

## BFS e DFS A CONFRONTO

Nella BFS i nodi che separano quelli visitati da quelli non visitati sono detti di **FRONTIERA** e sono una costante della BFS indipendentemente dalla struttura, salvati in una queue.

Nella DFS la allocazione di memoria è implicita quando viene effettuata la ricorsione poiché vengono salvate ogni volta le variabili locali nello **STACK** di chiamate.

DFS prende in input solo la radice corrente; per come è fatto l'algoritmo ogni chiamata a DFS arriverà sul figlio della radice corrente fino ad arrivare alla foglia, ciò significa che lo stack contiene una sequenza di padri-figli ovvero un percorso.

Il DFS allocherà memoria massima quando effettuerà il percorso dalla radice alla foglia al livello  $h$  (altezza).

Possiamo immaginare che questa dimensione sia  $M(m) = O(h)$  (limitato superiormente da  $h$ ). Il caso peggiore si presenta su un albero binario degenero che ha  $h=m$  quindi  $M(m) = O(m)$ .

(con  $m$  numero di nodi dell'albero)

Per la BFS la frontiera dipende dal livello che ha più nodi.

Il caso peggiore ci si presenta su alberi pieni al livello  $h$  dove il numero di foglie è  $\frac{m+1}{2}$  quindi  $M(m) = \Theta(m)$ .

Vediamo adesso i casi migliori:

Per il DFS c'è su un albero di altezza minima circa  $\lg_2 m$   $m(m) = \Omega(m \lg_2 m)$ .

Per il BFS il caso migliore è un albero degenero dove la frontiera contiene al più un nodo quindi

$$m(m) = \Theta(1)$$

### ALBERO BINARIO DI RICERCA (ABR)

È un albero binario **ORDINATO** definito nel seguente modo:

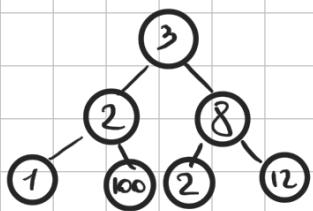
- $T$  è un albero vuoto
- $\forall x \in T \quad (\forall y \in X_{\leq x} \quad \text{val}(y) \leq \text{val}(x)) \wedge \forall y \in X_{> x} \quad \text{val}(x) \leq \text{val}(y))$

Quindi presi due nodi possiamo sempre individuare la relazione d'ordine.

Quest' albero ci permette inoltre di stabilire una relazione d'ordine tra un nodo e un suo sottosalbero (a differenza dello heap dove abbiamo una relazione d'ordine solo tra padre e figlio)

### OSS

Se imponiamo una proprietà meno forte non è detto che venga verificata. Ad esempio vediamo  $\forall x \in T (f_{sx}(x) \leq x \leq f_{dx}(x))$  dove  $f(x)$  definisce il figlio di  $x$ .



Questo albero **NON** è un ABR

### RICERCA SU ABR

Si noti che in un ABR ogni sottosalbero è un ABR

RICERCA ABR ( $T, k$ )

IF  $T \neq \text{NIL}$  THEN

  IF  $T \rightarrow \text{KEY} < k$  THEN

    RETURN RICERCA ABR ( $T \rightarrow dx, k$ )

  ELSE

IF  $T \rightarrow \text{KEY} > K$  THEN

RETURN RICERCA ABR ( $T \rightarrow s_x, K$ )

RETURN T

l' albero di ricorsa (nel caso peggiore) è un albero degenero perché l' algoritmo effettua solo una chiamata ricorsiva ad ogni ripetizione.

La complessità di quest' algoritmo dipende dall' altezza e non dal numero di nodi.

## INSEGNAMENTO ABR

L' inserimento deve mantenere la proprietà degli ABR.

Cerchiamo il nostro elemento K (quindi cerchiamo la sua posizione prevista), se c' è non lo inseriamo altrimenti abbiamo già la posizione dove inserire il nostro (facendo attenzione se inserirlo a destra o sinistra)

INSERT ( $T, k$ )

IF  $T \neq \text{NIL}$  THEN

IF  $T \rightarrow \text{KEY} < k$

$T \rightarrow d_x = \text{INSERT}(T \rightarrow d_x, k)$

insert ritorna la radice  
del sottobbero destro

ELSE IF  $T \rightarrow \text{KEY} > k$

$T \rightarrow s_x = \text{INSERT}(T \rightarrow s_x, k)$

ELSE

$x = \text{AllocaNodo}()$

$x \rightarrow \text{KEY} = k$

$x \rightarrow s_x = \text{NIL}$

$x \rightarrow d_x = \text{NIL}$

$T = x$

RETURN  $T$