1. Na rozmýšľanie (12 bodov)

Pozor na špecifikovanie operácie intervaláča

Keď používate nejakú dátovú štruktúru, tak je dôležité správne uviesť jej predpoklady. Pre riešenie úloh a), b) nestačí napísať: „Použijeme intervalový strom, kde vo vrcholoch budú 0 / 1 reprezentujúce zatvorenú / otvorenú pílu. A vo vrcholoch bude index najľavejšej otvorenej píly na intervale. Pri otvorení / zatvorení píly len zmeníme 0 / 1 v príslušnom liste.“ Nemáme totiž vysvetlené, čo sa deje so stromom pri zmene hodnoty v liste. Jednou možnosťou je celý proces zmeny opísať ako bol na prednáške. Avšak algoritmy z prednášky môžeme využiť. K tomu nám chýba podstatná informácia – špecifikovať, akú operáciu pri intervalovom strome používame. V materiáli https://ksp.sk/kucharka/intervalovy_strom/ je intervalový strom vysvetlený na súčet. Pokiaľ chceme používať inú operáciu, treba to špecifikovať.

Napríklad riešenie pre prvé dve operácie by stačilo v podobe: „Použijeme súčtový intervalový strom nad poľom 0 (zatvorená) a 1 (otvorená píla). Otvorenie a zatvorenie píly zodpovedá priamo týmto operáciám.“ Dokonca za plný počet by sme uznali riešenie:

Použijeme minimový intervalový strom nad poľom p dĺžky $n+1$, kde $p[i] = \infty$ pre zatvorenú pílu na kilometri i a $p[i] = i$ pre otvorenú pílu. Otvorenie a zatvorenie píly teda simulujeme operáciou zmeny listu v intervalovom strome. Ak sa spustí drevo na kilometer x, tak vrátime minimum intervalu $[x, n]$. Všetko sú bežné operácie s intervalovým stromom, ktoré majú zložitosť $O(\log n) + O(n)$ inicializácia.

Treba si dať však pozor, že výsledok operácie pre interval sa musí dať zrekonštruovať z výsledkov pre dva podintervaly. Napr. v úlohe 2.4 nemôžeme použiť intervaláč s operáciou „vráť pre interval dĺžku najdlhšej neklesajúcej postupnosti“.

Pozor na určenie časovej zložitosti

Väčšina z vás používala maximový (resp. s operáciou or) intervalový strom nad poľom 0 / 1 (otvorená / zatvorená píla). Uvedieme jeden z algoritmov pre tretiu otázku. Zamyslite sa, či je správny a akú má zložitosť.

Zmeny píl simulujeme zmenami prvkov v poli. Keď spustíme drevo na kilometer x, tak na koreň zavoláme nasledovnú funkciu.

1. Ak som v liste s pílou, tak vrátim jeho index. Ak v liste nie je píla, tak vrátim -1 (píla neexistuje).
2. Ak je v intervale ľavého syna otvorená píla (hodnota tohto syna je 1) a zároveň celý tento interval nie je naľavo od x, tak sa zavoláme na ľavého syna.
3. Pokiaľ ešte nemáme výsledok, tak sa zavoláme na pravého syna.

Logika je správna. Výsledok pre strom totiž nájdeme buď v ľavom podstrome, a to len keď tam je píla a nie je celý pred x; alebo v pravom podstrome. Problémom je časová zložitosť. Tento program totiž nemá zložitosť $O(\log n)$. Ak sú otvorené len píly 0 a n, tak program stále volá doľava až do spoločného rodiča vrcholov 0 a 1. Tam sa zavolá len na pravého syna a vráti -1. Toto sa deje potom z každého vrchola na ceste do koreňa – program sa volá stále doprava až do syna, čo robí

$$1 + 2 + \cdots + \log n = O((\log n)^2) \text{ operácií.}$$

Keď budeme menej šikovní, tak môžeme skončiť aj s lineárnym riešením, ktoré prehľadá celý strom. Na to si treba dať pozor.

Okrem toho v takýchto riešeniach treba ich časovú zložitosť odôvodniť.

Jedným zo správnych riešením tohto typu je nasledovné:

1. Ak sme v liste, vrátime jeho index.
2. Inak sa zavoláme na ľavého syna, ak to má zmysel. Ak dostaneme výsledok, tak ho vrátime, inak sa ešte zavoláme na pravého syna, ak to má zmysel, a vrátim výsledok z neho. Pod „ak to má zmysel“ myslíme, že syn má hodnotu 1 (v jeho podstrome je píla) a nie je pravda, že jeho podinterval je celý naľavo od x.

Časovú zložitosť vieme odhadnúť nasledovne. Algoritmus ide z koreňa po ceste k listu x. Niekde na tejto ceste (najskôr v liste x) vráti, že píla neexistuje. Táto fáza zaberie $O(\log n)$ času (taká je hĺbka stromu). Potom sa rekurzívne vynára – tiež $O(\log n)$, a to až do momentu, keď vyjdeme z ľavého syna a zavoláme sa do pravého. V tejto situácii pravýpodinterval musí byť celý za x a musí obsahovať pílu, keďže sa doňho voláme. Algoritmus teda nájde najľavejšiu pílu postupným sa vnáraním, čo tiež zaberie $O(\log n)$ času. Každá z týchto troch fáz má logaritmickú zložitosť, preto celý tento proces má tiež zložitosť $O(\log n)$.

Lazy loading

Viacerí ste v riešení spomínali, že ho možno zrýchliť lazy loadingom. Vo väčšine prípadov však išlo o nezmysel, ktorý odhaľoval, že intervalovým stromom a lazy loadingu nerozumiete. V štandardných riešeniach tejto úlohy totiž lazy loading nemá zmysel. Ten vieme použiť vtedy, keď robíme zmenu intervalu. Zadanie úlohy od nás vyžaduje zmeny len jedného miesta v poli a zodpovedanie otázky pre interval.

Avšak možno ho využiť pri nasledovnom riešení. Pre každý kilometer si budeme pamätať index najľavejšej a najpravejšej otvorenej píly a nad týmto poľom si vytvoríme intervalový strom. Otvorenie píly vieme simulovať zmenením týchto hodnôt na vhodnom intervale, podobne aj zatvorenie. Tieto operácie budeme robiť lazy. Keď spustíme drevo na kilometri x, tak odpoveď máme na indexe i (resp. tam bude, keď tam zídeme z koreňa). Detaily riešenia nechávame na vás (zložitosť, akú operáciu robíme v tomto strome, ...).

2. Na teoretické vedomosti (18 bodov)

V prvej časti tejto úlohy bolo treba popísať Dijkstrov algoritmus tak, ako sme ho mali na prednáške. Tu je veľmi dôležité, aby ten popis bol jasný a presne definovaný.

Veľa riešení spomenulo *spracované* a *nespracované* vrcholy. Nikde však nedefinovalo, čo to presne znamená. To totiž nie sú jasne definované slová alebo nejaké ustálené výrazy. Je to niečo, čo nám pomáha sa vyjadrovať zrozumiteľnejšie a jasnejšie, najprv ich však musíme mať definované.

Bolo teda dobré začať tým, že definujeme: *Spracované* vrcholy voláme tie, pre ktoré poznáme najkratšiu cestu, zo začiatku do týchto vrcholov.

Podobne aj v zvyšku popisu bolo dôležité definovať veci čo používame. Viacerí spomenuli haldu, ale nepopísali aké hodnoty do nej ukladajú, kedy presne ich tam vkladajú, ako vypočítajú hodnoty, ktoré tam patria. To všetko sú veci, ktoré si ako opravovateľ nemôžem domýšľať, ale musím vedieť vyčítať z textu.

Rovnako, viacerí ste zabudli vysvetliť, čo sa deje s vrcholmi, ktoré sú v halde viackrát.

Pri popise riešení sa teda snažte, aby bolo presne definované s akými hodnotami pracujete, ako ich počítate a čo s nimi ďalej robíte.

V druhej časti úlohy bolo treba vysvetliť, prečo Dijkstrov algoritmus nefunguje na grafe, v ktorom sú záporné hrany.

V riešeniach sa často spomínali existencie záporných cyklov. To samo o sebe ale nič neznamená, algoritmus by predsa mohol byť korektný aj pri ich existencii. A to presne bolo treba spraviť. Ukázať, že algoritmus korektný nebude, ak sa tam cykly vyskytnú.

K tomu sa dalo pristúpiť dvoma spôsobmi. V prvom stačilo vytvoriť nejaký príklad grafu so zápornými hranami a popísať, ako na ňom bude bežať Dijkstrov algoritmus (popísaný v predchádzajúcej úlohe) a že výsledné hodnoty naozaj budú spočítané zle. Tu je veľmi dôležitá práve tá druhá časť, ukázať, že algoritmus naozaj nefunguje. Bez toho si totiž iba môžeme domýšľať, či algoritmus zlyhá a ako veľmi.

Druhý spôsob riešenia bolo postupovať všeobecne, ukázať dôkaz správnosti Dijkstrovho algoritmu (teda, že nespracovaný vrchol s najkratšou vzdialenosťou od začiatku môžeme prehlásiť za spracovaný) a ukázať, kde tento dôkaz zlyhá kvôli zápornej hrane. Toto tvrdenie sa totiž spolieha na to, že ak mám aktuálne najkratšiu vzdialenosť, tá sa už skrátiť nemôže, to sa však využitím zápornej hrany stať môže.

Riešenei poslednej podúlohy vyžadovalo vytvoriť si nový graf. V tomto grafe jednotlivé vrcholy reprezentujú stav „v ktorom vrchole sa nachádzam a akej farby bola posledná hrana, ktorou som prešiel“. Medzi takýmito stavmi sa ľahko dajú pridať hrany vychádzajúc z hrán pôvodného grafu – ak vedie modrá hrana medzi v a w, tak v novom grafe bude hrana medzi $(v, cervena)$ a $(w, modra)$ a tiež hrana medzi $(w, cervena)$ a $(v, modra)$ (nové hrany sú orientované).

Ľubovoľná cesta v takomto grafe zaručí, že nepôjdu dve hrany rovnakej farby za sebou, a preto môžeme použiť Dijkstrov algoritmus.

Veľa z vás správne identifikovalo práve takéto riešenie. Opäť však bol problém s jasnosťou a presnosťou definácie. Buď nebol úplne presne definovaný stav, ktorý vrcholy reprezentujú, alebo ste nepopísali akým spôsobom upravíte hrany grafu. Opäť sa teda opakuje, že v popisoch treba byť čo najpresnejší a poriadne popísať všetky kľúčové časti.

Menšia chyba bola, ak človek nedodefinoval odkiaľ sa hľadá výsledná najkratšia cesta, prípadne do ktorého vrcholu má viesť.

3. Na programovanie (5 bodov)

Obe zadané funkcie boli z tých jednoduchších. Napriek tomu si treba dať pri implementácii pozor na drobné chyby, ktoré vedia výrazne ovplyvniť správnosť alebo efektivitu riešení.

Pri úlohe s umocňovaním bol chybný najčastejšie takto vyzerajúci program:

```
def umocni(a, b, m):
    if b == 0:
        return 1
    medzivysledok = umocni(a, b//2, m) * umocni(a, b //2, m)
    if b % 2 == 1:
        return (a * medzivysledok) % m
    return medzivysledok % m
```

Tento program vyzerá správne a dokonca píše správne hodnoty. Nie je však dosť efektívny. Problémom je, že `umocni(a, b//2, m)` volá dvakrát. Tým pádom ale stratí všetku efektivitu, lebo dookola pomaly prepočítava tú istú hodnotu v dvoch samostatných volaniach.

Správne riešenie si ako medzivýsledok poznačí hodnotu `umocni(a, b//2, m)` a keď ju chce využiť dvakrát, použije túto vypočítanú hodnotu, nevolá sa opäť rekurzívne.

Pri geometrickej funkcii bolo najväčším zdrojom chýb, ak ste sa rozhodli nepoužiť klasické vektorové násobenie, ktoré bolo na prednáškach. A snažili ste sa to spočítať pomocou uhlov alebo predpisov priamok.

Hoci takéto riešenia vedia viesť k dobrému riešeniu, je oveľa ľahšie v nich spraviť chybu alebo zabudnúť na nejaký príklad. Najčastejšie, priamky ktoré sú kolmé na os x nemajú definovaný predpis (lebo to nie sú korektné funkcie), prípadne pri počítaní tohto predpisu sa delí 0, čo v programe spôsobilo chybu.

Na prednáškach si ukazujeme klasické algoritmy a riešenia práve preto, že občas nechceme znova vymýšľať koleso, ale využiť osvedčené riešenie.