In questa lezione

Strutture dati elementari:

- **Pila**
- **Coda**
- **Loro uso nella costruzione di algoritmi.**

strutture dati (astratte)

Una struttura dati astratti consiste di uno o più insiemi con delle operazioni che possono modificarli.

Le operazioni sono descritte sulla base del loro effetto.

Esempio

La struttura dati in cui si ha un insieme di elementi con le operazioni

- **di inserimento, che ha come risultato quello di aggiungere una nuova elemento**
- **di cancellazione, che ha come effetto quello di eliminare un dato elemento dall'insieme, se presente**
- **di ricerca di una elemento, che fornisce la possibilità di recuperare un dato elemento nell'insieme, se presente, e ne segnala l'assenza altrimenti**

è chiamata dizionario (dictionary)

strutture dati (astratte): i dati

Gli elementi su cui si vogliono eseguire queste operazioni spesso sono complessi, sono oggetti rappresentati in memoria con strutture con molti campi.

Assumeremo che un campo in particolare, la chiave (key), è quella su cui effettuiamo le nostre operazioni. Se infatti assumiamo che le chiavi sono tutte diverse, allora abbiamo a che fare con un insieme. Gli altri dati dell'oggetto sono chiamati dati satellite (satellite data).

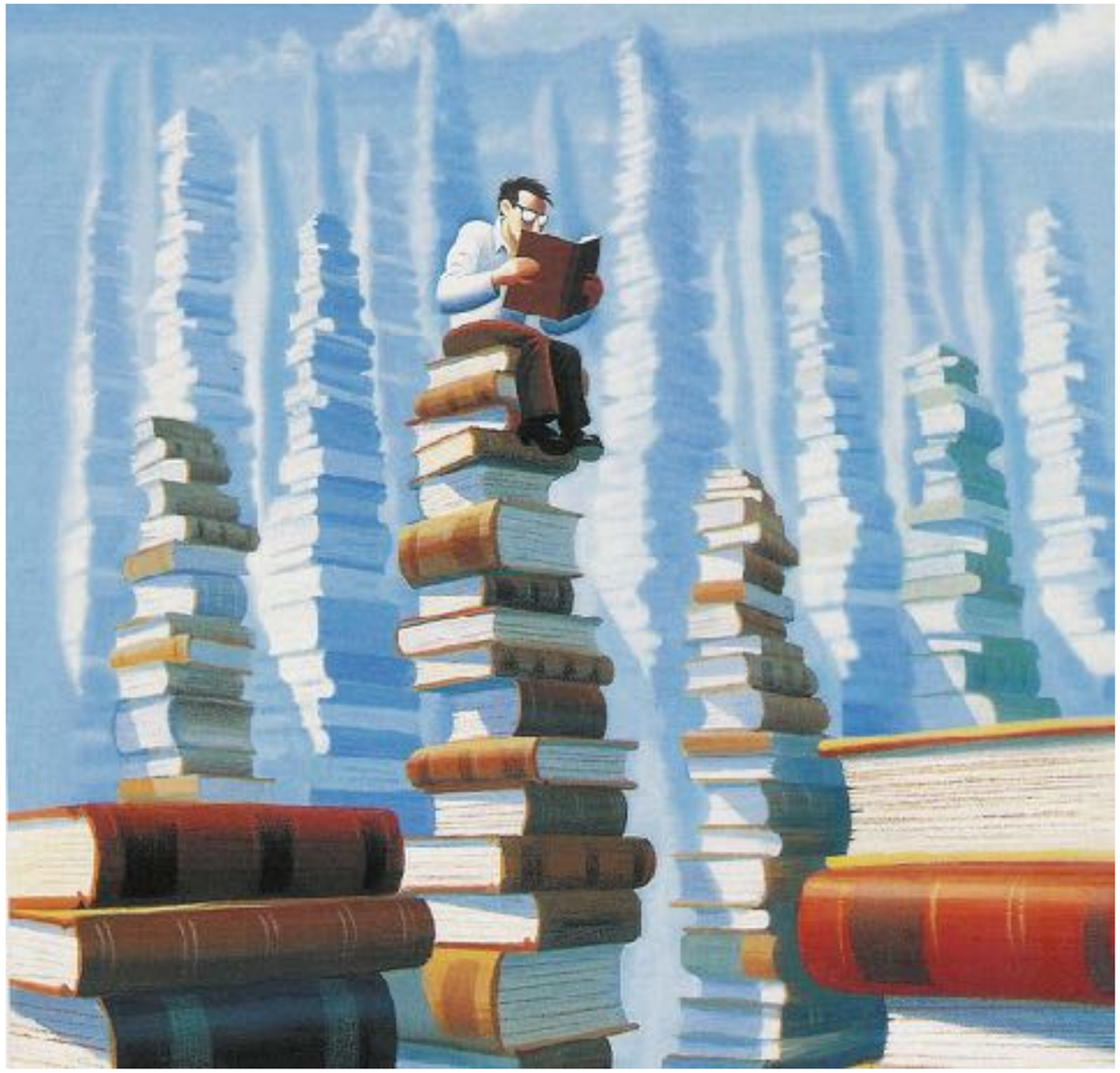
strutture dati lineari

Si tratta di collezioni in cui gli elementi sono organizzati in sequenza.

Gli elementi si possono inserire o prelevare solo alle estremità della sequenza.

Questo tipo di restrizioni caratterizza il tipo di dato astratto, per esempio in una pila gli elementi sono prelevati e aggiunti ad un solo estremo della sequenza.

Pila (stack)



Una pila è un insieme dinamico di dati in cui la cancellazione è vincolata a un elemento particolare: l'ultimo inserito.

- **solo l'elemento in cima è accessibile!!**
- **L'ultimo ad essere inserito è il primo ad essere estratto: Last In First Out.**

Operazioni

Le operazioni fondamentali (di interfaccia) sono:

- **push che inserisce un elemento in cima (top) alla pila**
- **pop che estrae l'elemento in cima alla pila**

Esempio: Ogni browser ha uno stack, che viene usato quando si vuole tornare indietro all'ultima pagina visitata prima della corrente.

Ogni volta che si apre una nuova pagina, il suo indirizzo va nello stack (push).

Quando si torna indietro si estrae dalla pila l'ultimo indirizzo inserito (pop).

Volendo si possono riesaminare tutte le pagine aperte nell'ordine contrario a quello di apertura.

Operazioni - 2

Oltre alle operazioni fondamentali push, che inserisce un elemento in cima (top) alla pila e pop, che estrae l'elemento in cima alla pila,

si definiscono le operazioni

- **pilaVuota che dà vero se la pila è vuota, falso altrimenti**
- **top che dà l'elemento in cima, senza estrarlo**

Implementazione

Per utilizzare una struttura dati astratta è necessario fornire un'implementazione, cioè scegliere come rappresentare i dati (la struttura dati concreta) e definire gli algoritmi per realizzare le operazioni.

In programmazione il concetto di interfaccia viene usato per "nascondere" l'implementazione e consentire l'uso delle funzionalità in modo indipendente dall'implementazione, esattamente come avviene per i tipi di dati elementari come i numeri interi.

Strutture dati astratte e concrete

In generale il processo è il seguente:

1. si definisce la struttura dati astratta

2. si sceglie la struttura dati concreta con cui rappresentare i dati in memoria

3. si definiscono gli algoritmi che realizzano gli obiettivi specificati nelle funzioni.

Array e liste concatenate sono le strutture dati concrete più usate nell'implementazione di una pila

Il problema delle parentesi ben bilanciate

Consideriamo l'insieme delle parentesi ben bilanciate, cioè di stringhe di parentesi, di tre tipi, in cui ogni parentesi aperta ha la sua corrispondente chiusa e tutte sono propriamente annidate.

Per esempio le seguenti sono ben bilanciate:

```
([({})[]])        ({[]}())    ([(()){[]}()])
```

mentre le seguenti NO

```
(([({)[]])}       ()))              ([]{}(()
```

Definizione ricorsiva.
Passo base: la stringa vuota, ε, è una stringa di parentesi ben bilanciata

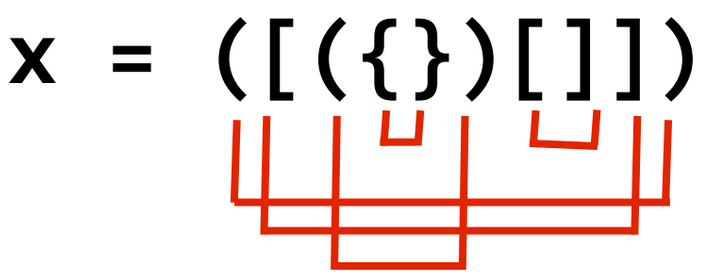
Passo ricorsivo: se x e y sono stringhe di parentesi ben bilanciate allora anche x(y), x[y], x{y} lo sono.

Uso uno stack?

Consideriamo il problema di stabilire se una stringa in input è una sequenza di parentesi ben bilanciata.

Per definizione una sequenza è ben bilanciata se ogni aperta ha una sua chiusa corrispondente e se sono correttamente annidate.

Guardando a questo esempio



leggendo da sinistra a destra ogni volta

l'ultima aperta sospesa è la prima a essere accoppiata.

Dunque serve uno stack!

l'algoritmo in bozza

Alg Verificaparentesi(x)
input: x è un'espressione aritmetica,
prec: le parentesi che occorrono in x sono del tipo: (,[,{
postc: restituisce vero se le parentesi in x sono ben bilanciata, falso altrimenti

Si impilano le aperte e si eliminano dalla cima dello stack le aperte in corrispondenza di ogni chiusa della stesso tipo.

Quindi quando si considera un carattere c:
se c è una parentesi aperta viene impilato
se c è una parentesi chiusa, viene confrontato con l'aperta in cima alla pila e se risultano dello stesso tipo si esegue una pop

Esempio di uso dello stack: l'algoritmo

Alg Verificaparentesi(x)

crea una pila vuota;
leggi il primo carattere di x e mettilo in c
finchè ci sono caratteri da leggere in x
se c non è una parentesi, passa al carattere successivo
se c è una parentesi aperta, esegui una push di c nella pila
altrimenti %c è una parentesi chiusa
se la pila è vuota, restituisci false
altrimenti %la pila non è vuota,
salva l'elemento in cima in d e esegui una pop
se la parentesi aperta d e quella chiusa c sono dello stesso tipo,
leggi il successivo carattere e mettilo in c
altrimenti restituisci false
se la pila è vuota, restituisci true altrimenti restituisci false

La pila in ogni passo di esecuzione del ciclo contiene le parentesi aperte non accoppiate con una chiusa fino a quel passo.

Ogni parentesi aperta viene inserita ed estratta dalla pila una sola volta, quindi $T(n) = \Theta(n)$, inoltre l'algoritmo usa uno spazio lineare per la coda

Esempio di esecuzione

x = ([ ] { { [ ] } ( ) })
0 1 2 3 4 5 6 7 8 9 10 11

x[0] = (va conservata, quindi push di (

x[1] = [va conservata, quindi push di [

x[2] =] chiude x[1] : pop

x[3], x[4] e x[5] vanno conservate, quindi si eseguono i relativi push

x[6] =] chiude x[5] : pop

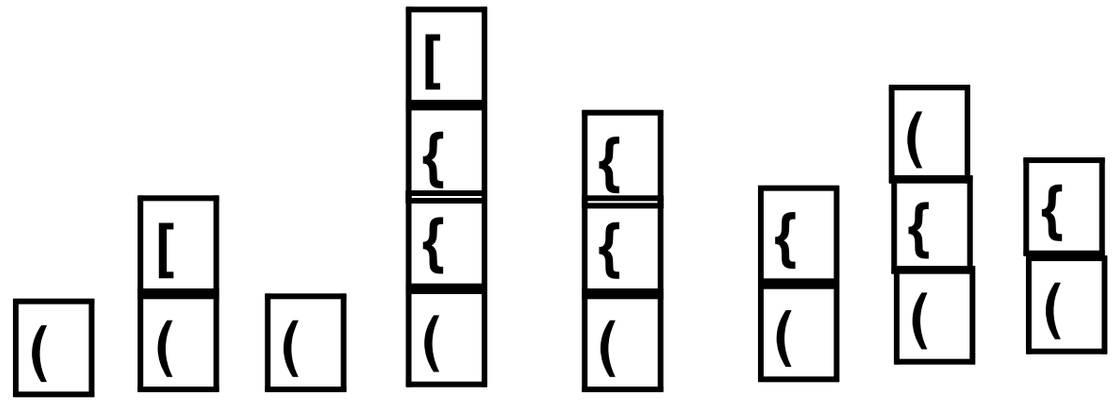
x[7] = } chiude x[4] : pop

x[8] = (va conservata, quindi push di (

x[9] =) chiude x[8] : pop

x[10] = } chiude x[3] : pop

x[11] =) chiude x[0] : pop



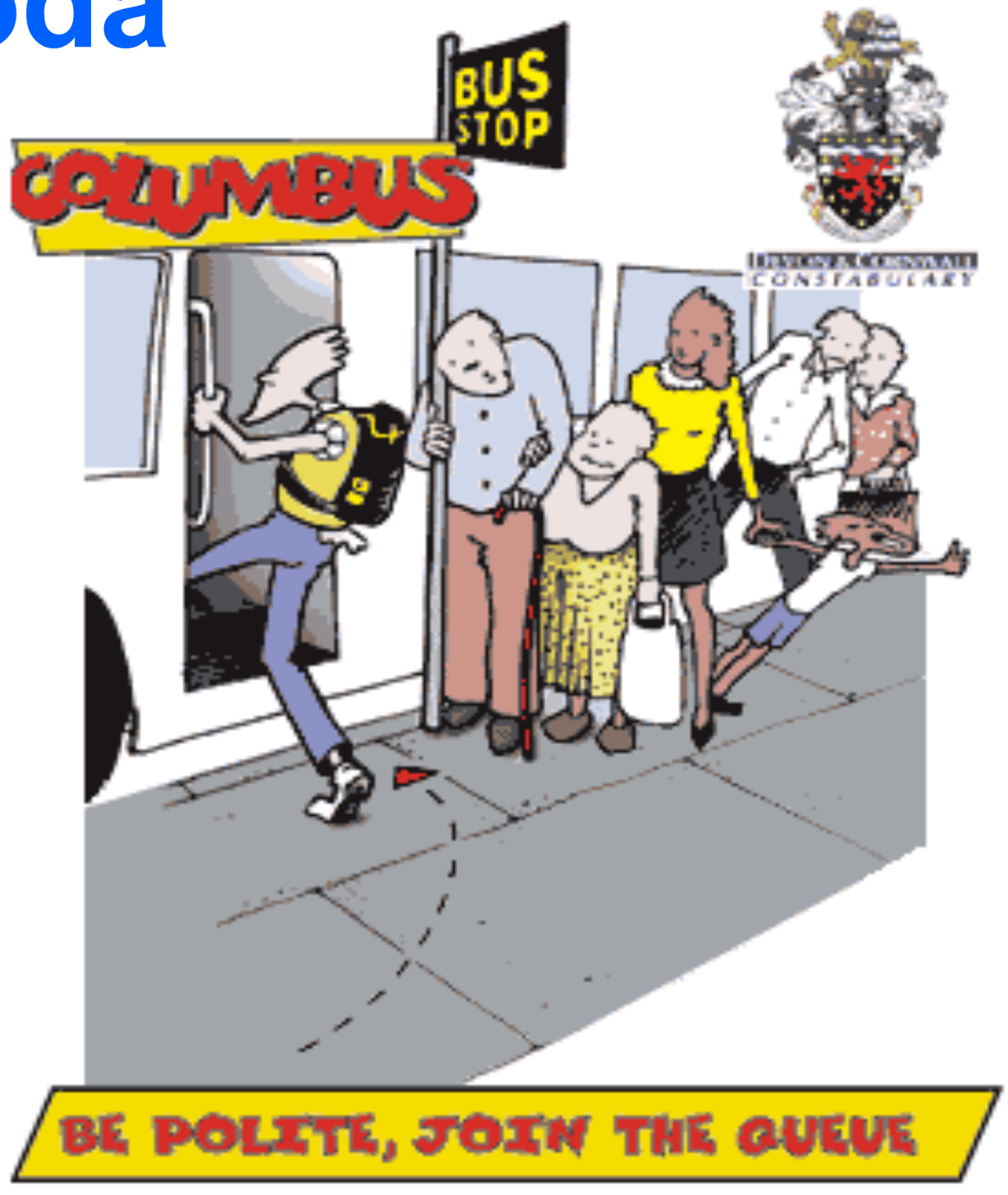
L'evoluzione della pila

Coda

Una coda è un insieme dinamico di dati in cui la cancellazione è vincolata a un elemento particolare: il primo inserito.

- **solo il primo elemento inserito è accessibile**
- **Il primo ad essere inserito è il primo ad essere estratto:**

First In First Out

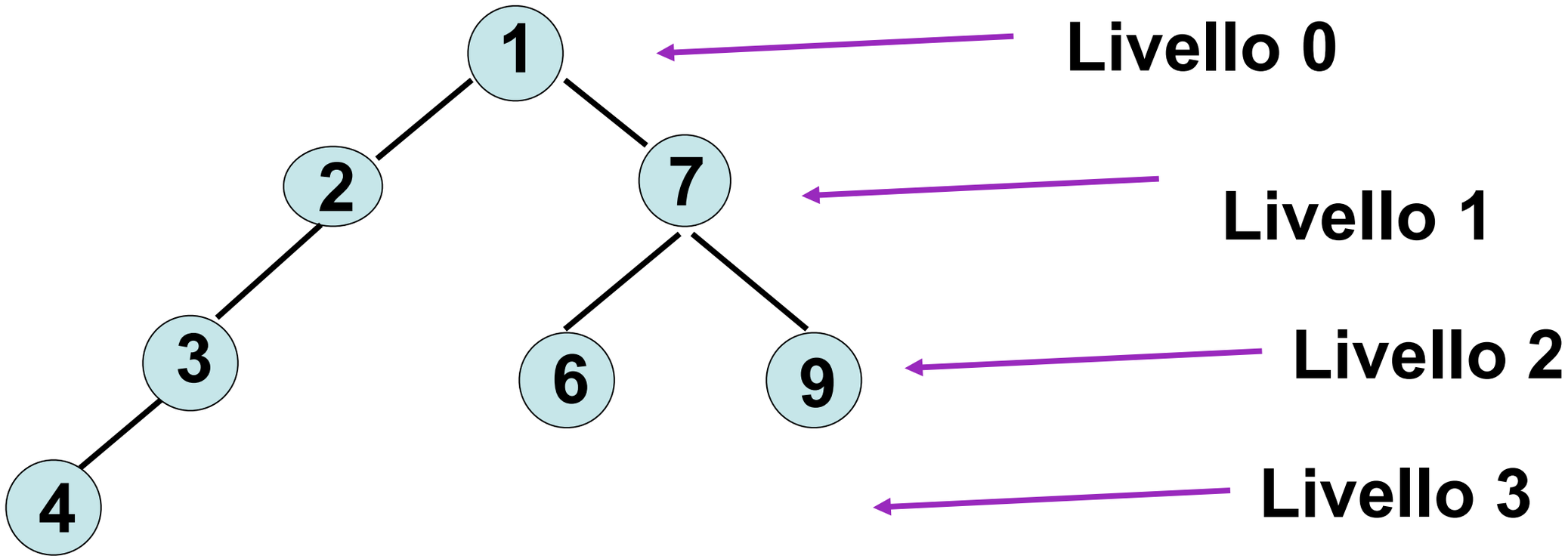


Le operazioni

Ogni coda ha un inizio, la testa (head) della coda, e una fine (tail). Le operazioni sono:

- **Accoda (Enqueue): inserisce un elemento alla fine della coda, (tail)**
- **estrazione (Dequeue): cancella l'elemento in testa alla coda, (head)**
- **front: dà, senza estrarlo, l'elemento in testa alla coda**
- **codaVuota: dà true se la coda è vuota, false altrimenti**

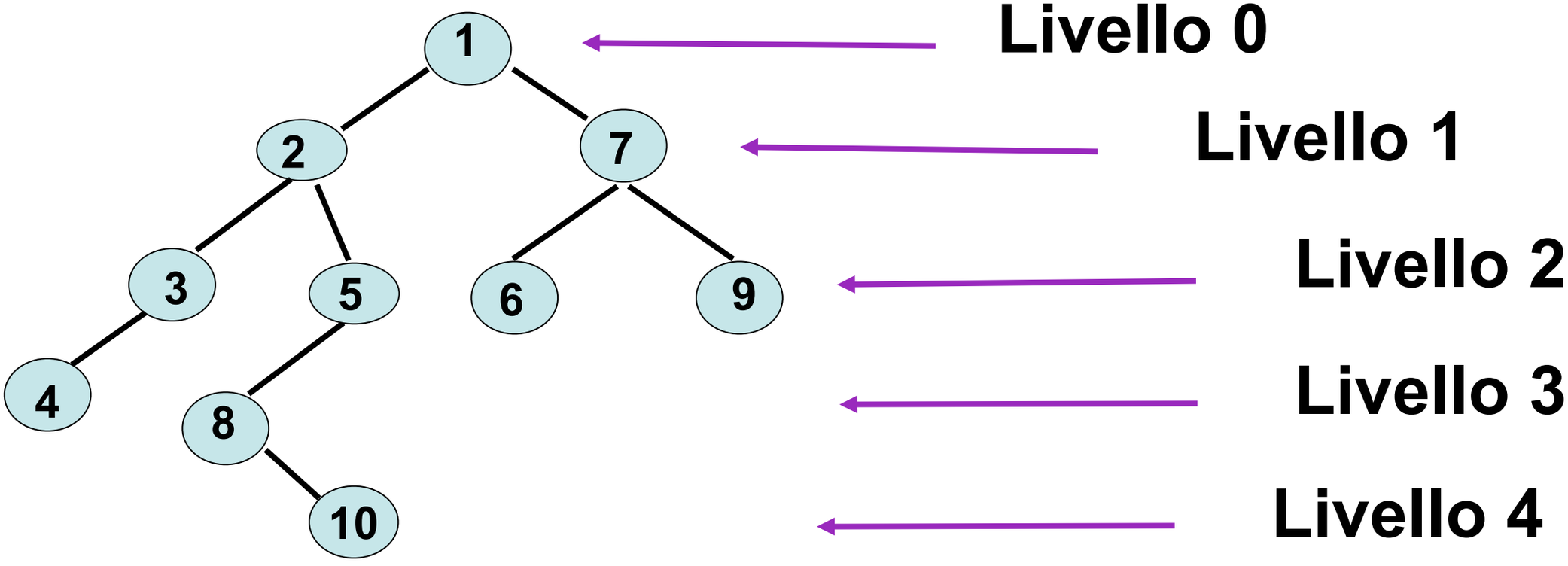
APPLICAZIONE DELLA CODA: VISITA PER LIVELLI DI UN ALBERO BINARIO



Si vuole visitare i nodi su ogni livello da sinistra a destra e in ordine discendente di livello.

Risultato visita: 1 27 369 4

VISITA PER LIVELLI DI UN ALBERO BINARIO



Risultato visita: 1 2 7 3 5 6 9 4 8 10
Osserviamo che se per esempio abbiamo visitato 6 il prossimo da visitare è suo fratello ma il successivo è il figlio del primo nodo sul suo livello.

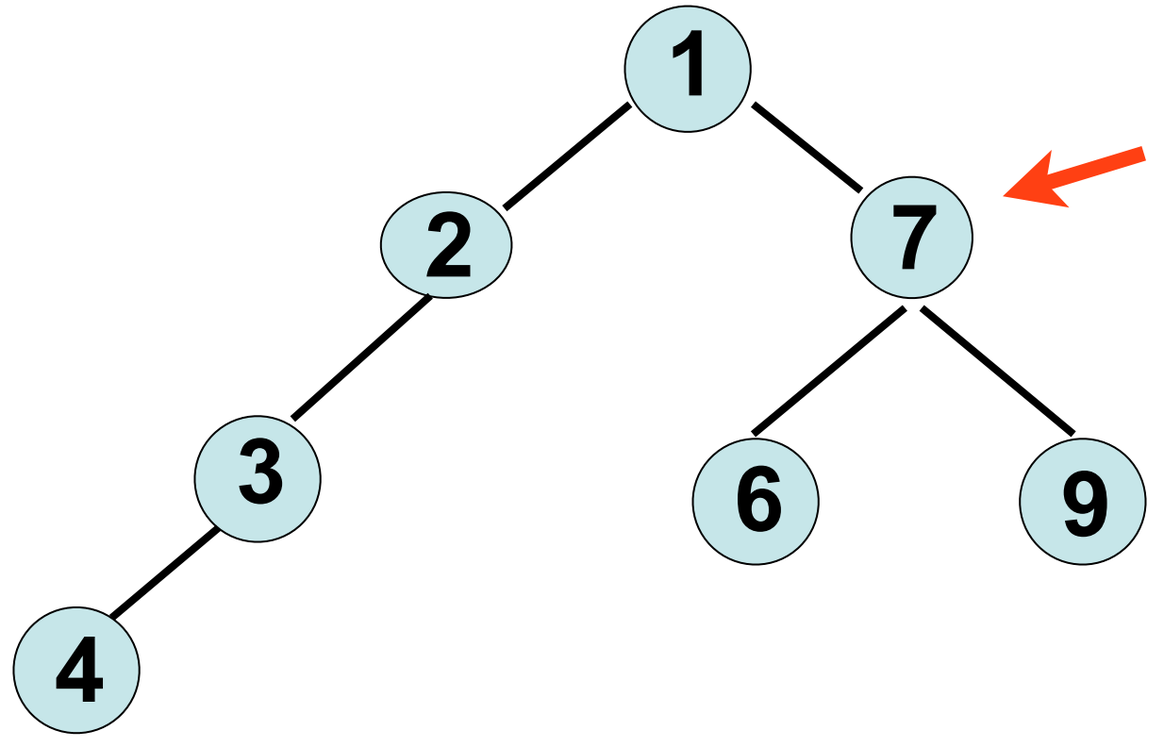
Allora se man mano che si visita un livello si tiene memoria degli eventuali figli, cioè dei nodi del livello successivo, il primo memorizzato è il primo da estrarre.

Quindi serve una coda!

Cosa fare in un passo generico

Alg VisitaPerLivelli(T)
//T è un albero binario
postc: stampa la sequenza dei nodi di T per livelli da sinistra a destra.

In un generico passo, potremmo trovarci alla fine di un livello i e nella necessità di stampare i nodi del livello successivo, partendo da sinistra.
Quindi bisogna che i nodi del livello i +1 siano stati accodati nell'ordine da sinistra verso destra, e l'unica possibilità di farlo è quando si stampa il loro padre.
Quindi l'azione da fare sul nodo in esame è la sua stampa e l'accodamento dei suoi figli.
Infine un nodo stampato non serve più e quindi deve essere tolto dalla coda.



Nodi nella coda prima della stampa di 7

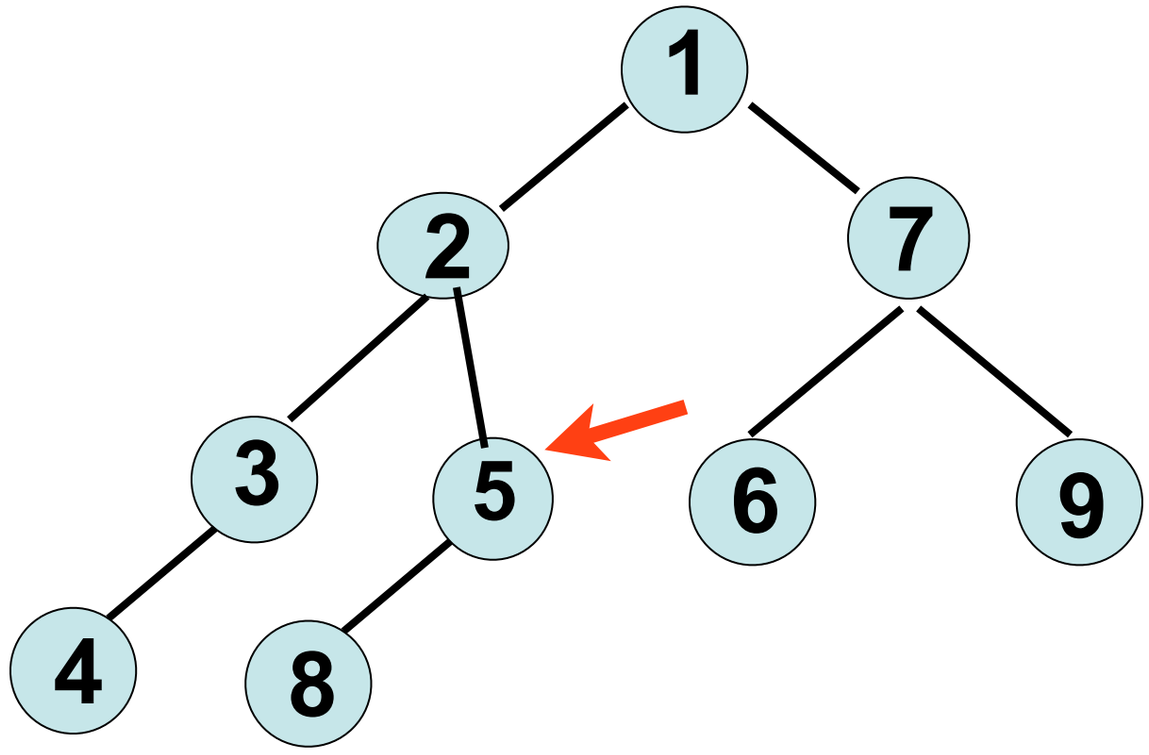
Q= | 7 | 3 |

Nodi nella coda dopo la stampa di 7

Q= | 3 | 6 | 9 |

Cosa fare in un passo generico

Una situazione del tutto analoga si verifica se consideriamo un passo intermedio in cui ci troviamo su un generico nodo di un livello.
Infatti quel nodo va stampato e tolto dalla coda, che deve contenere i restanti nodi di quel livello, ancora da stampare, e anche i figli dei nodi già stampati.
Quindi bisogna accodare i figli del nodo tolto e stampato.



Nodi nella coda prima della stampa di 5

Q= | 5 | 6 | 9 | 4 |

Nodi nella coda dopo la stampa di 5

Q= | 6 | 9 | 4 | 8 |

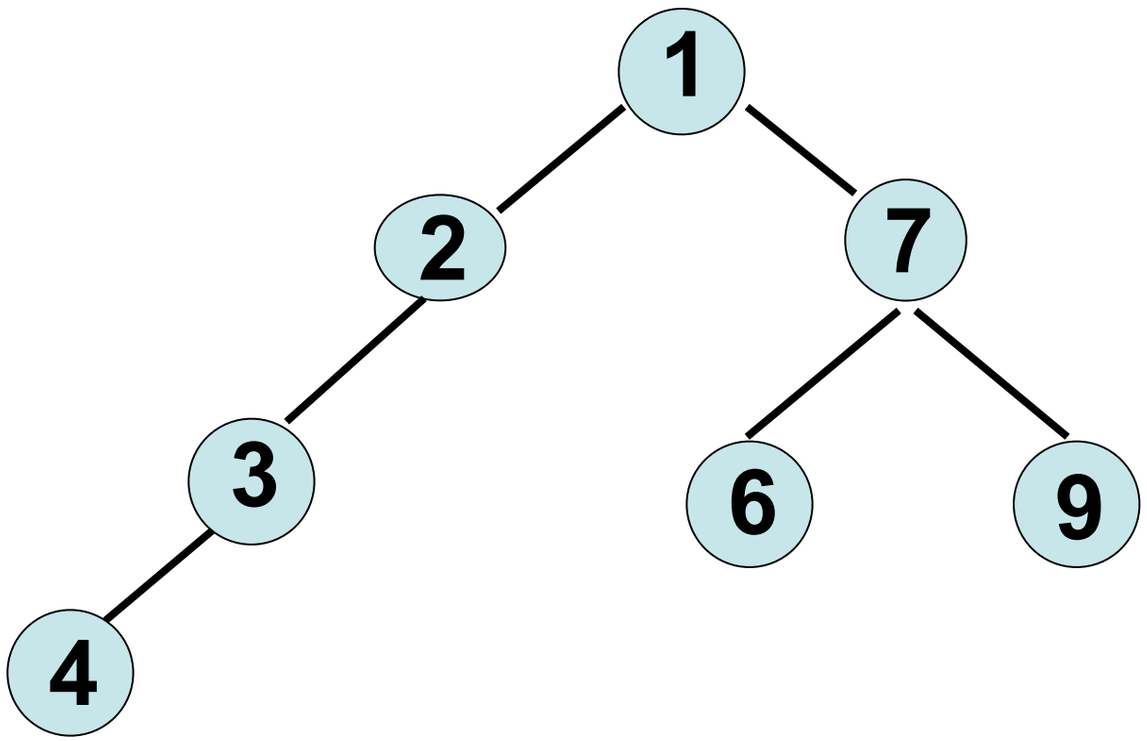
Visita per livelli: l'algoritmo

Dopo la visita di un nodo i la coda contiene i nodi del suo livello e alla sua destra, se ci sono, e a seguire i figli dei nodi alla sua sinistra, nel livello di i, compresi i figli di i.

```
Alg VisitaPerLivelli(T)
//T è un albero binario
postc: stampa la sequenza dei nodi di T
per livelli da sinistra a destra.
Q è una coda inizialmente vuota
accoda in Q il nodo radice, se l'albero è non vuoto
 while la coda non è vuota do
       y = dequeue(Q)
       stampa y
      if il figlio sinistro,x, di y è presente then Enqueue(Q,x)
      if se il figlio destro,x, di y è presente then Enqueue(Q,x)
```

N.B. L'algoritmo è formulato in modo da prescindere dalla rappresentazione in memoria di T, nel senso che la struttura che rappresenta un nodo potrebbe contenere, oltre al campo chiave, due o tre campi puntatore.

Visita per livelli: esempio di esecuzione



Contenuto coda all'inizio:

1

Nodo estratto e stampato: 1

2	7

Nodo estratto e stampato: 2

7	3

Nodo estratto e stampato: 7

3	6	9

Nodo estratto e stampato: 3

6	9	4

Nodo estratto e stampato: 6

9	4

Nodo estratto e stampato: 9

4

Nodo estratto e stampato: 4

Visita per livelli: analisi

```
Alg VisitaPerLivelli(T)
//T è un albero binario
postc: stampa la sequenza dei nodi di T
per livelli da sinistra a destra.
Q è una coda inizialmente vuota
accoda in Q il nodo radice, se l'albero è non vuoto
while la coda non è vuota do
        y = dequeue(Q)
        stampa y
if il figlio sinistro,x, è presente then Enqueue(Q,x)
if se il figlio destro,x,  è presente then Enqueue(Q,x)
```

Ogni nodo viene inserito ed estratto dalla coda una sola volta, quindi $T(n) = \Theta(n)$

Esercizi

Es. 10.1-6.
Mostrate come implementare una pila usando due code. Bisogna definire le operazioni Top,push e pop usando enqueue e dequeue.
Si analizzi la complessità asintotica delle operazioni push e pop in questa nuova implementazione.

Es. 10.1-7.
Mostrate come implementare una coda usando due pile. Bisogna definire le operazioni enqueue e dequeue usando push e pop. Si analizzi la complessità asintotica delle operazioni enqueue e dequeue in questa nuova implementazione.