Univerzitetni programerski maraton 2026
1. kolo — rešitve nalog

Tomaž Hočevar
tomaz.hocevar@fri.uni-lj.si

19. februar 2026

Lego stolp

Za izgradnjo stolpa moramo uporabiti vse kocke, odločiti se moramo samo, kako bomo obrnili katero izmed njih. Vrstni red kock ni pomemben. Ko se odločimo za orientacije posameznih kock, je dimenzija škatle definirana z največjo širino in največjo globino orientirane kocke. Naj bo X največja stranica katerekoli izmed kock (največja izmed daljših stranic kocke). Ta vrednost bo predstavljala širino škatle, sedaj pa moramo poskrbeti za čim manjšo globino. Za vse preostale kocke bomo njihovo daljšo stranico poravnali v smeri širine škatle (ker ne bo povečala širine), njihova krajša pa bo prispevala h globini. Če bi jo orientirali drugače, bi h globini s svojo daljšo stranico prispevala kvečjemu več, prispevek k širini pa se ne bi spremenil. Globina škatle bo torej enaka največji izmed krajših stranic kock. Izračunati moramo torej največjo izmed daljših stranic ter največjo izmed krajših stranic kock.

Mediane

Za vsako število, ki ga lahko izbrišemo iz seznama x, moramo določiti mediano nastalega seznama. Zaradi velike količine števil si ne moremo privoščiti za vsako izbrisano število urejati seznama, da najdemo mediano. Hitro lahko opazimo, da bodo odgovori precej podobni. Če imamo veliko števil in izbrišemo največje ali drugo največje, bo mediana v obeh primerih enaka. Za začetek uredimo števila v seznamu naraščajoče. Mediana se bo nahajala na indeksu (indeksirano od 0 naprej) $m = \lceil \frac{n-1}{2} - 1 \rceil$. Če izbrišemo neko število na indeksu $i > m$, se vrednost v urejenem seznamu na indeksu m ne spremeni, torej bo v tem primeru mediana x_m. V nasprotnem primeru pa se na indeks m premakne število z indeksa $m + 1$, torej bo pri $i \leq m$ postalo mediana število x_{m+1}.

Konektorji

Podano imamo rekurzivno strukturo datotečnega sistema, ki ga moramo izpisati v primerni obliki. Glavno oviro predstavljajo črte, ki povezujejo objekte v istem imeniku. Brez tega bi lahko pri rekurzivnem sprehodu po datotečnem sistemu hranili globino in izpisali primerno število presledkov pri vsakem izpisu. Vidimo lahko, da navpične črte povezujejo vse objekte, ki se nahajajo neposredno znotraj imenika. Torej se raztezajo od začetka do zadnjega neposredno vsebovanega objekta, pred vsebino tega zadnjega objekta pa jih ni več. Namesto globine bi bilo koristno hraniti indikatorje, kateri objekti na trenutni poti so zadnji. Če ne gre za zadnji objekt, izpišemo zamik z vertikalno črto, sicer pa zgolj primeren zamik s presledki.

Kozmično sevanje

Za začetek vzpostavimo zaščiteno območje z radijem 0 na začetni in končni točki, da lahko obravnavamo samo premike med območji. Vsi premiki med območji bodo potekali vzdolž daljice med središči območij, ker to predstavlja najkrajšo pot izven zaščitenega področja. Omejimo se torej lahko na iskanje poti med središči območij, pri čemer lahko za vsak par območij izračunamo dolžine nezaščitene poti med njima. To pa je pravzaprav običajen problem najkrajše poti v grafu. Vendar gre za zelo gost graf, zato uporabimo Dijkstrov algoritem s časovno zahtevnostjo $O(n^2)$ in ne $O(e \log n)$ (kar je $O(n^2 \log n)$ v tako gostem grafu)!

Mastermind

Gre za interaktivno nalogo, pri kateri morate najti strategijo igranja igre Mastermind, da boste uganili rešitev najkasneje v petem poskusu. Ni povsem očitno, kakšna strategija bo dovolj dobra. Pripravite si program, ki bo izračunal število poskusov v najbolj neugodnem primeru za vašo strategijo. Tako boste lahko prepričani, da je postopek dovolj dober.

Reševanje začnemo z množico vseh možnih rešitev. Z vsako poizvedbo dobimo razbitje te množice na manjše podmnožice glede na različne odgovore na poizvedbo. Ena izmed strategij je tipa minimax, kjer v vsaki potezi izberemo tako vprašanje, da bo ne glede na odgovor množica veljavnih rešitev čim manjša - minimiziramo torej maksimalno velikost množice v razbitju. S tem se izogibamo situacijam, kjer bi z neugodnim odgovorom izvedeli zelo malo. Najboljše poizvedbe niso nujno iz množice možnih rešitev, temveč je treba upoštevati vse možne poizvedbe, saj so lahko dober kriterij za razbitje množice čeprav ne morejo biti rešitev. V kolikor obstaja več enakovrednih poizvedb, izberemo tako, ki bi lahko bila tudi ena od rešitev.

Ker imamo opravka z več testnimi primeri, ne moremo vedno znova iskati optimalne poizvedbe. Vnaprej si lahko shranimo prvo potezo, ki je vedno enaka. Glede na enega izmed približno 15 odgovorov, si lahko shranimo tudi optimalno drugo potezo. Pravzaprav si lahko shranimo celotno drevo, ker je število možnih odgovorov (število črnih in belih žebljičkov) dokaj majhno, višina pravilnega drevesa pa tudi ne presega 5.

Velika združitev

Podan imamo seznam števil, ki jih združujemo v vedno večje komponente oz. barvamo z isto barvo. Pri tem ne barvamo strnjenih delov, temveč ločene vrednosti, ki so definirane s permutacijo. Strnjeni deli permutacije definirajo števila, ki jih moramo pobarvati enako v originalnem seznamu. Ob tem moramo odgovarjati na poizvedbe, koliko različnih barv se nahaja na danem strnjenem intervalu originalnega seznama. Izkaže se, da je presenetljivo težko najti rešitev, ki bi bila bolj učinkovita od naivnega združevanja in poizvedb z linearno časovno zahtevnostjo posamezne operacije.

Komponente bomo hranili z vpeljavo povezav med zaporednimi elementi v urejeni komponenti. To nam omogoči učinkovito izvajanje poizvedb o številu različnih barv. Odgovor je namreč enak številu elementov na intervalu, ki mu odštejemo število povezav, ki se v celoti nahajajo znotraj intervala. Če povezave predstavimo kot točke, se problem prevede na poizvedbe na pravokotnem območju (2D range query), kar lahko rešimo z 2D drevesno strukturo v $O(\log^2 n)$ na poizvedbo.

Načrtovati moramo še učinkovit način vzdrževanje te strukture s povezavami (ki opisuje barvne komponente) ob združevanje komponent glede na podano permutacijo. V permutaciji se lahko sprehodimo od začetka do konca intervala in pridobimo seznam barv, ki jih je treba združiti. Pri tem pa moramo vsako barvno komponento obravnavati enkrat in ne vseh njenih elementov, da ne dobimo rešitve s kvadratno časovno zahtevnostjo. To lahko dosežemo tako, da v permutaciji hranimo kazalce, ki nam omogočajo skočiti neposredno na konec intervala enake barve.

Seznam barv bomo združili tako, da bomo manjše barvne komponente združevali v največjo barvno komponento. Tako bo vsako število del kvečjemu $O(\log n)$ vstavljanj v večjo komponento. Za vsako barvo hranimo urejen seznam števil v drevesni strukturi. Sprehodimo se čez manjšo komponento in vstavljamo števila v večjo. Pri tem odstranimo povezavo iz manjše komponente ter zamenjamo povezavo v večji komponenti z dvema novima, ki sta posledica ravnokar vstavljenega/vrinjenega števila. Vsaka taka operacija ima časovno zahtevnost $O(\log^2 n)$, skupaj pa jih izvedemo največ $O(n \log n)$, da se vse združi v eno komponento.

Časovna zahtevnost vseh operacij združevanja je torej $O(n \log^3 n)$. Med tem pa imamo opravka še s poizvedbami, ki zahtevajo $O(\log^2 n)$ časa na poizvedbo. Faktor $\log^3 n$ je precej velik, zato potrebujete učinkovito implementacijo postopka, da je ta izrazito učinkovitejši od naivnega reševanja.