Vizualizácia intervalového stromu s lazy loadingom

Teória

Intervalový strom je dátová štruktúra určená na rýchle hľadanie určitej špecifickej vlastnosti na ľubovoľnom zadanom intervale. Touto vlastnosťou môže byť napríklad súčet, súčin, minimum alebo maximum na danom intervale. Intervaláč ktorý implementujem bude súčtový, ale miernym pozmenením implementácie vieme vytvoriť aj súčinový, minimový alebo maximový. Súčtový som si vybrala preto, lebo mi príde najjednoduchší na pochopenie jednotlivých vykonávaných funkcionalít.

Intervalový strom je rozumné si predstaviť ako úplný binárny strom vytvorený nad poľom prvkov s ktorými sa pracuje. To znamená, že hodnoty v listoch sú prvky poľa a hodnoty v inom vrchole je súčtom všetkých listov ktoré sú v jeho podstrome. Inými slovami, je to súčet intervalu pod ním. Napríklad pre pole [1, -5, 2, 2, 16, 7, 0, 2] bude intervalový strom vyzerať takto:

Hodnoty poľa vidíme v listoch. Každý vrchol okrem hodnoty zobrazuje aj rozsah ktorý doň spadá.

Prečo intervalový strom?

Môžme sa pýtať, načo nám intervaláč bude, veď zaberá viac pamäte ako samotné pole. To je síce pravda, a aj vytvorenie stromu zaberie viac času ako vytvorenie len samotného poľa. Ale ak máme vytvorený strom, odpoveď na otázku, aký je súčet na niektorom intervale vieme urobiť výrazne rýchlejšie, ako na poli. Druhá operácia, ktorú očakávame, že intervalový strom bude robiť dosť rýchlo je zmena hodnoty prvku. Tu si treba uvedomiť, že nestačí zmeniť len samotný prvok, ale zmena ovplyvní aj všetkých jeho predkov. Pripájam tabuľku porovnania horných odhadov časových zložitostí operácií na poli a na intervalovom strome pre n prvkov:

štruktúra	zostrojenie	súčet intervalu	zmena prvku
pole	O(n)	O(n)	O(1)
intervaláč	O(2n)≅O(n)	O($\log_2$ n)	O($\log_2$ n)

Z toho teda vyplýva, že si treba zvážiť, ktorú dátovú štruktúru použijeme v závislosti od toho, či očakávame viac požiadaviek na zmenu alebo na súčet. Ak prevažujú tie na súčet, z hľadiska časovej zložitosti je intervaláč lepšia voľba.

Konštukcia intervalového stromu

Úplne na začiatku treba intervalový strom vytvoriť. Sú dva spôsoby ako reprezentovať úplný binárny strom v pamäti. Buď v poli, kde na základe indexov platia vzťahy medzi otcom a synom. Alebo objektovo orientovane. Ja som sa rozhodla pre tento objektovo orientovaný prístup, takže môj intervalový strom je trieda, ktorá má podtriedu vrchol.

V oboch prípadoch je potrebné ako prvú vec zistiť, aká je najbližšia väčšia mocnina dvojky k počtu prvkov ktoré by mal intervalový strom uchovávať. To koľká mocnina to je určí, koľko vrstiev, resp akú hĺbku bude mať náš strom. Na základe toho budeme potom vieme alokovať veľkosť poľa, resp. počet vrcholov stromu ktoré je treba vytvoriť.

Následne je potrebné vytvoriť stromovú štruktúru, do vrcholov zapísať, aké listy sa nachádzajú v ich podstrome, teda ich rozsahy a do príslušných listov priradiť hodnoty vstupných prvkov. Takto sa rekurzívne dostaneme do listov a upravíme ich hodnotu. Následne, aby sme dostali korektný intervalový strom, pri vynáraní z rekurzie v každom vrchole upravíme hodnotu, a to tak, že do nej priradíme súčet hodnoty ľavého a pravého syna. Takto teda po vynorení až do koreňa dostaneme správne vyplnený intervalový strom.

Typy queries

Query je v podstate nejaká požiadavka alebo otázka, ktorú pošleme dátovej štruktúre a ona na základe nej niečo vykoná. V našom prípade sme už spomenuli 2, a to **súčet** a **zmenu**.

Zmena

Ak chceme zmeniť prvok v intervalovom strome, musíme ho identifikovať jeho indexom v poli nad ktorým je strom vytvorený. Ďalej musíme určiť, aká má byť nová hodnota na tomto mieste. Potom sa funkcia na zmenu zavolá na koreň stromu a zistí, kam sa má volať, či je list ktorý má meniť v ľavom alebo pravom podstrome. Následne sa na niektorý tento podstrom rekurzívne zavolá a v novom vrchole zase zisťuje, či sa má volať naľavo, či napravo. Takto prejde až k správnemu listu, v ňom zmení hodnotu na požadovanú a následne, keď sa vynára z rekurzie opravuje aj hodnoty jeho predkov, ktorých hodnota určite záleží od hodnoty listu, takže sa zmení. Toto bude pre každú požiadavku trvať asymptoticky O($\log_2$ n) času, pretože ak máme n prvkov v poli, hĺbka binárneho stromu s n listami bude najviac $\log_2$ 2n, a pri tejto query v každej hĺbke prejdeme práve jedným vrcholom, teda počet vrcholov ktoré prejdeme je logaritmický.

Súčet

Pri tejto query musíme zadať ľavý a pravý okraj intervalu, ktorého súčet chceme zistiť. Môj intervaláč funguje tak, že interval zadávame ako polouzavretý. Začiatočný prvok do súčtu patrí, koncový už nie. V súlade s týmto je aj anotácia rozsahov vo vrcholoch.
Na zisťovanie súčtu znova použijem rekurzívnu funkciu, ktorú zavolám na koreň stromu. Keď som práve v nejakom vrchole, funkcia skontroluje

- Či je rozsah vo vrchole mimo intervalu ktorý hľadám - vtedy nemá vrchol čo k súčtu pridať a returnuje sa 0 tzn. do výsledného súčtu neprispieva ničím
- Či je rozsah vo vrchole celý súčasťou rozsahu čo hľadám - vtedy returnujem hodnotu vrchola, keďže tá zastupuje nejaký úsek výsledného súčtu (toto je to miesto kde ušetrím čas prechádzania každého vrcholu v úseku)
- V inom prípade to znamená, že vo vrchole sa nachádza časť toho čo hľadám a časť toho, čo nie, inými slovami, prechádza cez rozsah vrcholu ľavý alebo pravý okraj intervalu, ktorý hľadám. V takom prípade sa delegujem s rovnakou požiadavkou na synov vrchola, lebo tí spravujú len určitú časť, čo môže vyhovovať zadanému rozsahu presne.

No a čo s tým časom, ktorý to zaberie, ako sme vo vyššieuvedenej tabuľke prišli na to, že je to logaritmus? Je to preto, že v v každej vrstve(hĺbke) skúmam vždy najviac 4 vrcholy a počet vrstiev je ($\log_2$ 2n). Najviac 4 vrcholy preto, lebo z vrstvy o 1 vyššie sa môžem delegovať najviac z 2 vrcholov, pretože rozsah má práve 2 okraje.

Lazy zmena

Ako už samotný názov projektu napovedá, intervalový strom som rozšírila o lazy loading. To znamená, že je schopný poskytnúť aj tretí typ query, ktorý som nazvala lazy zmena. Lazy zmena je query, ktorá dokáže veľmi rýchlo zmeniť nielen jeden prvok pôvodného poľa, ale niekoľko prvkov súvisle vedľa seba. Inak povedané, dokáže zmeniť všetky prvky na nejakom intervale. Vie to urobiť v (asymptotickom) čase O($\log_2$ n). V mojom prípade lazy zmena funguje tak, že ku každému listu v danom rozsahu pripočíta zadanú hodnotu lazy. Samozrejme, aj táto funkcia sa dá prispôsobiť podľa toho na čo ju chceme využiť. Môže napríklad odčítavať konštantu, násobiť konštantou, zvyšovať prvky o 1...
Funguje to tak, že každý vrchol si okrem svojej hodnoty a rozsahu pamätá aj hodnotu v premennej lazy. Počiatočný stav lazy v každom vrchole je 0, ale môže sa zmeniť zavolaním tejto query. Ak zavoláme lazy zmenu na strom, správanie bude veľmi podobné ako pri hľadaní súčtu. V každom vrchole sa pýtame:

- Je rozsah vo vrchole mimo intervalu ktorý hľadám - nič sa nemení, vrátim sa
- Je rozsah vo vrchole celý súčasťou rozsahu čo hľadám - vtedy zmením svoju hodnotu lazy na tú, ktorú udáva query. A zároveň si updatnem svoju hodnotu tak, že pripočítam počet listov v mojom podstrome vynásobený hodnotou lazy v query. Takto som akoby zmenila celý rozsah listov v podstrome daného vrchola.
- V inom prípade to znamená, že rozsah vrcholu prechádza cez rozsah vrcholu ľavý alebo pravý okraj intervalu, ktorý mením. V takom prípade sa delegujem s rovnakou požiadavkou na synov vrchola, lebo tí spravujú len určitú časť, čo môže vyhovovať zadanému rozsahu presne.

Časovú zložitosť vypočítame úplne analogicky s dôkazom zložitosti pre query súčet tj. najviac 2 volania na každej úrovni, teda najviac 4 spracovávané vrcholy na jednej úrovni.
Ak pri niektorej query prechádzame vrcholom, v ktorom je hodnota lazy nenulová, a potrebujeme sa delegovať na jeho synov, pred delegovaním sa na synov zavolá funkcia, ktorá updatne ich hodnotu podľa lazy otca a nastaví im hodnotu lazy (platnú pre ich potomkov).

Ako to vizualizovať?

Ako rozumný spôsob vizualizácie som zvolila úplný binárny strom so statickým počtom využívaných vrcholov. Tieto vrcholy môžu meniť hodnotu, ale ich počet je nemenný. Aby bolo jasné, aká operácia je vizualizovaná a v ktorom vrchole sa práve niečo vykonáva a aj na krajšie ukázanie rekurzie a ľahšie pochopenie zložitosti operácií som zvolila vyznačovanie práve spracovávaného vrcholu orámovaním daného vrcholu. Informácie o jednotlivých orámovaniach, farbách a významoch sa nachádzajú v spodnej časti okna, v legende.

Prvý stĺpec legendy hovorí o podfarbení hodnôt vo vrchole, prvou farbou je podfabená aktuálna hodnota vrcholu -tá sa nachádza v hornej polovici vrcholu. Zároveň je ňou podfarbený rozsah indexov ktoré vrchol zastupuje. Toto sa nachádza v spodnej polovici štvorca reprezentujúceho vrchol. V prípade ak vo vrchole je hodnota lazy rozdielna od nuly, zobrazí sa táto hodnota vo vrchole a podfarbí sa druhou farbou v tomto prvom stĺpci. Napríklad takto:

Označenie vykonávanie operácií je označené orámovaním vrcholu príslušnou farbou podľa legendy. Napríklad takto:

Ako bolo popísané už vyššie, keď pracujeme s intervalovým stromom, chceme ho reprezentovať ako úplný binárny strom. Čo ale ak počet prvkov vo vstupnom poli nie je mocninou dvojky? Potom jednoducho bude existovať vrchol s nejakou neutrálnou hodnotou, ktorý nemôžeme meniť, nechceme sa doň rekurzívne volať a chceme zakázať posielať query s takýmto vrcholom vovnútri. Pre peknosť vizualizácie je teda vždy zobrazený úplný binárny strom a vrcholy ktoré nie sú používané majú šedé podfarbenie. Informáciu o tejto farbe znova nájdeme v legende.

Ovládanie

Pred spustením

Je niekoľko vecí, ktoré môžeme nastaviť ešte pred spustením programu. Informácie o týchto zmenách je možné nájsť aj v súbore s projektom v súbore readme.txt .

- **PRVKY**
  - pred spustením je možné zadať hodnoty prvkov s ktorými bude intervalový strom pracovať
  - prvky treba zapísať do súboru **input.txt**
  - počet prvkov musí byť menší, rovný 16, inak nebude možná prehľadná vizualizácia
  - prvky môžu byť ľubovoľné jedno a dvojciferné celé čísla
  - každý zadaný prvok musí byť na samostatnom riadku

- **RÝCHLOSŤ VIZUALIZÁCIE = DELAY**
  - delay hovorí o tom, ako rýchlo prebieha vizualizácia
  - delay sa dá nastaviť pred spustením vizualizácie v súbore **delay.txt**
  - odporúčaný delay je 500-800

Spustenie

Pre spustenie programu je potrebné spustiť súbor **projekt.py** pomocou nejakého Python interpeteru.

Po spustením

Po spustení sa zobrazí grafické okno, v ktorom okrem nakresleného intervalového stromu a legendy nájdeme aj ovládací panel. Tento sa nachádza v pravej časti.

Panel obsahuje 3 tlačidlá s menami jednotlivých queries, tlačidlá s číslami, tlačidlo ENTER na potvrdenie zadaného čísla. V hornej časti sa nachádza čierne pole v ktorom sú napísané doposiaľ nepotvrdené cifry tvoriace zadávané číslo (na obrázku číslo 12). Nad týmto panelom sa postupne zobrazujú **pokyny**, akú hodnotu treba zadať, a aké už boli zadané.

Zadávanie query:

1. stlačíme tlačidlo s názvom príslušnej query(nepotvrdzujeme ENTERom)
2. nad panelom sa zobrazí nápis "Vybraná query: [meno tej query]" a pokyn k zadaniu prvého parametra
3. zadávame postupne číslice prvého parametra, na konci potvrdíme ENTERom
4. nad panelom sa znova zobrazí oznam o hodnote prvého parametra a o tom, že máme zadať druhý
5. rovnakým spôsobom zadáme a potvrdíme druhý parameter
6. ak query vyžaduje len 2 parametre, začne sa automaticky vykonávať
7. ak na query treba aj tretí parameter, zadáme ho rovnako a následne sa vykoná
8. ak zadaná query bola suma, po skončení vizualizácie sa nad panelom zobrazí jej výsledok

Ukončenie

Grafické okno sa ukončuje zavretím samotného okna.

Po ukončení

Po ukončení programu si môžeme pozrieť históriu vykonaných queries. Nájdeme ju v súbore **output.txt** ktorý sa nachádza v priečinku s projektom.
