**Algoritmy a dátové štruktúry II (IV003)** **SADA PROBLÉMOV 1**

**Autor**: Michal Lukáč, 430614

**Spoluriešiteľ**: Kristína Kaiserová

**Úloha 1**

Je treba nájsť optimálny prvok postupnosti a ich vzájomne priradených váh je prvok, ktorý splňuje nasledújúce dve podmienky: a . Takýto prvok je vpodstate váhovým mediánom. Obyčajný medián(prípadne Kvantil ) je hodnota ktorá delí zoradený zoznam hodnôt na dve rovnako početné poloviny. Je treba vytvoriť algoritmus ktorý nájde takýto medián v Ο(n). Jednoduchým spôsobom by bolo zoradiť postupnosť čísel a váhovy medián dopočítať na takejto zoradenej postupnosti. To by však trvalo Ο(n.log(n)).

V našom prípade môžeme využiť algoritmus SELECT(z prednášok) pre problém i-teho prvku, ktorý využíva princíp rekurzie. Zložitosť tohoto algoritmu je lineárna voči dĺžke postupnosti teda O(n). Preto aby sme našli váhový median musíme nájsť pomocou SELECT median postupnosti . Na základe mediánu rozdelíme prvky do dvoch množín a to množiny A a B. Sumu váh množiny A označíme ako . Potom na základe sumy váh voláme rekurzívne nášu funkciu WSELECT pre nájdenie váhového mediánu.

***WSELECT(X,W,M=1/2)***

*1. x = SELECT(X,n/2)*

*2. = SUM(xi<x)*

*3. if +≥ M & : return x*

*5. if < M :*

*6. A, = prvky väčšie než x*

*7. WSELECT(A,, M-SUMW(xi<x)*

*8. else*

*9. A, = prvky menšie rovn než x*

*10. WSELECT(A,, M)*

Pokiaľ je suma väčšia ako ½ potom medián je menší ako x. Pokiaľ je suma + väčšia alebo rovná ako ½ vrátime hľadaný medián. Pokiaľ je suma + menšia ako ½ potom prvok x nemôže byť medián podľa zadaných podmienok.

Zložitosť nášho algoritmu je O(n). Algoritmus SELECT dokáže nájsť medián za O(n). Každé volanie WSELECT narába iba s polovičným počtom prvkov z predošlého volania. Potom môžeme složitosť tohto rekurzívneho algoritmu ako T. Na základe Master Theorem analyzujeme túto rovnicu a dostávame že T(n) O(n).

**Algoritmy a dátové štruktúry II (IV003)** **SADA PROBLÉMOV 1**

**Autor**: Michal Lukáč, 430614

**Spoluriešiteľ**: Kristína Kaiserová

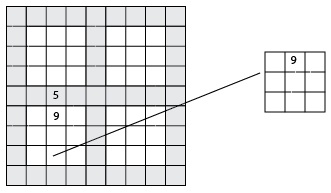
**Úloha 2**

Daná je matica n x n. Cieľom programu je nájsť Esem teda body ktoré sú v matici najväčšie vo svojom 4 okolí(na krajoch matice 3 okolí alebo 2 okolí). Požadované je aby algoritmus vykonal O(n) a bol rekurzívny.

Zvolíme rekurzívny algoritmus Divide and Conquer. Algoritmus delí množinu na menšie časti a časť spracuje rekurzívne. Na koniec skombinuje výsledky. Celková cena je T(n) = Croz + T(n..) + Ckomb.

MAXMatrix(X)

1. Získame prvý, prostredný a posledný riadok a stĺpec. Počet prvkov je 6n, keďže jeden riadok/stĺpec má n prvkov. Toto rozdelenie nám rozdelí maticu na 4 kvadranty.
2. Nájdeme maximum v týchto 6n elementov a priradím do premennej max
3. Ak je hodnota max väčšia než je okolie tak označíme prvok ako Esem
4. Inak je tento prvok menší, takže nemôže byť Esem. Väčší prvok je teda v jednom z 4 kvandrantov a preto rekurzívne zavoláme na kvadrant funkciu MAXMatrix

**

Analýza zložitosti algoritmu:

Každým zanorením sa zredukuje matica na polovicu teda z nxn na pre O(n).

T(n) = T(n/2) + cn

T(n) = T(n/4) + cn/2 + cn

= T(1) + cn(1 + ½ + 1/4 + 1/8 + ...) = O(n)

Korektnosť algoritmu:

Pokiaľ sa rekurzívne zanoríme do jedného z kvadrantov vieme, že v tomto kvadrante je globálny prvok pre ktorého sme sa do tohto kvadrantu zanorovali. Preto ako invariant, ktorý platí vždy, je že počas rekurzie nikdy najväčší prvok neklesne.

**Algoritmy a dátové štruktúry II (IV003)** **SADA PROBLÉMOV 1**

**Autor**: Michal Lukáč, 430614

**Spoluriešiteľ**: -

**Úloha 3**

Cieľom je vytvoriť 234 strom s operáciami pre MINIMUM, INSERT, DECREASE KEY, DELETE, EXTRACT MIN s časovou zložitosťou najviac O(logn). Kde n je množina kľučov uložených v listoch.

Dôležitou úlohou pre zachovanie časovej zložitosti O(logn) je, že výška h pre tento strom logn. Jedinou môžnosťou ako zvýšiť výšku stromu je rozdelením koreňového uzla pri zaplnení. Z toho vyplýva že všetky listy sú v rovnakej hĺbke. Kĺuče sú uložené v listoch zoradené z ľava do prava. Každý vrchol x obsahuje hodnotu small[x], ktorá je rovná najmenšiemu kľúču uloženému v listu podstromu s koreńom x.

Môžeme povedať, že každá štruktúra pre uzol môže byť reprezentovaná následne:

1. Každý uzol obsahuje hodnotu x.n teda počet uľožených ukazateľov v uzle x. Maximálny počet ukazateľov je 4.
2. Ukazatel x.left, x.leftmiddle, x.rightmiddle, x.right na uzly alebo listy
3. Booleovskú hodnotu x.leaf, ktorá uchováva pravdivú hodnotu True pokiaľ je uzol list, inak je FALSE a je to vnútorný uzol.
4. Hodnota x.key pokiaľ je uzoľ list
5. Hodnota x.small pokiaľ je uzoľ vnútorným uzlom

Nech T označíme náš strom. Štruktúru T obsahuje okrem koreňového uzlu, aj ukazateľ na list najmenšieho uzlu. Pri inicializácií vieme, že prvý vložený prvok je list.

**INSERT**

Pri vložení nového listu do stromu T postupujeme následne. Spočítame si počet synov a pokiaľ je počet následníkov 4, teda strom je plný vzniká nový uzol.

INSERT(k):

r = T.root

if r.n == 4

1. Vytvorim nový uzol S, S.leaf = false, s.left = r, s.n =1 a nastavím T.root = S
2. Zavoĺam SPLITCHILD(S,1)
3. Zavolám INSERTNONFULL(S,k)

else:

4, Zavolám INSERTNONFULL(r,k)

Funkcia SPLITCHILD(x,i) slúži pre rozdelenie i-teho následníka x na dve časti a spojenie týchto uzlov späť do x. Pokiaľ má uzol, ktorý rozdeľujeme označme ho y, 4 následníkov, potom takýto uzol rozdelíme na dve časti y a z. V takomto uzlu taktiež nastaví novú najmenšiu hodnotu small[x]. Na riadku označeného ako 3 v INSERT vložíme novú hodnotu do x. Zavedieme si taktiež operáciu INSERTNONFULL ktorá bude definovaná ako:

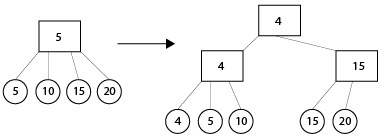
INSERTNONFULL(x,k):

if x.leaf:

1. Nájdeme miesto pre k
2. Zvýšime x.n o 1

else:

1. Nájdeme index i pre list/uzol pre hodnotu k. Tento uzol označíme xi
2. Pokiaľ tento uzol je plný zavolaj SPLITCHILD(x,i)
3. Zavolaj INSERTNONFULL(xi,k)



Na obrázku vidíme, že po pridaní 5, 10, 15, 20 do stromu dostávame koreň so 4 listami. Pokiaľ pridáme v takejto chvíli 4 do stromu. Zavolá sa funkcia SPLITCHILD, ktorá rozdelí koreň stromu na dve časti. Môžeme si všimnúť, že strom rastie vždy od koreňa, preto platí, že hĺbka stromu pre všetky listy je rovnaká. Zoberme si teraz napríklad situáciu kedy by bola do uzla s hodnoutou small=4 a tromi listami 4, 5, 10 pridané ďaľšie číslo 11. Po tejto operácií sa tento uzol zaplní. Pri pridávaní 12 do stromu T sa postupuje následne. Vo funkcí INSERT keďže koreňový uzol nie je plný, tak sa priamo zavolá INSERTNONFULL(koreň=x,k). V INSERTNONFULL sa prejdú ukazatele a pozre sa do hodnôt small. Zistí sa, že hodnota 15 je väčšia ako 12 preto sa 12 musí pridať do predchádzajúceho uzlu. Tento uzol je však zaplnený preto sa zavolá SPLITCHILD a koreňový uzol dostane ďaľší uzol ako následníka.

MINIMUM

Keďže vieme, že v hodnote small je hodnota, ktorá sa rovná najmenšiemu kľúču uloženému v listu podstromu, potom vieme, že v koreňovom vrchole sa nachádza najnižšia hodnota. Predpokľadajme, že naľavo sa uchovávajú vždy menšie hodnoty než, napravo. Potom môžeme minimum nájsť pozrením sa vždy v ceste do najľavejšieho uzlu a nakoniec listu. Prípadne si môžeme zobrať štruktúru T, ktorá obsahuje ukazateľ na najmenší list.

EXTRACT MIN

Nájdenie hodnoty minima pomocou funkcií MINIMUM a odstránenie ju z uzla. Nastavenie hodnoty, ktorá je v nasledujúcom liste ako small a taktiež nastavenie v štruktúre stromu a prejdenie stromu a nastavenie tejto hodnoty do uzlov daného podstromu.

DECREASE KEY

---

DELETE

Volanie delete si môžeme predstaviť podobne ako je vo fibonačiho halde. A to zavolaním funkcie DECREASEKEY(x,-inf) a EXTRACTMIN.

**Algoritmy a dátové štruktúry II (IV003)** **SADA PROBLÉMOV 1**

**Autor**: Michal Lukáč, 430614

**Spoluriešiteľ**: -

**Úloha 4**

Uvažujme zoznam S prirodzených čísel. Nad zoznamami vykonávame nasledujúce operácie. INSERT(cena operácie je 1), MIN-ALL(cena operácie je N-teda dĺžka zoznamu), MIN-ONE, DELETE.

**1.Libovolná postupnosť n operací typu INSERT a MIN-ALL má složistosť O(n)**

**NIE**

Protipríklad: Uvažujme, že vložíme prvky 1,1,1,1,1,1,2 a následne zavolám MIN-ALL. Ostanú mi prvky 1,1,1,1,1,1. Pokiaľ znovu pridám prvok napríklad 2 a znovu zavolám funkciu MIN-ALL opäť mi ostane 1,1,1,1,1,1. Neustalé môžem MIN-ALL volať na hodne veľkým číslom seznamom o velikosti n a operácia INSERT nedokáže zaplatiť operáciu MIN-ALL.

**2.Ľubovolná postupnosť n operací typu INSERT A MIN-ONE má zložitosť O(n)**

**ÁNO**

Dôkaz pomocou metódy účtov:

|  |  |  |
| --- | --- | --- |
| Operácia | Cena | Kredity |
| INSERT | 1 | 3 |
| MIN-ONE | |L| | 0 |

Operáciu INSERT zaplatíme 1 kreditom, 2 kredity dáme na účet. Operáciu MIN-ONE zaplatíme kreditmy z účtov. Počas celého výpočtu platí invariant počet kreditov na účte je rovný počtu prvkov v zásobníku. Z toho plynie, že zostatok na účte nikdy neklesne pod 0.

Pokiaľ by sme zvolili, že počet kreditov za operáciu INSERT bude iba 2, tak nám to nebude stačiť. Uvažujme, že postupne pridávame prvky zaradom 1,1,1,1,1,1,2 túto postupnosť by sme zaplatili 7 kreditmi a 7 vložíme na účet. Následne zavoláme operáciu MIN-ONE ktorá si vezme 7 kreditov. To nám však nebude stačiť pretože ak by nám 1 ostala a pridáme napríklad 2,4,5. Na účte máme iba 3 kredity a operácie MIN-ONE by vyžadovala v tejto chvíli 4 kredity. **Preto musí byť operácia INSERT ohodnotená 3 kreditmi.** Súčet kreditov vykonaných operácií je menšie alebo rovné 3n. Preto platí O(n).

**3.Ľubovolná postupnosť n operací typu INSERT A DELETE má zložitosť O(n)**

**NIE**

Protipríklad: Uvažujme podobne ako v 1 prípade, že vložíme prvky 1,1,1,1,1,1,2 a následne zavolám DELETE(S,1) s časovou zložitosťou O(n). Zostanú mi prvky 1,1,1,1,1,1. Operáciu DELETE(S,1) však môžeme volať viackrát za sebou. Preto takúto operáciu opäť nezaplatíme pre akúkoľvek postupnosť.

**4.Ľubovolná postupnosť n operací typu INSERT a DELETE má zložitosť O(n), pričom DELETE sa volá vždy s iným parametrom.**

**ANO**

Dôkaz pomocou metódy účtov:

|  |  |  |
| --- | --- | --- |
| Operácia | Cena | Kredity |
| INSERT | 1 | 3 |
| DELETE | |L| | 0 |

V tomto prípade vieme, že operácia DELETE sa volá vždy s inou hodnotou i. Preto platí aj invariant že počet kreditov na účte je väčší rovno 0.

Preto pokiaľ pridávame postupnosť prvkov 1,1,1,1,1,1,2. Na účte mi zostalo 14 kreditov. Zavolaním DELETE(S,1) nám ostane na účte 1,1,1,1,1,1 a zmazaná bola iba dvojka. Na účte nám zostalo 7 kreditov. Zo zadania vieme, že delete nemôže byť volaná po sebe s takým istým číslom. Keďže táto podmienka platí tak DELETE nemôže byť zavolaná s parametrom 1 ale iným číslom. Uvažujme, že k postupnosti pridáme prvok 3. Na účte je momentálne 10 kreditov. Následným zavolaním DELETE s ľubovolným parametrom okrem 1(keďže ten bol volaný naposledy). Máme zostatok kreditov na zmazanie všetkých prvkov. Celková zložitosť postupnosti n operácií je menšia alebo rovná súčte kreditov vykonaných operácií a teda súčet kreditov vykonaných operacií je menšie alebo rovno 3n. Preto platí O(n).

**Algoritmy a dátové štruktúry II (IV003)** **SADA PROBLÉMOV 1**

**Autor**: Michal Lukáč, 430614

**Spoluriešiteľ**: -

**Úloha 5**

**a)**

Cieľom je nájsť lokáciu takú, ktorou minimalizujeme súčet kde je vzdialenosť medzi dvoma lokáciami. Objektívnu funkciu, ktorú minimalizujeme môžeme zapísať ako:

Pokiaľ chceme nájsť minimum tejto funkcie musíme ju zderivovať podľa p a následne si ju rozpíšeme do dvoch častí podobne ako je podmienka v prvej úlohe.

Váha bodu reprezentujúceho medián sa vo výpočte nezahrňuje. Táto zderivovaná funkcia je neklesajúca pretože ak sa p zväčší tak počet čísel sa nezníži. Derivácia v bode p je menšia ako nule pokiaľ p je menšie ako akékoľvek . Obdobne je derivácia v bode p väčšia ako 0 pokiaľ p je väčšie než akékoľvek . Potom existuje taký bod q, pre ktorý platí, že derivácia v bode p je menšia-rovná 0 pokiaľ p < q. Taktiež platí, že derivácia je väčšia ako 0 pokiaľ p > q. Potom tento bod je globálne minimum.

**b)**

Riešenie dvojrozmerného lokalizačného problému je nájdenie mediánu ako pre súradnicu x, tak pre súradnicu y.

*2DLOKAL(P,W):*

*x = WSELECT(P.X,W.X,1/2)*

*y = WSELECT(P.Y,W.Y,1/2)*

*return Point(x,y)*