Liste concatenate

Misael Mongiovì
Massimo Orazio Spata

Insiemi Dinamici

Gli insiemi sono fondamentali per l'informatica e la matematica. Mentre gli insiemi matematici sono immutabili, gli insieme manipolati dagli algoritmi possono crescere, ridursi o cambiare nel tempo.

Questi insiemi sono detti ***dinamici***.

Sequenze lineari

- Insieme finito di elementi disposti consecutivamente.
- Gli elementi hanno un ordine, che può essere o meno rilevante. Ogni elemento può quindi avere associato un indice di posizione (univoco).
- L'operazione più elementare è l'accesso ai singoli elementi.

Modalità di Accesso

- Accesso diretto (array)
  - Accediamo direttamente all'elemento a_i senza dover attraversare la sequenza.
- Accesso sequenziale (liste)
  - Raggiungiamo l'elemento attraversando la sequenza a partire da un suo estremo.
  - Costo O(i)
  - Accedere a a_{i+1} (da a_i) costa O(1)

Allocazione della Memoria

- Array e liste corrispondono a modi diversi di allocare la memoria
- Array: le locazioni di memoria corrispondenti ad elementi contigui sono contigue.
  - Il nome dell'array corrisponde alla locazione del primo elemento (a[0])
  - Se x è l'indirizzo di a, a[0] si trova all'indirizzo x
  - a[1] si trova all'indirizzo x+1, ecc.

Allocazione della Memoria

- Liste: gli elementi si trovano in locazioni di memoria non necessariamente contigue.
  - Ogni elemento memorizza, oltre al proprio valore, l'indirizzo del valore successivo.

- **Vantaggio**: le liste si prestano bene a implementare sequenze dinamiche.

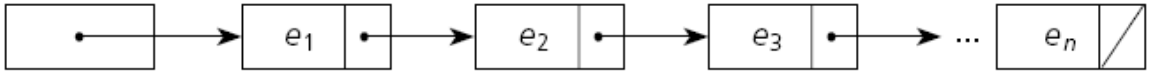
Array di dimensione variabile

- E’ necessario poter ridimensionare l’array (allocarne uno nuovo di diversa dimensione)
- L’operazione richiede O(n) tempo per ciascuna nuova allocazione
- Si può fare meglio?
  - Costo cumulativo O(n) per ciascun gruppo di *n* operazioni consecutive
  - Costo (distribuito): O(1) per operazione

Liste

una **lista concatenata** è una sequenza di elementi collegati l'uno all'altro da un puntatore

gli elementi si compongono cioè di due parti: una che contiene l'informazione (chiave) e l'altra costituita da un puntatore al successivo elemento



- *Liste semplici:* ogni elemento contiene un unico puntatore che lo collega al nodo successivo
- *Liste doppiamente concatenate:* ogni elemento contiene due puntatori, uno all'elemento precedente e l'altro al successivo
- *Liste circolari semplici:* l'ultimo elemento si collega al primo in modo che la lista può essere attraversata in modo circolare.
- *Liste circolari doppiamente concatenate:* l'ultimo elemento si collega al primo elemento e viceversa

Dichiarazione di un elemento

si può definire un elemento, o "nodo" della lista, mediante i costrutti struct o class

```
struct Node {
 int data;
 Node* next;
};
```

```
class Node {
public :
 int data;
 Node* next;
};
```

Costruzione di una lista

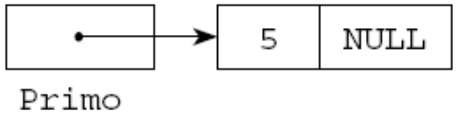
La creazione di una lista concatenata implica i seguenti passi:

Passo 1. Dichiarare il tipo del nodo ed il puntatore di testa
Passo 2. Allocare memoria per un nodo utilizzando l'operatore new assegnandone l'indirizzo ad un puntatore
Passo 3. Creare iterativamente il primo elemento ed i successivi
Passo 4. Ripetere finché vi siano nodi da immettere

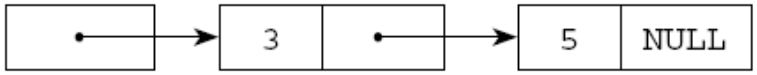
```
class Node {
public :
   Node* next;
   int data;
   Node (Node* n, int d): next(n), data(d) {}
};
```

Costruzione di una lista

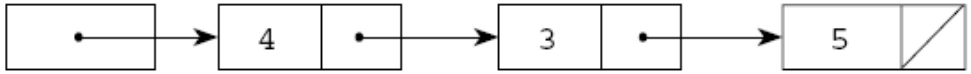
```
Node* primo = NULL; // lista vuota
primo = new Node(primo, 5); // primo elemento
```




```
primo = new Nodo(primo, 3);
```



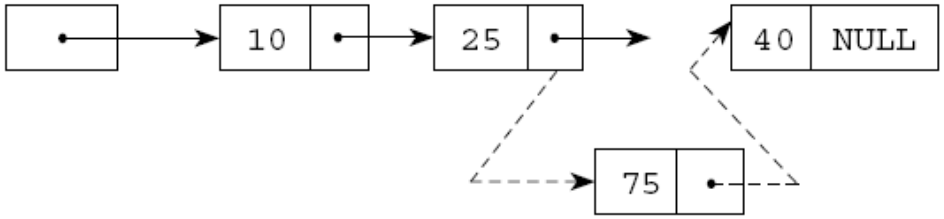

```
primo = new Nodo(primo, 4);
```

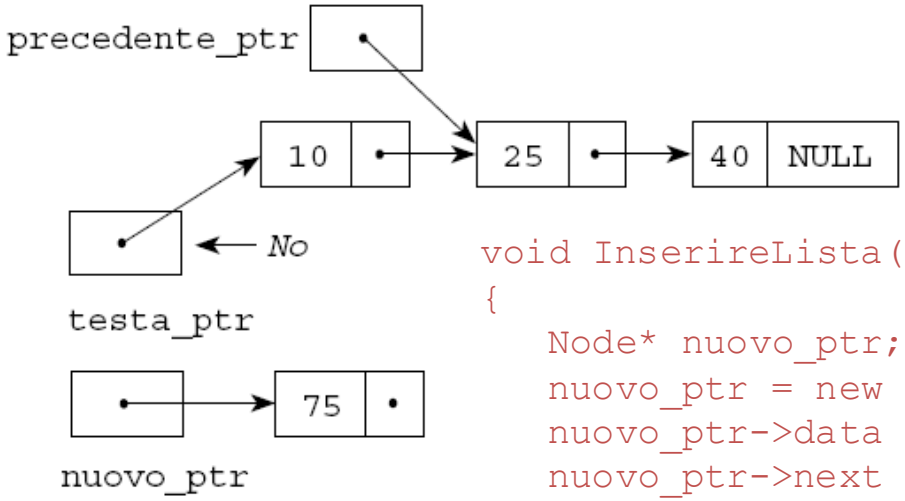


Inserimento in testa

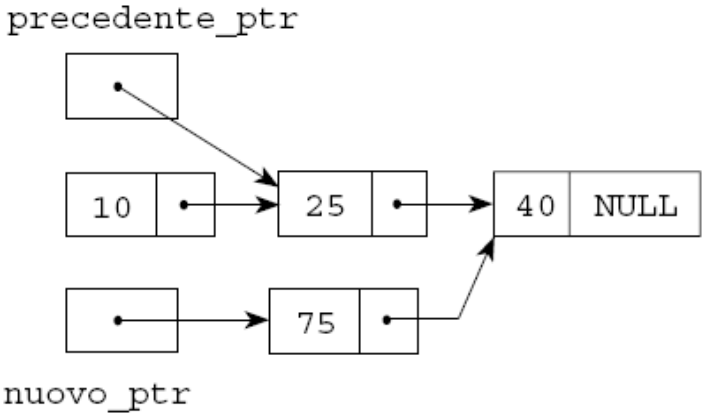
```
void InserTestaLista (Nodo* head, int data)
{
   Node* nuovo_ptr;
   nuovo_ptr = new Node;     // assegna nuovo nodo
   nuovo_ptr->data = data;   // mette elemento in nuovo nodo
   nuovo_ptr->suc = head;    // collega nuovo nodo
                             // al fronte della lista
   head = nuovo_ptr ;        // muove puntatore testa
                             // e punta al nuovo nodo
}
```

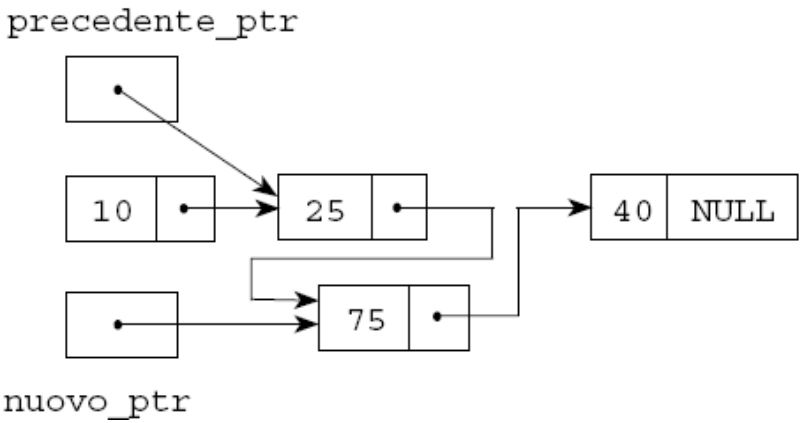
Inserimento in un posto predeterminato






```
void InserireLista(Node* precedente_ptr,int data)
{
   Node* nuovo_ptr;
   nuovo_ptr = new Nodo;
   nuovo_ptr->data = data;
   nuovo_ptr->next = precedente_ptr->next;
   precedente_ptr->next = nuovo_ptr;
}
```





Ricerca di un elemento

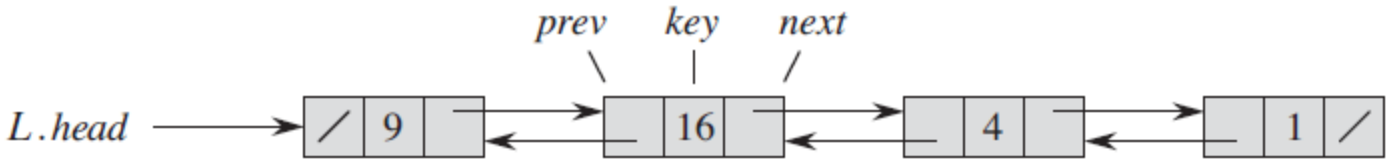
- Molto simile alla ricerca in array
- Semplicemente alla fine viene restituito un puntatore all'elemento trovato

Rimozione di un elemento

1. cercare il nodo che contiene il dato, farlo puntare da `pos` e far puntare da `ant` il nodo che lo precede (se non è il primo)
2. mettere il puntatore `suc` del nodo puntato da `ant` al puntatore `suc` del nodo puntato da `pos`
3. se `pos` è il puntatore di testa si mette `p` al campo `suc` del nodo puntato da `pos`
4. si libera la memoria occupata dal nodo puntato da `pos`

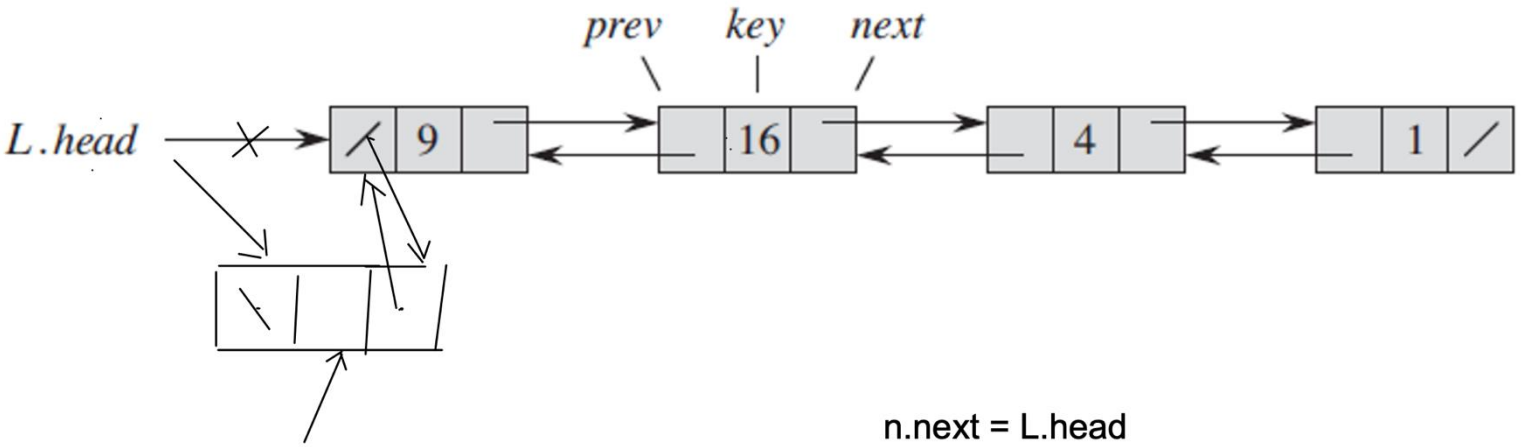
```
void Cancellare(int data) // l'attributo 'p' punta alla testa della lista
{
    Nodo* ant=NULL, *pos =p;
      // ricerca del nodo da cancellare
    ......
      // si cancellerà il nodo puntato da pos
    if(ant!=NULL)
       ant->next = pos->next;      // non è il primo
    else
        p = pos->next;    // è il primo
    delete pos;
 }
```

Lista doppiamente concatenata




```
class Nodo
{
   public :
   Nodo *prev, *next;
   int data;                        // dato di tipo intero
   Nodo(Nodo *pre, Nodo *suc, int d = 0)
     : prev(pre), next(suc), data(d) {}
};
```

Inserimento in testa




```
n.next = L.head
n.prev = nullptr
if (L.head != nullptr)
    L.head.prev = n
L.head = n
```

Inserimento in testa a lista doppiamente concatenata

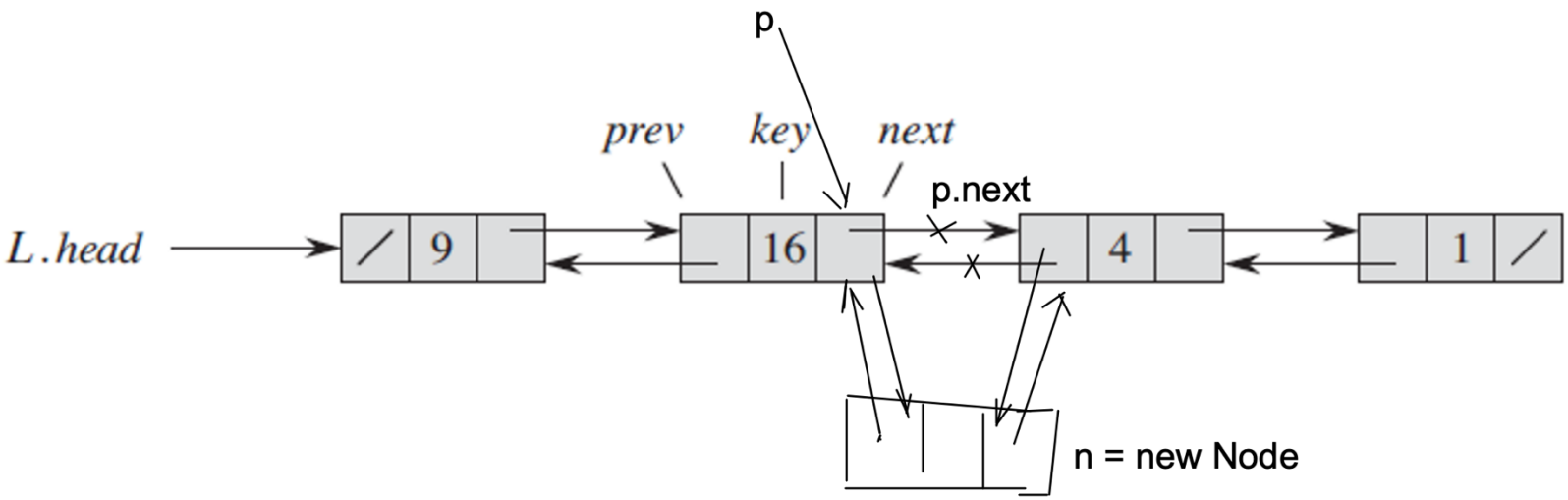
1. allocare dinamicamente un nuovo nodo, assegnarne l'indirizzo ad un puntatore nuovo e copiarvi dentro l'elemento e che si vuole inserire

2. assegnare il campo next del nuovo nodo al puntatore di testa head, ed il campo prev del primo nodo, se esiste, al nodo nuovo. Se la lista è vuota non fare niente

3. far puntare head al nodo nuovo

```
LIST-INSERT(L, x)
1  x.next = L.head
2  if L.head ≠ NIL
3       L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

Inserimento dopo un elemento p




```
n.next = p.next
n.prev = p
if (p.next !=nullptr)
   p.next.pred = n
p.next = n
```

Inserimento in una determinata posizione (pos) in lista doppiamente concatenata

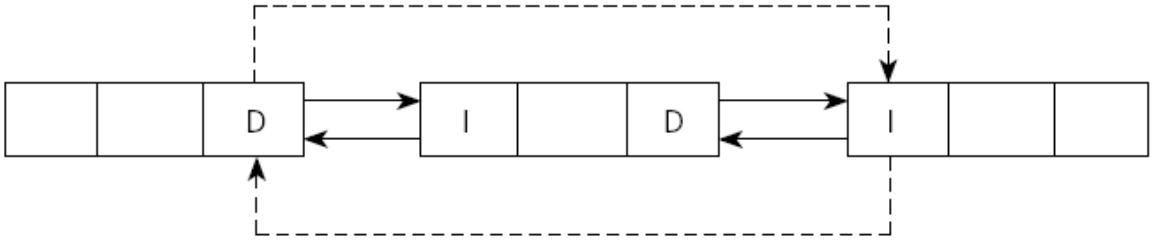
1. cercare la posizione dove bisogna inserire il nodo

2. allocare dinamicamente memoria per il nuovo nodo e copiarvi il contenuto informativo

3. far puntare il campo next del nuovo nodo al nodo pos->next l'attributo prev del nodo successivo di pos, se esiste, deve puntare al nuovo nodo

4. far puntare l'attributo next del puntatore pos->prev, se esiste, al nuovo nodo; l'attributo prev del nuovo nodo, farlo puntare a pos->prev

Rimozione da lista doppiamente concatenata



1. ricerca del nodo che contiene il dato; si deve avere l'indirizzo del nodo da rimuovere (`p`)
2. l'attributo `next` del nodo antecedente (`p->prev`) deve puntare all'attributo `next` del nodo da rimuovere (se non è il primo nodo della lista); se `p` è il primo della lista, l'attributo `head` deve puntare all'attributo `next` del nodo da rimuovere `p`
3. l'attributo `prev` del nodo successivo a quello da cancellare deve puntare all'attributo `prev` del nodo da rimuovere (se non è l'ultimo nodo della lista); se `p` è l'ultimo nodo della lista non fare niente
4. si libera la memoria occupata dal nodo rimosso `p`

Inserimento in lista circolare

1. allocare memoria al nodo `nuovo` e riempirlo di informazione

2. se la lista è vuota far puntare il `next` di `nuovo` al nodo stesso, e far puntare la testa della lista al `nuovo`

3. se la lista non è vuota si deve decidere dove collocare il `nuovo`, conservando l'indirizzo del nodo antecedente `ant`; collegare l'attributo `next` di `nuovo` con l'attributo `next` del nodo antecedente `ant`; collegare l'attributo `next` del nodo antecedente `ant` con il `nuovo`.

Rimozione da lista circolare

1. cercare il nodo `ptrnodo` che contiene il dato conservando un puntatore all'antecedente `ant`

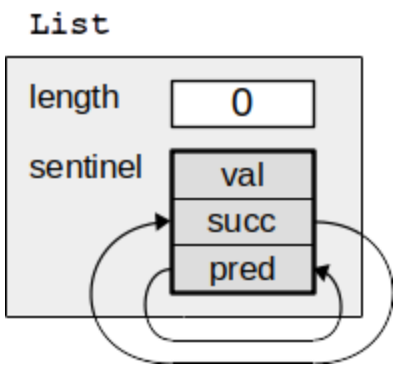
2. far puntare il campo `next` del nodo antecedente `ant` dove punta il campo `next` del nodo da cancellare; se la lista conteneva un solo nodo si mette a `NULL` il puntatore `head` della lista

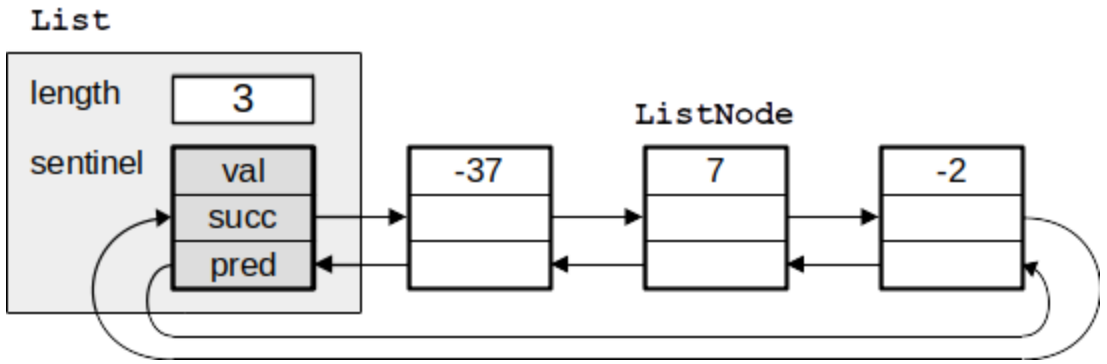
3. se il nodo da rimuovere è quello puntato dal puntatore d'accesso alla lista, `head`, e la lista contiene più di un nodo, si modifica `head` per mandarlo dove punta il campo `next` del nodo puntato da `head` (se la lista rimane vuota fare prendere a `head` il valore `NULL`).

4. da ultimo, si libera la memoria occupata dal nodo eliminato

Lista circolare con sentinella

In una lista con sentinella è presente un nodo speciale, detto "sentinella", che non contiene alcuna informazione utile ma serve solo per marcare l'inizio (o la fine) della lista. La sentinella consente di accedere in tempo O(1) al primo o ultimo elemento della lista: il primo elemento della lista è il successore della sentinella, mentre l'ultimo elemento della lista è il predecessore.





Esercizi

Implementare:

- Una lista concatenata di interi
- Una lista concatenata di tipi generici
- Una lista concatenata ordinata
- Una lista doppiamente linkata
- Una lista doppiamente linkata ordinata
- Una lista (doppia) circolare (con e senza sentinella)

Per ciascuna struttura dati implementare le operazioni di inserimento, rimozione, ricerca e l'operatore operator<<.

Esercizio

Definire una semplice classe Rettangolo. Successivamente definire:

- Una lista di rettangoli con un metodo per aggiungere rettangoli e un metodo che prende in input due valori numerici, ed elimina dalla lista tutti i rettangoli con area compresa tra i due valori forniti dall'utente