

## Řešení

Ze zadání vím, že musíme pracovat s binární haldou. Začneme tedy nejjednodušším řešením a to je postupné nalezení a odebírání nejmenšího prvku, dokud nezískáme  $K$  jako kořen haldy. Toto je vlastně (pravděpodobně) ten algoritmus využitý na té přednášce a v tomto případě by měl časovou komplexitu  $\mathcal{O}(K \log N)$ . Protože máme zaručeno, že  $K$  je řádově menší než  $N$ , nemusíme řešit případ, kde by bylo rychlejší hledat od největšího prvku a můžeme se soustředit na hledání nejmenšího.

I přes to, že komplexita  $\mathcal{O}(K \log N)$  není vůbec špatná, určitě se dá zlepšit. Hlavně, v rámci  $\mathcal{O}$  notaci ztratíme veškeré konstanty, a protože celá komplexita programu pravděpodobně (zatím předpokládám) bude větší než  $\mathcal{O}(\log N)$ , většina optimalizací budou v počtu kroků v algoritmu.

Jako první pokus o řešení, zkusím napsat algoritmus, který bude hledat  $K$ -tý nejmenší prvek v binárním haldě, bez toho, abych pokaždé odstranil nejmenší prvek a hledal nový kořen.

---

### Algorithm 1 Hledání $K$ -tého nejmenšího členu

---

**Input:**  $K$ , *heap*

- 1:  $savedVals[] \leftarrow \text{ordered array}$  ▷ An array of ordered, saved values
- 2: Insert  $heap[1]$  into  $savedVals$
- 3:  $curInd \leftarrow 1$  ▷ Indexing from 1  
▷ Refers to the index of the last "stable" value in  $savedVals$
- 4: **while**  $K > curInd$  **do**
- 5:     **if**  $savedVals[curInd]$  has children nodes **then**
- 6:          $BinaryInsert(savedVals, \text{start: } curInd+1, \text{value: } savedVals[curInd].leftChild)$
- 7:          $BinaryInsert(savedVals, \text{start: } curInd+1, \text{value: } savedVals[curInd].rightChild)$   
▷ A slightly modified binary search for the correct range to which it can put the value  
▷ This will always have  $\mathcal{O}(\log N)$  and  $\Omega(\log N)$  complexity
- 8:     **end if**
- 9:      $curInd \leftarrow curInd + 1$
- 10: **end while**

**Output:**  $savedVals[K]$

---

Ted se pokusím algoritmus vysvětlit.

Máme poli  $savedVals$ , který na začátku jenom kořen haldy. Víme, že je na správném místě (je to nejmenší prvek na místě 1) a přidáme jeho potomky pomocí upraveného binary search, který věřím je dost triviálně upravený, že nemusím dokazovat jeho komplexitu. Protože jsme věděli, že předchozí prvek byl na správném místě, ten  $BinaryInsert$  mohl začít od následujícího indexu. Tímhle skončí první cyklus algoritmu.

Na následujícím cyklu je nejmenší prvek na indexu o jeden větší, ten máme zase zaručený, že je na správném místě a opět přidáme jeho potomky pomocí  $BinaryInsert$ . V případě že nejmenší prvek nemá žádné potomky, tak se jenom zmenší oblast *nestabilních* čísel (t.j. čísla,

která nemusí být na správné pozici). To je pro algoritmus jenom výhodné, takže to zanedbáme při počítání  $\mathcal{O}$  complexity.

V případě, že každý prvek v *savedVals* má potomky, na každém kroku přidáme 2 *nestabilní* prvky, ale jeden předchozí *nestabilní* prvek se stane *stabilní*. To znamená, že na každém kroku získáme 1 *stabilní* prvek a oblast *nestabilních* prvků se zvýší o 1. A protože skončíme první cyklus algoritmu s jedním *stabilním* prvek a dvěma *nestabilními* prvky, je zřejmé, provedeme  $K$  cyklů, abychom našli  $K$ -tý prvek a oblast *nestabilních* prvků bude mít velikost  $K+1$ . Tedy nejvýše  $K$  krát provedeme  $2 \log K$ , což vychází jako  $\mathcal{O}(K \log K)$ .