Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Strutture dati fondamentali: Alberi 2

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Rappresentazione di albero binario in memoria.
Descriveremo tre diverse rappresentazioni:

- **Memorizzazione tramite record e puntatori**
- **Rappresentazione posizionale**
- **rappresentazione tramite vettore dei padri**

Memorizzazione tramite record e puntatori:

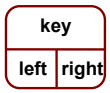
Il modo più naturale di rappresentare e gestire gli alberi **binari** è per mezzo dei puntatori.
Ogni singolo nodo è costituito da un record contenente:

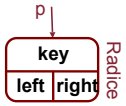
key: le opportune informazioni pertinenti al nodo stesso;

left: il puntatore al figlio sinistro (oppure *None* se il nodo non ha figlio sinistro);

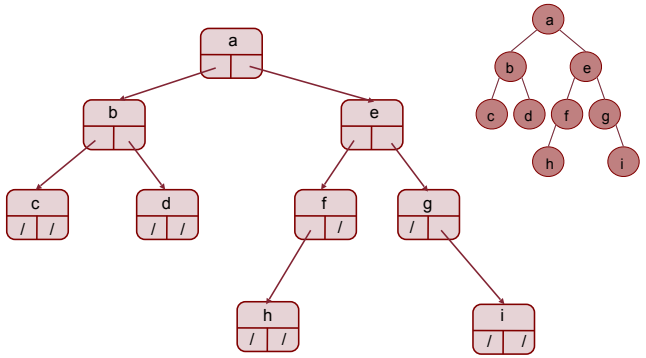
right: il puntatore al figlio destro (oppure *None* se il nodo non ha figlio destro);

L'albero viene acceduto per mezzo del puntatore alla radice.





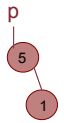
Memorizzazione tramite record e puntatori (segue):



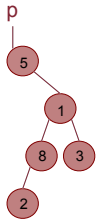
```
class NodoAB:
   def __init__(self, key=None, left=None, right=None):
      self.key = key
      self.left = left
      self.right=right
```

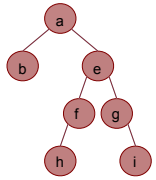
```
>>> p=NodoAB(5)
```


```
>>> p.right=NodoAB(1)
```




```
>>> p.right.left=NodoAB(8)
>>> p.right.right=NodoAB(3)
>>> p.right.left.left=NodoAB(2)
```





ESERCIZIO: scrivere una funzione $generaAlbero(n, m)$ che genera un albero a caso con n nodi aventi chiavi nell'intervallo $[1,\ldots,m]$

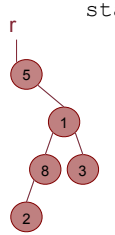
```
import random

def generaAlbero(n,m):
    if n==0:
        return None
    p=NodoAB(random.randint(1,m))
    s=random.randint(0,n-1)
    p.left=generaAlbero(s,m)
    p.right=generaAlbero(n-1-s,m)
    return p
```

```
>>> r=generaAlbero(10,100)
```

ESERCIZIO: scrivere una funzione $stampa(r)$ che stampa le chiavi dei nodi dell'albero r. Stampa ricorsivamente in base a questa regole:
Prima la chiave del nodo e poi indentate le chiavi del sottoalbero di sinistra e quelle del sottoalbero di destra.

```
def stampa(p, h=0):
    if p==None:
        print('| '*h, '-')
    else:
        print('| '*h, str(p.key))
        stampa(p.left, h+1)
        stampa(p.right, h+1)
```



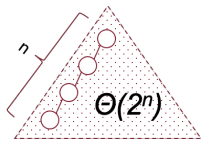

```
>>> stampa(r)
5
| -
| 1
| | 8
| | | 2
| | | | -
| | | | -
| | | -
| | 3
| | | -
| | | -
```

Rappresentazione posizionale:

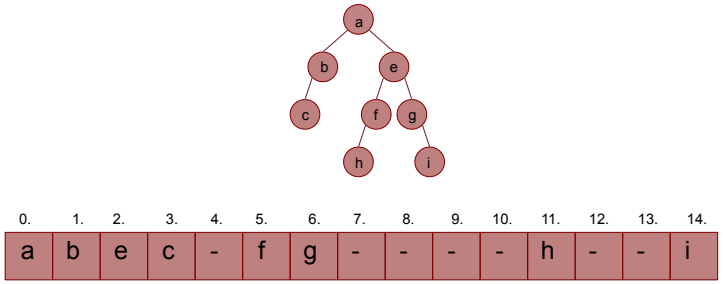
L'abbiamo già discussa in merito allo heap.
I nodi vengono memorizzati in un vettore, nel quale la radice occupa la posizione di indice 0 ed i figli sinistro e destro del nodo in posizione i si trovano rispettivamente nelle posizioni 2i +1 e 2i+2.

Svantaggi rispetto alla gestione mediante puntatori:
Se h è l'altezza dell'albero richiede l'allocazione di un vettore in grado di contenere un albero binario completo di altezza h (quindi un vettore che contenente $2^{h+1} - 1$ elementi) e se l'albero non è abbastanza "denso" chi verifica un notevole spreco di spazio.

Albero «degenere»:



Rappresentazione posizionale (segue):

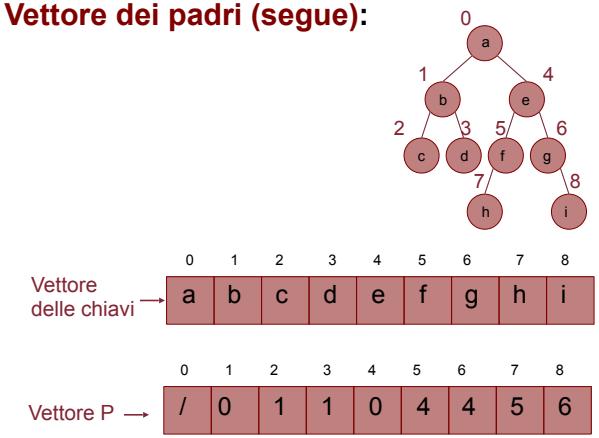


Vettore dei padri:

Costituito da un vettore ***P*** in cui ogni elemento è associato ad un nodo dell'albero.
Introducendo una biezione tra gli n nodi dell'albero e gli indici $0,\ldots,n-1$, l'elemento $P[i]$ del vettore P contiene l'indice del padre del nodo i nell'albero.

Questo metodo di memorizzazione funziona senza alcuna modifica anche per alberi non necessariamente binari, in cui cioè ogni nodo può avere un numero qualunque di figli.

Vettore dei padri (segue):



Confronto fra le strutture dati: come trovare il padre di un nodo

Struttura a puntatori:

- al momento non siamo in grado di farlo: dobbiamo accedere all'albero tramite il puntatore alla sua radice, ma poi non possiamo scorrere la struttura come fosse una lista, perché non sappiamo se dirigerci a destra o a sinistra dovremo quindi imparare a navigare tra i nodi dell'albero (visita di alberi) .

Rappresentazione posizionale:

- il padre del nodo *i* è banalmente in posizione $\left\lfloor \frac{i-1}{2} \right\rfloor$.

Vettore dei padri:

- L'indice del padre di ogni nodo i è memorizzato direttamente nell'elemento P[i] del vettore.

Confronto fra le strutture dati: **determinare se il nodo abbia 0, 1 o 2 figli**

Struttura a puntatori:

- si verifica se i campi left e right siano settati a $None$ oppure no.

Rappresentazione posizionale:

- si vede se gli elementi di indice $2i+1$ e $2i+2$ esistono e sono settati a '-' oppure no.

Vettore dei padri:

- Si deve scorrere l'intero vettore P e contarvi il numero di occorrenze dell'elemento i (l'indice del nodo di cui vogliamo contare i figli).

Confronto fra le strutture dati: **determinare la distanza di un nodo dalla radice**

Struttura a puntatori:

- come nel caso della ricerca del padre di un nodo per farlo dovremo prima vedere come navigare tra i nodi dell'albero.

Rappresentazione posizionale:

- il livello del nodo i (e quindi la sua distanza dalla radice) è banalmente $\lfloor \log_2 i \rfloor$.

Vettore dei padri:

- vettore dei padri: a partire da i risaliamo di padre in padre passando per $P[i]$, $P[P[i]]$, $P[P[P[i]]]$, ecc. fino a giungere alla radice; ciò richiede tempo proporzionale ad h.

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi

- Dire quant'è la massima lunghezza di un vettore che è necessario allocare per poter memorizzare sempre un albero di n nodi con la rappresentazione posizionale.
- Progettare un algoritmo che, dato un albero binario memorizzato tramite vettore dei padri, restituisca il vettore relativo alla rappresentazione posizionale dello stesso albero. Calcolare il costo computazionale dell'algoritmo.
- Progettare un algoritmo che, dato un albero binario memorizzato tramite rappresentazione posizionale, restituisca il vettore dei padri dello stesso albero. Calcolare il costo computazionale dell'algoritmo.