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 usporiadaných trojic Input, využíváme v zanorení rozsah od from do to
výstup: vypočítaná silueta v poli usporiadanej n-tice Output
if to = from then
    return Input[from] // rekurzívna zarážka, vracia pole 3 čísiel
half ← (to + from)/2
LeftArray ← ComputeSiluete (Input, from, half)
RightArray ← ComputeSiluete (Input, half + 1, to)
pl ← 0 // index i-te dvojice v poli left
pr ← 0 // index i-te dvojice v poli right
Output ← ∅
while pl < ⌈|LeftArray|/2⌉ ∨ pr < ⌈|RightArray|/2⌉ do
    left ← LeftArray.TakePair (pl) // funkcia TakePair (i) vráti i-tou dvojici z poľa
    right ← RightArray.TakePair (pr)
    if left.x < right.x then
        prevRight ← RightArray.TakePair (pr − 1)
        if left.y ≥ prevRight.y then
            přidej left do Output
        else
            if prevRight.x ≤ left.x then
                přidej (left.x, prevRight.y) do Output
            else
                přidej left do Output
            fi
        fi
        pl ← pl + 1
    else
        prevLeft ← Left.TakePair (pl − 1)
        if right.y ≥ prevLeftArray.y then
            přidej right do Output
        else
            if prevLeft.x ≤ right.x then
                přidej (right.x, prevLeft.y) do Output
            else
                přidej right do Output
            fi
        fi
        pr ← pr + 1
    fi
od
// vložíme poslednú x-vú nepoužitú súradnicu do Output
if LeftArray.last ≤ RightArray.last then
    přidej RightArray.last do Output
else
    přidej LeftArray.last do Output
return (Output)
```

Popis algoritmu: Algoritmus postupuje metodou rozdeluj a panuj. V prvom kroku voláme procedúru ComputeSiluete ($Input, 0, počet\ trojic\ v\ Input$), ktorá rozdelí vstupné pole na podproblémy až na jednotlivé trojice reprezentujúce siluety jednej budovy,

kde sa rekurzia zastaví. V kroku "panuj"následne spojujeme vždy 2 dané siluety. Siluety si môžeme predstaviť na jednej ose, kde ich prechádzame zľava po x-ovej osy cez všetky dvojice a spracovávame vždy prvú nespracovanú dvojicu, z poslednej bereme len x súradnicu pre uzavretie siluety. Dvojice z poľa získavame pomocou funkcie TakePair (i), ktorá vracia i-tu dvojicu z poľa, pre hodnoty mimo poľa vráti nulu na danej pozícií dvojice(napr. posledná dvojica je $(x, 0)$, -1 dvojica je $(0, 0)$)
V cykle vždy vezmeme dvojice na ktorých sa aktuálne nachádzame v oboch siluetách. Následne určíme, ktorá je viac vľavo po x-ovej osy. Aby bola dvojica relevantná pre výslednú siluetu musí splňovať jednu z nasledujúcich podmienok:

1. y súradnica dvojice je väčšia ako posledná spracovaná y-súradnica z druhého poľa, vtedy pridáme danú dvojicu do výsledku, lebo prevyšuje aktuálnu výšku druhej siluety

2. ak je mešia ako predchádzajúca to znamená že daná hrana, môže vytvárať prienik s hranou druhej siluety alebo druhá silueta sa nachádza mimo prvej siluety, čo testuje zanorený esle-if

Korektnost: Algoritmus je korektný ak je konečný a parciálne korektný. Pre konečnosť ukážeme, že rekurzia skončí a, že spájanie dvoch polí v rekurzívnom voláni tiež skončí. Pri rekuzii vidíme, že každým zanorením sa vstupné pole zmenší o polovicu, teda existuje zanorenie kedy vstup bude obsahovať len jednu trojicu a rekurzia zastaví. Podmienka cyklu kontroluje či sme spojili už všetky dvojice z polí, keďže každým prechodom cyklu zvýšime *pl* alebo *pr* a veľkosti polí sa nemenia môžeme tvrdiť, že cyklus zastaví. Takže aj celý algoritmus je konečný.
Pre parciálnu korektnosť platí nasledujúce. Ak vstup obsahuje jednu trojicu tak výstup algoritmu je táto trojica. Môžeme induktívne tvrdiť, že pridaním ďalšej trojice môžu vzniknúť dva prípady:

1. dané siluety majú prienik: V tomto prípade sa algoritmus rozhoduje, či sa daná silueta nachádza v tej druhej(viď popis algoritmu) ak áno berieme výšku z predchádzajúcej siluety. V prípade, že sa hrany pretínajú musíme započítať vyšiu výšku v na danej x-ovej súrandici.

2. dané siluety nemajú prienik: Z algoritmu vidíme, že do výsledneho riešenie ich zreťazí za seba podľa x-ovej súradnice.

Keďže algoritmus vždy berie len jednú dvojicu zo spájaných polí až kým obe polia neprejde nenastane teda situácia, že by na konci cyklu existovala nejaká nespracovaná dvojica. Následne na uzavretie siluety algoritmus vezme hodnotu najďalej na x-ovej osy, takže algoritmus korektne uzavrie a spojí 2 siluety teda je korektný.

Asymptotická časová složitost: tohoto algoritmu je $\mathcal{O}(n \cdot \log(n))$, rekurzia sa volá $\log_2 n$-krát (pólením pôvodneho vstupu). V každom rekurzívnom zanorení (okrem posledného, ktoré v konštantnom čase vráti trojicu) algoritmus spája 2 polia vo while cykle.

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Toto spájanie prebehne v lineárnom čase vzhľadom k dĺžke oboch polí, keďže v každom kroku spracujeme jednu dvojicu a operácie v cykle sú v konštantnom čase. Vieme, že v rekurzívnych volaniach v hĺbke i spojujeme maximálne 2^i polí, kde každé z týchto polí môže mať dĺžku najviac $n/2^i$. Keďže počet spracovávaných bodov je v každej vrstve rekurzie nanajvýš n, dokážeme ju vyriešiť v $\mathcal{O}(n)$. Z čoho plynie , že algoritmus prebehne v $\mathcal{O}(\log(n))$ rekurzívnych volaniach o zložitosti $\mathcal{O}(n)$, takže asymptotická časová zložitosť algoritmu je $\in \mathcal{O}(n \cdot \log(n))$.

Tento algoritmus bol testovaný na ľuďoch. Behom vývoja nebol zabitý žiaden živý tvor, i keď dvom skoro hrablo.