Sada: 2 Příklad: 1 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Procedura computeSiluete(*Input*, *from*, *to*)

vstup: pole uspořádaných trojic *Input*, využíváme v zanoření rozsah od *from* do *to*

výstup: vypočtená silueta v poli uspořádané n-tice *Output*

```
if to = from then
    return Input[from] // rekurzivní zarážka, vrací pole 3 čísel
fi
half ← (to + from)/2
Output[0] ← computeSiluete (Input, from, half)
Output[1] ← computeSiluete (Input, half + 1, to)
// nechť u těchto polí můžeme pomocí funkce takePair (i) vzít i-tou dvojici, přičemž
   poslední číslo utvoří dvojici s NaN
dvojice souřadnic first(x, y) ← Min (Output[0].takePair (0), Output[1].takePair (0))
   // vybere pole s minimem x, z tohoto pole vezme i y
if first ∈ Output[0] then
    chosen ← 0
else
    chosen ← 1
counter[chosen] ← 1
counter[!chosen] ← 0
NewOutput ← prázdné pole
while first.y ≠NaN do
    second(x, y) ← Output[!chosen]. takePair (counter[!chosen])
    if second.y > first.y then
        přidej dvojici first do NewOutput
        first ← second
        second ← Output[chosen]. takePair (counter[chosen])
        counter[chosen] ← counter[chosen] + 1
        chosen ←!chosen
    fi
    following ← Output[!chosen].takePair (counter[!chosen])
    if second.x < folowing.x then
        // NaN je vždy větší než libovolné číslo
        přidej dvojici first do NewOutput
        first(x, y) ← Min (Output[0].takePair (counter[0]), Output[1].takePair
        (counter[1]))
        if first ∈ Output[0] then
            chosen ← 0
        else
            chosen ← 1
        counter[chosen] ← counter[chosen] + 1
    fi
od
přidej first.x do NewOutput
return (NewOutput)
```

Sada: 2 Příklad: 1 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Popis algoritmu: samotná myšlenka algoritmu je jednoduchá. Vstup rozdělíme na jednotlivé trojice reprezentující siluetu jedné budovy. V kroku "panuj"pak spojujeme vždy 2 dané siluety. Nejtěžším krokem je spojení. Musíme procházet dvojici siluet a ověřovat, která je v dané pozici vyšší a tu přidáme do výsledné siluety. To se v algoritmu děje od řádku 7.
Prvně musíme vybrat siluetu, která má první budovu s menší souřadnicí x. Pomocná funkce Min dostane jako vstup 2 dvojice a ty porovná podle první složky - x. Poté musíme zjistit, ze kterého pole jsme dvojici vybrali a správně nastavit počítadla vybraných dvojic. Zápisem !*chosen* myslím "negaci", tedy druhou možnou hodnotu.
Nejzajímavější částí algoritmu je cyklus. Ten se zastaví v případě, že jsme již prošli obě pole, což lze testovat třeba tak, že otestujeme, jestli je ve dvojici $first$ číslo. To nám rozšiřuje nároky na funkci Min- ta musí fungovat i pro NaN, v takovém případě vždy volí libovolné číslo. Pro 2 NaN ukládá do dvojice NaN. V Cyklu s každým průchodem posouváme dvojici *second*, kterou testujeme vůči 2 různým věcem. Zaprvé ji porovnáváme s y hodnotou první dvojice, zadruhé testujeme, zdali už jsme se v pomyslném počítadle na ose x neposunuli na další budovu v první siluetě. Do výstupu přidáváme první dvojici za podmínky, že vstoupíme do jednoho z if bloků. První z podmínek v případě splnění prohazuje siluety, ze kterých budeme vybírat, druhá jen posouvá dvojici na další budovu stejné siluety.

Korektnost:

Asymptotická časová složitost: algoritmu je dle zadání $\mathcal{O}(n \cdot \log(n))$. Rekurze se volá $\log_2(n)$-krát přičemž v k-tém rekurzivním zanoření (první volání bez rekurze odpovídá $k = 0$) budu spojovat celkem 2^k m-tic, kde m odpovídá maximálně $\frac{3 \cdot 2^{\log_2(n)-k}}{2^k}$, kde n je počet budov (počet trojic). m jsem odvodil tak, že v nejspodnějším patře spojuji trojice, každé vyšší patro délku m-tice maximálně zdvojnásobí. Po úpravě a roznásobení získáme hodnotu $3n$, která odpovídá maximálnímu počtu operací (zanedbáváme konstantní počet operací porovnání) na každé úrovni zanoření rekurze. To tedy spolu s $\log_2(n)$ počty zanoření dává $\mathcal{O}(n \cdot \log(n))$