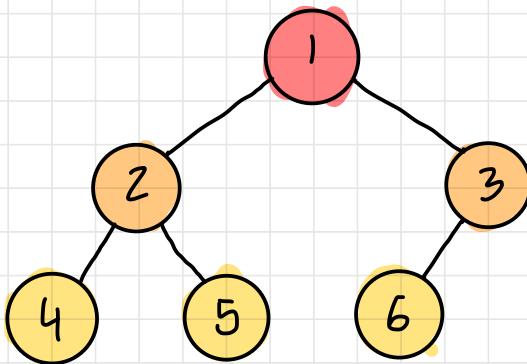


Kopce Binarne ,

Kopiec Binarny



↑
mot used - indeksowanie od 1 upraszcza nam obliczenia

- Dzieci są mniejsze / większe od rodzica
- Kopce binarne służą do implementacji kolejki priorytetowej
- rodzic $\rightarrow \lfloor \frac{i}{2} \rfloor$
- lewe dziecko $\rightarrow 2i$:
- prawe dziecko $\rightarrow 2i + 1$
- Heapify przywraca porządek kopcowy w złożoności czasowej $O(n \log n)$ $\rightarrow \Theta(n)$ dla wszystkich wierzchołków
 - ↑ dowód w zadaniach z egzaminu

Time complexity

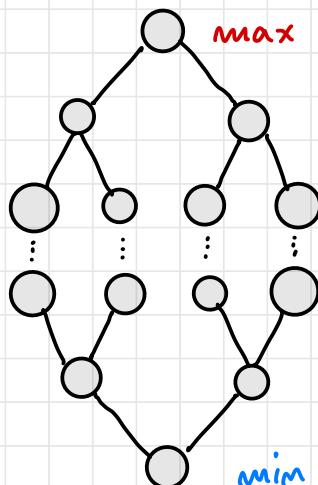
Operation	Description	Complexity
Build heap	Builds a heap from a binary tree	$O(n)$
Decrease key	Decreases an existing key to some value	$O(\log n)^*$
Delete	Deletes a node given a reference to it	$O(\log n)^*$
Extract minimum	Removes and returns the minimum value	$O(\log n)$
Find minimum	Returns the minimum value	$\Theta(1)$
Insert	Inserts a new value	$O(\log n)$
Union	Combine the heap with another to form a valid binary heap	$O(n), O(k \log n)^{**}$

Min - Max

Istnieją dwie wersje kopca min-maxowego.

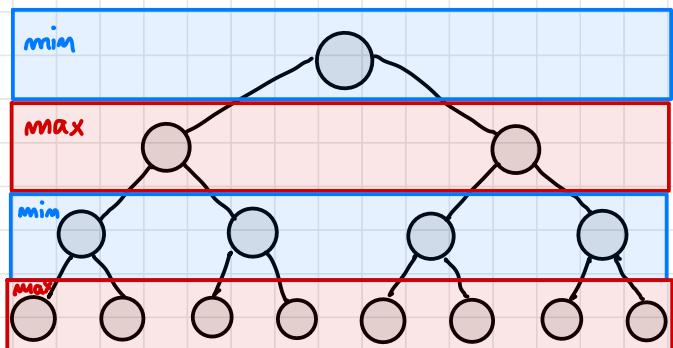
1. Wersja ze sklejonymi kopcami

„Sklejamy” liście tych kopców i modyfikujemy funkcje wyciągające potomków danego węzła.



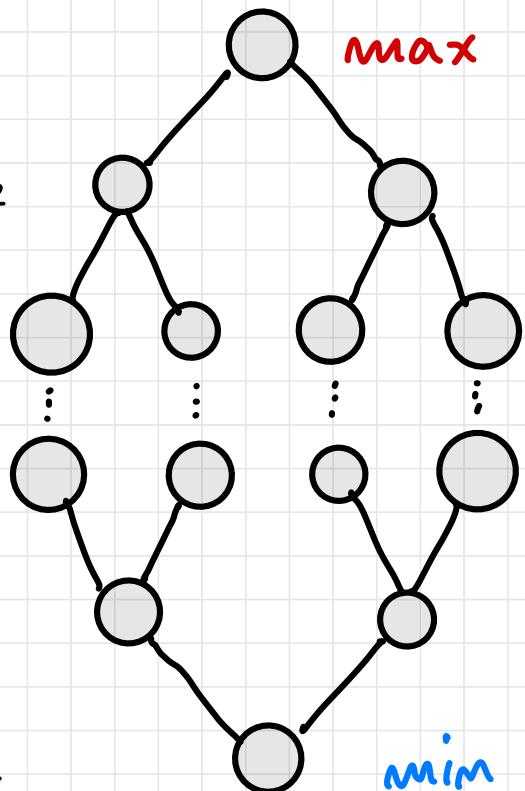
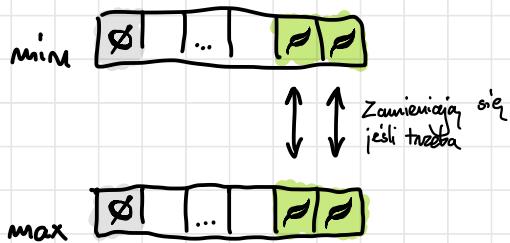
2. Wersja z poziomaniem min oraz max kopca.

Ta wersja operuje w rezultacie na abstrakcyjnych kopcach 4-argument.



Wersja I

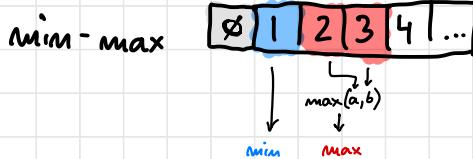
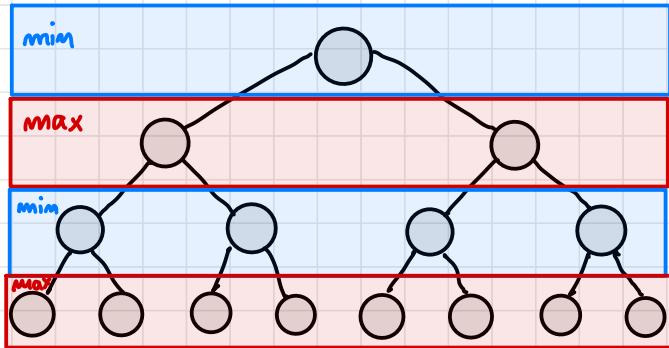
Będącym przechowywać 2 kopce ze sobą sklepane w most. sposób:



Teraz gdy operacje wykonujemy na kopcu min, to zawsze najmniej patrząc na lisc' oraz odpowiadającym mu lisc' z przeciwnego kopca i zwracamy mniejszy robiąc swap. Jeśli swap zostanie wykonany, wykonujemy heapify na przeciwnym kopcu.

Versja II

Wszystko przechowujemy w jednym kopcu.



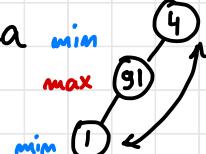
Każdy wierzchołek ma parzysty lub nieparzysty poziomie jest największy lub najmniejszy od jego potomków.

Heapify składa się teraz z 2 faz:

1. Sprawdź czy wierzchołek jest na swoim poziomie. Jeśli nie, zamień go z rodzicem



2. Wykonaj standardowy heapify dla kopca 4-stnego na poziomie mim lub max.



Zadania z Egzaminu

1. Napisz szybką procedurę budowy kopca:

func buduj_kopiec_max(K)

for ($i = \frac{m}{2}; i \geq 1; i++$)

 przesun_nizzej(K, i)

func przesun_nizzej(K, i)

$k = i$

(kopiec max)

do

$j = k$

guard

if

$2j \leq m$

and

$K[2j] > K[k]$

|

$k = 2j$

if $2j < m$ and $K[2j+1] > K[k]$

$k = 2j + 1$

swap($K[j], K[k]$)

while $k \neq j$

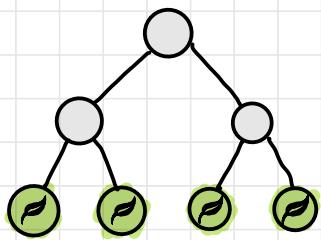
2. W jakim czasie działa procedura budowania kopca?

Mozemy zauwazyc, ze ma kazdyem kopcu

lisie \rightarrow nie beda przesunięte

mniej, poniewaz sa na samym dole.
Liczba lisow w kopcu wynosi $\pm \frac{n}{2}$.

Rodzice tych lisow moza byc przesunieti tylko o jeden element mniej itd.
Jest $\frac{m}{4}$ wezlow rodzicow.



Zauwazamy zatem pewna prawidlowosc:

$$0 \cdot \frac{m}{2} + 1 \cdot \frac{m}{4} + 2 \cdot \frac{m}{8} + \dots + k \cdot \frac{m}{2^{k+1}}$$

$\uparrow \quad \uparrow \quad \uparrow$
lisie rodzice dziedzicze

Tatwo zaobserwować, że otrzymaliśmy szereg harmoniczny,
zatem złożoność czasowa powinna być liniowa.

$O(n)$

3. Opisz algorytm tworzenia kopca, którego złożoność określa poniższe równanie. Czy to najlepszy algorytm?

$$T(n) = \sum_{i=1}^n \log i$$

Jest to algorytm, który n razy wykonuje operację `insert` ($O(\log n)$), która polega na wstawieniu elementu do kopca i przesunięciu go w góre.

Lepszy algorytm, to taki, gdzie wypełniamy tablicę kopca elementami, a następnie przesuwamy te elementy w dół. Wtedy złożoność wynosi $O(n)$. - patrz poprzednie zadanie

4. Jaka jest złożoność poniższej funkcji tworzenia kopca w tablicy K?

func buduj_kopiec (K[1..n])

for i=2 to n

przesuń wyżej(K, i)

$$\sum_{i=1}^n \log i = \log(n!) \leq \log n^n = n \log n$$

A zatem $O(n \log n)$

5. W której wersji deletemu ma kopię spodziewamy się wykonać mniej porównań i dlaczego?

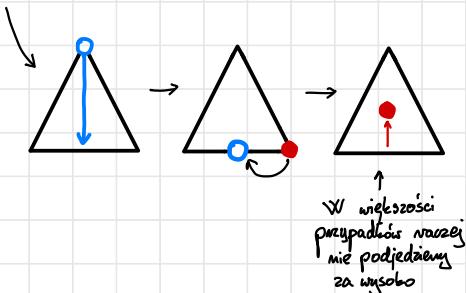
- a) Usuwany element zastępujemy skrąnieniem prawaym liściem
- b) Przepychanie dzieci w dół

a) Tutaj wykonamy $2 \log n$ porównań. Na każdym etapie musimy porównać dzieci, a następnie z mniejszym z nich porównać wierzchołek.

b) Usunięty element staje się dzieciem i przenosimy ją w dół wykonując porównanie tylko między dziećmi, a następnie wstawiamy element z końca tablicy do tej dzieci i podciągamy do góry.

Wykonamy max $2\log n - 1$ porównań (średnio $\log n$)

Zwycięzca: wersja b



6. Udowodnij, że przynajmniej jedna operacja `min`, `deletemin` lub `insert` wykonywanych na kolejce priorytetowej wymaga w najgorszym wypadku czasu $\Omega(\log n)$.

`min`:

$O(1)$, bo ta informacja znajduje się na szczytzie

kopca:
tutaj
↓



`deletemin`:

Wymaga $O(\log n)$ operacji

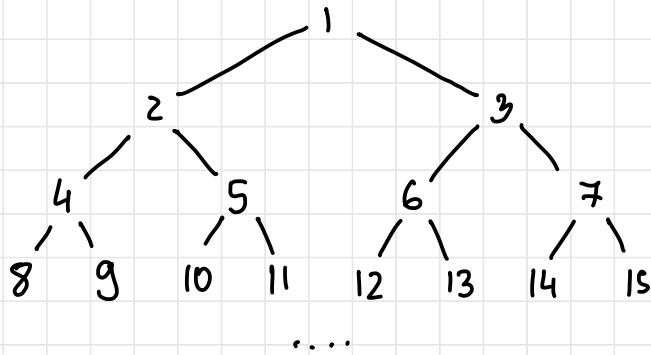
`insert`:

Wymaga $O(\log n)$ operacji

Nie jesteśmy w stanie zrobić tego szubiek. Gdyby to było możliwe, to moglibyśmy przedstawić kopiec szubiek z $O(m \log m)$

7. W kopcu umieszczone m kłuczy 1, 2, 3, ..., m
W korzeniu znajduje się 1.

Na jakiej maksymalnie wysokości znajduje się klucz k?



Znajduje się na poziomie $\log(k)$