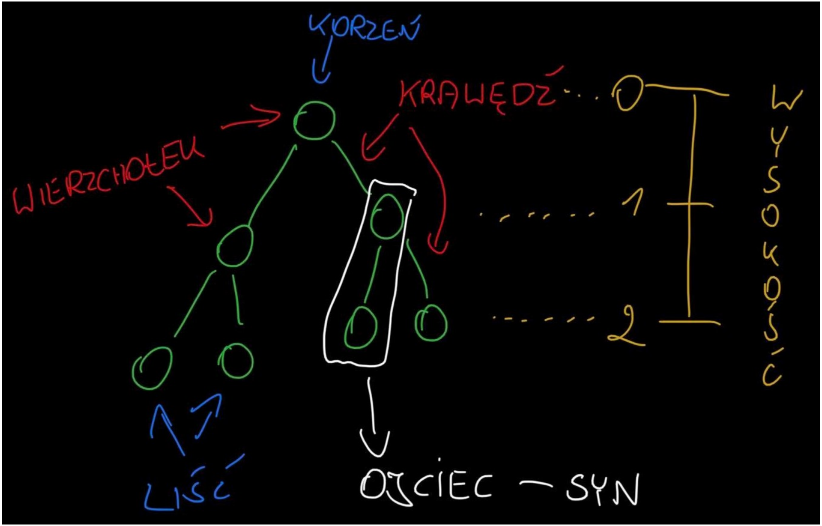
Wprowadzenie do Algorytmiki. Kopce. Szybkie potęgowanie macierzy.

Artur Laskowski

17 stycznia 2022, Poznań

Drzewo



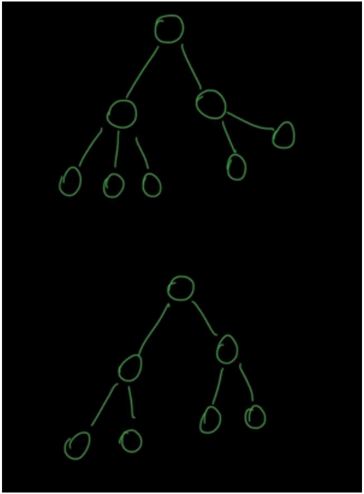
Drzewo

Drzewo

Drzewo binarne

Ojciec ma maksymalnie 2ch synów

Drzewo jest zbalansowane, jeżeli różnica wysokości dowolnej pary liści nie przekracza 1.



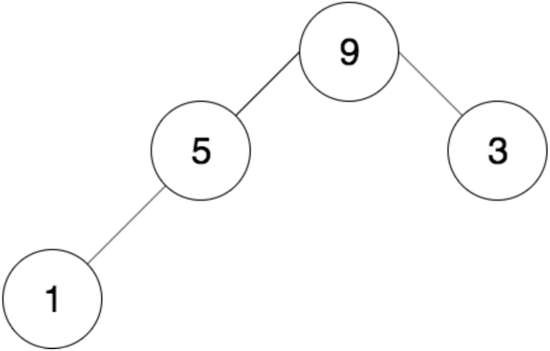
Kopiec

Kopiec jest również nazywany *stogiem*, albo *kolejką priorytetową*.

Wykorzystuje reprezentacę za pomocą drzewa binarnego.

Kopiec

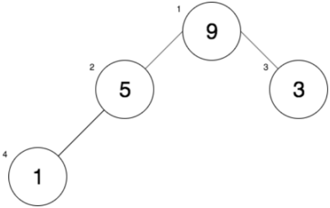
Wartość w wierzchołku ojca jest zawsze większa od wartości w wierzchołkach synów.



Indeksowanie

Indeksowanie:
o - indeks ojca
$s1, s2$ - indeksy synów

$$o = \lfloor s1/2 \rfloor$$
$$o = \lfloor s2/2 \rfloor$$
$$s1 = 2 \cdot o$$
$$s2 = 2 \cdot o + 1$$



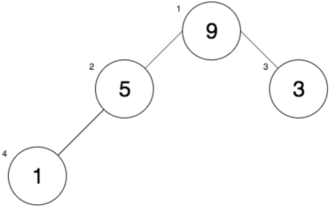
Reprezentacja

Potrafimy wyliczyć indeksy ojca i syna

Numeracja jest ciągła

Możemy reprezentować kopiec za pomocą tablicy

```
int array[] = {0, 9, 5, 3, 1};
```

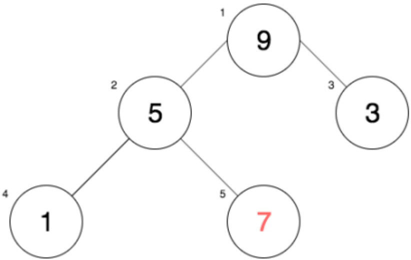


Dodawnia elementu

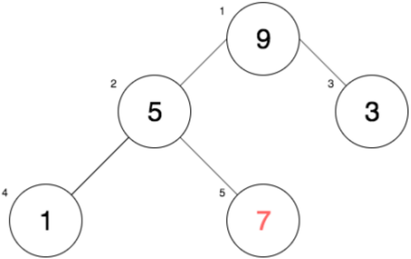
Dodajemy element na końcu kopca

Przywracamy strukturę kopca

```
int array[] = {0, 9, 5, 3, 1};
int array_size = 5;

void insert(int e) {
    ++array_size;
    array[array_size] = e;
    push_up(array_size);
}

void push_up(int idx) {
    int son = idx;
    int father = idx / 2;
    while (father > 0 && array[father] < array[son]) {
        swap(array[father], array[son]);
        son = father;
        father = son / 2;
    }
}
```



Dodawanie elementu



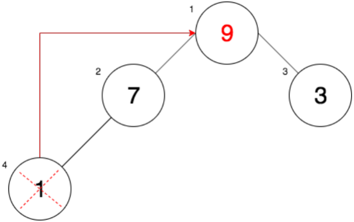
Usuwanie elementu

Usuwamy korzeń

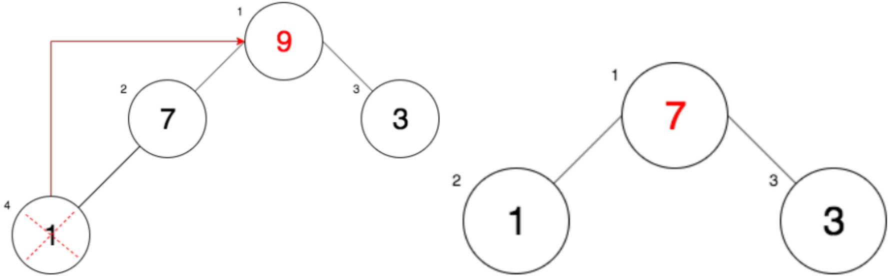
Zastępujemy korzeń elementem ostatnim

Przywracamy strukturę kopca

```
int array[] = {0, 9, 5, 3, 1};
int array_size = 4;

int delmin() {
    int res = array[1];
    array[1] = array[array_size];
    --array_size;
    push_down(1);
    return res;
}

void push_down(int idx) {
    int father = idx;
    int son = 2 * idx;

    while(son <= array_size) {
        if(son < array_size && array[son+1] > array[son]) {
            ++son;
        }
        if(array[son] > array[father]) {
            swap(array[son], array[father]);
            father = son;
            son = father * 2;
        } else {
            break;
        }
    }
}
```



Usuwanie elementu



Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Wstawianie elementu: **?**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

- Wstawianie elementu: **O(log n)**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Usuwanie elementu: **?**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Usuwanie elementu: **O(log n)**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Pobieranie największego: **?**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Pobieranie największego: **O(log n)**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Tworzenie kopca o n elementach: **?**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

Tworzenie kopca o n elementach: **O(n log n)**

Złożoność obliczeniowa

Wysokość kopca jest logarytmiczna.

- Wstawianie elementu: **O(log n)**
- Usuwanie elementu: **O(log n)**
- Pobieranie największego: **O(log n)**
- Tworzenie kopca o *n* elementach: **O(n log n)**

Inne kopce

Kopiec Fibonacciego:

- Usuwanie minimum: **O(log n)**
- Reszta operacji: **O(1)**
- Duża stała
- Makabryczna implementacja

Kopce

Kopce można łatwo wykorzystać do sortowania liczb:

- Utwórz kopiec
- Wykonaj n razy operację usunięcia minimum
- Czas działania **O(nlogn)**
- Stabilnie, w miejscu

Zadanie

Należy wykonać naprawy fragmentów dróg:

- Każdy fragment remontowany jest dokładnie 1 dzień
- Siła zniszczenia fragmentu to p_i
- Remont danego odcinka można rozpocząć dopiero w dniu d_i
- Najgorsze odcinki mają być naprawione najszybciej
- Jaka będzie kolejność remontowania?

Szybkie potęgowanie

Naiwnie a^n jest obliczane jako:

$$a^n = a \cdot a \cdot \ldots \cdot a$$

Takie podejście jest niepraktyczne dla dużych a, lub n.

Szybkie potęgowanie

Jednak:

$$a^{(b+c)} = a^b \cdot a^c$$

$$a^{2b} = a^b \cdot a^b = (a^b)^2$$

$$a^{X_{10}} = a^{Y_2}$$

Szybkie potęgowanie - algorytm

$$a^{X_{10}} = a^{13_{10}} = a^{1101_2} = a^{2^3} \cdot a^{2^2} \cdot a^{2^0} = a^8 \cdot a^4 \cdot a^1$$

$$n = \log_2 X \quad \begin{cases} a^1 & = a^0 \cdot a \\ a^2 & = a^1 \cdot a \\ a^3 & = a^2 \cdot a \\ & \vdots \\ a^{n-1} & = a^{n-2} \cdot a \\ a^n & = a^{n-1} \cdot a \end{cases}$$

Liczby Fibonacciego szybkim potęgowaniem

$$\begin{vmatrix} F_{N+1} \\ F_N \end{vmatrix} = M \times \begin{vmatrix} F_N \\ F_{N-1} \end{vmatrix}$$

$$\begin{vmatrix} A & B \\ C & D \end{vmatrix} \times \begin{vmatrix} F_N \\ F_{N-1} \end{vmatrix} = \begin{vmatrix} A \cdot F_N + B \cdot F_{N-1} \\ C \cdot F_N + D \cdot F_{N-1} \end{vmatrix}$$

$$\begin{vmatrix} F_{N+1} \\ F_N \end{vmatrix} = \begin{vmatrix} A \cdot F_N + B \cdot F_{N-1} \\ C \cdot F_N + D \cdot F_{N-1} \end{vmatrix} = \begin{vmatrix} 1 \cdot F_N + 1 \cdot F_{N-1} \\ 1 \cdot F_N + 0 \cdot F_{N-1} \end{vmatrix}$$

$$\begin{vmatrix} A & B \\ C & D \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix}$$

Liczby Fibonacciego szybkim potęgowaniem

$$\begin{vmatrix} F_{N+1} \\ F_N \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix} \times \begin{vmatrix} F_N \\ F_{N-1} \end{vmatrix}$$

$$\begin{vmatrix} F_{N+1} \\ F_N \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix}^N \times \begin{vmatrix} F_1 \\ F_0 \end{vmatrix}$$

Laboratoria

https://www.hackerrank.com/wda-09-2021