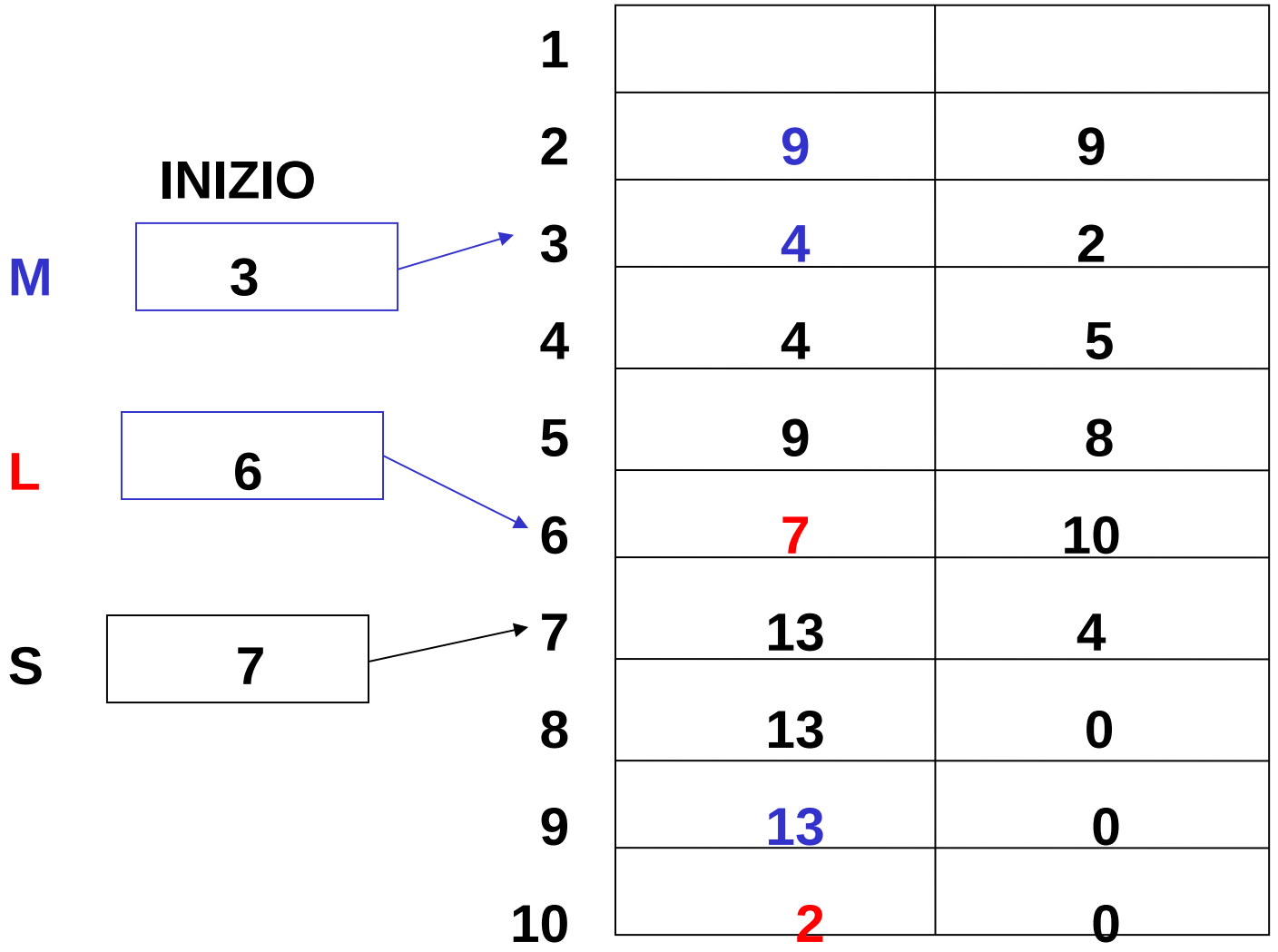
- viene sempre utilizzato un vettore (array monodimensionale) per l'implementazione della lista, ma si riesce a superare, attraverso i riferimenti, il problema dell'aggiornamento (inserimento o cancellazione di un elemento)
- Si realizzano i riferimenti mediante cursori, cioè variabili intere o enumerative, il cui valore è interpretato come indice di un vettore.
- Si definisce un vettore spazio che
  - contiene tutte le liste, ognuna individuata da un proprio cursore iniziale
  - contiene tutte le celle libere, organizzate in una lista, detta "listalibera"

Realizzazione con cursori /2

- Esempio: disponiamo di tre diverse liste l, m, s
  l = <7, 2> m = <4, 9, 13> s = <13, 4, 9, 13>
- Possiamo usare un unico vettore spazio per rappresentare le tre liste.
- La componente di spazio ha due campi: nel campo “elemento” è memorizzato il contenuto del nodo, nel campo “successivo” il riferimento al prossimo nodo
- La sequenza degli elementi che formano la lista è ricostruibile iniziando dalla componente dell’array corrispondente al valore di inizio, nel cui campo “elemento” è memorizzato il valore del primo elemento.
- Seguendo i cursori si trovano gli elementi successivi della lista. Infatti, utilizzando i cursori, si può definire la posizione pos(i) dell’elemento i–esimo di l uguale al valore del cursore alla cella del vettore spazio che contiene l’elemento (i-1) – esimo, se 2 <= i <= n+1, uguale a 0 se i = 1

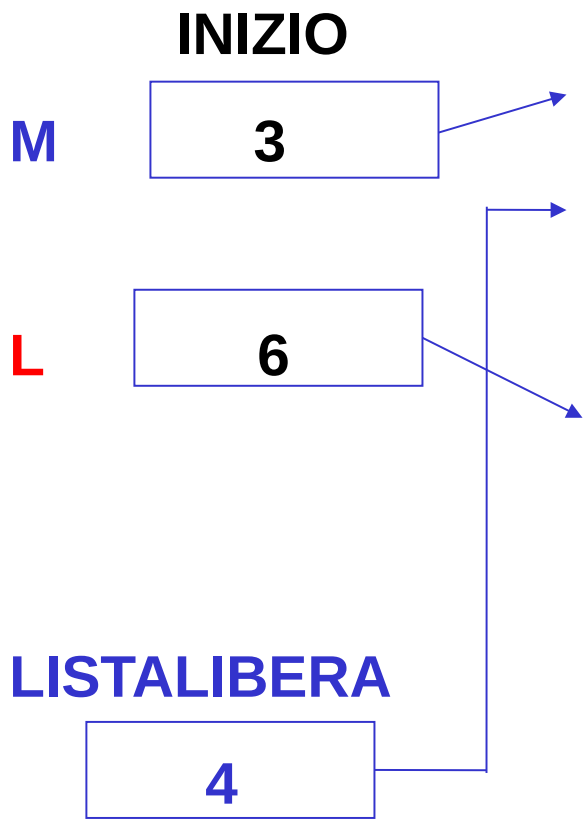
Realizzazione con cursori /3

- Analogamente si definisce pos(n+1) uguale al cursore alla cella di spazio che contiene l'elemento n – esimo se n >= 1, o uguale a 0 altrimenti. La lista vuota si indica con l=∅.



Realizzazione con cursori /4

- Per poter aggiornare una lista così realizzata sorge il problema di individuare la posizione di una componente libera nell'array. Si usa una listalibera memorizzata nello stesso array per raccogliere in modo collegato le componenti libere dell'array



1		8
2	9	9
3	4	2
4		5
5		1
6	7	10
7		0
8		7
9	13	0
10	2	0

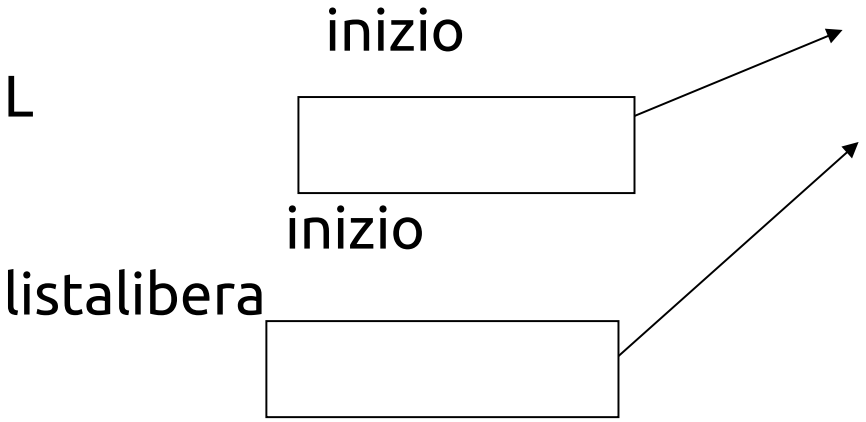
Realizzazione con cursori /5

- Quando si deve inserire un nuovo elemento nella lista, si preleva una componente della listalibera e la si utilizza per memorizzare il nuovo elemento, collegandolo in modo opportuno agli altri elementi della lista.
- Una possibile dichiarativa:
  - definizione di costanti:
    maxlung = … /* costante intera positiva */
  - definizione di tipi:
    posizione intero in 0..maxlung
    lista di tipo posizione
    tipoelem di tipo intero
    componente tipo strutturato con componenti
    - elemento di tipo tipoelem
    - successivo di tipo posizione
  - definizione di variabili:
    listalibera: variabile di tipo lista
    spazio: array di maxlung elementi di tipo componente

Realizzazione con cursori /6

- La listalibera rappresenta un serbatoio da cui prelevare componenti libere dell'array e in cui riversare le componenti dell'array che non sono piu' utilizzate per la lista

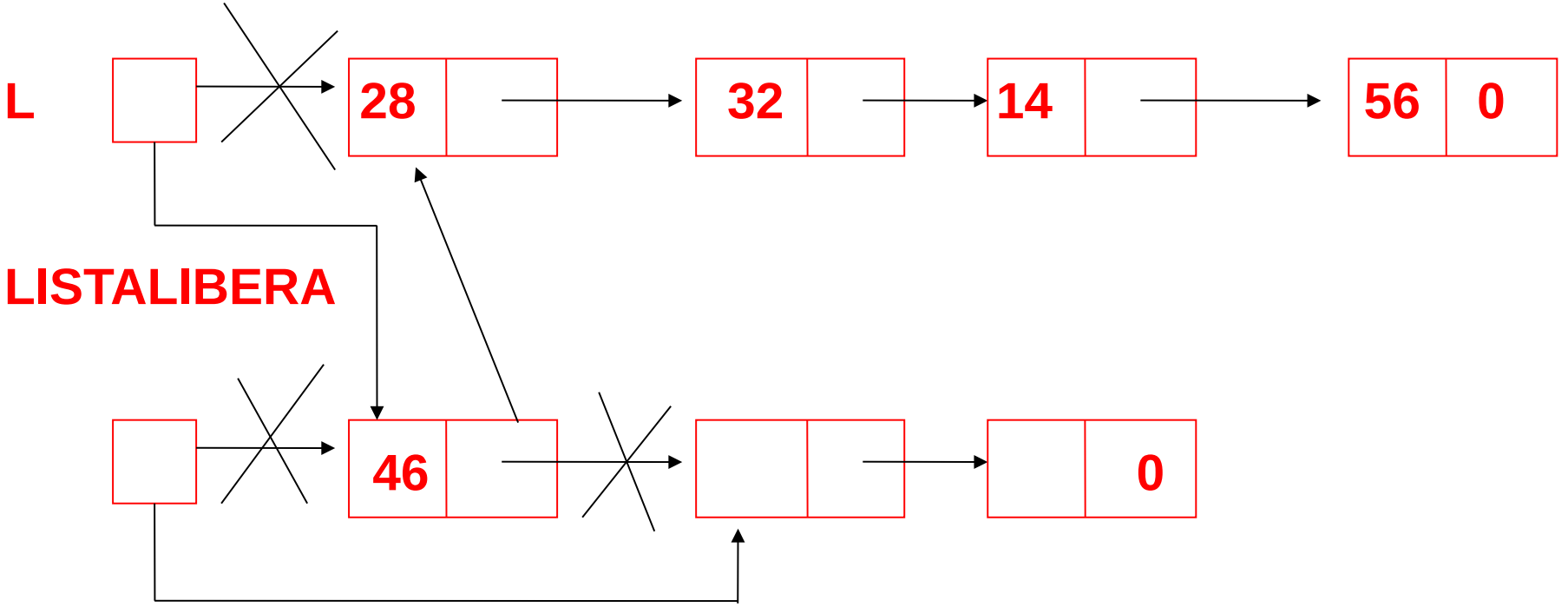
L = [28, 32, 14, 56]



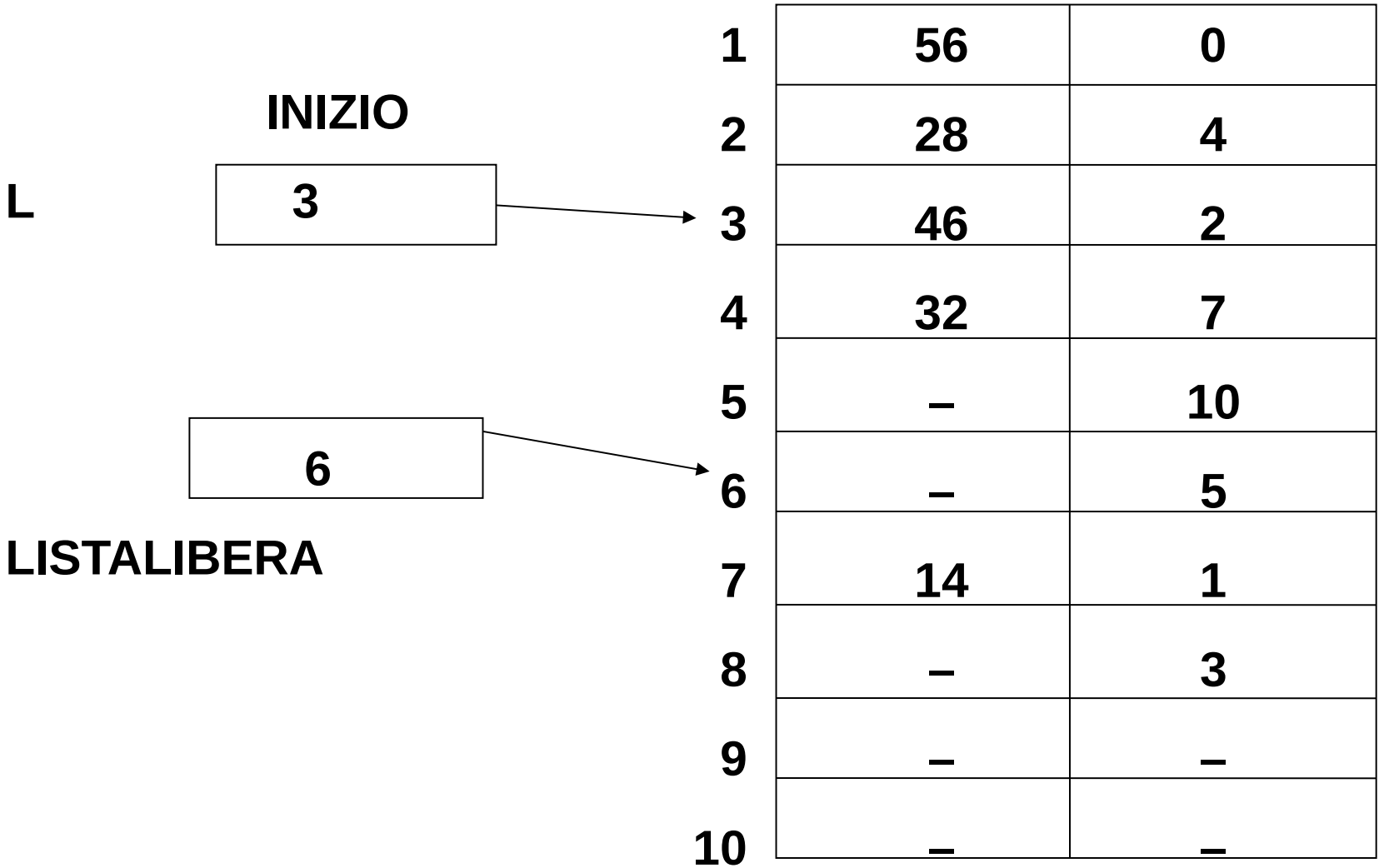
1	56	0
2	28	4
3	–	6
4	32	7
5	–	10
6	–	5
7	14	1
8	–	0
9	–	8
10	–	9

Realizzazione con cursori /7

- Se si vuole inserire un nuovo elemento (= 46) in testa alla lista si fa uso della prima componente della lista libera



Realizzazione con cursori /8



Realizzazione con cursori /9

- Le operazioni eseguibili sulla listalibera sono
  - la inizializzazione, che serve a collegare tra loro le varie componenti dell'array che fanno parte inizialmente della listalibera;
  - il prelievo di una componente della listalibera se esiste;
  - la restituzione di una componente alla listalibera.

```
inizializza_lista_libera()
    listalibera = 1
    for i = 1 to maxlung – 1 do
        spazio[i].successivo = i+1
        spazio[maxlung].successivo = 0
```

Realizzazione con cursori /10

- È immediato verificare, prima di definire le operazioni di inslista e canclista, che con questa rappresentazione l'inserimento e l'eliminazione di un elemento non richiedono lo spostamento di altri elementi della lista, grazie alla listalibera.
- Rimangono i problemi connessi all'uso dell'array cioè all'esigenza di definire una dimensione.

Considerazioni

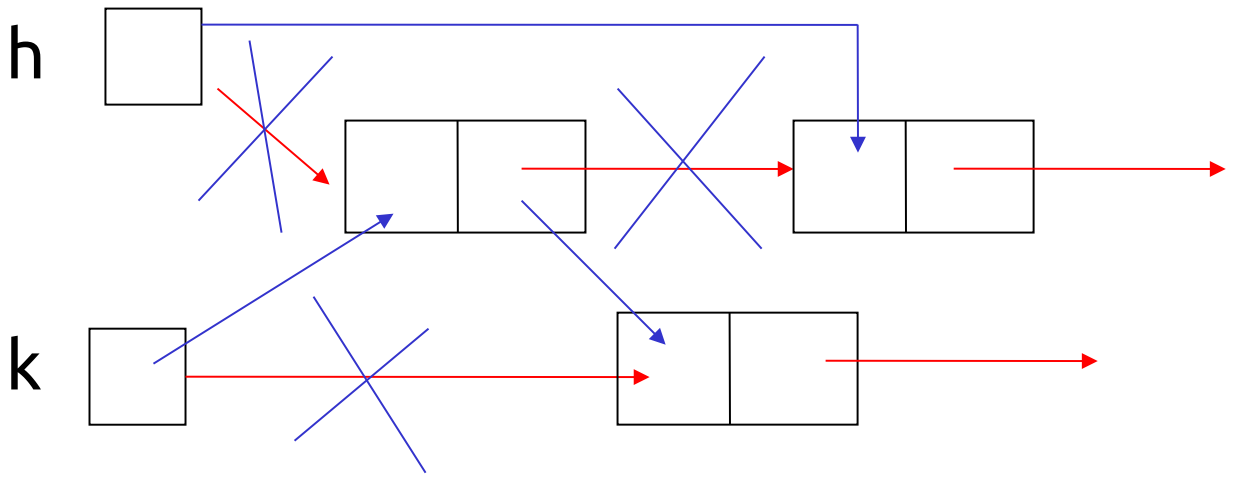
- Uno degli svantaggi della rappresentazione sequenziale è superato
  - l'inserimento e la eliminazione di un elemento non richiedono lo spostamento di altri elementi nella lista
- La complicazione data dalla necessità di gestire la lista libera è compensata dal fatto che gli aggiornamenti richiedono un numero limitato di operazioni
- Rimangono gli svantaggi connessi all'uso dell'array
  - la dimensione dell'array rappresenta un limite alla crescita della lista e la quantità in memoria utilizzata non dipende dalla lunghezza effettiva della lista
- Rispetto alla rappresentazione sequenziale vi è un'ulteriore occupazione di memoria, vista la necessità di memorizzare i riferimenti.

Spostamento

- È fondamentale disporre della procedura sposta(h,k) che trasferisce la cella puntata da h spostandola prima della cella puntata da k

sposta(riferimento h, riferimento k)

temp = k
k = h
h = spazio[h].successivo
spazio[k].successivo = temp



Inserimento

```
inslista(tipoelem a, posizione p, lista l)
    if listalibera = 0 then error
    elseif p = 1 then
        sposta(listalibera, l)
        spazio[l].elemento = a
    else
        sposta(listalibera, spazio[p].successivo)
        spazio[spazio[p].successivo].elemento = a
```

Eliminazione

```
canclista(posizione p, lista l)
    if p = 1 then
    sposta(l, listalibera)
    else
    sposta(spazio[p].successivo, listalibera)
```

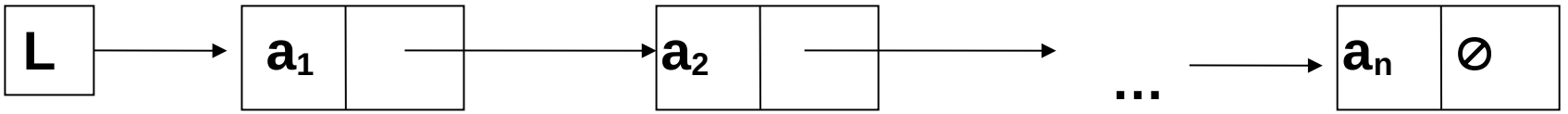
Benchè si siano date le realizzazioni usando uno pseudo linguaggio, è evidente che la realizzazione con cursori è implementabile in tutti i linguaggi che dispongono di vettori e consentono l'astrazione funzionale mediante procedure o funzioni

Realizzazione con puntatori

- Un'altra possibile realizzazione di una lista è quella mediante l'uso congiunto del tipo puntatore e del tipo record.
- Questa è sicuramente la più efficace realizzazione della rappresentazione collegata
- Il tipo puntatore è un tipo di dato i cui valori rappresentano indirizzi di locazioni di memoria. Le operazioni usualmente disponibili su una variabile di tipo puntatore p sono:
  - l'accesso alla locazione il cui indirizzo è memorizzato in p
  - la richiesta di una nuova locazione di memoria e la memorizzazione dell'indirizzo in p (new)
  - il rilascio della locazione di memoria il cui indirizzo è memorizzato in p (delete)

Rappresentazione collegata realizzata mediante puntatori

- Una possibile realizzazione è quella di una lista monodirezionale semplificata. Vi è una struttura di n elementi o “celle”, tale che l’i – esima cella contiene l’i – esimo elemento della lista e l’indirizzo della cella che contiene l’elemento successivo
  - La prima cella è indirizzata da una variabile l di tipo puntatore
  - L’ultima cella punta a un valore convenzionale nil
  - Gli indirizzi sono noti alla macchina ma non al programmatore
  - la posizione pos(i) è uguale al valore del puntatore alla cella che contiene l’i – esimo elemento (atomo) 1 <= i <= n

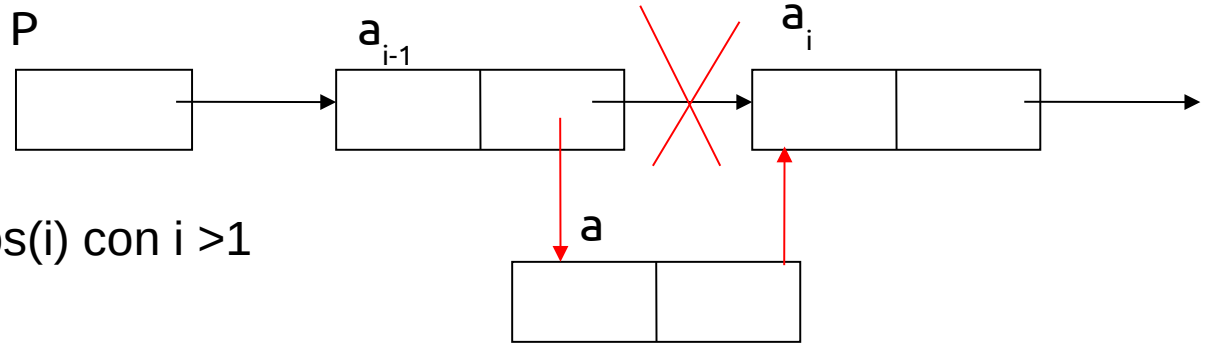


Rappresentazione collegata realizzata mediante puntatori /2

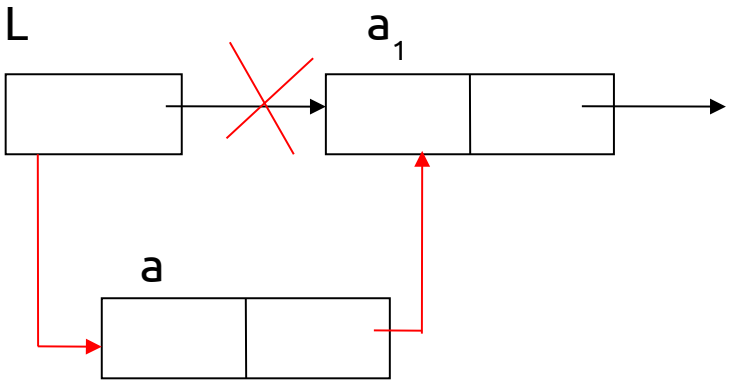
- Per numero di elementi diverso da zero (n > 1) pos(i) è il puntatore alla cella contenente l'i-esimo elemento
- Mentre per n = 0 ovvero quando la lista e' vuota L = NULL
- Una possibile dichiarazione di tipo

posizione: tipo puntatore a cella
cella: tipo strutturato con componenti
    elemento di tipo tipoelem
    successivo di tipo posizione
lista: alias per il tipo posizione

L'aggiornamento dei puntatori



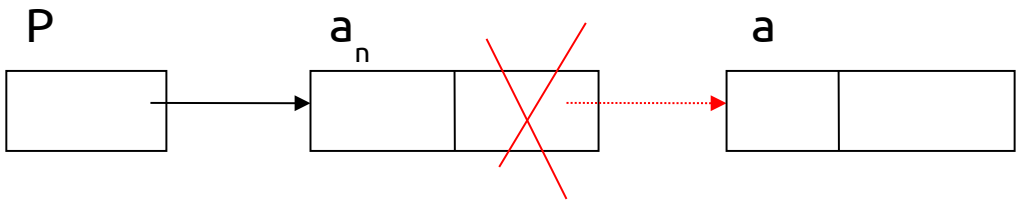
Inslista in P=pos(i) con i >1



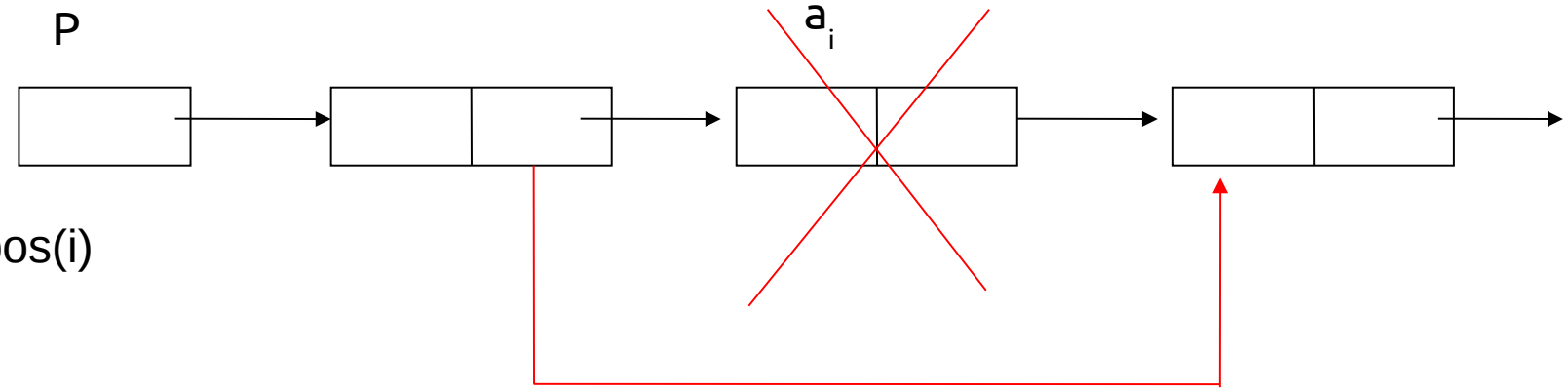
Inslista in P=pos(1)

L'aggiornamento dei puntatori /2

Inslista in P=pos(n+1)

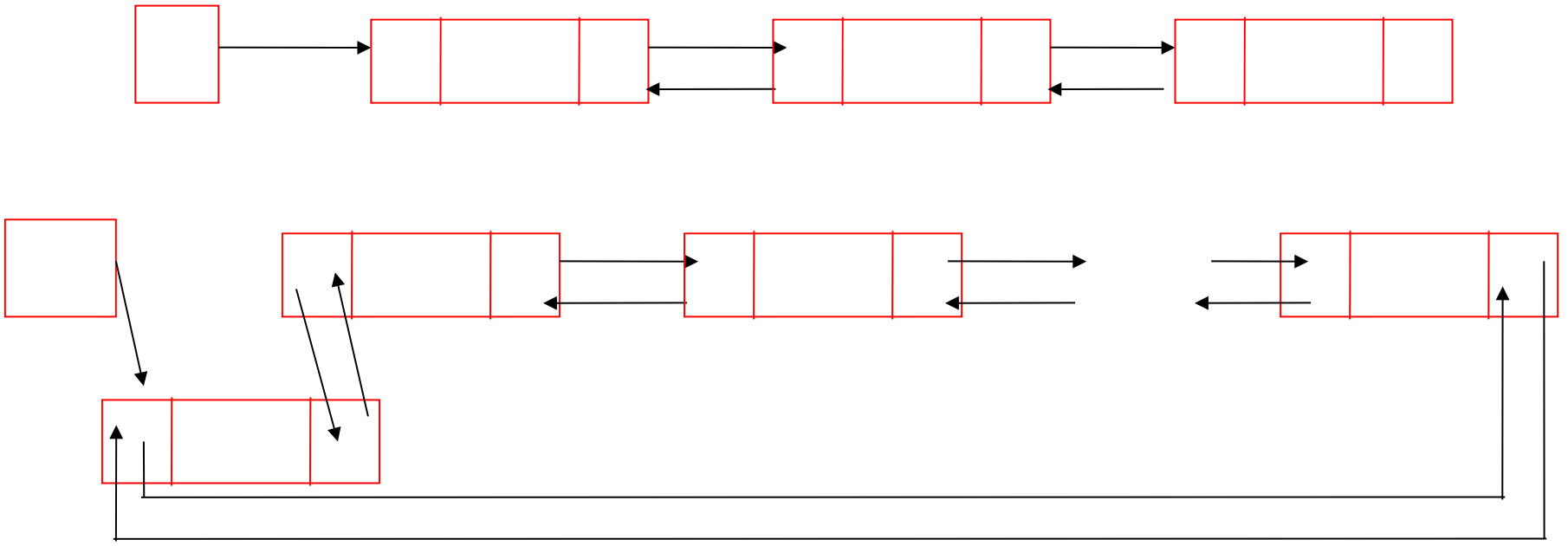


Canclista P=pos(i)



Una variante della rappresentazione collegata (a doppi puntatori o simmetrica)

- Ogni elemento contiene, oltre al riferimento al nodo successivo, anche il riferimento al precedente.
- La lista può essere “espansa” con una cella in piu’ per la realizzazione circolare



Una variante della rappresentazione collegata (a doppi puntatori o simmetrica)2/

- Con questa rappresentazione si ha il vantaggio di:
  - poter scandire la lista in entrambe le direzioni
  - poter individuare facilmente l'elemento che precede
  - poter realizzare le operazioni di inserimento senza dover usare variabili aggiuntive

Esercizi su Liste

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 è un particolare tipo di lista in cui l'ordine sequenziale degli elementi è legato ad una relazione d'ordine (<=) definita sugli elementi.
- Pertanto se
  - l = < a1 , a2 , . . . , an >
- allora
  - a1 <= a2 <= . . . <= an
- Quando si devono effettuare operazioni di inserimento e di cancellazione di elementi in una lista lineare ordinata è necessario stabilire prima la posizione in cui va fatta la variazione.

Ricerca in una lista lineare ordinata /2

- Consideriamo una lista di nomi messi in ordine alfabetico nella quale intendiamo inserire un nuovo nome, ad esempio

pos.	Nome
1.	anna
2.	antonio
3.	carlo
4.	davide
5.	elena
6.	filippo
7.	giulio

- Per prima cosa è 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. È dunque necessario

    scandire la lista finché
    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)

Ricerca in una lista lineare ordinata /3

- La parte centrale dell'algoritmo di ricerca sarà
  - mentre il nome cercato segue nell'ordine alfabetico il nome corrente di lista passa al nome successivo nella lista
- Nel condurre una ricerca in una lista l va seguito l'ordine logico e non quello fisico. Se la struttura è realizzata con puntatori

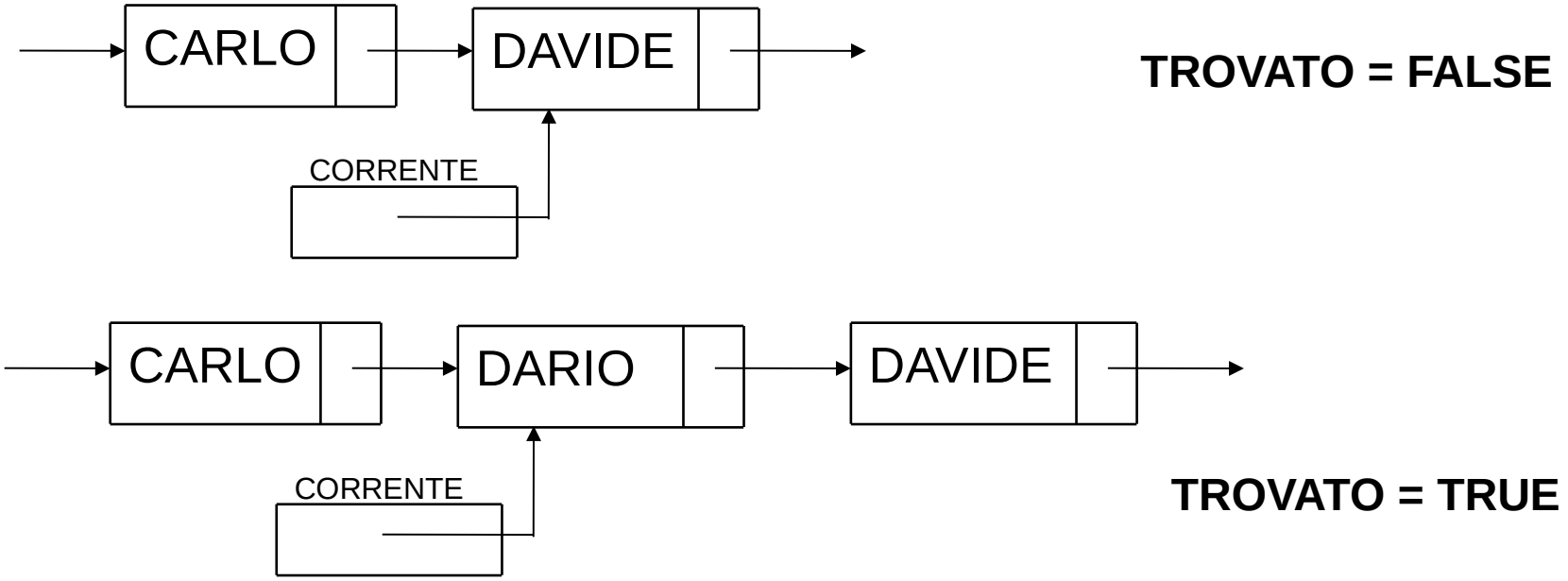
  cella: tipo strutturato con componenti
  - nome: di tipo tiponome
  - successivo: di tipo puntatore a cella

  lista: puntatore a cella
  posizione: puntatore a 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 basterà effettuare l'assegnazione corrente = succlista (corrente, l) per scorrere la catena finché non si trovi la posizione in cui il nome va inserito (in caso di inserimento) o è presente (in caso di ricerca)
- Alla procedura chiamante possiamo restituire una variabile booleana trovato che aiuti a sapere se l'elemento ricercato c'è o non c'è

Ricerca in una lista lineare ordinata /4

- Inoltre, può essere utile restituire il valore del puntatore corrente che punterà 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)

Ricerca in una lista lineare ordinata /4

- Cosa succede se l'elemento cercato e' l'ultimo elemento nella lista?
- Ci attendiamo che ci venga segnalato che quello è l'ultimo elemento e che la lista è 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)

Ricerca in una lista lineare ordinata /5

ALGORITMO

definisci il nome cercato e l'inizio lista
inizializza precedente e corrente a inizio lista
poni trovato a falso
mentre la ricerca continua e non e' finita la lista esegui:
    a) se il nome cercato precede alfabeticamente il nome corrente allora
        a. 1) interrompi la ricerca
    altrimenti
        a'. 1) poni precedente a corrente
        a'. 2) aggiorna corrente puntando all'elemento successivo
stabilisci se il nome cercato e' trovato
restituisci precedente, corrente e trovato

Ricerca in una lista lineare ordinata /6

```
Costanti: namesize = 20
Tipi:
nameformat: array di namesize elementi di tipo char
listpointer: puntatore a listnode
listnode: tipo strutturato con componenti
    - listname: di tipo nameformat
    - next: di tipo listpointer
/* ricerca nella lista ordinata la presenza di searchname */
listsearch(searchname: di tipo nameformat, current: di tipo listpointer per
    riferimento, previous: di tipo listpointer, listhead: di tipo listpointer;
    found: di tipo boolean per riferimento)
/* asserisci; listhead punta alla testa della lista */
previous = nil
notfound =  true
current = listhead  /* se vero indica il prossimo nodo da esaminare */
found = false
/* invariante: searchname appare alfabeticamente dopo name nel nodo precedente
 * se previous node = nil */
while ( (not found) and (current = nil) ) do
    with valore puntato da current do
    if searchname <= listname then
    notfound = false
    else
        /* sposta i puntatori al nodo successivo */
    previous = current
    current = next
    /* 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
```

Ricerca in una lista lineare ordinata /7

```
CON ASTRAZIONE

Costanti:
dimnome = 20
Tipi:
tiponome: array di dimnome elementi di tipo char
p_cella: puntatore a cella
cella: tipo strutturato con componenti
    - nome: di tipo tiponome
    - successivo: di tipo p_cella
lista: di tipo p_cella
posizione: di tipo p_cella

CERCA_IN_LISTA (cercato: di tipo tiponome; corrente: di tipo posizione per riferimento,   precedente:
di tipo posizione, L: di tipo lista, trovato: di tipo boolean per
    riferimento)
corrente = PRIMOLISTA(L)
precendente = corrente
continua = TRUE
trovato = FALSE
while (continua    and not FINELISTA(corrente, L)) do
     nome_corr = LEGGILISTA(corrente, L)
     if cercato <= nome_corr then
          continua = FALSE
     else
          precedente = corrente
          corrente = SUCCLISTA(corrente, L)
          if not FINELISTA(corrente, L) then
               TROVATO = (cercato = nome_corr)
```

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 può 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 è sostanzialmente fondato sul seguente ciclo:

    mentre non è finita lista1 e non è finita lista2
    confronta i due elementi correnti di lista1 e lista2
    memorizza l'elemento minore in lista3
    aggiorna l'elemento corrente della lista interessata

Fusione di liste ordinate /2

```
fusione(Lista1 di tipo Lista, Lista2: di tipo Lista, Lista3 : di tipo Lista per
riferimento)

crealista(Lista3)
p1 = primolista(Lista1)
p2 = primolista(Lista2)
p3 = primolista(Lista3)
while (not finelista(p1, Lista1) and not finelista (p2, Lista2)) do
    elem1 = leggilista(p1, Lista1)
    elem2 = leggilista(p2, Lista2)
    if elem1 < elem2 then
        inslista(elem1, p3, Lista3)
        p1 = succlista (p1, Lista1)
    else
        inslista(elem2, p3, Lista3)
    p2 = succlista (p2, Lista2)
    p3 = succlista (p3, Lista3)
while not finelista (p1, Lista1) do
    inslista (leggilista(p1, Lista1), p3, Lista3)
    p1 = succlista (p1, Lista1)
    p3 = succlista (p3, Lista3)
while not finelista (p2, Lista2) do
    inslista (leggilista(p2, Lista2), p3, Lista3)
    p2 = succlista (p2, Lista2)
    p3 = succlista (p3, Lista3)
```

Ordinamento di una lista

- Data una lista, ordinare gli elementi in ordine crescente rispetto ad una relazione d'ordine >=
- Utilizzeremo a tale scopo il metodo noto come ordinamento naturale (o natural merge sort), generalmente utilizzato per ordinare i file.
- Data una lista l = a1 a2 . . . an , diremo che una sottosequenza ai, ai+1 . . . ak costituisce una catena (o run) se accade che:

  ai-1 > ai
  aj <= aj+1 per ogni j = i, i+1, … , k -1
  ak > ak+1
- Una lista è quindi una sequenza di catene.
- Esempio
  - l = 82 16 14 15 84 25 77 13 75 4
  - l è ottenuta dalla concatenazione di sei catene di varia lunghezza

Ordinamento di una lista /2

- Una proprietà interessante è che la fusione di due sequenze di n catene produce una singola sequenza di n catene.
- Così, 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 più log2 n passi la lista sarà completamente ordinata
- Esempio
  - l = 82 16 14 15 84 25 77 13 75 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

Ordinamento di una lista /3

- Ma come determinare le sequenze di catene da fondere?
- Si può 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 può anche essere minore di n/2, poiché dopo la distribuzione due o più 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 può creare una nuova lista l il cui grado di ordinamento è maggiore. L'algoritmo di ordinamento alterna fasi di distribuzione a fasi di fusione, fino a quando si ottiene una lista con un'unica catena.

Ordinamento di una lista /4

```
OrdinamentoNaturale ( L: di tipo lista per riferimento)
repeat
    crealista(A)
    crealista(B)
    distribuisci(L, A, B)
    numero_catene = 0
    crealista(L)
    merge(A, B, L, numero_catene)
until numero_catene = 1
```

Ordinamento di una lista /4

La fase di distribuzione si basa sul richiamo della procedura copiacatena.

```
distribuisci (l: di tipo lista; a e b: di tipo lista per
riferimento);
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)
```

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 <).

Ordinamento di una lista /5

```
merge(A e B: di tipo lista; L: di tipo lista per riferimento;
    numero_catene : di tipo integer per riferimento )


pa = primolista(A)
pb = primolista(B)
pl = primolista(L)
while not finelista(pa,A) and not finelista(pb,B) do
    fondiCatena(pa, A, pb, B, pl, L)
    numero_catene = numero_catene + 1
    while not finelista(pa,A) do
        copiaCatena(pa, A, pl, L);
        numero_catene = numero_catene + 1
    while not finelista(pb,B) do
        copiaCatena(pb, B, pl, L)
        numero_catene = numero_catene + 1
```

Ordinamento di una lista /6

```
fondiCatena(pa: di tipo posizione per riferimento;
   A: di tipo lista;
   pb: di tipo posizione per riferimento;
   B: di tipo lista;
   pl: di tipo posizione per riferimento;
   L: di tipo lista per riferimento)
```

```
 repeat
  if leggilista(pa, A) < leggilista(pb, B) then
       copia(pa, A, pl, L, finecatena)
    if finecatena then
         copiaCatena (pb, B, pl, L)
  else
    copia(pb, B, pl, L, finecatena)
    if finecatena then
       copiacatena(pa, A, pl, L)
  until finecatena
```

Ordinamento di una lista /7

```
copiaCatena (  pa : posizione per riferimento;
   A : lista;
   pb : posizione per riferimento;
   B : lista );

  repeat
    copia (pa, A, pb, B, finecatena)
  until finecatena


copia (pa : posizione per riferimento;
   A : lista;
   pl : posizione per riferimento;
   L : lista per riferimento;    finecatena : boolean per riferimento);

  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))
```