Algorytmy i struktury danych
Lista 7

Zadanie 2.

Pokaż w jaki sposób zbudować kopiec z losowej tablicy długości n w czasie liniowym od wielkości danych. Jak to wygląda dla kopca d-arnego.

a) Budowa kopca minimalnego z tablicy A o długości n (indeksowanie od 0):

```
function BuildHeap(A):
     for i in range floor(n/2) to 1 // i-- :
          Heapify(A,i)

function Heapify(A, i):
    left = Left(i)
    right = Rigth(i)
    if left < n and right < n then
        lowest = i
        if A[left] < A[lowest] then
            lowest = left
        if A[right] < A[lowest] then
            lowest = right
        if lowest != i then
           swap(A[lowest], A[i])
           Heapify(A, lowest)

function Left(i):
    return 2 * i + 1

function Right(i):
    return 2 * i + 2
```

Kluczową funkcją jest Heapify, która przywraca własność kopca dla poszczególnych węzłów o indeksach i w tablicy A, a działa następująco:

(a) Wybiera dzieci zgodnie z własnością kopca, lewe posiada indeks $2 * i + 1$ ($2 * i$ dla indeksowania od 1), a prawe $2 * i + 2$ ($2 * i + 1$ dla indeksowania od 1)

(b) Po sprawdzeniu czy indeksy dzieci nie wykraczają poza tablicę (jeżeli tak, to węzeł o indeksie i jest liściem) szukane jest minimum wartości dla indesów, Left(i), Right(i) oraz i. Zabieg ten ma na celu przywrócenie własności minimalnego kopca binarnego.

(c) W przypadku gdy minimum nie znajduje się w i, następuje zamiana elementów w A (ojciec musi mieć mniejszą wartość niż dziecko)

W celu obliczenia złożoności całego algorytmu, należy skorzystać z faktu, że n-elementowy kopiec binarny posiada co najwyżej $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ węzłów o wysokości h. Złożoność obliczeniowa samej funkcji Heapify to $k * O(1)$ gdzie k to wysokość kopca, a O(1) to kosz znalezienia minimum. Zbudowanie kopca będzie miało zatem kosz:

$$\sum_{\substack{\cancel{n=0} \\ h = 0}}^{\ln n} \left\lceil \frac{n}{2^{h+1}} \right\rceil * O(h) = O(n)$$

Więc nasz algorytm jest poprawny.

b) Kopiec d-arny

Kopiec d-arny to kopiec, dla którego każdy węzeł posiada d dzieci. Pseudokod również w wersji indeksowania od 0.

```
function BuildHeap(A, d):
     for i in range floor(n/2) to 1 // i-- :
          Heapify(A,i, d)

function Heapify(A, i, d):
    lowest = i
    for j in range 0 to d:
        c = Child(i, j)
        if c < n and A[c] < A[lowest] then
            lowest = c
    if lowest != i then
       swap(A[lowest], A[i])
       Heapify(A, lowest)

function Child(i, j, d):
    return d*i + (j + 1)
```

Powyższy algorytm posiada funkcję Child, która umożliwia przypisanie węzłowi więcej niż dwoje dzieci (j to kolejne dzieci). Dodatkowo w funkcji Heapify pojawiła się pętla for wykonująca się d razy (wcześniej niejawnie występowała pętla wykonująca się 2 razy - dziecko prawe oraz dziecko lewe).
Korzystając ponownie z własności o liczbie węzłów w stosunku do wysokości, tym razem zakładając, że liczba dzieci to d zamiast 2, otrzymamy złożoność:

Tutaj założone jest, że 'd' jest stałe. W innym wypadku powinno się tam pojawić O(d*h), a wynik byłby funkcją od 'n' i 'd'

$$\sum_{\substack{\cancel{n=0} \\ h=0}}^{\ln n} \left\lceil \frac{n}{d^{h+1}} \right\rceil * O(h) = n * \sum_{h\,\cancel{n}=0}^{\ln n} \frac{h}{d^h} = O(n)$$

Zbudowanie kopca również będzie kosztować O(n).

W powyższej sumie brakuje dużego O przed wszystkim.