

Strutture dati

Programmazione
Corso di laurea in Informatica

AA2003/04
© M.A. Alberti

1

Programmazione
Strutture dati

Le strutture dati

- I programmi lavorano su informazioni
- Le informazioni sono spesso organizzate in tavole
 - Che non sono semplicemente una massa di dati numerici
 - Ma è che coinvolgono relazioni strutturali tra i dati
- In genere c'è molta più informazione strutturale tra i dati di quella che sia necessario rappresentare

AA2003/04
© M.A. Alberti

2

Programmazione
Strutture dati

Proprietà delle strutture dati

- Le strutture dati hanno proprietà **statiche** e **dinamiche** che possono essere molto diverse
- Le strutture dati devono essere rappresentate e implementate nei linguaggi di programmazione
- Spazio di memoria per l'archiviazione
- La rappresentazione influenza le funzionalità
 - Creazione, manipolazione, ricerca, accesso

AA2003/04
© M.A. Alberti

3

Programmazione
Strutture dati

Rappresentazione della struttura dati

- Decidere caso per caso quanta struttura rappresentare
- Occorre considerare non solo la struttura dati ma anche la classe d'operazioni da effettuare sui dati
- La rappresentazione dipende tanto dalle proprietà intrinseche delle struttura dati quanto dalle funzionalità che si desiderano
- **Forma e funzione**

AA2003/04
© M.A. Alberti

4

Programmazione
Strutture dati

Tabelle d'informazioni

- Nella forma più semplice una tabella è una lista lineare di elementi
- La sua relazione strutturale più importante consente di rispondere a domande come
 - Chi è il primo della lista?
 - Chi è l'ultimo?
 - Chi precede e chi segue un dato elemento?
 - Quanti elementi ci sono nella lista?
- In altri casi una tabella può essere a 2 o più dimensioni
 - Può avere una struttura gerarchica ad albero
 - Può essere una struttura più complessa con ramificazioni a reticolo

AA2003/04
© M.A. Alberti

5

Programmazione
Strutture dati

Liste lineari

- Un insieme di $n \geq 0$ di nodi le cui proprietà strutturali coinvolgono essenzialmente solo le posizioni lineari (a 1-dimensione) relative dei nodi
- Se $n \geq 0$, $x[1]$ è il primo nodo allora $\forall k$ con $1 < k < n$, il k -esimo nodo $x[k]$ è preceduto dal nodo $x[k-1]$ ed è seguito dal nodo $x[k+1]$; inoltre $x[n]$ è l'ultimo nodo

AA2003/04
© M.A. Alberti

6

Programmazione
Strutture dati

Operazioni sulle liste

- Le operazioni che vogliamo eseguire su una lista, includono (non esclusivamente):
 - Accedere al k -esimo nodo
 - Inserire un nuovo nodo in una posizione opportuna
 - Eliminare il k -esimo nodo
 - Unire 2 o più liste
 - Fare una copia di una lista
 - Determinare il numero di nodi in una lista
 - Ordinare i nodi di una lista secondo un criterio
 - Cercare le occorrenze di un nodo con un particolare valore

AA2003/04
© M.A. Alberti

7

Programmazione
Strutture dati

Particolari liste

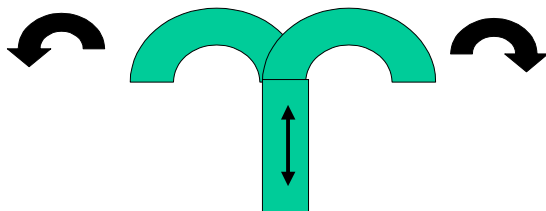
- Sono frequenti liste lineari in cui inserimenti, cancellazioni e accesso ai dati avvengono quasi sempre solo in prima o in ultima posizione
 - **Stack** o **pila**. Inserimenti e cancellazioni (e spesso anche gli accessi) avvengono solo a un estremo della lista
 - **Coda**. Inserimenti avvengono a un estremo della lista e le cancellazioni all'altro
 - **Deque** (coda a doppi estremi). Inserimenti e cancellazioni possono avvenire ai 2 estremi della lista

AA2003/04
© M.A. Alberti

8

Programmazione
Strutture dati

Esempio



Un binario morto ad una stazione di testa
è rappresentabile con uno stack.

Il primo treno ad uscire è l'ultimo che è entrato

AA2003/04
© M.A. Alberti

9

Programmazione
Strutture dati

Nomenclatura

- Spesso vengono usati altri nomi per indicare stack e liste in genere
 - **LIFO**. Last In First Out, per stack
 - **FIFO**. First In First Out, per le code
- In particolare per gli stack si usano i termini
 - **Top** o cima per indicare il primo elemento dello stack
 - **Bottom** o fine per indicare l'ultimo elemento
 - **Pop** per eliminare l'elemento al top
 - **Push** per inserire un nuovo dato

AA2003/04
© M.A. Alberti

10

Programmazione
Strutture dati

Rappresentazione di una lista

- A seconda della classe di operazioni più frequenti è utile una particolare rappresentazione
- Non esiste una singola rappresentazione che rende le operazioni tutte ugualmente efficienti
 - L'accesso random alla k -esimo nodo è difficile se contemporaneamente inseriamo e cancelliamo nodi
- Distinguiamo rappresentazioni di liste in funzione delle principali operazioni da eseguire

AA2003/04
© M.A. Alberti

11

Programmazione
Strutture dati

Liste statiche vs dinamiche

- Le liste statiche sono quelle che adottano una rappresentazione statica
 - Viene allocata un'area di memoria di dimensione prefissata quindi hanno una dimensione fissa
- Le liste dinamiche adottano una rappresentazione dinamica
 - L'area di memoria per archiviare i nodi viene riservata dinamicamente in esecuzione quando serve
 - Possono avere una dimensione non prefissata

AA2003/04
© M.A. Alberti

12

Programmazione
Strutture dati

Implementazione di pile

- `public void push(Object o)`
 - Aggiunge l'oggetto passato come parametro sullo stack. Se lo stack è pieno solleva una `OutOfBoundsStackException`
- `public Object pop()`
 - Riporta il riferimento dell'oggetto che sta al top dello stack. Se lo stack è vuoto solleva una `EmptyStackException`
- `public boolean empty()`
 - Riporta `true` se lo stack è vuoto, `false` altrimenti
- `Stack()`
 - Il costruttore che costruisce uno stack vuoto

AA2003/04
© M.A. Alberti

13

Programmazione
Strutture dati

L'eccezione per stack vuoto

```
public class EmptyStackException extends  
    RuntimeException  
{  
}
```

Un'eccezione definita come estensione di `RuntimeException` quindi non controllata
Viene sollevata dal metodo `pop()`

AA2003/04
© M.A. Alberti

14

Programmazione
Strutture dati

L'eccezione per stack pieno

```
public class OutOfBoundsStackException  
    extends RuntimeException  
{  
}
```

Un'eccezione definita come estensione di `RuntimeException` quindi non controllata
Può essere sollevata dal metodo `push()`

AA2003/04
© M.A. Alberti

15

Programmazione
Strutture dati

Implementazione statica

- Per un'implementazione statica di stack, possiamo adottare un'array di oggetti
- Lo stack viene rappresentato come un array di riferimenti a `Object`
- L'array sarà predimensionato in modo conveniente
 - Occorre tenere presente la dimensione massima dell'array
 - Potrà sempre essere possibile andare in overflow

AA2003/04
© M.A. Alberti

16

Programmazione
Strutture dati

Implementazione statica

- Ogni posizione dell'array rappresenta un nodo dello stack
- La posizione `0` viene usata per contenere l'elemento più in basso dello stack. Un indice indica la cima dello stack
- `Stack.java` e l'eccezioni
 - `EmptyStackException.java`
 - `OutOfBoundsStackException.java`

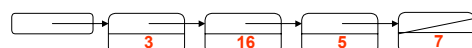
AA2003/04
© M.A. Alberti

17

Programmazione
Strutture dati

Implementazione dinamica

- Si usa una lista lineare concatenata
 - Oltre all'informazioni del nodo è necessario archiviare l'indirizzo del nodo successivo
 - Ogni nodo contiene un campo per contenere il riferimento all'oggetto con il valore che ci interessa archiviare



AA2003/04
© M.A. Alberti

18

Programmazione
Strutture dati

Rappresentazione del nodo

- La classe che rappresenta il nodo dello stack

```
Class NodoStack {
    Object dato;
    NodoStack pros;
}
```
- La definizione è ricorsiva, cioè il nodo è definito in termini di sé stesso

AA2003/04
© M.A. Alberti

19

Programmazione
Strutture dati

Rappresentazione dello stack

- Per definire lo stack è sufficiente dare il riferimento alla cima dello stack
 - La cima sarà `null` nel caso di stack vuoto

```
public class Stack {
    private NodoStack cima;
    ... altri metodi
}
```

AA2003/04
© M.A. Alberti

20

Programmazione
Strutture dati

Costruttore e metodi

```
public Stack() {
    cima = null;
}

public boolean empty() {
    return cima == null;
}
```

AA2003/04
© M.A. Alberti

21

Programmazione
Strutture dati

Il metodo `push()`

- Il metodo ha come parametro il riferimento all'oggetto che deve inserire alla cima dello stack
 - Dovrà creare un nuovo nodo
 - Copiare il riferimento passato come parametro nel campo `dato` del nuovo nodo
 - Inserire il nuovo nodo alla cima della lista

AA2003/04
© M.A. Alberti

22

Programmazione
Strutture dati

Il metodo `pop()`

- Il metodo non ha parametri. Deve eliminare il nodo alla cima dello stack
 - Dovrà memorizzare il riferimento contenuto nella cima
 - Eliminare la cima
 - Riportare all'ambiente il riferimento memorizzato
- Se la lista è vuota deve sollevare un'eccezione
 - `EmptyStackException`

AA2003/04
© M.A. Alberti

23

Programmazione
Strutture dati

Classi interne

- La classe `NodoStack` viene usata solo dalla classe `Stack` e quindi viene dichiarata `static` e `private`
- È una risorsa privata della classe `Stack` che avrà accesso perciò ai suoi campi
- `Stack.java`

AA2003/04
© M.A. Alberti

24

Programmazione
Strutture dati

Le code

- Negli stack l'elemento che si preleva è l'ultimo che si è inserito (**LIFO**)
- Nelle code gli elementi si inseriscono ad un estremo e prelevano all'altro (**FIFO**)
 - Quindi si preleva nell'ordine in cui si è inserito
 - Ad esempio: una lista d'attesa

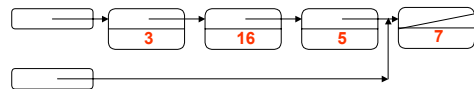
AA2003/04
© M.A. Alberti

25

Programmazione
Strutture dati

Implementazione dinamica

- Si può usare una lista concatenata
- Utile avere anche un riferimento all'ultimo nodo della lista



AA2003/04
© M.A. Alberti

26

Programmazione
Strutture dati

Implementazione in Java

- Rappresentazione del singolo nodo

```
class NodoCoda {
    Object dato;
    NodoCoda pros;
}
```
- Rappresentazione della coda

```
public class Coda {
    private NodoCoda primo, ultimo;
    private static NodoCoda { ... }
    ... altri metodi
}
```

AA2003/04
© M.A. Alberti

27

Programmazione
Strutture dati

Operazioni sulla coda

- Inserire nodi
 - Avviene alla fine della lista
- Eliminare nodi
 - Avviene all'inizio della lista

AA2003/04
© M.A. Alberti

28

Programmazione
Strutture dati

Costruttore e metodi

- Il costruttore costruisce una coda vuota

```
public Coda() {
    primo = ultimo = null;
}
```
- Funzione per inserire un nodo
 - Costruire il nuovo nodo da inserire
 - Collegare il nuovo nodo dopo l'ultimo
 - Aggiornare il riferimento all'ultimo nodo

```
public void aggiungi (Object o) { ...
}
```

AA2003/04
© M.A. Alberti

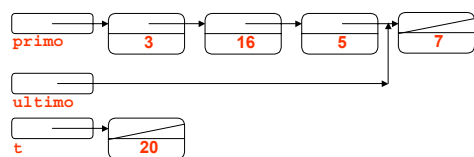
29

Programmazione
Strutture dati

Inserire nodi

- Creare il nuovo nodo con il dato riferito da `o`

```
NodoCoda t = new NodoCoda();
t.dato = o;
t.pros = null;
```



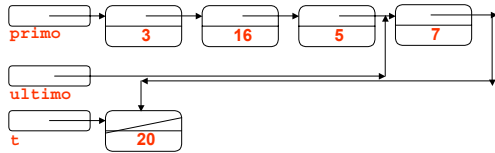
AA2003/04
© M.A. Alberti

30

Programmazione
Strutture dati

...inserire nodi...

- Collegare dopo l'ultimo nodo
`ultimo.pros = t;`



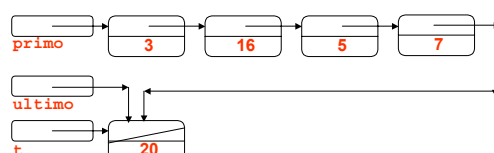
AA2003/04
© M.A. Alberti

31

Programmazione
Strutture dati

...inserire nodi...

- Aggiornare il riferimento all'ultimo nodo
`ultimo = t;`



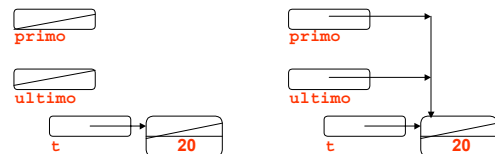
AA2003/04
© M.A. Alberti

32

Programmazione
Strutture dati

...inserire nodi...

- Ma se la coda era vuota allora
`if (primo == null) primo = ultimo = t;`



AA2003/04
© M.A. Alberti

33

Programmazione
Strutture dati

Eliminare nodi

- Controllare se la coda è vuota e nel caso sollevare l'eccezione `CodaVuotaException`
`if (primo == null) throw new CodaVuotaException();`
- Eliminare il riferimento al primo nodo e riportare all'ambiente il riferimento contenuto nel campo `dato`
`Object risultato = primo.dato;`
`primo = primo.pros;`

AA2003/04
© M.A. Alberti

34

Programmazione
Strutture dati

...eliminare nodi...

- Se la lista rimane vuota dopo il prelievo, occorre ricordarsi di aggiornare ultimo
`if (primo == null) ultimo = null;`

[Coda.java](#) con l'eccezioni
[CodaVuotaException.java](#) e il driver
[ProvaCoda.java](#)

AA2003/04
© M.A. Alberti

35

Programmazione
Strutture dati