Ročníkový projekt 1

O projekte

Keďže je v druhom semestri môjho študijného odboru povinný predmet Ročníkový projekt 1, ktorého súčasťou je aj vytvoriť ku svojmu projektu webovú stránku na ktorej bude dokumentácia ku projektu, tak tu tá stránka je.
Názov môjho ročníkového projektu je: "Vizualizácia dátovej štruktúry Treap"

Teoretický rozbor

Úvod

Projekt je vytvorený v jazyku Python, s využitím grafickej knižnice `tkinter`. Cieľom môjho projektu bolo vizualizovať dátovú štruktúru Treap.

Treap je náhodný binárny vyhľadávací strom, ktorý sa dá použiť ako implementácia dátovej štruktúry `set`. Táto dátová štruktúra ponúka možnosť operácií `insert, delete, find` v priemernom čase `O(n log(n))`, kde `n` je počet prvkov v množine. Rozoberme si teraz, čo tieto slová znamenajú.

Strom

Strom v informatike označuje súvislý graf, ktorý neobsahuje žiadny cyklus. Toto znamená, že ak si jeden vrchol zvolíme ako koreň stromu, tak jeho susedov nazývame ako jeho *synov*, resp. rodovo korektne *potomkov*, resp. *deti*, a opačne, tento vrchol nazývame ako *otec* alebo *rodič* jeho *synov*.

Binárny strom

Binárny strom je špeciálny prípad stromu, kde každý vrchol má najviac 2 synov. Týchto synov zvykneme označovať ako *ľavého a pravého syna*.
Úplný binárny strom je taký strom, kde každý vrchol okrem listov má 2 synoch, a všetky listy majú od koreňa rovnakú vzdialenosť.
Hĺbka úplného binárneho vyhľadávacieho stromu s `n` vrcholmi je `O(log(n))`.

Binárny vyhľadávací strom (Binary search tree, BST)

BST je špeciálny prípad binárneho stromu, kde je v každom vrchole uložená hodnota, ktorú nazývame *kľúč*, a pre každý vrchol platí, že jeho ľavý syn, resp. všetky vrcholy v jeho ľavom podstrome majú od neho kľúč menší, a pravý vrchol má od neho kľúč väčší. To znamená, že na zistenie, či v BST je nejaký vrchol sa nám stačí pozrieť na najviac toľko vrcholov, koľko je najhlbší vrchol, čo je za ideálnej situácie, ako sme si povedali v prechádzajúcom odstavci `O(log(n))`.
Pozrime sa na to, ako tieto operácie fungujú.

Zisti, či je v BST vrchol s nejakým kľúčom (Find)

Začnime operáciou `Find`. Táto funguje rekurzívne, presne podľa definície BST. To znamená, že sa v každom koreni pozrieme, že či je kľúč v tomto koreni rovný tomu, čo hľadáme. Ak áno, tak sme ho našli, a môžeme sa vynoriť z rekurzie, a ak nie, tak sa pozrieme, že či je kľúč, ktorý hľadáme väčší alebo menší ako kľúč v tom vrchole kde sme. Podľa tohoto sa rekurzívne zavoláme buď na ľavého, alebo pravého syna.
Pseudokód:

```
funkcia Find(vrchol, kľúč): #vracia True/False
  Ak vrchol.kľúč == kľúč:
 vráť True
  Inak:
 Ak vrchol.kľúč > kľúč:
   vráť Find(vrchol.ľavý_syn, kľúč)
 Inak:
   vráť Find(vrchol.pravý_syn, kľúč)
```

Keďže v momente, ako sa začneme vynárať z rekurzie sa už nikdy nezanoríme, a maximálne sa môžeme zanoriť tak hlboko, ako je hĺbka najhlbšieho vrcholu, a v *priemernom* prípade je najhlbší vrchol `O(n log(n))` hlboko, tak časová zložitosť tejto operácie je `O(log(n))`.

Vlož do BST vrchol s nejakým kľúčom (Insert)

Táto operácia prebieha veľmi podobne, ako operácia *Find*, a to tak, že robíme úplne rovnakú vec, ako keď by sme hľadali vrchol, ktorý chceme vložiť. Keď prídeme k miestu, kde už žiaden vrchol nie je, a chceli by sme tam pokračovať v hľadaní, tak ho tam umiestnime.
Pseudokód (predpokladáme, že taký vrchol v BST nie je):

```
funkcia Insert(vrchol, kľúč): #nevracia nič
  Ak vrchol.kľúč == kľúč:
 vráť
  Inak:
 Ak vrchol.kľúč > kľúč:
   Ak vrchol.ľavý_syn != None:
  Insert(vrchol.ľavý_syn, kľúč)
   Inak:
  vrchol.ľavý_syn = Vrchol(kľúč)
 Inak:
   Ak vrchol.pravý_syn != None:
  Insert(vrchol.pravý_syn, kľúč)
   Inak:
  vrchol.pravý_syn = Vrchol(kľúč)
```

Čo sa týka zložitosti, tak podobne ako operácia *Find*, aj tu sa po vynorení z rekurzie nikdy znovu nezanoríme, čo znamená, že maximálne sa pozrieme na toľko vrcholov, koľko je najväčšia hĺbka, a rovnako ako pri *Find*, v *priemernom* prípade je najhlbší vrchol `O(log(n))` hlboko, tak časová zložitosť tejto operácie je `O(log(n))`.

Vymaž z BST vrchol s nejakým kľúčom (Delete)

Na začiatku táto operácia funguje rovnako ako *Find*, teda že nájde, kde náš vrchol na vymazanie je. Následne môže nastať niekoľko možností:

1. **Vrchol je list**

   V tomto prípade stačí proste vrchol vymazať, keďže je to list, tak nemá žiadneho syna, a teda nám to BST nenaruší.

2. **Vrchol má jedného syna**

   V tomto prípade je treba nahradiť aktuálny vrchol svojim jediným synom. BST ostane stále validné.

3. **Vrchol má dvoch synov**

   Toto je najzložitejší prípad, kde si ale stačí uvedomiť, že vrchol, ktorý by mal ísť na miesto vrcholu, ktorý mažeme je ten vrchol, ktorého kľúč je najväčší menší, alebo najmenší väčší od kľúča vrcholu, ktorý mažeme.

Takže pseudokód (predpokladáme, že taký vrchol v BST je):

```
funkcia Delete(vrchol, kľúč): #vracia vrchol
  Ak vrchol.kľúč == kľúč:
 Ak vrchol.ľavý_syn == None && vrchol.pravý_syn == None:
   Vráť None
 Ak vrchol.ľavý_syn == None:
   Vráť vrchol.pravý_syn
 Ak vrchol.pravý_syn == None:
   Vráť vrchol.ľavý_syn
 w = vrchol.ľavý_syn
 pokiaľ w!=None:
   w=w.pravý_syn
 vráť w

  Inak:
 Ak vrchol.kľúč > kľúč:
   vrchol.ľavý_syn = Delete(vrchol.ľavý_syn, kľúč)
 Inak:
   vrchol.pravý_syn = Delete(vrchol.pravý_syn, kľúč)
 vráť vrchol
```

Časová zložitosť tento operácie je z rovnakého dôvodu ako pri *Find* a *Insert* `O(log(n))`.

Treap

Problémom prístupu, ktorý sme popísali je to, že zložitosť operácií je závislá od hĺbky BST, a v najhoršom prípade (napr. vkladáme prvky v usporiadanom poradí) nedostaneme strom hĺbky `O(log(n))`, ale strom hĺbky `O(n)` (taký spájaný zoznam), a teda sme si zložitosťou oproti normálnemu poľu vôbec nepomohli.
Riešenie je, nejakým spôsobom tento strom vyvažovať, aby bol vždy dostatočne široký a mal dostatočne malú hĺbku.
Ako jedno z riešení, ktoré je veľmi jednoduché na naprogramovanie je použitie Treapu. Názov Treap vznikol spojením slov Tree a Heap, teda strom a halda. Ide o strom, kde v každom vrchole máme rovnako ako doteraz uložený kľúč, ale aj náhodne vygenerovanú hodnotu (volajme ju priorita), a v prípade, ak sa pozreráme na kľúče, tak náš strom je validný BST, a ak sa pozeráme na priority, tak náš strom je validná halda.
Dôvod, prečo tento prístup spôsobí, že náš strom bude dostatočne plytký je, že v prípade, ak do BST pridávame vrcholy v náhodnom poradí, tak vo väčšine prípadov (v priemernom prípade) bude hĺbka BST približne `O(log(n))`, a v našom strome sú prvky náhodne

preusporiadané, čo je to isté, ako keby boli pridávané v náhodnom poradí.

Ako takéto niečo implementovať?

Začnime trochu z iného konca. Vytvorme si dve funkcie, pomocou ktorých už operácie Insert a Delete vytvoríme veľmi jednoducho. Tieto dve operácie budú split a merge, a budú robiť to, že budú rozdeľovať treap na dva validné treapy tak, že v jednom budú všetky kľúče menšie ako nejaká hodnota, a v druhom väčšie. Operácia merge bude robiť to, že spojí dva validné treapy, kde všetky kľúče v jednom sú menšie ako v druhom, do jedného validného treapu.

Operácia `Insert`

S využitím funkcií `merge` a `split`, ktoré už máme, je táto funkcia jednoduchá. Bude fungovať ako rekurzívna funkcia, ktorá sa volá na pravý, alebo ľavý podstrom podľa kľúča prvku, ktorý chceme vkladať, ale v prípade, ak sa zavoláme na prvok, ktorého priorita je menšia priorita prvku, ktorý chceme vložiť, tak vieme, že aby bola dodržaná podmienka, že to musí byť validná halda, tak ten prvok sa musí nachádzať presne na tom mieste. To znamená, že zavoláme funkciu split na ten vrchol, a z dvoch výsledných stromov sa stanú podstromy prvku, ktorý vkladáme.

Operácia `Delete`

Táto operácia funguje ešte jednoduchšie ako operácia `Insert`. Tu jednoducho rekurzívne nájdeme prvok, ktorý chceme vymazať, a následne zavoláme funkciu `Merge` na ten prvok, ktorý chceme vymazať, a jej výsledok bude namiesto toho podstromu, ktorého koreň bol prvok, ktorý sme vymazali.

Operácia `Find`

Operácie `Find` funguje úplne rovnako ako v normálnom nevyvažovanom BST.

Časový harmonogram

Okrem dokumentácie musí stránka k projektu obsahovať aj časový harmonogram, aby bolo vidieť, ako sme si rozložili čas, alebo tak.

Dátum	Čo sa vtedy stalo
19.2.2020	Úvodná hodina RP 1
do 15.3.2020	Výber témy
do 15.3.2020	Hľadanie zdrojov
do 25.4.2020	Programovanie projektu
do 15.6.2020	Výroba tejto dokonalej stránky

Návod na ovládanie

Síce je ovládanie projektu výlučne grafické, ale mám pocit, že v rámci RP 1 je vyžadované, aby bolo ovládanie popísané. Takže:

Rozdelenie obrazovky

Obrazovka je rozdelená do dvoch častí, pravej (časť A) a ľavej (časť B). V časti B je numerická klávesnica na zadávanie queries.
Queries sa zadávajú tak, že uživateľ zadá kľúč, a následne stlačí, ktorá query to má byť (Find, Delete, Insert).
Následne sa v časti A, ktorá je rozdelená na časti A1, A2, A3, A4 zobrazí táto query. V časti A1 je celý čas zobrazený Treap, v pravom hornom rohu časti A1 je popis, čo program momentálne robí, a v ľavom hornom rohu je znázornený vrchol, pre ktorý túto query robíme.
Všetky vrcholy sú znázornené kružnicami, v ktorých je napísaný kľúč (označený ako "k") a priorita (označená ako "p").
V častiach A2 a A4 sa znázorňujú čiastkové výsledky (pravý a ľavý podstrom) pre operáciu Split, a v časti A3 zase výsledky operácie Merge.

Súbory na stiahnutie

Odkaz na .zip balík s projektom.
Odkaz na .pdf s dokumentáciou.

vytvoril: **Marcel Palaj**, palaj.marcel@gmail.com, 2020, 1AIN2