Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa 09

Strutture collegate - 1

A. Miola

Febbraio 2008

Contenuti

- Limitazioni degli array nella gestione della loro dimensione
- Gestione dinamica della memoria
- Definizione di strutture collegate lineari
- Inserimenti, eliminazioni e modifiche di elementi
- Scansione, accesso e ricerca di elementi

Gestione della memoria

- In questa dispensa verrà affrontato il problema della gestione della memoria durante l'esecuzione del programma (gestione dinamica della memoria) al fine di memorizzare le informazioni di interesse per una applicazione
- Inizialmente saranno discusse le limitazioni degli array nella gestione dinamica della memoria e successivamente saranno presentate le strutture collegate lineari che offrono un meccanismo estremamente flessibile per la gestione dinamica della memoria

Limitazioni degli array

- La dimensione di un array è stabilita al momento della sua creazione e non può essere modificata successivamente
- Gli elementi di un array sono allocati sequenzialmente in memoria
- Un array utilizza uno spazio di memoria proporzionale alla sua dimensione, indipendentemente dal numero di elementi validi effettivamente memorizzati
- Operazioni di inserimento o cancellazione di elementi in un array richiedono la successiva creazione di nuovi array di dimensione più grande o più piccola a seconda delle esigenze

Gestione della dimensione di un array . . .

- Alcune di tali limitazioni possono essere superate gestendo in maniera opportuna gli array, in modo da allocare sufficiente memoria per contenere i dati di una applicazione dinamicamente durante la sua esecuzione
- Consideriamo ad esempio il problema della gestione di un elenco di persone (rappresentato da un array di oggetti della classe Persona), come tipicamente potrebbe avvenire in una procedura di gestione di una anagrafe comunale

. . . Gestione della dimensione di un array

- **Nell'esempio – ovviamente - non è nota a priori la quantità di persone da gestire**
  - **Possiamo quindi dimensionare un array a con un valore di stima massima - sovradimensionamento**
  - **Ogni elemento dell'array a sarà un riferimento ad una persona e via via che si aggiungono persone vengono utilizzati successivi elementi dell'array a e in ogni istante potremmo avere una delle due situazioni seguenti**
    - **Non tutti gli elementi dell'array a hanno assegnato un valore – quindi c'è un uso parziale delle disponibilità di memoria**
    - **Tutti gli elementi dell'array a hanno assegnato un valore – quindi non sarà più possibile inserire nuove persone nell'elenco**

. . . Gestione della dimensione di un array

- L'inserimento di una nuova persona nell'elenco risulta quindi immediato nel caso di posizioni dell'array ancora disponibili
- Nel caso in cui non ci siano posizioni disponibili dovremo
  - Creare un nuovo array b – ad esempio di dimensione doppia di quello a originario
  - Copiare nel nuovo array b tutti gli elementi dell'array a e quindi inserire la nuova persona nell'elenco – cioè nell'array b
  - Assegnare al riferimento a dell'array originario il riferimento al nuovo array b

. . . Gestione della dimensione di un array

- La cancellazione di una persona dall'elenco richiede lo spostamento – all'indietro di una posizione – di tutti gli elementi dell'array che seguono quello da eliminare, conservando l'allocazione sequenziale della memoria
- Una soluzione più efficiente richiede anche di valutare quante posizioni sono ancora disponibili dopo la cancellazione effettuata ed eventualmente ottimizzare lo spazio – ad esempio se le disponibilità superano la metà del totale si può
  - Creare un nuovo array b – ad esempio di dimensione metà di quello a originario
  - Copiare nel nuovo array b tutti gli elementi dell'array a
  - Assegnare al riferimento a dell'array originario il riferimento al nuovo array b

Gestione dinamica della memoria . . .

- L'esempio mostra l'esigenza di una gestione dinamica della memoria in cui vengono allocate risorse di memoria ogni volta che ce n'è bisogno e vengono rilasciate quando queste non sono più necessarie
- Nel momento in cui vogliamo definire e manipolare sequenze di oggetti, abbiamo quindi bisogno di un ulteriore meccanismo per allocare memoria in maniera dinamica, che deve tener conto del fatto che in generale la dimensione delle sequenze non è nota a priori e varia nel tempo ad ogni operazione di inserimento o cancellazione

. . . Gestione dinamica della memoria . . .

- Gli array prevedono – di per sé - una gestione statica della dimensione della memoria, in quanto una volta creati la loro dimensione non può essere modificata
  - Ricordiamo che la creazione dell'array avviene al tempo di esecuzione – *attenzione quindi in questo caso all'uso della parola statica*
  - La gestione dinamica della memoria viene quindi "simulata" come nell'esempio visto
- In un array la successione logica degli elementi di una sequenza viene realizzata con una successione fisica degli elementi che vengono allocati sequenzialmente

. . . Gestione dinamica della memoria . . .

- La strada da seguire è quindi quella di svincolare la sequenzialità fisica da quella logica degli elementi di una sequenza allocando i singoli elementi in modo casuale in memoria
- Le strutture collegate che ora introduciamo sono definite quindi appositamente per consentire di allocare e deallocare memoria in maniera dinamica, a seconda delle esigenze del problema nella memorizzazione dei dati di interesse per l'applicazione, e quindi della loro dimensione variabile nel tempo

. . . Gestione dinamica della memoria

- In effetti abbiamo già visto la possibilità di allocare memoria per creare nuovi oggetti, invocando tramite l'operatore new i metodi costruttori di una classe
- Le strutture collegate soddisfano pienamente a questa esigenza e vengono realizzate in maniera tale da consentire facilmente la modifica della propria struttura, gli inserimenti e le cancellazioni in posizioni specifiche
- Esse vanno ad affiancare le strutture di array come ulteriore strumento di rappresentazione e gestione di sequenze di dati

Strutture collegate lineari

- Introduciamo quindi le strutture collegate lineari, dette anche liste, che consentono di rappresentare sequenze di oggetti - cioè in cui ogni elemento ha un solo successore che non necessariamente sarà allocato nelle celle adiacenti di memoria
- Esistono anche le strutture collegate non lineari - gli alberi - che sono un esempio di sequenze in cui ogni elemento ha uno o più successori

La classe NodoLista

- La classe base per una struttura collegata lineare (o **lista**) è una classe i cui oggetti rappresentano le informazioni associate ad un singolo elemento (o **nodo**) della struttura

```
class NodoLista {
  public TipoElemento info;
  public NodoLista next;

}
```

- Questa classe definisce
  - due variabili di istanza pubbliche:
    - `info` che contiene l'informazione di interesse e può essere di un qualsiasi tipo da specificare
    - `next` che è un riferimento al prossimo (successivo) nodo (elemento) della lista – *c'è una definizione induttiva*
  - **implicitamente** un costruttore `NodoLista`

Creazione e collegamento di nodi . . .

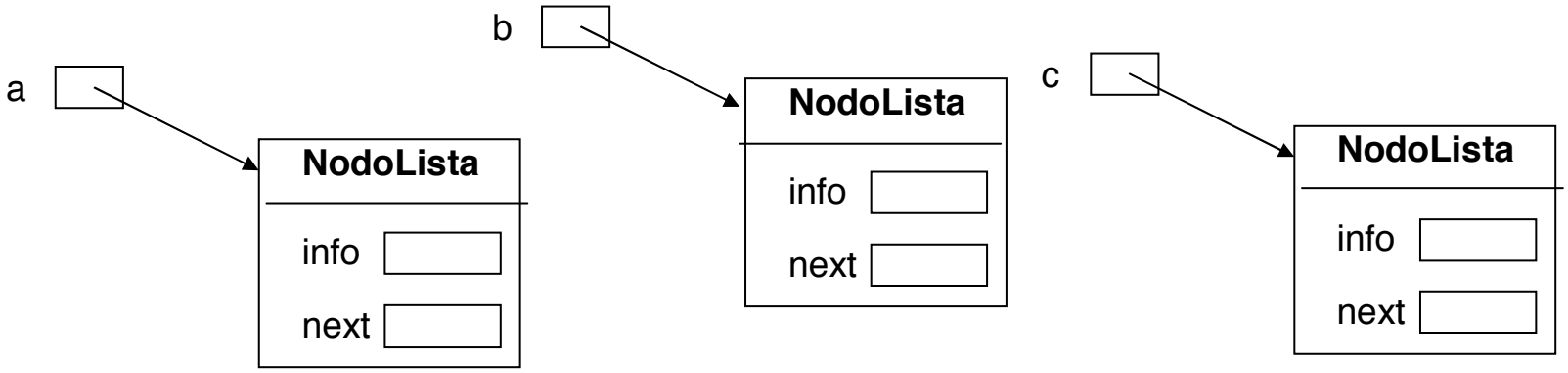
- Il seguente frammento di programma crea una struttura lineare di 3 nodi contenenti stringhe

```
class NodoLista {
    public String info;
    public NodoLista next;
}
public class ProvaLista {
    public static NodoLista crea3NodiABC() {
            NodoLista a = new NodoLista();
            NodoLista b = new NodoLista();
            NodoLista c = new NodoLista();
            a.info = "A"; a.next = b;
            b.info = "B"; b.next = c;
            c.info = "C"; c.next = null;
            return a; }
}
```

. . . Creazione e collegamento di nodi . . .

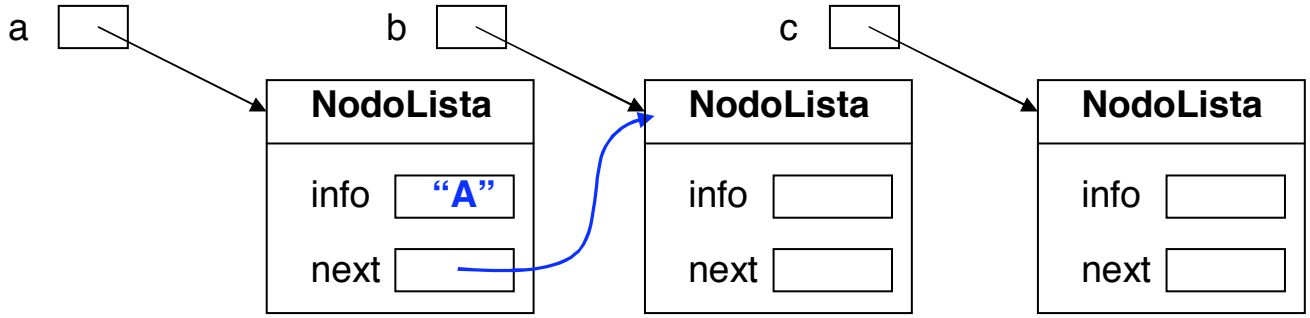
Le seguenti figure illustrano passo passo la rappresentazione in memoria della lista creata con il metodo `crea3NodiABC` descritto sopra

```
NodoLista a = new NodoLista();
NodoLista b = new NodoLista();
NodoLista c = new NodoLista();
```



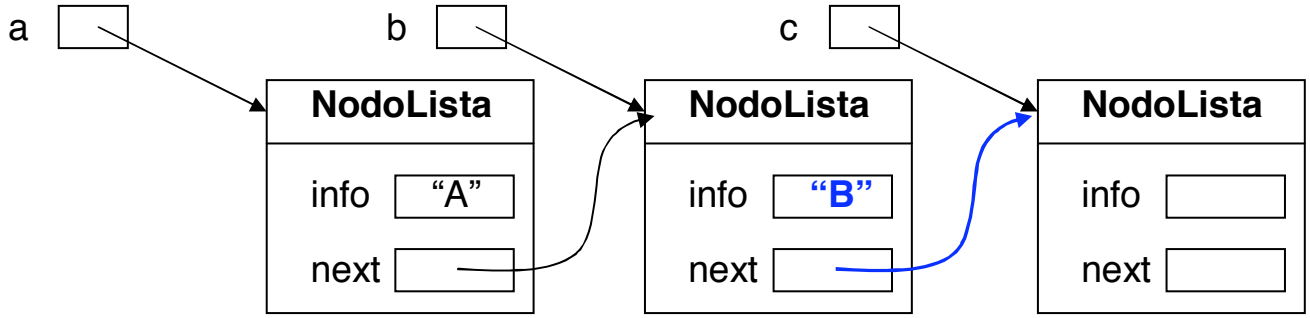
. . . Creazione e collegamento di nodi . . .

```
a.info = "A"; a.next = b;
```



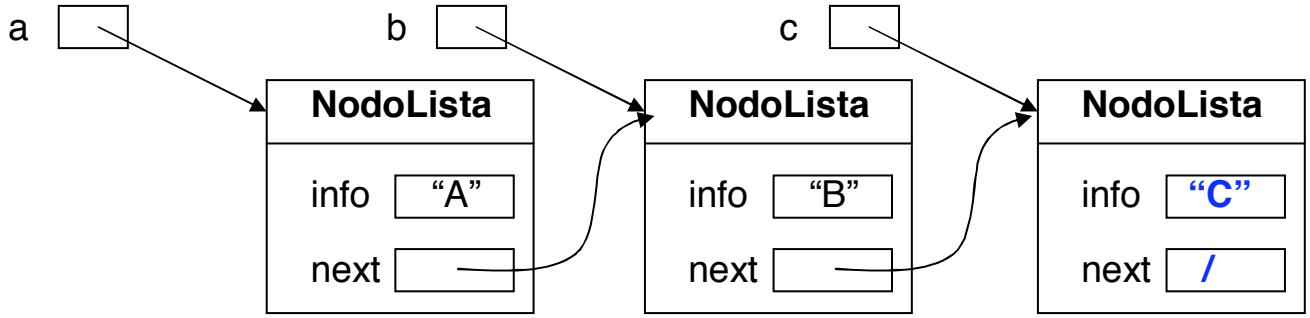
. . . Creazione e collegamento di nodi . . .

```
b.info = "B"; b.next = c;
```



. . . Creazione e collegamento di nodi

```
c.info = "C"; c.next = null;
```



Operazioni sulle strutture collegate lineari

- **Le operazioni più comuni sulle strutture collegate lineari sono:**
  - modifica, inserimento ed eliminazione di un nodo,
  - accesso ad un nodo,
  - ricerca di un'informazione nella struttura,
  - verifica che la struttura sia vuota,
  - calcolo della dimensione della struttura,
  - conversioni da e verso array, stringhe, ecc.
- **Si noti che :**
  - Le operazioni di inserimento ed eliminazione di un elemento modificano la configurazione della struttura collegata
  - L'operazione di modifica dell'informazione in un nodo non ne modifica la struttura, ma solo il suo contenuto
  - Le altre operazioni invece non modificano né la struttura né il contenuto della struttura collegata lineare

Nota bene

- Le operazioni che sono esaminate in questa dispensa vengono descritte attraverso frammenti di codice corredati, in genere, di rappresentazioni grafiche che intendono favorire la comprensione dei meccanismi tecnici alla base dell'implementazione
- Nella successiva dispensa verranno presentate varie applicazioni che utilizzaranno le tecniche di implementazione che qui vengono introdotte

Modifica dell'informazione in un nodo

- Sia p (il riferimento a) un nodo qualsiasi della struttura collegata, vogliamo modificare la sua informazione con il valore della stringa x

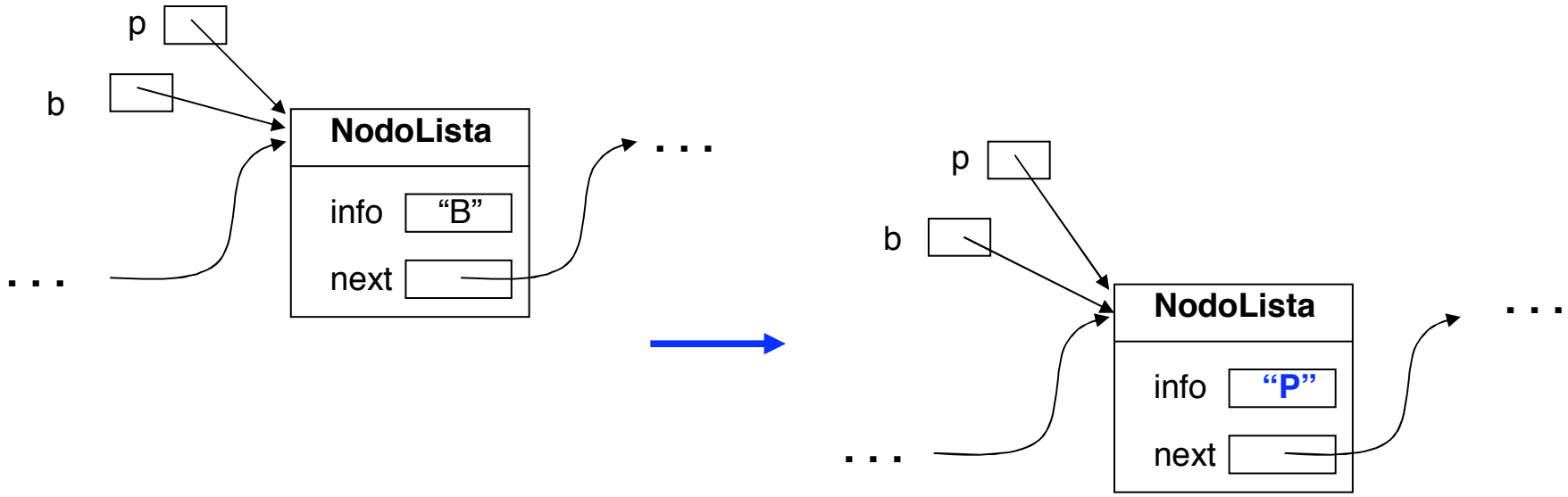
```
NodoLista p = ...// p è il riferimento al nodo
String x = ... // x è la stringa che vogliamo
               // memorizzare in p
p.info = x;
```

In questo caso non è necessario modificare la struttura collegata, ma solamente il contenuto dell'informazione del nodo in questione

Rappresentazione dell'operazione

Ad esempio

```
NodoLista p = b       // p è il riferimento al nodo
                      // b dell'esempio precedente
String x = "P"        // x è la stringa che vogliamo
                      // memorizzare in p
p.info = x;
```



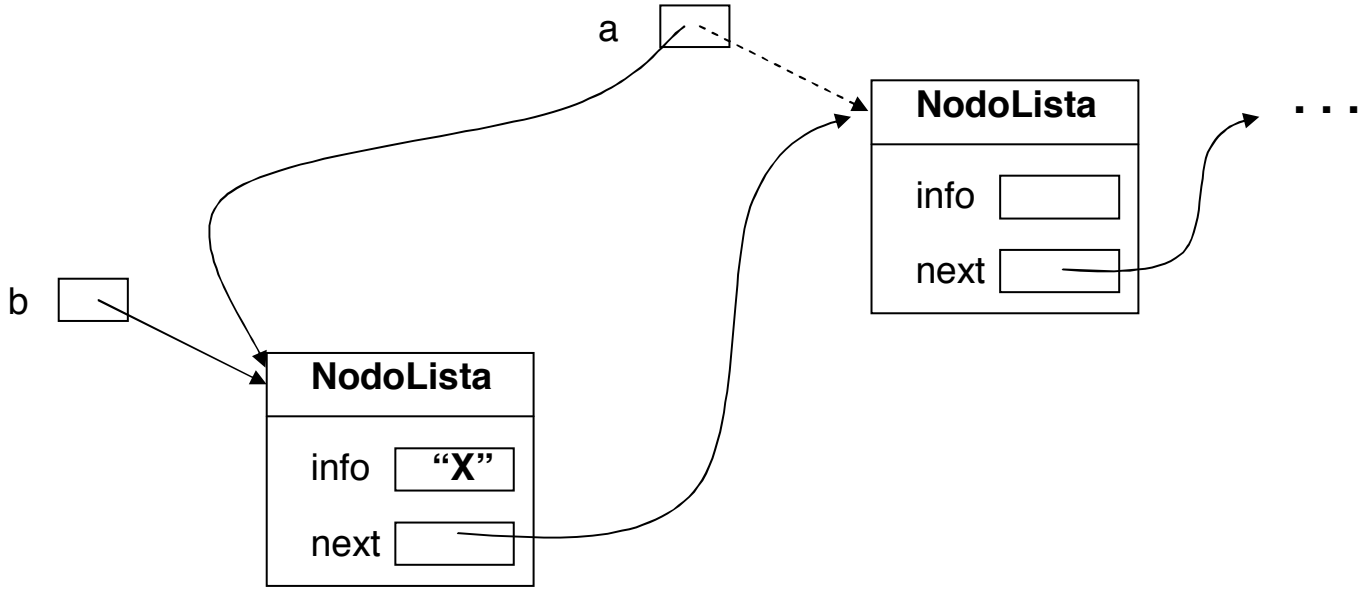
Inserimento di un nodo in prima posizione

- Sia a il (riferimento al) primo nodo della struttura, vogliamo inserire un nuovo nodo, prima di a, contenente come informazione una stringa x e modificare a in modo da mantenere il riferimento al primo nodo della struttura
- Si tratta di aggiungere un elemento all'inizio della struttura ovvero in testa

```
NodoLista a = ...
String x = "X"
NodoLista b = new NodoLista();
b.info = x;
b.next = a;
a = b;
```

Rappresentazione dell'operazione

Le frecce tratteggiate indicano il contenuto delle variabili prima dell'esecuzione delle istruzioni considerate, mentre quelle piene indicano il contenuto delle variabili modificate dall'esecuzione delle istruzioni

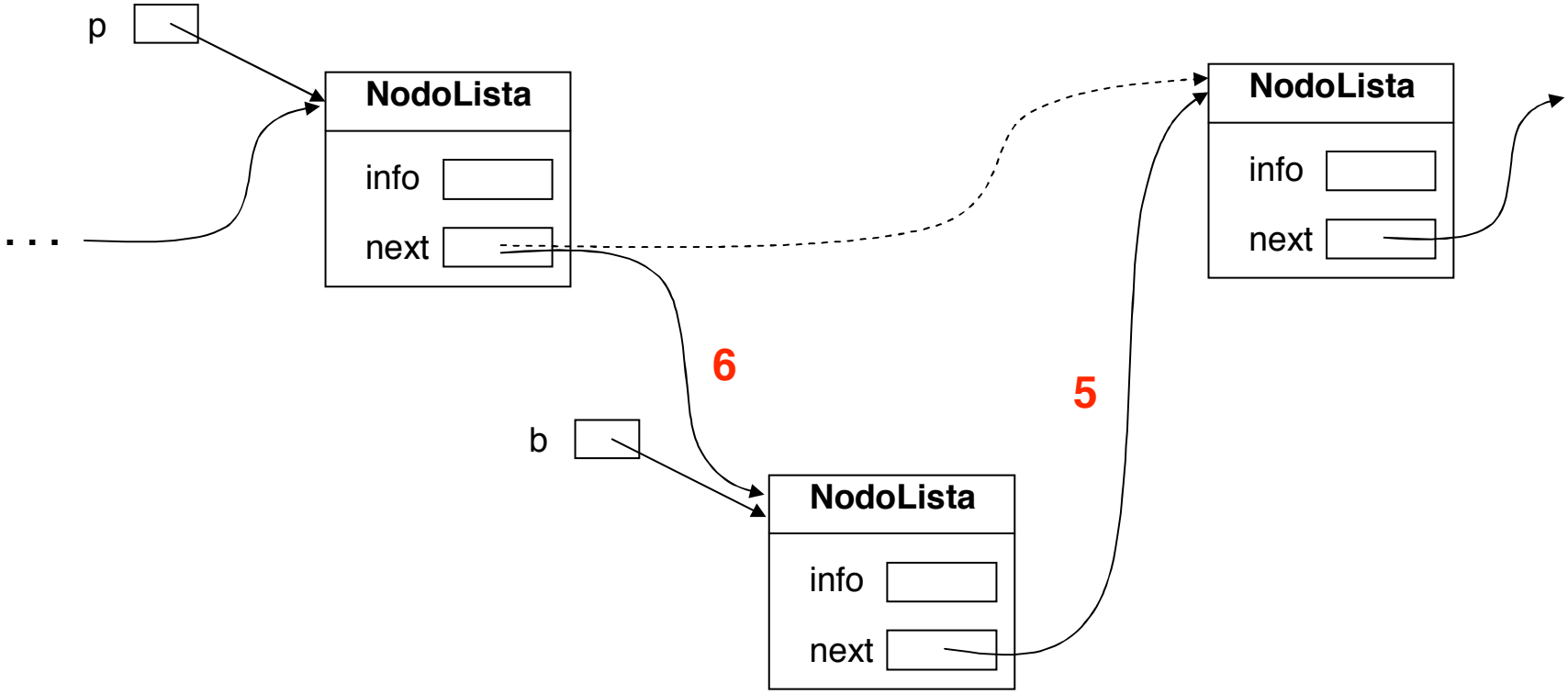


Inserimento di un nodo in posizione intermedia

- Vogliamo inserire un nuovo nodo, contenente la stringa x, dopo il nodo (riferito da) p

```
String x = ...                          // 1
NodoLista p = ...                       // 2
NodoLista b = new NodoLista();          // 3
b.info = x;                             // 4
b.next = p.next;                        // 5
p.next = b;                             // 6
```

Rappresentazione dell'operazione



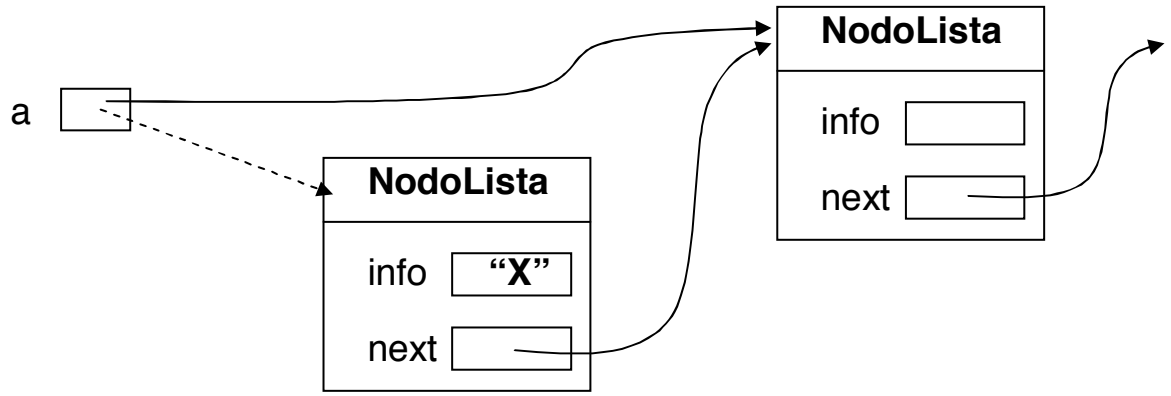
L'ordine con cui vengono eseguite le sei operazioni indicate è significativo, in quanto, ad esempio, l'inversione delle ultime due istruzioni 5 e 6 comporterebbe la perdita del riferimento al nodo successivo a p

Eliminazione di un nodo in prima posizione

- Sia a il (riferimento al) primo nodo della struttura, vogliamo eliminare il nodo puntato da a e modificare a in modo da mantenere il riferimento al nuovo primo nodo (cioè l'ex-secondo - se esiste) della struttura

```
NodoLista a = ...
a = a.next;
```

Rappresentazione dell'operazione



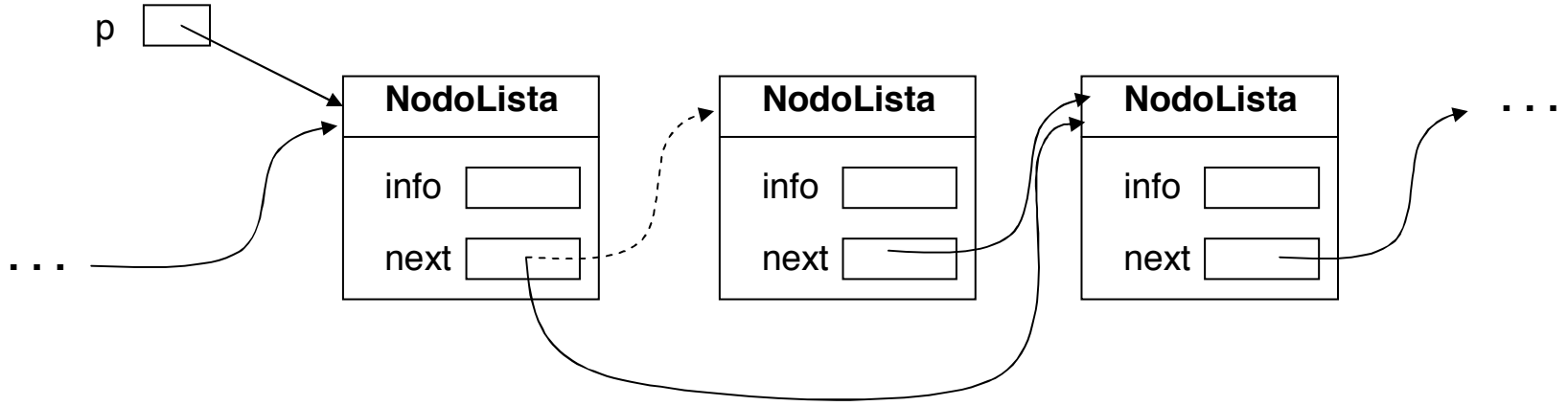
L'operazione descritta considera correttamente anche il caso in cui l'elemento da eliminare è l'unico della lista (caso in cui a.next vale null), nel qual caso si restituisce la lista vuota (cioè a = null).

Che succede del nodo eliminato ?

Eliminazione di un nodo in posizione intermedia

- Vogliamo eliminare il nodo successivo al nodo (riferito da) p

```
NodoLista p = ...
p.next = p.next.next;
```



Anche in questo caso l'operazione tratta correttamente anche il caso in cui l'elemento da eliminare sia l'ultimo

Ciclo di scansione di una struttura collegata lineare . . .

- Per effettuare operazioni su tutti gli elementi di una struttura collegata lineare, oppure su uno specifico elemento caratterizzato da una proprietà, è necessario effettuare cicli di scansione per accedere a tutti gli elementi

. . . Ciclo di scansione di una struttura collegata lineare

- Lo schema di ciclo per accedere a tutti i nodi della struttura il cui primo nodo è puntato dalla variabile **a** è il seguente

```
NodoLista a = ...
NodoLista p = a;
while (p!=null) {
      // elaborazione del nodo puntato da p
      p = p.next;
}
```

Inserimento di un nodo in ultima posizione

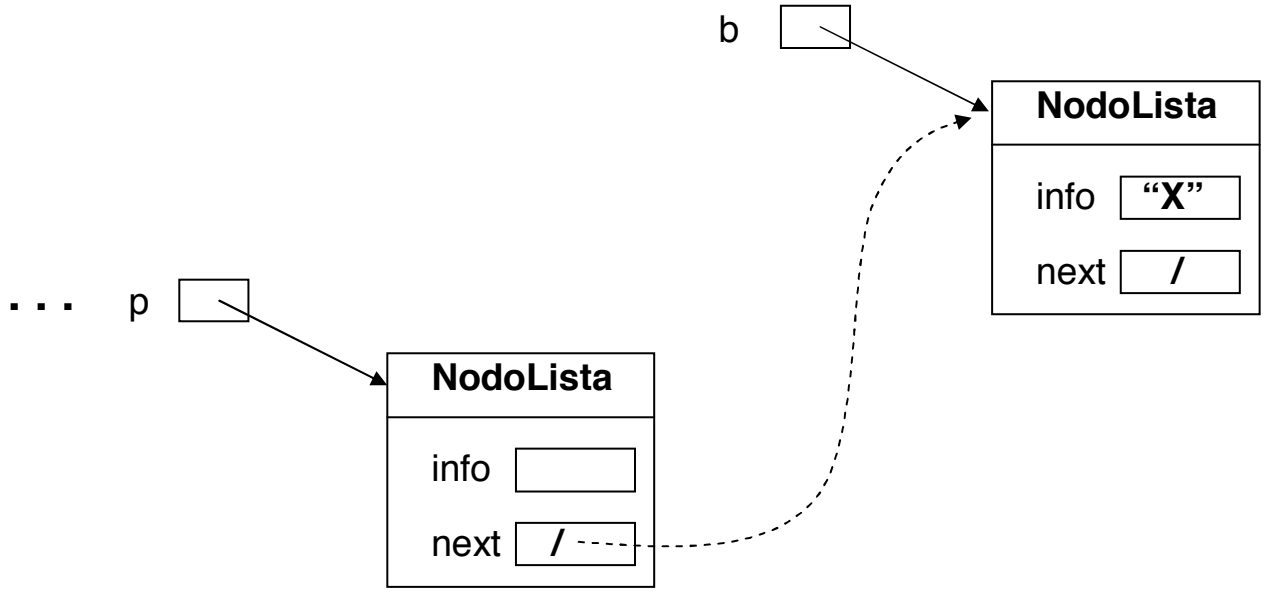
- Sia a il (riferimento al) primo nodo di una struttura, vogliamo inserire (aggiungere) un nuovo nodo, contenente come informazione una stringa x, come ultimo elemento della struttura, senza modificare il riferimento a al primo nodo della struttura
- Si tratta di aggiungere un elemento in fondo alla struttura ovvero in coda
- Bisogna prima scandire tutta la struttura, a partire dal suo primo nodo fino all'ultimo (quello che ha null come valore della variabile next)

Inserimento di un nodo in ultima posizione

- Il riferimento all'ultimo nodo può essere individuato con una operazione di scansione simile a quella vista in precedenza

```
NodoLista a = ...
NodoLista p = a;
while (p.next !=null) p = p.next;
   // il nodo puntato da p e' l'ultimo
String x = "X"
NodoLista b = new NodoLista();
b.info = x;
b.next = null
p.next = b;
```

Rappresentazione dell'operazione



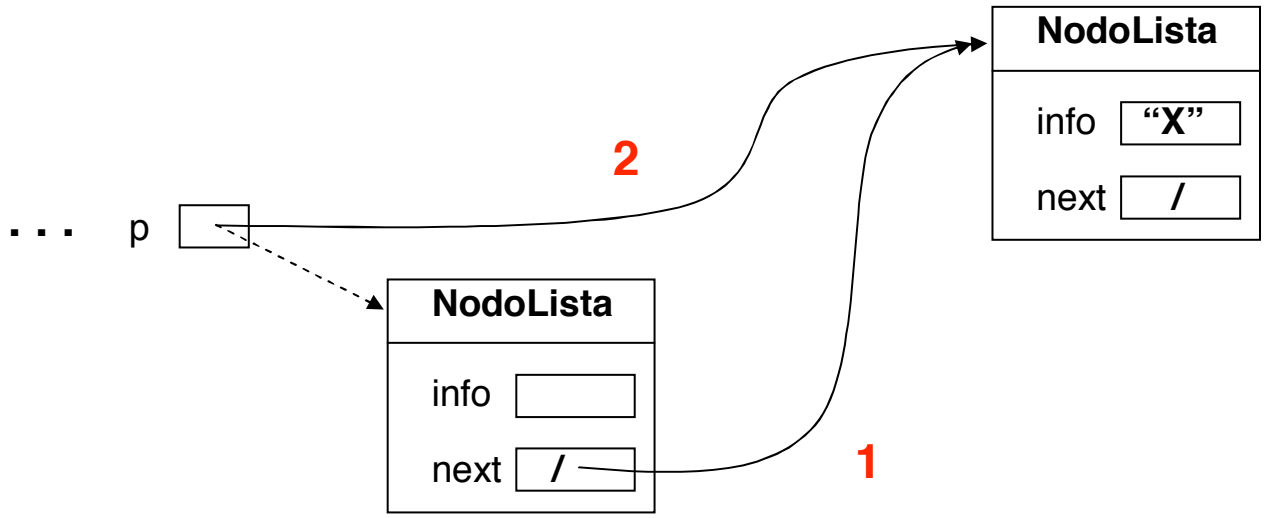
Ovviamente il riferimento a al nodo iniziale non è cambiato mentre la struttura complessiva sì perché ha un elemento in più alla fine

Inserimento di un nodo in ultima posizione

- In effetti posso anche procedere in modo alternativo come segue

```
NodoLista a = ...
NodoLista p = a;
while (p.next !=null) p = p.next;
   // il nodo puntato da p e' l'ultimo
String x = "X"
p.next = new NodoLista();
p = p.next;
p.info = x;
p.next = null;
```

Rappresentazione dell'operazione



Ovviamente, anche in questo caso, il riferimento a al nodo iniziale non è cambiato mentre è da notare che il precedente ultimo nodo è ora il penultimo e non può più essere acceduto direttamente ma solo tramite una scansione opportuna dell'intera struttura a partire dal nodo iniziale

Ricerca di una informazione nella struttura . . .

- Sia **a** il (riferimento al) primo nodo della struttura collegata e **x** una stringa, vogliamo calcolare il riferimento al nodo che contiene la stringa **x** come informazione

. . . Ricerca di una informazione nella struttura

```
NodoLista a = ...
String x = ...
NodoLista p = a;
while (p!=null && !p.info.equals(x))
            p = p.next;
if (p==null) {
   // La stringa x non e' presente
   // nella struttura
      ... }
else {
   // p e' il riferimento al nodo
   // contenente la stringa x

      ... }
```

Accesso ad un nodo tramite la sua posizione . . .

- Sia a il (riferimento al) primo nodo della struttura collegata e k un valore intero, vogliamo calcolare il riferimento al nodo k-esimo (iniziando a contare da 0) della struttura se esiste, oppure segnalare che tale nodo non esiste

. . . Accesso ad un nodo tramite la sua posizione

```
NodoLista a = ...
int k = ...
NodoLista p = a;
for (int i=0; i<k && p!=null; i++)
        p = p.next;
if (p==null) {
   // k >= dimensione della struttura
   //. . . lanciare una eccezione . . .
}
else {
   // p e' il riferimento al nodo in
   // posizione k e va quindi elaborato
}
```