

Binary Search Trees

Giorgio Audrito

Online, 4 aprile 2023

Introduzione

Gli insiemi: `std::set<T>`

Che cosa possono fare?

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi
- accedere al minimo e massimo elemento

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi
- accedere al minimo e massimo elemento

E se invece volessimo...

- trovare il k -esimo elemento?

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi
- accedere al minimo e massimo elemento

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)?

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi
- accedere al minimo e massimo elemento

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo?

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi
- accedere al minimo e massimo elemento

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi?

Gli insiemi: `std::set<T>`

Che cosa possono fare?

- aggiungere e rimuovere elementi
- controllare la presenza di un elemento qualunque
- trovare gli elementi nell'insieme più vicini ad un dato elemento
- sapere il numero totale di elementi
- accedere al minimo e massimo elemento

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Tutto troppo lento (se $k, ans \in \Omega(N)$).

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(V)$

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(V)$
- troppo se $V \gg N$

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(V)$
- troppo se $V \gg N$
- comunque non ci aiutano per unioni/intersezioni

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree sparso

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree sparso

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(N \log V)$

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree sparso

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(N \log V)$
- bene ma non benissimo: se $N \approx 10^6$ e $V \approx 10^{18}$, sono
 - $3\times$ più lento di $\mathcal{O}(\log N)$
 - $20\times$ più spazioso di $\mathcal{O}(N)$

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

Segment tree sparso

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(N \log V)$
- bene ma non benissimo: se $N \approx 10^6$ e $V \approx 10^{18}$, sono
 - $3\times$ più lento di $\mathcal{O}(\log N)$
 - $20\times$ più spazioso di $\mathcal{O}(N)$
- comunque non ci aiutano per unioni/intersezioni

Gli insiemi: `std::set<T>`

E se invece volessimo...

- trovare il k -esimo elemento? $\mathcal{O}(k)$
- sapere quanti elementi sono in un certo intervallo (o somme, ecc...)? $\mathcal{O}(ans)$
- somme (o altre operazioni associative) su elementi in un intervallo? $\mathcal{O}(ans)$
- fare unioni e intersezioni tra insiemi? $\mathcal{O}(N)$

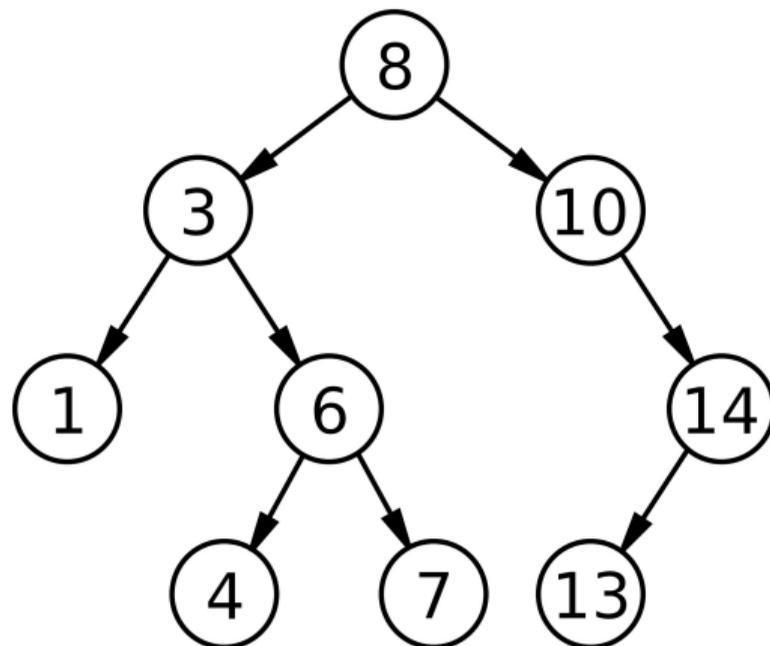
Segment tree sparso

- operazioni associative in $\mathcal{O}(\log V)$, memoria $\mathcal{O}(N \log V)$
- bene ma non benissimo: se $N \approx 10^6$ e $V \approx 10^{18}$, sono
 - $3\times$ più lento di $\mathcal{O}(\log N)$
 - $20\times$ più spazioso di $\mathcal{O}(N)$
- comunque non ci aiutano per unioni/intersezioni

ci tocca capire come sono fatti i set per poterli personalizzare...

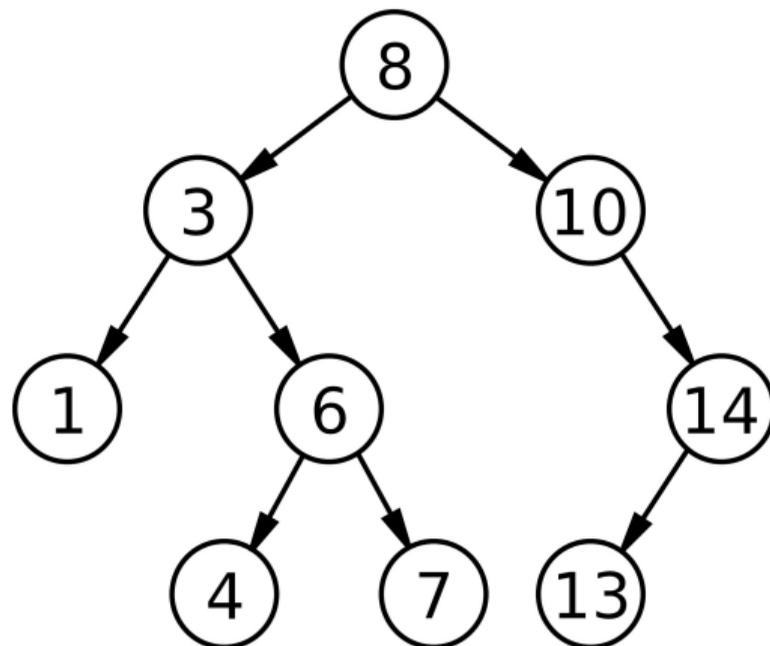
Binary Search Trees

- elementi in un albero binario
- a sinistra gli elementi più piccoli
- a destra gli elementi più grandi
- ... so sempre come navigare
- ... posso mantenere dati associativi



Binary Search Trees

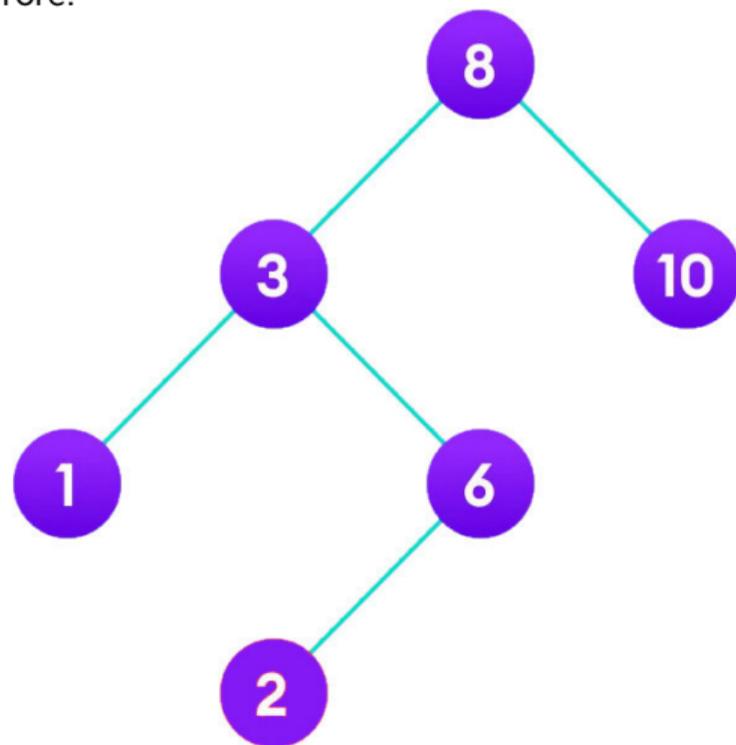
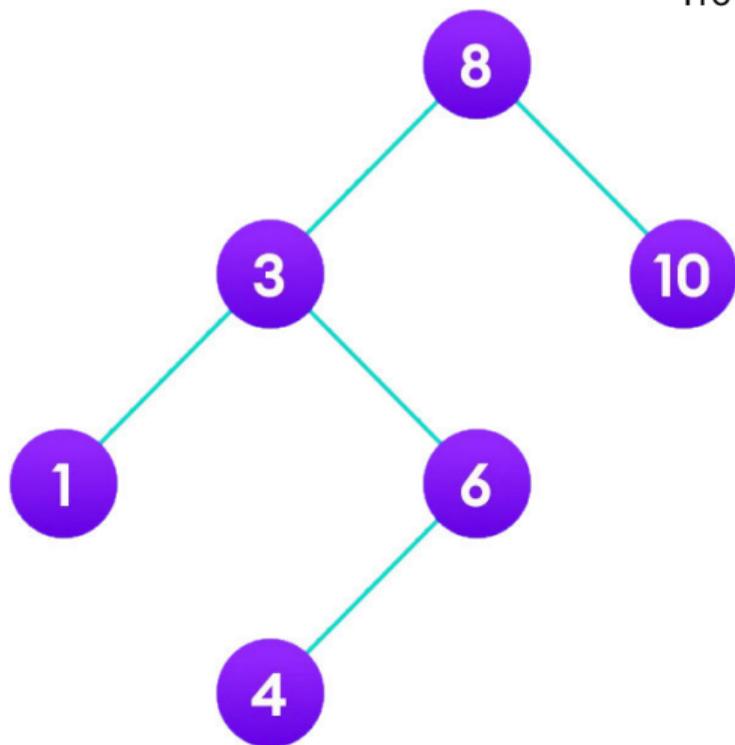
- elementi in un albero binario
- a sinistra gli elementi più piccoli
- a destra gli elementi più grandi
- ... so sempre come navigare
- ... posso mantenere dati associativi



ci possono anche essere alberi non binari ma non ci interessano ora

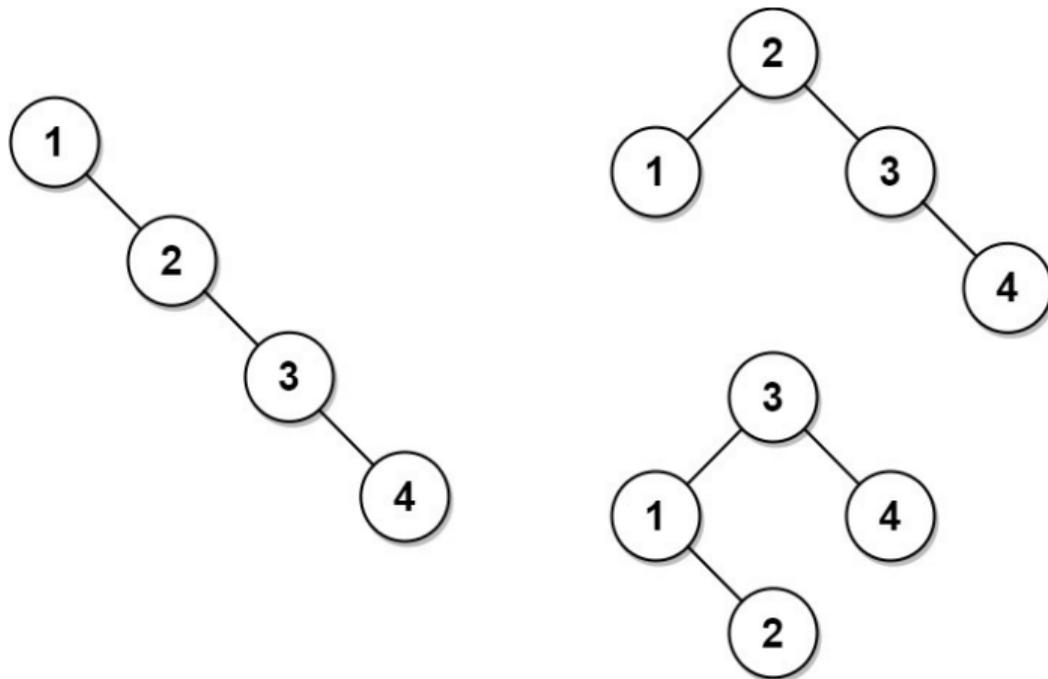
Binary Search Trees

Trova l'errore:



Binary Search Trees

Ci possono essere molti BST con gli stessi elementi:



Quale preferiremmo?

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi
- richiede di modificare l'albero (tree rotations)

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi
- richiede di modificare l'albero (tree rotations)
- moltissimi modi per farlo, noi vediamo i **treap**

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi
- richiede di modificare l'albero (tree rotations)
- moltissimi modi per farlo, noi vediamo i **treap**

Treap = Tree + Heap

- struttura dati **randomizzata**, l'albero è bilanciato in media

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi
- richiede di modificare l'albero (tree rotations)
- moltissimi modi per farlo, noi vediamo i **treap**

Treap = Tree + Heap

- struttura dati **randomizzata**, l'albero è bilanciato in media
- genero priorità random per ogni elemento da inserire

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi
- richiede di modificare l'albero (tree rotations)
- moltissimi modi per farlo, noi vediamo i **treap**

Treap = Tree + Heap

- struttura dati **randomizzata**, l'albero è bilanciato in media
- genero priorità random per ogni elemento da inserire
- forzo un ordinamento a **heap**: padre maggiore (o minore) dei figli

Balanced Binary Search Trees

- mantenere bilanciamento mentre si aggiungono elementi
- richiede di modificare l'albero (tree rotations)
- moltissimi modi per farlo, noi vediamo i **treap**

Treap = Tree + Heap

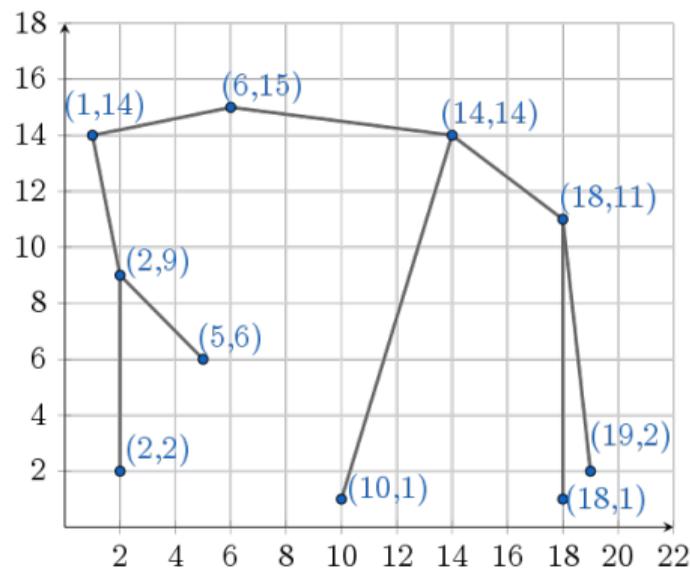
- struttura dati **randomizzata**, l'albero è bilanciato in media
- genero priorità random per ogni elemento da inserire
- forzo un ordinamento a **heap**: padre maggiore (o minore) dei figli
- servono anche per avere array con operazioni su range rapide (implicit treap)

Treaps

Treaps

Treap = Tree + Heap

- struttura dati **randomizzata**, l'albero è bilanciato in media
- genero priorità random per ogni elemento da inserire
- forzo un ordinamento a **heap**: padre maggiore (o minore) dei figli
- servono anche per avere array con operazioni su range rapide (implicit treap)



detto anche **cartesian tree** perché si disegna bene

Treaps

Ma esiste almeno un Treap dati elementi e priorità?

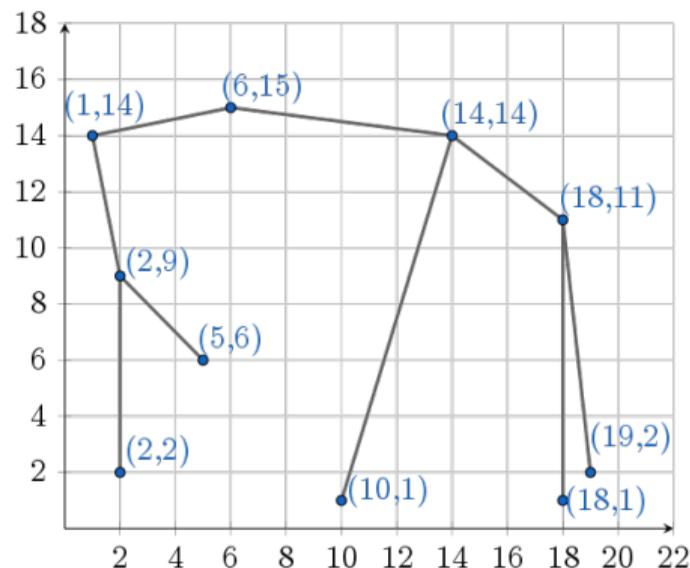
Sì: ordina gli elementi per priorità e poi inseriscili naturalmente nell'albero senza ribilanciare

Possono esistere Treap diversi per gli stessi dati?

No! Si può dimostrare ricorsivamente

Come mantenere la proprietà Heap?

Tramite **split** e **merge**...



Split!

`split(T, k)`

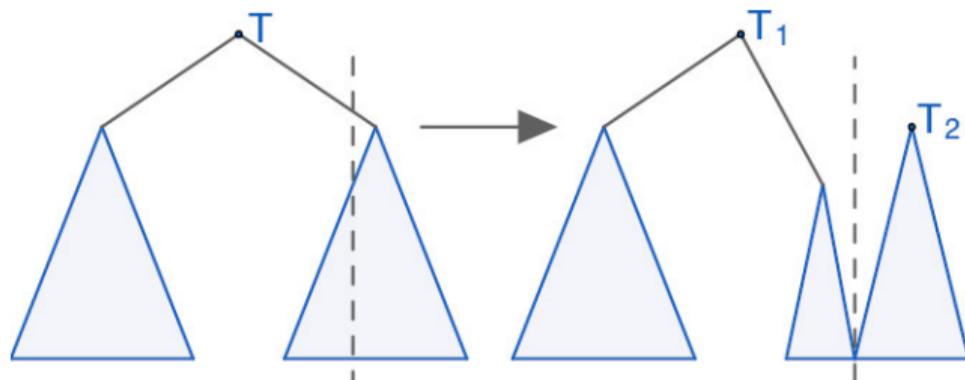
- restituisce due treap L e R
- L con gli elementi $\leq k$
- R con gli elementi $> k$

se $T.\text{root.val} \leq k$:

- $RL, RR = \text{split}(T.R, k)$
- $L = \text{treap}(T.\text{root}, T.L, T.RL)$
- return L, RR

se $T.\text{root.val} > k$:

- $LL, LR = \text{split}(T.L, k)$
- $R = \text{treap}(T.\text{root}, T.LR, T.R)$
- return LL, R



Merge!

`merge(L, R)`

- assume che $L < R$
- unisce i due treap

se $L.\text{root.pri} \geq R.\text{root.pri}$:

- $R = \text{merge}(L.R, R)$
- $T = \text{treap}(L.\text{root}, L.L, R)$
- return T

se $L.\text{root.pri} < R.\text{root.pri}$:

- operazione simmetrica

