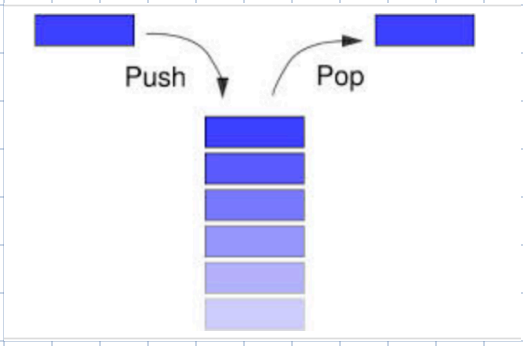
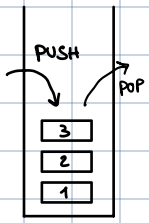
STACK / PILA

POLITICA: L.I.F.O.





isEmpty() → TRUE SE È VUOTA, FALSE ALTRIMENTI

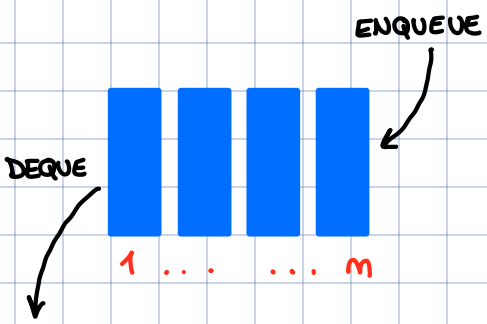
PUSH (e) → AGGIUNGE e IN CIMA

POP() → RESTITUISCE L'ELEMENTO IN CIMA

TOP() → RESTITUISCE L'ELEMENTO IN CIMA SENZA TOGLIERLO

CODA / QUEUE

POLITICA: F.I.F.O.



isEmpty() → TRUE SE È VUOTA, FALSE ALTRIMENTI

ENQUEUE (e) → AGGIUNGE e COME ULTIMO ELEMENTO

DEQUEUE() → RESTITUISCE IL PRIMO ELEMENTO TOGLIENDOLO

FIRST() → RESTITUISCE IL PRIMO ELEMENTO SENZA TOGLIERLO

DIZIONARIO

INSIEME DI COPPIE (CHIAVE, ELEMENTO)

INSERT (K, V) → AGGIUNGE AL DIZIONARIO LA COPPIA (K, V)

DELETE (K) → RIMUOVE LA COPPIA CON CHIAVE K

SEARCH (K) → RESTITUISCE L'ELEM. CON CHIAVE K, ALTRIMENTI NULL

TECNICHE DI RAPPRESENTAZIONE DEI DATI

RAPPRESENTAZIONE INDICIZZATA → ARRAY

- PROPRIETÀ FORTE → INDICI DELLE CELLE SONO NUMERI CONSECUTIVI

- PROPRIETÀ DEBOLE → NON POSSIAMO AGG. NUOVE CELLE

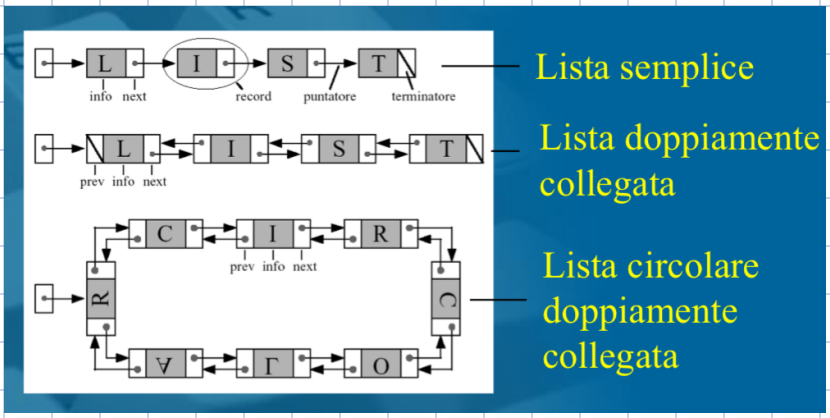
- PRO → ACCESSO DIRETTO AI DATI TRAMITE INDICI
- CONTRO → DIMENSIONE FISSA (RIALLOCAZIONE RICHIEDE TEMPO LINEARE)

RAPPRESENTAZIONE COLLEGATA → RECORD COLLEGATI DA PUNTATORI
↳ NUMERATI CON INDIRIZZO DI MEMORIA
↳ CREATI e DISTRUTTI INDIVIDUALMENTE e DINAMICAMENTE
↳ COLLEGAMENTO TRA RECORD A e RECORD B TRAMITE PUNTATORE

- PROP. FORTE → POSSIAMO AGG. O TOGLIERE RECORD
- PROP. DEBOLE → GLI INDIRIZZI DEI RECORD NON SONO SEMPRE CONSECUTIVI

- PRO → AGGIUNTA E RIMOZIONE IN TEMPO COSTANTE
- CONTRO → ACCESSO SEQUENZIALE AI DATI

ESEMPIO



STRUTTURE COLLEGATE

COSTRUZIONE DI UN DIZIONARIO

CON STRUTTURE INDICIZZATE (ARRAY)

Metodo più semplice: array non ordinato (sovradimensionato)

Insert → costa O(1) – inserisco dopo ultimo elemento
Search → costa O(n) – devo scorrere l'array
Delete → costa O(n) – delete = search + cancellazione

Array ordinato:

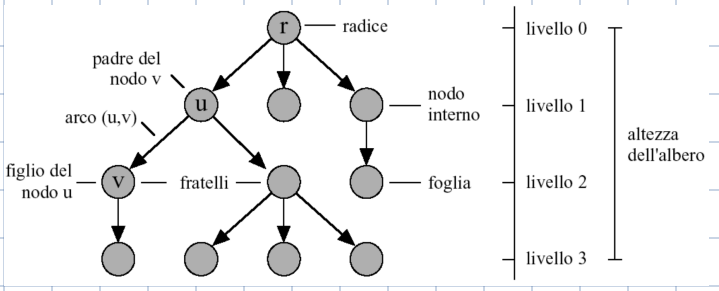
Search → O(log(n)) – ricerca binaria
Insert → O(n)
Ho bisogno di:
O(log(n)) confronti → per trovare la giusta posizione in cui inserire l'elemento
O(n) trasferimenti → per mantenere l'array ordinato
(Ricorda che O(n) + O(log(n)) = O(n))
Delete → O(n) (come per Insert)

CON STRUTTURE LEGATE (LISTE)

Lista non Ordinata
Search – O(n)
Insert – O(1)
Delete - O(n)

Lista Ordinata
Search – O(n) non posso usare la ricerca binaria
Insert – O(n) devo mantenere ordinata la lista
Delete – O(n)

ALBERI



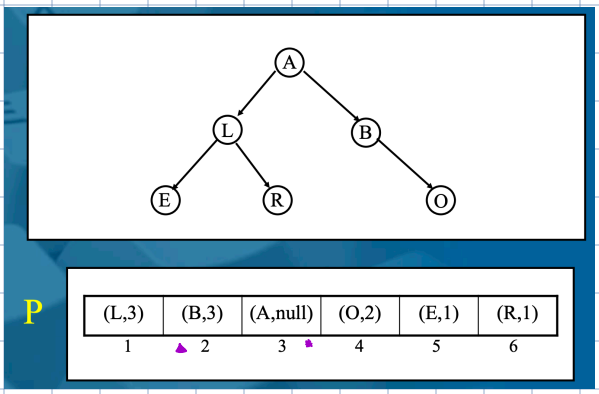
GRADO DI UN NODO → # DI FIGLI

ALBERO RADICATO → ALBERO CON UN NODO SPECIFICO CONTRASSEGNATO COME RADICE

RAPPRESENTAZIONI INDICIZZATE PER ALBERI

VETTORE DEI PADRI

NODO → CONTIENE COPPIA (VALORE, INDEX - PADRE)



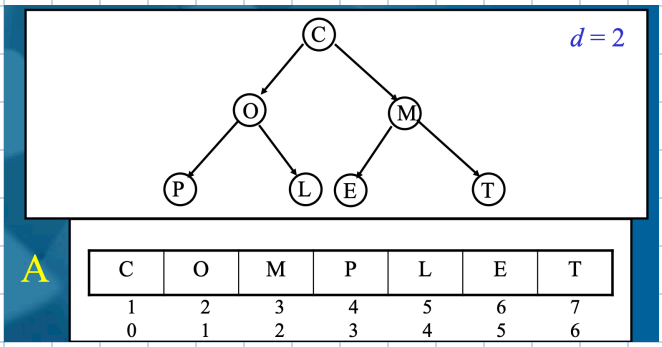
- # ARBITRARIO DI FIGLI
- TEMPO PER INDIVIDUARE IL PADRE DI UN NODO $O(1)$
- TEMPO PER INDIVIDUARE UNO o PIÙ FIGLI DI UN NODO $O(m)$

▲ NODO B HA COME PADRE A CHE SI TROVA IN POSIZIONE 3 NELL' ARRAY

* NODO A NON HA NESSUN PADRE

VETTORE POSIZIONALE

NODI SALVATI NELL' ARRAY PER LIVELLI



- J-ESIMO FIGLIO DI i IN POSIZ. → $d \cdot i + J$
- PADRE DI i IN POSIZ → $\lfloor (i-1)/d \rfloor$

ATTENZIONE LE FORMULE SOPRA SONO PER GLI ARRAY CHE PARTONO DA 0

$(d \cdot i) + J$

J-ESIMO FIGLIO CHE VOGLIO

FATTORE DI RAMIFICAZIONE

INDEX PADRE

$\lfloor (i-1)/d \rfloor$

FATTORE DI RAMIFICAZIONE

INDEX DEL NODO SU CUI SONO

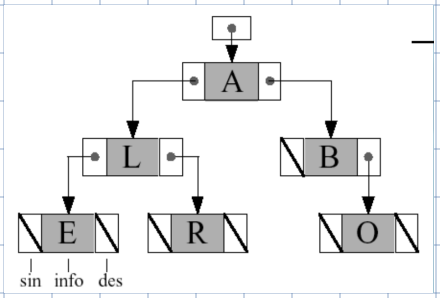
MOLTIPLICANDO $i \cdot d$ FACCIO IN MODO DI "SALTARE" LE CELLE CHE CI SONO PRIMA DEI FIGLI DEL NODO IN POS. $\underline{i}$

- # DI FIGLI ESATTAMENTE d
- SOLO PER ALBERI COMPLETI O QUASI

- TEMPO PER INDIVIDUARE IL PADRE DI UN NODO $O(1)$
- TEMPO PER INDIVIDUARE LO SPECIFICO FIGLIO DI UN NODO $O(1)$

RAPPRESENTAZIONI COLLEGATE PER ALBERI

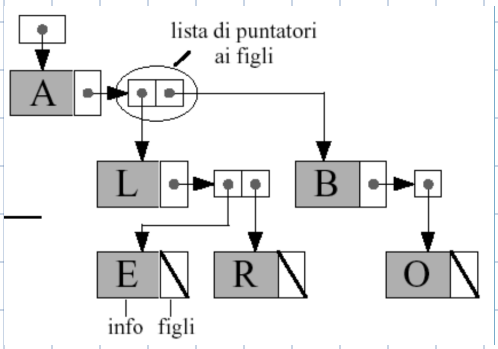
CON PUNTATORI AI FIGLI



OGNI NODO HA:

- DATO CONTENUTO
- PUNTATORE AL FIGLIO SX
- PUNTATORE AL FIGLIO DX
- NUMERO LIMITATO DI FIGLI

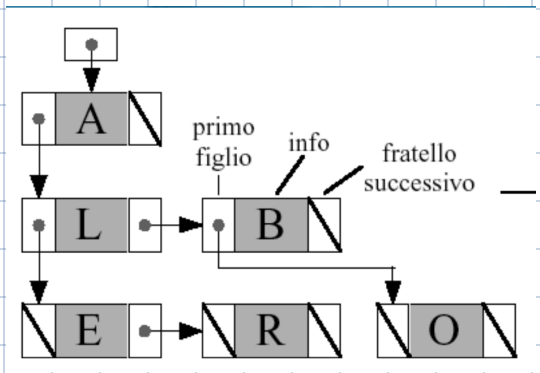
CON LISTE DI PUNTATORI AI FIGLI



OGNI NODO HA:

- DATO CONTENUTO
- PUNTATORE A LISTA DI PUNTATORI CHE PUNTANO AI FIGLI
- NUMERO ARBITRARIO DI FIGLI

TIPO PRIMO FIGLIO - FRATELLO SUCCESSIVO



OGNI NODO HA:

- DATO CONTENUTO
- PUNTATORE AL FIGLIO
- PUNTATORE AL FRATELLO SUCCESSIVO

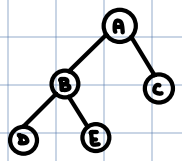
VISITA IN PROFONDITÀ (DFS) (PER ALBERI BINARI)

UTILIZZA STRUTTURA DATI STACK / PILA ⟶ L.I.F.O.

algoritmo visitaDFS(*nodo r*)
  Pila S
  S.push(*r*)
  while (**not** S.isEmpty()) **do**
    $u \leftarrow$ S.pop()
    if ($u \neq$ null) **then**
      visita il nodo u
      S.push(figlio destro di *u*)
      S.push(figlio sinistro di *u*)

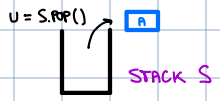
1 2 3

OGNI VOLTA CHE ESAMINA UN NODO PRENDE I SUOI FIGLI E LI PUSHA NELLO STACK, METTENDOLI IN CIMA SARANNO I PROSSIMI AD ESSERE ESTRATTI QUINDI PROSEGUE IN PROFONDITÀ.



1 CREO LO STACK S E INSERISCO NELLO STACK IL NODO R (LA RADICE SE INIZIAMO DA LI)

2 FINCHÉ LO STACK NON È VUOTO ESTRAGGO L'ELEMENTO IN CIMA (CIOÈ L'ULTIMO AGGIUNTO)



3 SE IL NODO ESTRATTO NON È NULL ALLORA LO VISITO/ESAMINO FACENDO LE OPPORTUNE OPERAZIONI CHE DEVO COMPIERE, DOPODICHÉ INSERISCO IN CIMA ALLO STACK IL FIGLIO DX E POI IL FIGLIO SX.

ORDINE DI VISITA: A, B, D, E, C

COMPLESSITÀ: $T(m) = O(m)$!! → OGNI NODO VIENE INSERITO ED ESTRATTO DALLA PILA UNA SOLA VOLTA

→ TEMPO SPESO SU OGNI NODO $O(1)$

↳ SE SO INDIVIDUARE I FIGLI DEL NODO IN TEMPO COSTANTE

VISITA DFS RICORSIVA (PER ALBERI BINARI)

algoritmo `visitaDFSRicorsiva`(*nodo r*)

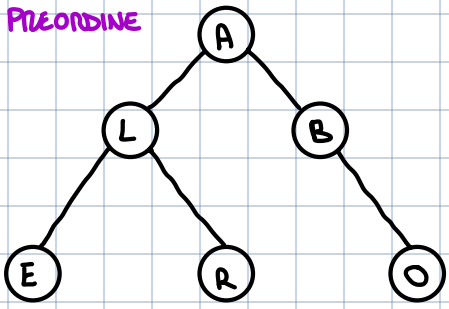
1. **if** ($r \neq$ `null`) **then**
2. *visita il nodo r*
3. `visitaDFSRicorsiva`(figlio sinistro di r)
4. `visitaDFSRicorsiva`(figlio destro di r)

VISITA IN PREORDINE: RADICE, SOTTOALBERO SX, SOTTOALBERO DX

VISITA SIMMETRICA: SOTTOALBERO SX, RADICE, SOTTOALBERO DX — SCAMBIO RIGA 2 e 3

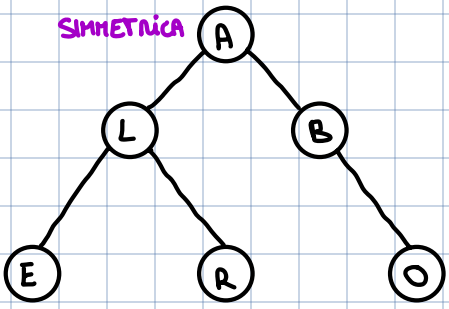
VISITA IN POSTORDINE: SOTTOALBERO SX, SOTTOALBERO DX, RADICE — SPOSTO RIGA 2 DOPO 4

PREORDINE



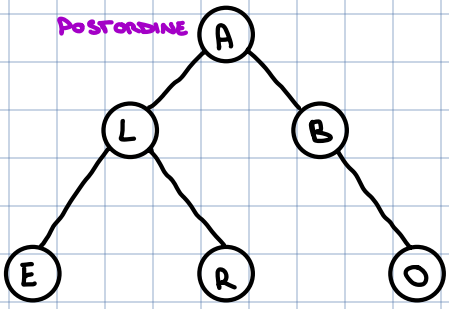
A, L, E, R, B, O

SIMMETRICA



E, L, R, A, B, O

POSTORDINE



E, R, L, O, B, A

VISITA IN AMPIEZZA (BFS) (PER ALBERI BINARI)

UTILIZZA STRUTTURA DATI CODA/QUEUE → F.I.F.O.

```
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)
```

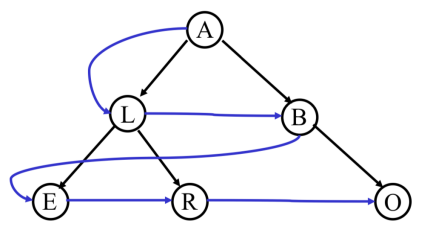
1 VENGONO ESAMINATI TUTTI I NODI LIVELLO PER LIVELLO, DA SX A DESTRA VISTO CHE VIENE PUSHATO PRIMA IL FIGLIO SX.

2 SICCOME LA CODA ESTRAI GLI ELEMENTI IN ORDINE DI ARRIVO, LA VISITA È "ORIZZONTALE" LIVELLO PER LIVELLO.

1 CREO LA QUEUE E CI INSERISCO IL NODO r

2 FINCHÉ LA CODA NON È VUOTA ESTRAGGO IL PRIMO NODO USANDO LA DEQUE() (IN ORDINE DI ARRIVO)

3 INSERISCO CON LA ENQUEUE() I FIGLI SX e DX, VERRANNO ESTRATTI IN ORDINE DI ARRIVO QUINDI PRIMA SX.



ORDINE DI VISITA: A, L, B, E, R, O

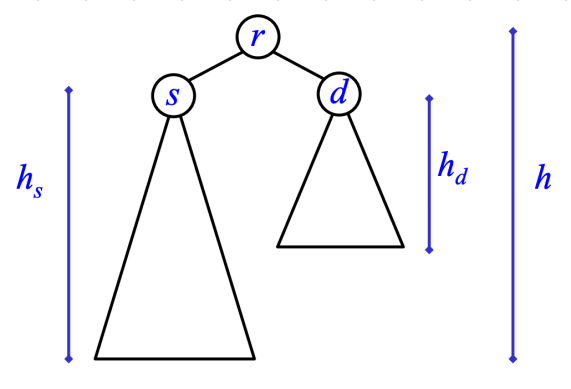
COMPLESSITÀ: $T(m) = O(m)$!! → OGNI NODO VIENE INSERITO ED ESTRATTO DALLA PILA UNA SOLA VOLTA

→ TEMPO SPESO SU OGNI NODO $O(1)$

↳ SE SO INDIVIDUARE I FIGLI DEL NODO IN TEMPO COSTANTE

CALCOLO ALTEZZA DI UN ALBERO CON VISITA

POSSIAMO UTILIZZARE GLI ALGORITMI DI VISITA ANCHE PER CALCOLARE L'ALTEZZA DI UN ALBERO



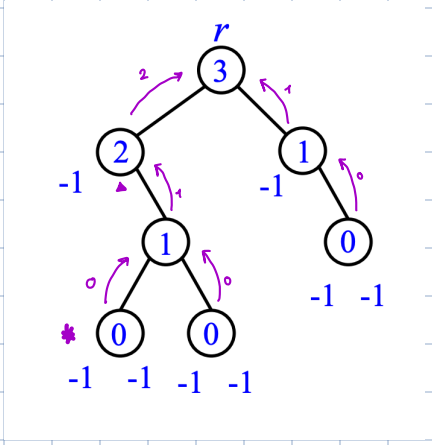
$$h = 1 + \max\{h_s, h_D\}$$

PER LA RADICE

MASSIMO TRA L'ALTEZZA DEL SOTTOALBERO SX e SOTT. DX

CalcolaAltezza (nodo r)

1. **if** (r = null) **then return** -1
2. sin = CalcolaAltezza(figlio sinistro di r)
3. des = CalcolaAltezza(figlio destro di r)
4. **return** 1+max{sin, des} *



▲ SCEGLIE SEMPRE IL MAX TRA h_s e h_D, IN QUESTO CASO SCEGLIE TRA -1 e 1 → 1

POI CI SOMMA 1

SOMMANDO 1 AI NODI NULL CHE RESTITUISCONO -1, OTTENIAMO 0.

COMPLESSITÀ: $O(m)$!!

DFS RICORSIVA ALBERI

SCENDE VISITANDO LA RADICE, PRIMA IL NODO SINISTRO, QUANDO RISALE VISITA IL FIGLIO DESTRO. UT

PREORDINE: RADICE, SX, DX

SIMMETRICA: SX, RADICE, DX

POSTORDINE: SX, DX, RADICE $O(m)$

UTILIZZATA PER VISITARE TUTTI I NODI DELL' ALBERO SCENDENDO IN PROFONDITÀ.

BFS

SCENDE ESPLORANDO L'ALBERO PER LIVELLI, UTILE PER TROVARE AD ESEMPIO IL CAMMINO MINIMO VERSO UN NODO IN UN ALBERO NON PESATO. COSTO $O(m)$

CALCOLA ALTEZZA ALBERO

SCENDE IN PROFONDITÀ UTILIZZANDO LA DFS, OGNI VOLTA CHE RISALE RITORNA 1 + MAX (SOTTOALBERO SX, SOTTOALBERO DX), L'1 SAREBBE L'ARCO DI PROFONDITÀ COLLEGATO ALLA RADICE. $O(m)$