Cvičenie 2

Príklad 2.1.

Máme permutáciu n prvkov uloženú v poli A. Inverziou nazveme takú dvojicu indexov i a j, že $i < j$ a $A[i] > A[j]$. Spočítajte počet inverzií tejto permutácie.

Náčrt riešenia 2.1.

Postupne prechádzame pole a pýtame sa: koľko čísel väčších ako aktuálny prvok sme už predtým stretli? Počet inverzií je súčet týchto odpovedí.

Aby sme vedeli na otázky odpovedať v $O(\log n)$, vytvoríme nové pole, v ktorom na x-tej pozícii budeme mať 1, ak sme už stretli číslo x, 0 v opačnom prípade. Potom, keď chceme vedieť, koľko prvkov je väčších ako y, spýtame sa na súčet intervalu $(x, n+1)$.

Ak nemáme radi intervalové stromy, dala sa táto úloha vyriešiť aj upraveným algoritmom merge-sort. Vždy, keď vo funkcii merge presúvame prvok z druhej polovice, zvýšime počítadlo inverzií o toľko, koľko prvkov ešte ostalo v prvej polovici (pretože toľko väčších čísel prvok práve predbehol).

Pre čitateľov so stále neuhasenou túžbou po rýchlejšom algoritme odporúčam článok od Chana a Pătraşcua *Counting inversions, offline orthogonal range counting, and related problems.* (2010), napríklad tu: https://people.csail.mit.edu/mip/papers/invs/paper.pdf.

Príklad 2.2.

Máme tabuľu, ktorá má H riadkov, každý so šírkou W. Postupne lepíme na túto tabuľu oznamy. Každý oznam je papierik s výškou 1 a šírkou w_i. Pre každý oznam zistite, do ktorého najvyššieho riadku ho vieme nalepiť?

Oznamy sa nemôžu navzájom prekrývať a do daného riadku ho vždy lepíme čo najviac vľavo. Na začiatku máme teda v každom riadku W voľného miesta, keď do neho nalepím papierik s dĺžkou w_i, ostane už len $W - w_i$ miesta.

Náčrt riešenia 2.2.

Pamätáme si pre každý riadok, koľko máme voľného miesta. Nad týmto postavíme maximový intervalový strom.

Ak chceme nájsť prvý riadok, v ktorom je aspoň x miesta, začneme v koreni intervalového stromu. Následne putujeme naprieč stromom až ku listom. Vždy, keď je v ľavom synovi hodnota aspoň x, presunieme sa do ľavého syna. Inak sa presunieme do pravého syna.

List v ktorom skončíme je naľavejší list s hodnotou aspoň x.

Príklad 2.3.

Máme postupnosť n jednotiek a núl. Postupne máme spracovávať dva druhy operácií:

- v intervale (z, k) zmeň všetky jednotky na nuly a naopak
- povedz dĺžku najdlhšej neklesajúcej podpostupnosti (nie nutne súvislej) – teda vyber čo najviac prvkov tak, že najskôr vyberáš iba 0 a potom iba 1

Náčrt riešenia 2.3.

Prvý, možno jediný neintuitívny krok je povedať si, že chceme použiť intervalový strom. Ani to však nie je až tak neintuitívne, ak vieme, že chceme spracovávať nejaké intervaly. Od tohto momentu už vieme riešenie získať čisto analyticky.

Čo by sme si museli pamätať vo vrcholoch, aby sme vedeli odpovedať na druhú otázku? Tak mohli by sme si napríklad v každom vrchole pamätať dĺžku najdlhšej neklesajúcej podpostupnosti, ktorá sa nachádza v príslušnom podstrome. Túto hodnotu si označme v_{01}. Ak by sme si pamätali toto, odpoveď na druhú otázku je hodnota tejto premennej pre koreň.

Vieme však túto hodnotu efektívne spočítať z rovnakých hodnôt, ktoré patria synom príslušného vrcholu? Odpoveď je, že nie. Potrebujeme na to niečo pridať. Napríklad, počet jednotiek v podstrome (v_1) a počet núl v podstrome (v_0). Potom ak máme vrchol v, u je jeho ľavý podstrom a w jeho pravý podstrom, tak ľahko zistíme, že $v_{01} = \max(u_0 + w_{01}, u_{01} + w_1)$. A hodnoty v_0 a v_1 vieme rátať ešte jednoduchšie. Máme teda vyriešenú druhú časť úlohy.

K tomu však potrebujeme pridať aj úpravu intervalu. Tú budeme chcieť samozrejme robiť cez lazy-loading operáciu. Pozrime sa teda, či spĺňame potrebné podmienky. Skladať dokopy dve takéto operácie je ľahké. Ak chcem flipnúť nejaký podstrom a potom ho chcem flipnúť znova, tak s ním v podstate nemusím nič robiť. Väčší problém je, či vieme upraviť hodnoty v_0, v_1 a v_{01} dostatočne rýchlo, ak nám prišla daná úprava, teda všetko v podstrome sa otočí.

Hodnoty v_0 a v_1 sa iba vymenia. Čo však s hodnotou v_{01}? Ani toto však nie je taká veľká prekážka. Stačí, že si budeme pamätať a počítať aj dĺžku najdlhšej nerastúcej podpostupnosti (teda najskôr jednotky a potom nuly) – hodnota v_{10}. Túto hodnotu vieme počítať obdobne ako hodnotu v_{01} a pri prevrátení podstromu sa tieto dve hodnoty jednoducho vymenia.

V tomto momente platia všetky potrebné vlastnosti, ktoré majú naše funkcie spĺňať a môžeme takéto riešenie použiť.

Príklad 2.4.

Máme zadaný zakorenený strom s n vrcholmi. Každý vrchol má priradené jedno celé číslo. Postupne musíme spracovávať nasledovné operácie:

1. Zvýš číslo vo vrchole u o y.
2. Ak je priemerná hodnota čísla v podstrome s koreňom u menšia ako x, zvýš všetky čísla v tomto podstrome o y.
3. Nastav hodnotu každého prvku v podstrome s koreňom u na minimálnu hodnotu spomedzi čísel v tomto podstrome.
4. Povedz číslo vo vrchole u.

Náčrt riešenia 2.4.

Vrcholy stromu vieme očíslovať číslami 1 až n v takzvanom postorder poradí – najprv priradíme čísla ľavému podstromu, potom pravému podstromu a nakoniec koreňu. Číslo i-teho vrchola označíme `Out[i]`. Navyše si každý vrchol vie pamätať najmenšie číslo v jeho podstrome (najvyššie má on sám), ktoré označíme `In[i]`. Pri tomto postorder číslovaní majú vrcholy podstromu s koreňom v postupne čísla `In[v]` až `Out[v]` – čiže každému podstromu zodpovedá súvislý interval.

Potom všetky operácie typu urob niečo s podstromom v a zisti niečo o podstrome vieme robiť ako operácie urob niečo s intervalom `In[v]..Out[v]` a zisti niečo o intervale `In[v]..Out[v]`. Na to vieme použiť intervalový strom s príslušnými lazy-loadingovými flagmi.

Všimnite si tiež, že v našom prípade potrebujeme robiť dva rôzne typy updatov, rozmyslite si, že tieto lazy flagy, ktoré im zodpovedajú sa dajú v pohode skladať dokopy a preto platia všetky potrebné podmienky.