3. domácí úkol | Vilém Zouhar

1

Úkol := (*ohodnocení*, *deadline*)

1.1 Popis

Nabízený algoritmus selže už na triviálním případu, kdy máme úkoly $(10, 5), (5, 1)$. Hladově vyřešíme ten první, ale tím nám propadne ten druhý. Chtěli bychom to přesně naopak. Seřazení se nám bude hodit, ale ještě navíc budeme požadovat, aby se každý úkol dělal co nejpozději (tak je to totiž matfyzácké). Uděláme si okýnka, kterých bude N. Pak od nejvíce hodnotné budeme zezadu tato okýnka plnit.

1.2 Korektnost

Kdyby plnění různých úkolů trvalo různé dny, tak bychom se mohli dopracovat k batohu. Tady ale využíváme toho, že když má nějaký úkol větší váhu, tak si co nejúsporněji vybere své okýnko.

1.3 Pseudokód

```
sorted(tasks, key=value)
okynka = [false]*N
for task in tasks:
        for i in in range(min(N,task.deadline),-1): # odzadu
                if not okynka[i]:
                        okynka[i] = task
                        break
output(okynka)
```

1.4 Složitost

Dvě smyčky velikosti N, N a sortění $N \cdot log(N)$. Předpokládáme, že deadlines jsou v řádku počtu úkolů. Tedy celkově $O(N^2)$ časově a $O(N)$ paměťově.

2

2.1 Popis

Z DFS inorder průchodu jakéhokoliv BVS dostaneme setřízenou posloupnost. Zdegenerujeme si tedy tímto způsobem náš vstupní strom na spojový seznam (vzestupně připojíme vždy do levého syna) a z tohoto seznamu budeme stavět plně vyvážený BVS. Chtěli bychom si rekurzivně vždy vzít vždy polovinu zpracovávané části, tu prohlásit za kořen a připojit levý a pravý podstrom ze dvou zbytků.

2.2 Korektnost

Výsledný strom musí být perfektně vybalancovaný, neboť jej stavíme ze setřízené posloupnosti.

2.3 Pseudokód

```
a = DFS_inorder(tree)

p = a # prvni prvek ze seznamu

get_balanced(k):
        left = get_balanced(k/2)
        root = p
        p = p.right_node
        right = get_balanced(N - k/2 - 1)
        root.left_node = left
        root -> right_node = right
        return root

output (get_balanced(N/2))
```

2.4 Složitost

DFS inorder můžeme procházet chytře a nepotřebujeme k tomu ani zásobník. Mohlo by se nám totiž stát, že na vstupu dostaneme degenerovaný BVS, který by chtěl řádově lineární zásobník. Stačí si vždy ve stromochodovi pamatovat, odkud jsme přišli. Pokud zeshora, tak chceme jít do levého syna, pokud v levého syna, pak chceme jít do pravého syna a pokud z pravého syna, tak chceme jít nahoru. Takto dokážeme projít celý strom bez použití paměti naví. Samotná rekurzivní funkce naivně vypadá na $O(N \cdot \log(N))$, ale jedná se o rekurzi typu: $T(n) = 2 \cdot T(n/2) + O(1)$, tedy $a = 2$, $b = 2$, $\log_b(a) = 1 \Rightarrow T(n) \in O(n)$. Jelikož se bude zanořovat nejvýše do hloubky budovaného stromu, tak chce také pro zásobník nejvíce $O(\log(N))$ paměti.
Celkově tedy $O(\log(N))$ paměti a $O(N)$ času.