Algoritmi e Strutture Dati

Lezione 14

28 ottobre 2022

Tipo Pila

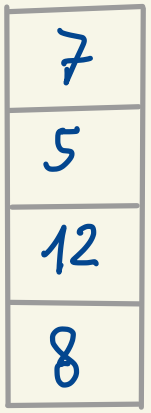
Stack

Collezione di dati con organizzazione *Last–In–First–Out*

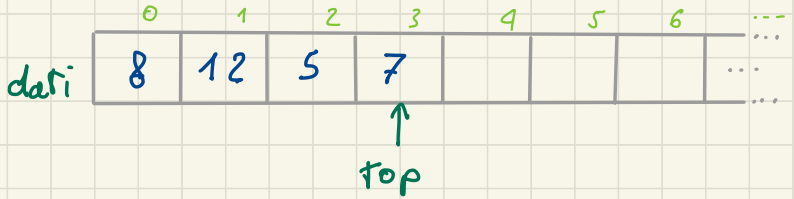
LIFO

Operazioni

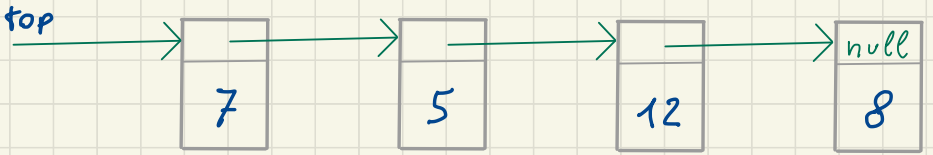
- isEmpty() → boolean
- push(elemento)
- pop() → elemento
- top() → elemento



PILA: implementazione mediante array



PILA: implementazione mediante liste



7
5
12
8

Tipo Coda

Collezione di dati con organizzazione *First–In–First–Out*

FIFO

Operazioni

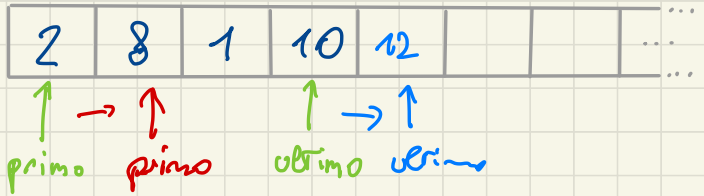
- isEmpty() $\rightarrow$ boolean
- enqueue(elemento)
- dequeue() $\rightarrow$ elemento
- first() $\rightarrow$ elemento

4 10 8 3 11

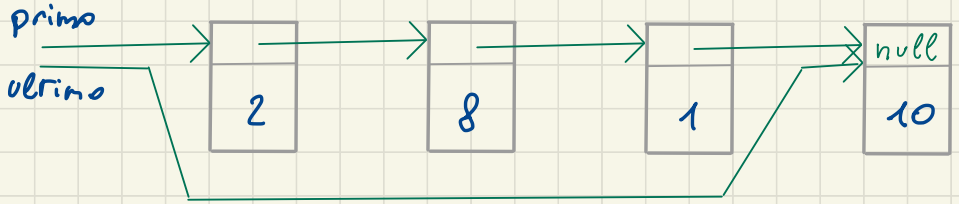
CODA: implementazione mediante array

dequeue

enqueue (12)



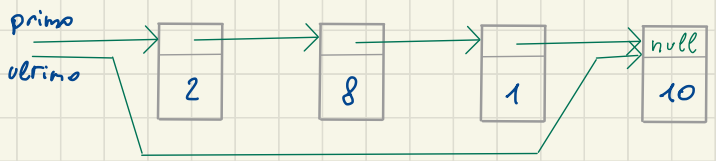
CODA: implementazione mediante liste



coda vuota

primo = null

ultimo = null

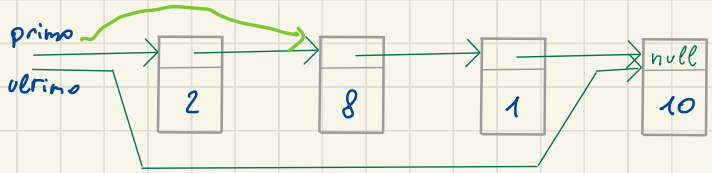


FUNZIONE isEmpty() → boolean

IF primo = null THEN RETURN true
ELSE RETURN false

FUNZIONE first() → elemento

RETURN primo.dato



primo null

FUNZIONE dequeue ( ) → elemento

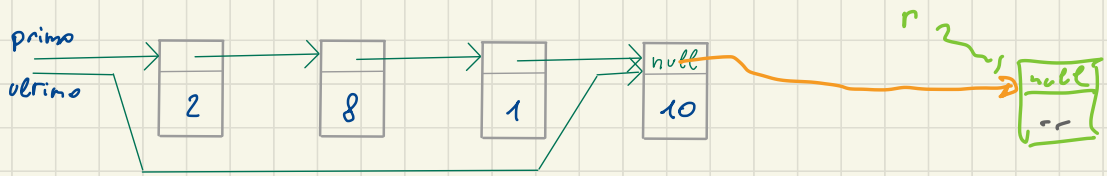
x ← primo.dato

primo ← primo.pros

IF primo = null THEN

ultimo ← null

RETURN x




```
PROCEDURA enqueue (elemento x)
    r ← riferimento a un nuovo nodo
    r.dato ← x
    r.pros ← null
    IF primo = null THEN   // coda vuota
    |   primo = r
    |   ultimo = r
    ELSE
    |   ultimo.pros ← r
    |   ultimo ← r
```

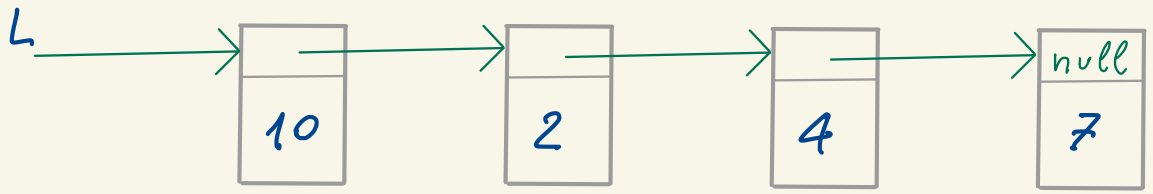
primo null
ultimo null

Operazioni in tempo $O(1)$

Dalle liste agli alberi

Liste lineari

- Collezioni di nodi collegati tramite riferimenti



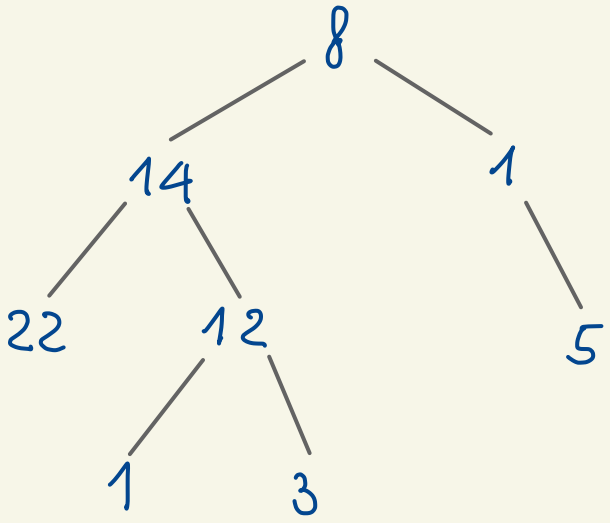
- Ad ogni nodo è possibile associare un *unico successore*

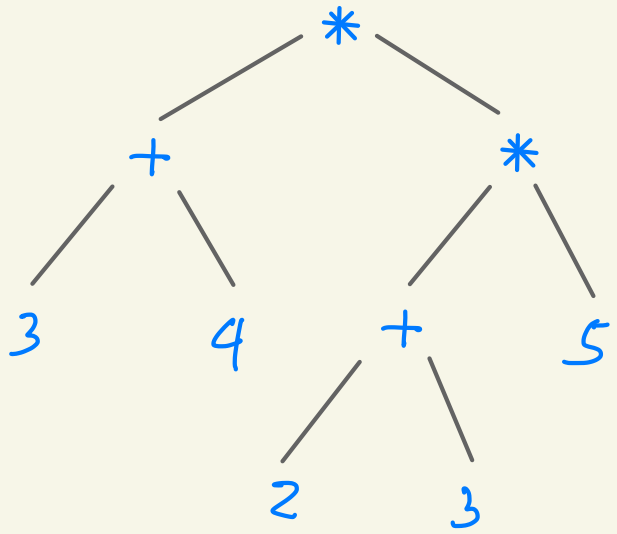
Mediante riferimenti/puntatori è possibile definire strutture più complicate, ma più flessibili

Dalle liste agli alberi

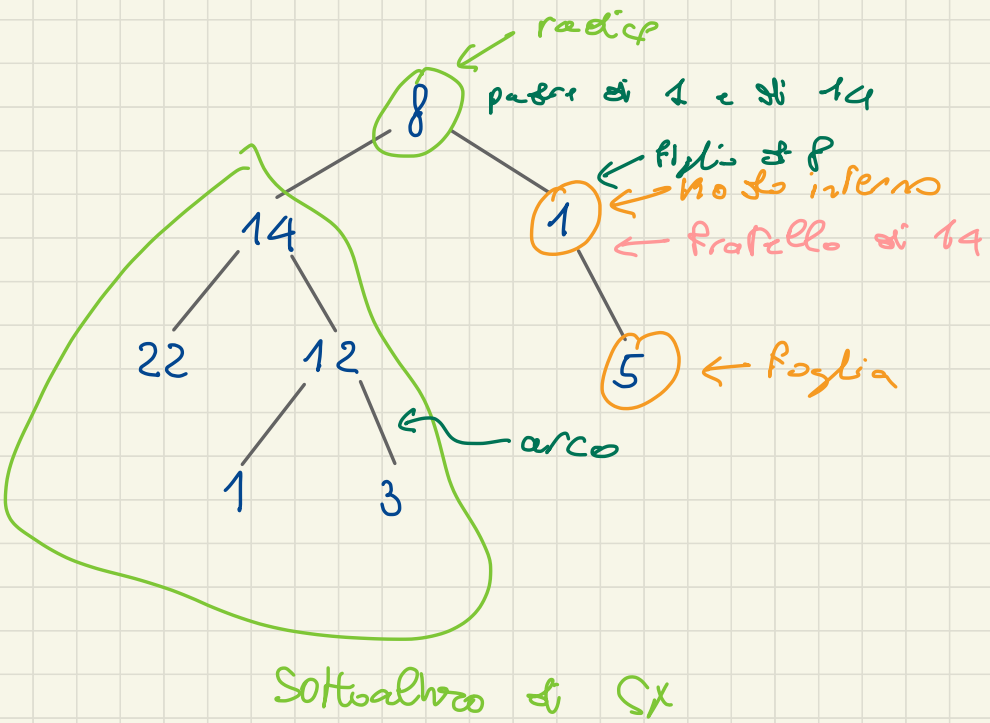
Alberi binari

- Ad ogni nodo possono essere associati *due* "successori" detti *figlio sinistro* e *destro*





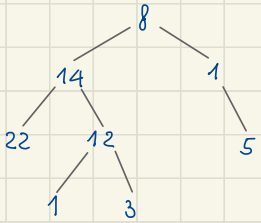
(3 + 4) * (2 + 3) * 5

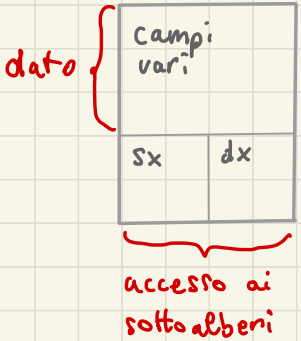


Profondità o livello di un nodo:

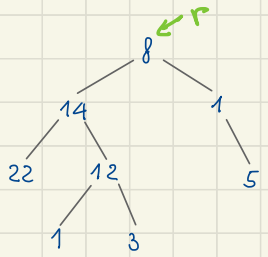
di archi attraversati per raggiungere il nodo dalla radice

Altezza = max prof dei nodi


8
14
1
22
12
5
1
3


dato
campi vari
sx
dx
accesso ai sottoalberi

VISITE AD ALBERI BINARI

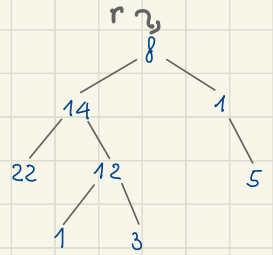



```
ALGORITMO visitaGenerica(AlberoBinario r)
    S ← {r}
    WHILE S ≠ ∅ DO
        preleva un nodo x da S
        visita x
        S ← S ∪ {figli di x}
```

VISITA in AMPIEZZA

S = coda

```
ALGORITMO visitaInAmpiezza (AlberoBinario r)
    C ← coda vuota
    C.enqueue(r)
    WHILE NOT(C.isEmpty()) DO
        x ← C.dequeue()
        IF x ≠ null THEN
            visita nodo associato a x
            C.enqueue(x.sx)
            C.enqueue(x.dx)
```




```
ALGORITMO visitaGenerica (AlberoBinario r)
    S ← {r}
    WHILE S ≠ ∅ DO
        preleva un nodo x da S
        visita x
        S ← S ∪ {figli di x}
```

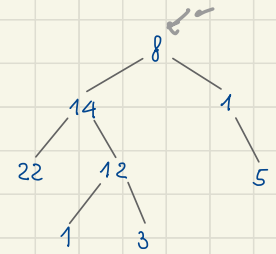
8 14 1 22 12 5 1 3

C 8 ~~14~~ ~~1~~ ~~22~~ ~~12~~ ~~null~~ ~~5~~ ~~null~~ ~~null~~ ~~1~~ ~~3~~ ~~null~~ ~~null~~ ~~null~~ ~~null~~ ~~null~~ ~~null~~

VISITA in PROFONDITA'

S ≡ Pila

```
ALGORITMO visitaProfondità (AlberoBinario r)
    P ← pila vuota
    P.push(r)
    WHILE NOT (P.isEmpty()) DO
        x ← P.pop()
        IF x ≠ null THEN
            visita nodo associato a x
            P.push(x.dx)
            P.push(x.sx)
```



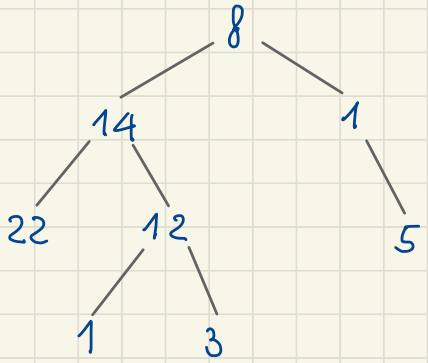

```
ALGORITMO visitaGenerica (AlberoBinario r)
    S ← {r}
    WHILE S ≠ ∅ DO
        preleva un nodo x da S
        visita x
        S ← S ∪ {figli di x}
```

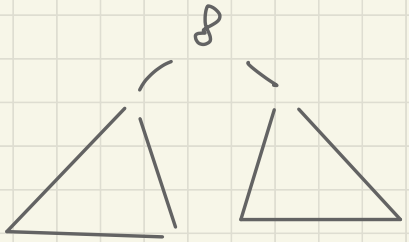
8 14 22 12 1 3 1 5

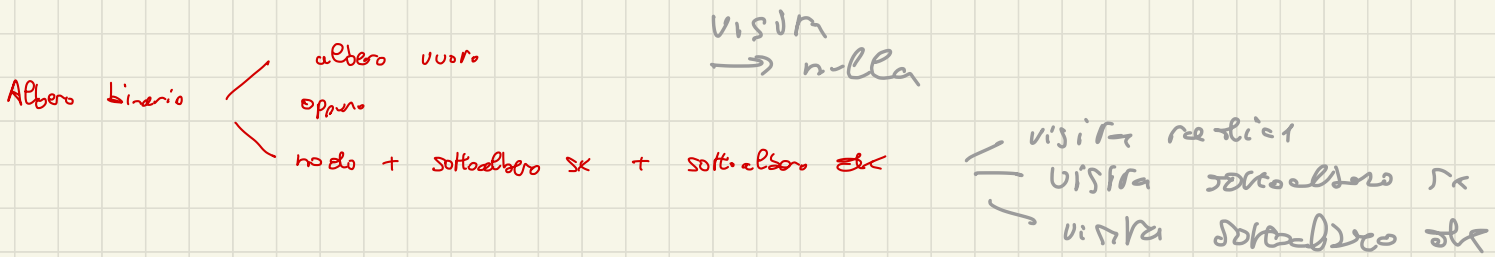
P 8 1 14 12 22 null null 1 3 1 null null 5 null

Albero binario

- albero vuoto
- oppure
- nodo + sottoalbero SX + sottoalbero DX

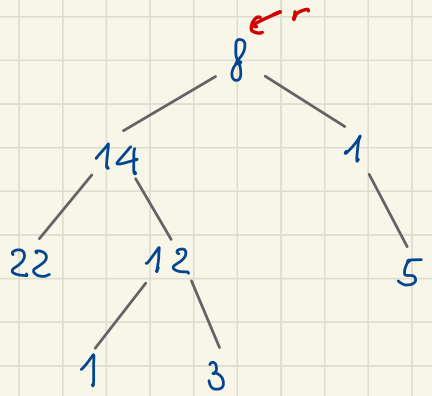







```
ALGORITMO visitaInProfondità (AlberoBinario r)
    IF r ≠ null THEN    // e' albero non è vuoto
        visita la radice
        visitaInProfondità (r.sx)
        visitaInProfondità (r.dx)
    // else non fare nulla
```

```
ALGORITMO visitaProfondità (alberoBinario r)
  IF r ≠ null THEN
    visita la radice
    visitaInProfondità (r.sx)
    visitaInProfondità (r.dx)
```



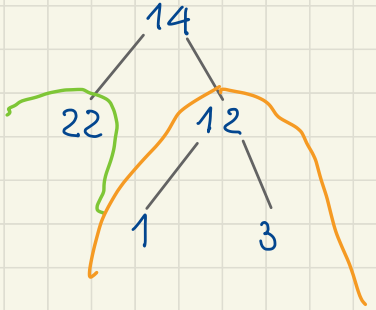
8 14 22 12 1 3 1 5

sx dx

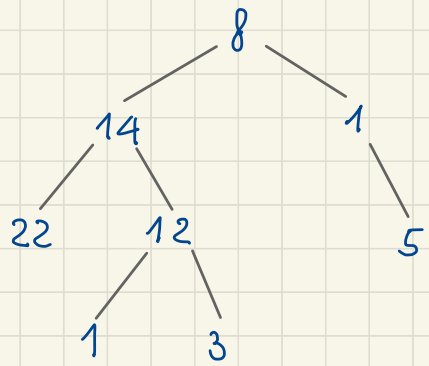
VISITA IN PROFONDITÀ

in ordine ANTICIPATO

(Pre-ordine)



```
ALGORITMO visitaProfondità (albero binario r)
  IF r ≠ null THEN
    visita la radice
    visitaInProfondità (r.sx)
    visitaInProfondità (r.dx)
```




```
ALGORITMO visitaProfondità (albero binario r)
  IF r ≠ null THEN
    visitaInProfondità (r.sx)
    visitaInProfondità (r.dx)
    visita la radice
```

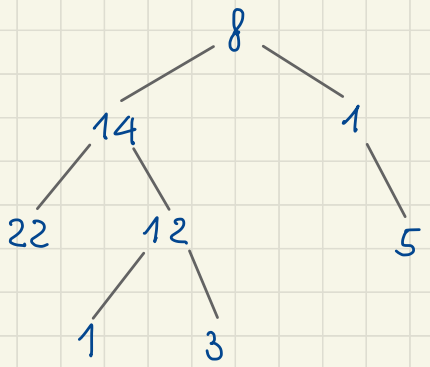
22 1 3 12 14 5 1 8

sx dx

ORDINE POSTICIPATO

POST-ordine

```
ALGORITMO visitaProfondita (alberoBinario r)
  IF r ≠ null THEN
     visitaInProfondita (r.sx)
     visita la radice
     visitaInProfondita (r.dx)
```

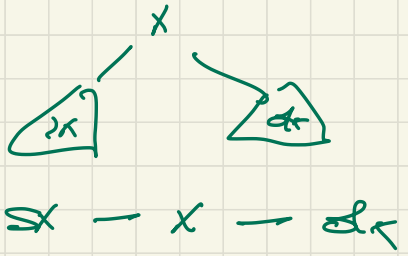


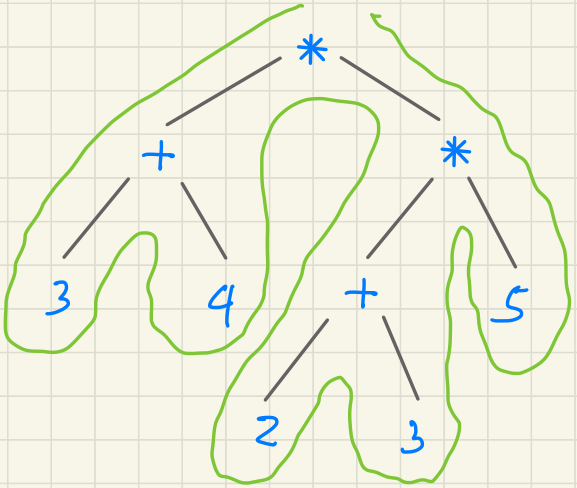
22 14 1 12 3 8 1 5

sx dx

ORDINE SIMMETRICO

In-Ordine





$(3+4) * (2+3) * 5$

ANTICIPATO * + 3 4 * + 2 3 5

SIMMETRICO 3 + 4 * 2 + 3 * 5

POSTICIPATO 3 4 + 2 3 + 5 * *

NOTAZIONE POSTFISSA POLACCA INVERSA