Titolo unità didattica: Strutture dati elementari

Titolo modulo: Tipo di dato Albero

Tipo di dato albero e visite

Argomenti trattati:

- ✓ Proprietà degli alberi
- ✓ Rappresentazioni indicizzate
- ✓ Rappresentazioni collegate
- ✓ Visita di alberi
- ✓ Visita in profondità
- ✓ Visita in ampiezza

Prerequisiti richiesti: nozioni base degli algortimi

Prof. Angelo Ciaramella

Alberi

- In molte situazioni elaboriamo collezioni di oggetti su cui sono definite delle relazioni gerarchiche
  - Albero genealogico
  - Dati strutturati

- Un albero è un grafo connesso aciclico e non orientato

Alberi

- Un albero (radicato) è un albero libero in cui uno dei vertici si distingue da tutti gli altri
  - La radice

- Un albero radicato è una coppia $T = (N, A)$ costituita da un insieme N di nodi e da un insieme $A \in N \times N$ di coppie di nodi detti archi

- Ogni nodo v (tranne la radice) ha un solo genitore (padre) u tale che $(u, v) \in A$

Alberi

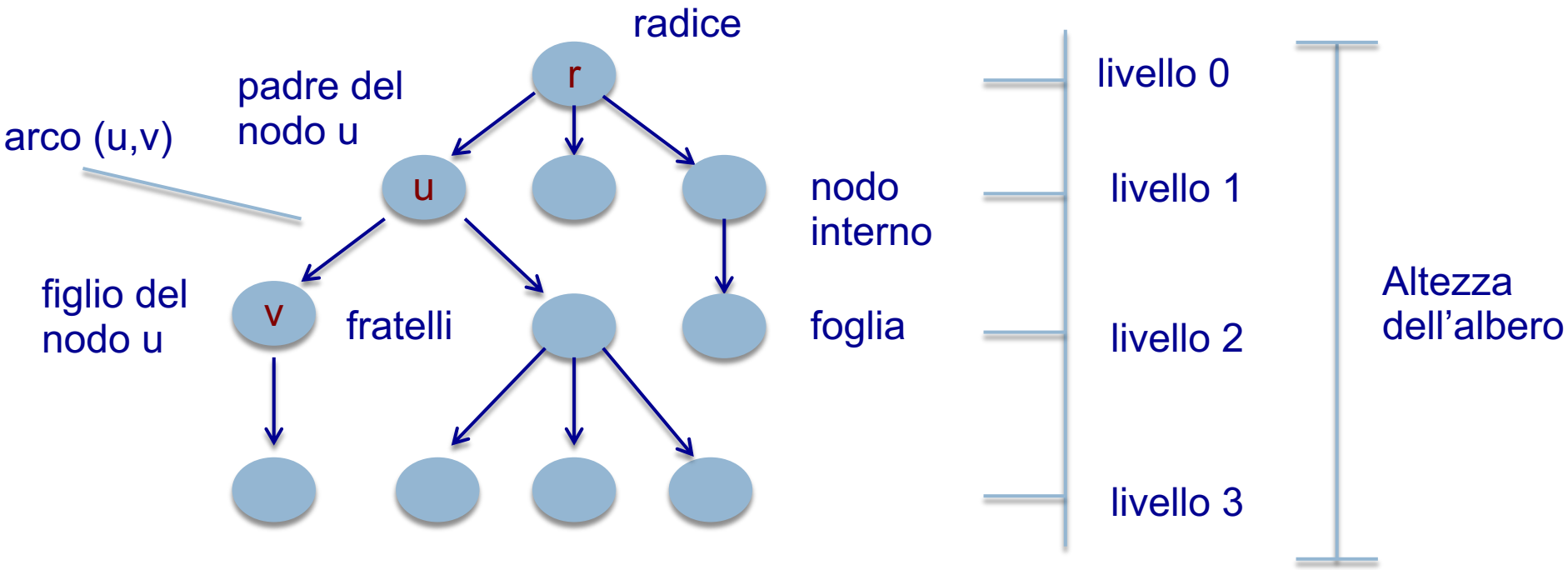
- Un nodo u può avere zero o più figli v tali che (u, v) $\in$ A e il loro numero viene chiamato grado del nodo
- Un nodo senza figli è chiamato foglia
- Nodi che non sono né foglia né la radice sono chiamati nodi interni
- La profondità (o livello) di un nodo è il numero di archi che bisogna attraversare per raggiungerlo a partire dalla radice. Definizione ricorsiva
  - La radice ha profondità zero
  - Se un nodo ha profondità k, tutti i suoi figli hanno profondità k+1

Alberi

- Nodi con lo stesso genitore vengono detti fratelli

- L'altezza di un albero è la massima profondità a cui si trova una foglia

Esempio di albero radicato



Alberi

- Molte strutture dati efficienti sono basati su alberi

- Particolare rilievo hanno alcune classi di alberi con vincoli strutturali
  - Facilitano la rappresentazione o consentono di realizzare operazioni su collezioni di elementi in modo efficiente
  - I principali vincoli strutturali riguardano il grado dei nodi e la profondità
  - Un albero d-ario in cui tutti i nodi tranne le foglie hanno grado d
  - Per $d = 2$ l'albero è detto binario
  - Un albero d-ario in cui tutte le foglie sono sullo stesso livello è completo

Tipo dato Albero

```
tipo Albero:

dati:
  un insieme di nodi (di tipo nodo) e un insieme di archi

operazioni:
    numNodi()→ intero
      restituisce il numero di nodi presenti nell'albero

    grado(nodo v)→ intero
      restituisce il numero di figli del nodo v

    padre(nodo v)→ nodo
      restituisce il padre del nodo v nell'albero, o null
      se v è la radice

    figli(nodo v) → (nodo, nodo, …, nodo)
      restituisce i figli del nodo v
```


Tipo dato Albero

aggiungiNodo(*nodo u*)→ nodo
inserisce un nuovo nodo v come figlio di u nell'albero e lo restituisce. Se v è il primo nodo ad essere inserito nell'albero, esso diventa la radice

aggiungiSottoalbero(*Albero a, nodo u*)
inserisce nell'albero il sottoalbero a in modo che la radice di a diventi figlia di u

rimuoviSottoalbero(*nodo v*)→ Albero
stacca e restituisce l'intero sottoalbero radicato in v. L'operazione cancella dall'albero il nodo v e tutti i suoi discendenti.

Rappresentazioni indicizzate

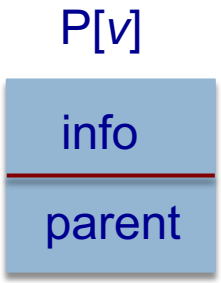
- Due delle più comuni rappresentazioni di alberi basate su array
  - Vettore padri
  - Vettore posizionale
- Entrambe richiedono spazio O(n) per un albero di n nodi
- Idea di base
  - Rappresentare ogni nodo dell'albero con una cella di un array che contiene l'informazione associata al nodo
  - Facile realizzazione ma la cancellazione e l'inserimento sono difficoltosi

Vettori padri

- Sia T = (N, A) un albero con n nodi numerati da 0 a (n - 1)

- Un vettore padri è un array P di dimensione n le cui celle contengono coppie (info, parent)

$$v \in [0, n-1]$$



record relativo alla cella v

- P[v].parent = u se e solo se vie è un arco (u, v) in A. Se v è la radice P[v].parent = null

Vettori padri

- Da ogni nodo è possibile risalire in tempo O(1) al proprio padre (padre(v))

- Per trovare il figlio richiede una scansione dell'array in tempo O(n) (figli(v))

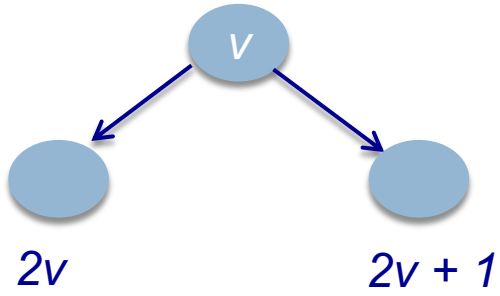
Vettore posizionale

- Nel caso particolare di alberi d-ari completi ($d \geq 2$) è possibile usare una rappresentazione indicizzata dove ogni nodo ha una posizione prestabilita nella struttura

- Sia $T = (N, A)$ un albero d-ario con n-nodi numerati da 1 a n
  - Un vettore posizionale è un array P di dimensione n tale che P[v] contiene l'informazione associata al nodo v
  - l'informazione associata all'i-esimo figlio di v è in posizione P[d v + i] per i compreso tra 0 e (d-1)

Vettore posizionale

- Ad esempio in un albero binario completo

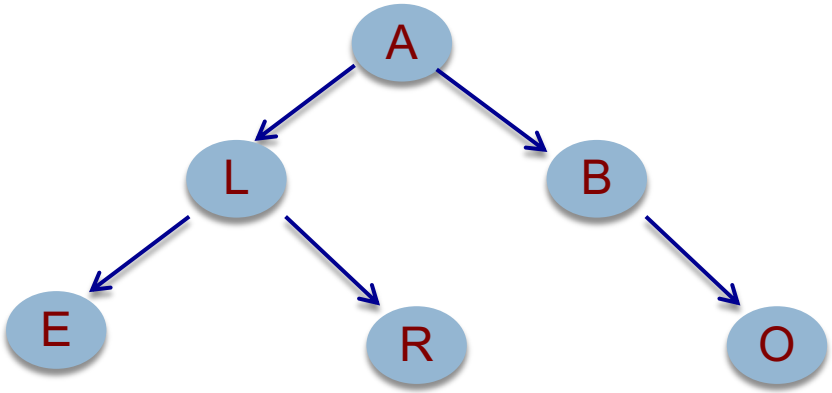


- Per semplicità di indicizzazione la posizione 0 dell'array non è utilizzata
- L'operazione padre(v) può essere realizzata in tempo costante
- L'operazione figli(v) richiede tempo O(grado(v))

Rappresentazioni collegate

- Le rappresentazioni collegate di alberi sono più flessibili di quelle indicizzate
  - Permettono in modo efficiente di supportare modifiche alla struttura di un albero
- Idea di base
  - Rappresentare ogni nodo con un record
  - Il record contiene l'informazione associata al nodo e puntatori che consentono di raggiungere altri nodi

Esempio di albero



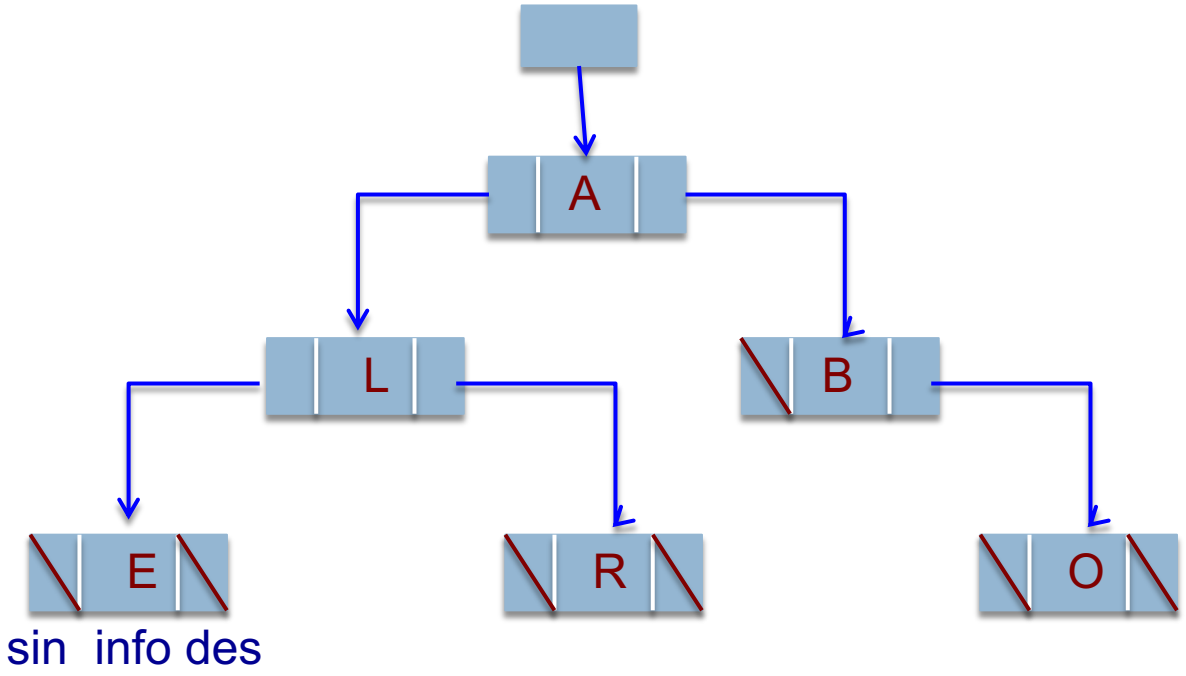
Esempio di albero

Puntatori ai figli

- Se ogni nodo dell'albero ha grado al più d è possibile mantenere in ogni nodo un puntatore a ciascuno dei possibili figli

- Lo spazio richiesto sarà O(nd) → O(n) per d costante

Puntatori ai figli



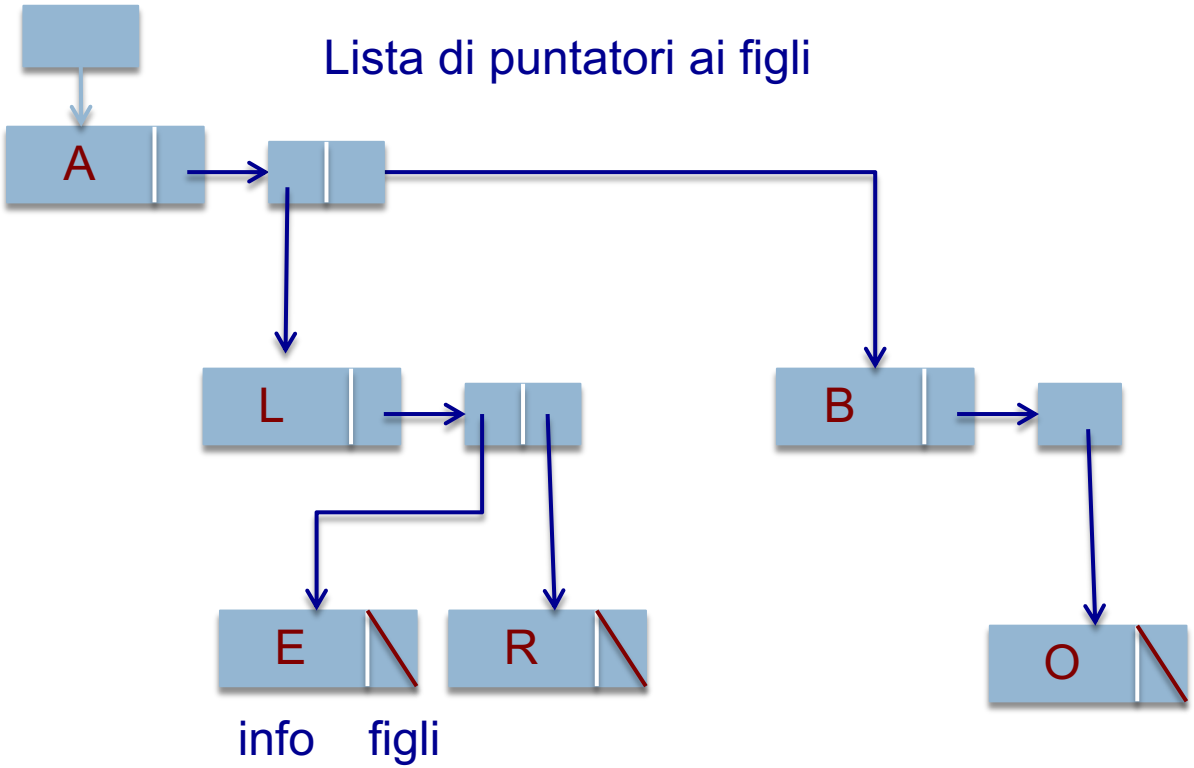
Rappresentazione collegata

Lista figli

- Se il numero massimo di figli non è noto a priori è possibile associare ad ogni nodo una lista di puntatori ai suoi figli
  - Questa lista può essere a sua volta rappresentata in modo indicizzato o collegato

- Lo spazio richiesto per rappresentare un albero con n nodi è O(n) indipendentemente dal numero di figli di un nodo

Lista figli



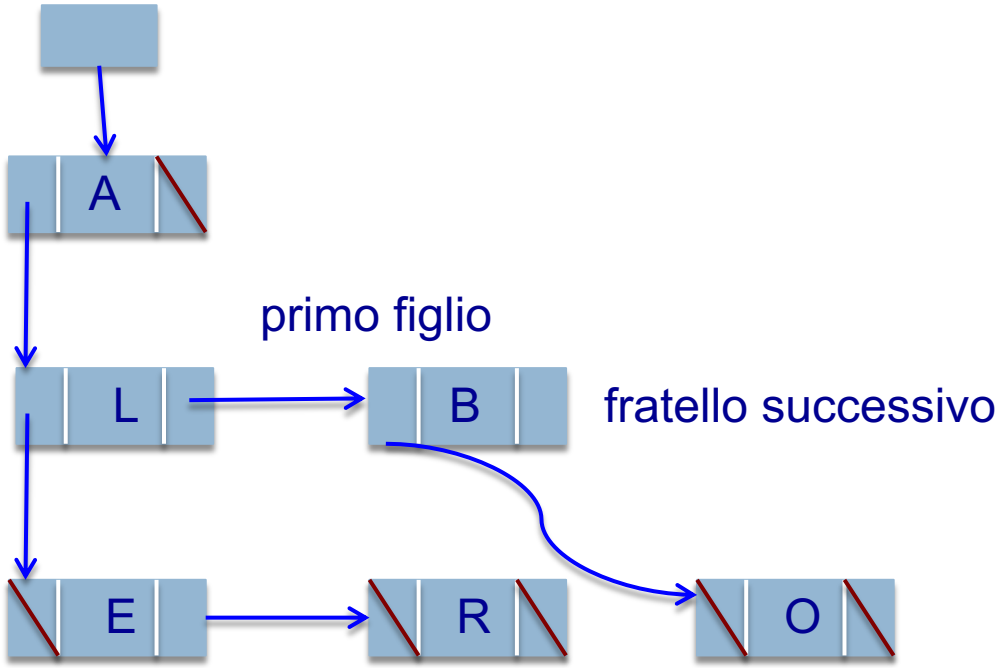
Rappresentazione con lista di puntatori ai figli

Primo figlio-fratello successivo

- Una variante alla soluzione precedente senza usare una struttura dati addizionale
  - Per ogni nodo è possibile mantenere un puntatore al primo figlio (posto a null se non ci sono figli) e un puntatore al fratello successivo

- Lo spazio richiesto è O(n)

Primo figlio-fratello successivo



Rappresentazione primo figlio-fratello successivo

Visita generica di un albero

- In molti casi è necessario attraversare un albero visitandone tutti i nodi
- Diversamente da collezioni di oggetti rappresentati in modo lineare negli alberi è necessario seguire tutti i rami possibili partendo dalla radice
- L'algoritmo di visita generica applicato alla radice di un albero con n nodi termina in O(n) iterazioni
  - Lo spazio usato è O(n)

Visita generica di un albero

```
algoritmo visitaGenerica (nodo r)

S ←{r}

while (S ≠ ∅ ) do

   estrai un nodo u da S
   visita il nodo u

   S ← S ∪{figli di u}
```

Algoritmo di visita generica a partire da un nodo r

Visita in profondità

- Nell'algoritmo di visita generica
  - possiamo rappresentare l'insieme dei nodi aperti S mediante il tipo Pila

- Otteniamo la visita in profondità (Depth-First Search o DFS)

Pila di dati

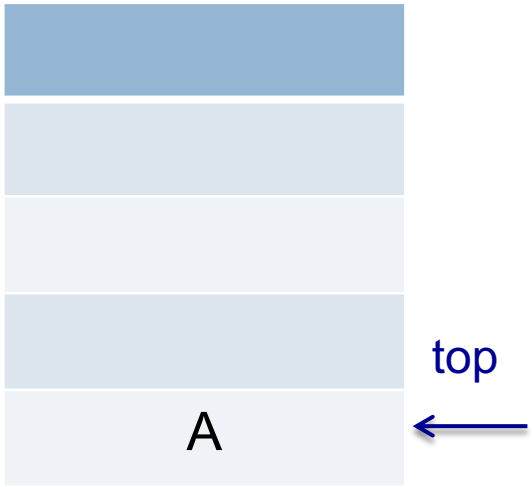
- Una pila di dati (o stack) è una lista ordinata dove gli inserimenti e le cancellazioni vengono effettuati ad un estremo della lista chiamata top

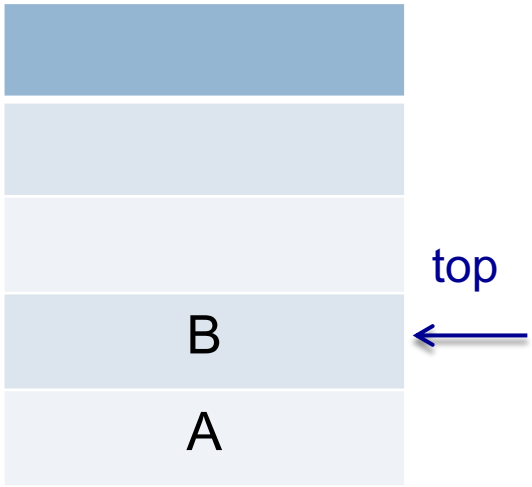
- Questa particolare disciplina di accesso è chiamata LIFO (Last-In-First-Out)

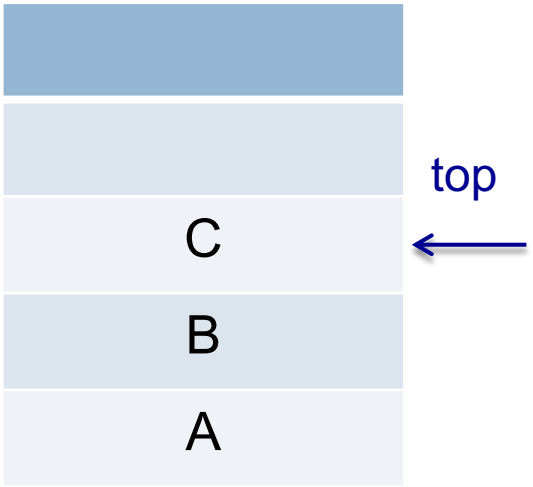
Tipo dato Pila

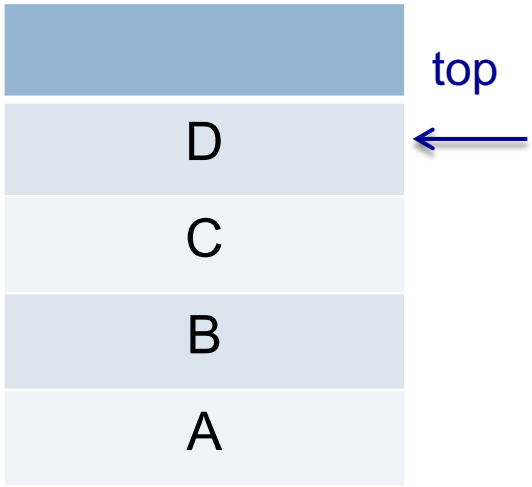
```
tipo Pila:

dati:
  una sequenza S di n elementi

operazioni:
    isEmpty() → result
      restituisce true se S è vuota, e false
      altrimenti

    push(elem e)
      aggiunge e come ultimo elemento di S

    pop() → elem
      toglie da S l'ultimo elemento e lo
      restituisce

    top() → elem
      restituisce l'ultimo elemento di S (senza eliminarlo)
```

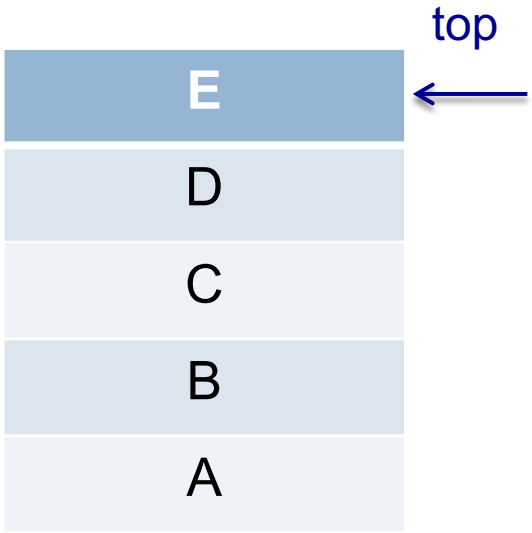

Operazioni sulla Pila

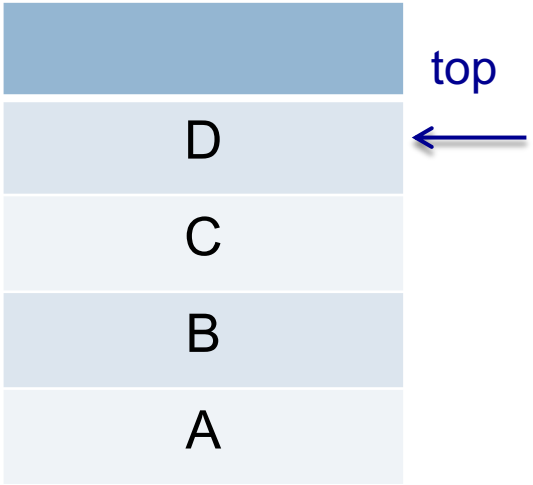












Esempio di utilizzo della Pila

Visita in profondità

```
algoritmo visitaDFS (nodo r)

Pila S
S.push(r)

while (not S.isEmpty()) do

      u ← S.pop()

      if (u ≠ null) then
        visita il nodo u
        S.push(figlio destro di u)
        S.push(figlio sinistro di u)
```


Visita in profondità

- In una visita in profondità si prosegue la visita dall'ultimo nodo lasciato in sospeso
  - tenderemo a seguire tutti i figli sinistri andando in profondità fino a che non si raggiunge la prima foglia sinistra
- L'ordine di visita dell'albero dell'esempio precedente è ALERBO
- Impilando prima il figlio sinistro si ottiene l'effetto simmetrico

Visita in profondità ricorsiva

- Esiste una realizzazione ricorsiva dell'algoritmo DFS

- La pila S non appare esplicitamente nell'algoritmo in quanto è implicito nella ricorsione

Visita in profondità

```
algoritmo visitaDFSRicorsiva (nodo r)

if (r = null) then return

visita il nodo r

visitaDFSRicorsiva (figlio sinistro di r)
visitaDFSRicorsiva (figlio destro di r)
```


Visita in profondità ricorsiva

- Posizionando diversamente l'operazione di visita del nodo r possiamo ottenere varianti dell'algoritmo in cui l'ordine di visita dei nodi cambia
  - Visita in preordine
    - Si visita prima la radice
    - I risultati sull'albero di esempio: ALERBO
  - Visita simmetrica
    - Si effettua prima la chiamata ricorsiva sul figlio sinistro, poi si visita la radice
    - I risultati sull'albero di esempio: ELRABO
  - Visita in postordine
    - Si effettuano prima le chiamate ricorsive e infine si visita la radice
    - I risultati sull'albero di esempio: ERLOBA

Visita in ampiezza

- Partendo dall'algoritmo di visita generica e rappresentando l'insieme S mediante il tipo di dato Coda
  - otteniamo la visita in ampiezza (Breadth-First search o BFS)

Coda

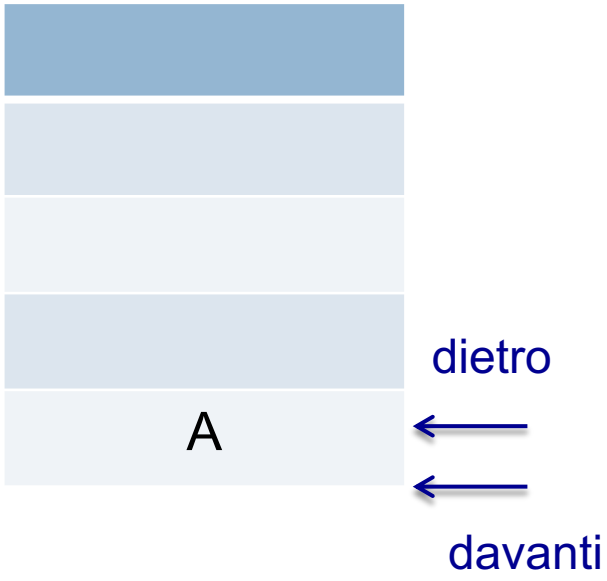
- Una coda (queue) è una lista ordinata nella quale tutti gli inserimenti avvengono ad un'estremo e tutte le cancellazioni avvengono all'estremo opposto

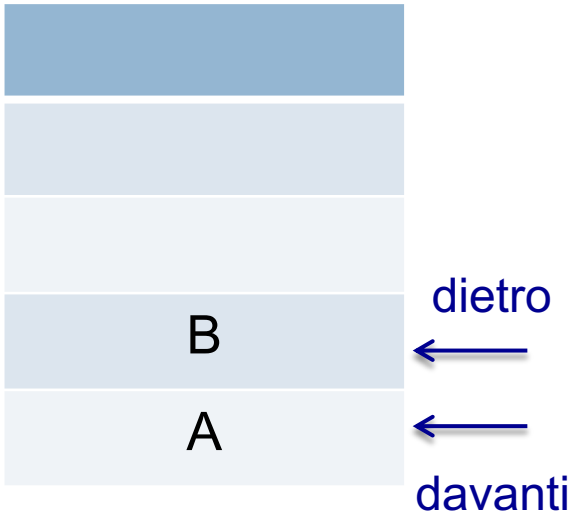
- Questa particolare disciplina di accesso è chiamata FIFO (First-In-First-Out)

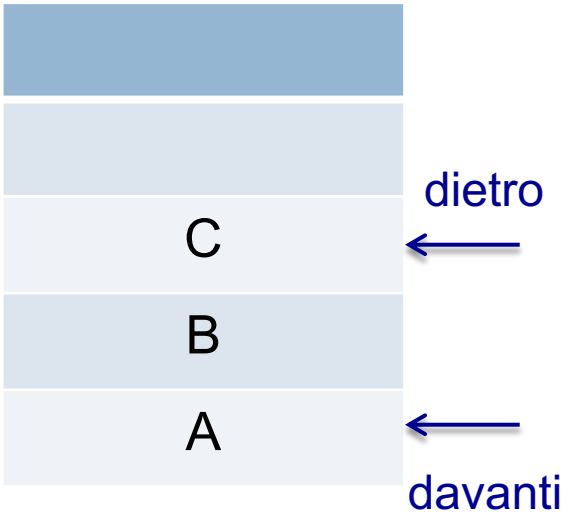
Tipo dato Coda

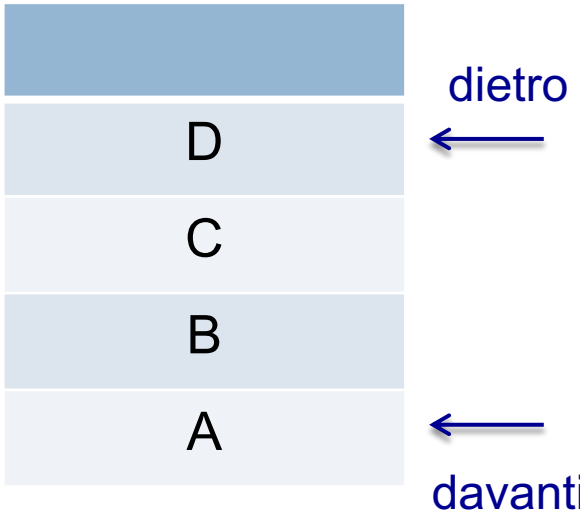
```
tipo Coda:

dati:
  una sequenza S di n elementi

operazioni:
    isEmpty()→ result
      restituisce true se S è vuota, e false
      altrimenti

    enque(elem e)
      aggiunge e come ultimo elemento di S

    dequeue()→ elem
      toglie da S il primo elemento e lo restituisce

    top() → elem
      restituisce il primo elemento di S (senza eliminarlo)
```

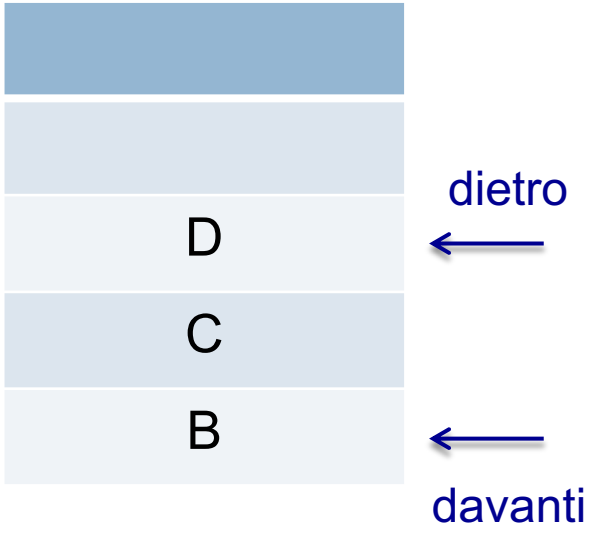

Operazioni sulla coda











Esempio di utilizzo di una coda

Visita in ampiezza

```
algoritmo visitaBFS (nodo r)

Coda C
C.enqueue(r)

while (not C.isEmpty()) do
      u ← C.dequeue()

      if (u ≠ null) then
        visita il nodo u
        C.enqueue(figlio sinistro di u)
        C.enqueue(figlio destro di u)
```


Visita in ampiezza

- La caratteristica principale della visita in ampiezza è che i nodi vengono visitati per livelli
  - Si visita prima la radice, poi i figli della radice, poi i figli dei figli, e così via

- I risultati sull'albero di esempio: ALBERO