Korespondenční Seminář z Programování

38. ročník | **KSP** | **Listopad 2025**

Vzorová řešení první série třicátého osmého ročníku KSP

38-1-1 One-shot burza

Základní algoritmus

Pojďme se nejdřív podívat na to, co po nás vlastně zadání chce: Chceme získat dvojici pozic v poli, která maximalizuje jistou funkci – Plesnivousův výdělek. Konkrétně, taková funkce by pro pozice i, j a pole p vypadala takto: $f(i,j) = \lfloor K/p[i] \rfloor \cdot (p[j] - p[i])$. (Nakoupíme $\lfloor K/p[i] \rfloor$ kusů uhlí za cenu $p[i]$ a pak ho prodáme za cenu $p[j]$.)

Nabízí se tuto funkci prostě spočítat pro všechny dvojice i, j a z nich si průběžně pamatovat minimum. Takový algoritmus by běžel v $\mathcal{O}(N^2)$. S tím se přeci nemůžeme spokojit, pojďme to zlepšit.

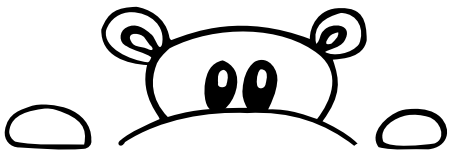
Zrychlujeme

Pojďme si teď zafixovat den nákupu i. Kdy se nám vyplatí prodat? Prodávat chceme za maximální cenu, to dá rozum, jenže hledat pro každé i maximální j by nám asymptotickou časovou složitost nezlepšilo. Proto si na tento úkol povoláme datovou strukturu, která nám pro každou pozici řekne, jaké je po ní ve zbytku pole maximum. Ti z vás, kteří se k tomuto nedostali: schválně si zkuste takovou strukturu teď vymyslet, než budete číst dál.

Taková datová struktura není vůbec složitá – zkonstruujeme ji postupným průchodem dnů odzadu. Budeme ji například reprezentovat polem, pojmenujme ho M. Víme, že za posledním prvkem už žádné maximum není. Také víme, že za předposledním prvkem je maximum po něm ten poslední. Dál jednoduše pro každý prvek uložíme ten větší z $M[i+1]$ a $p[i+1]$ – maximum za i-tým prvkem je buď stejné jako maximum za $(i+1)$-tým prvkem, nebo je to $(i+1)$-tý prvek. Pro účel výstupu si nebudeme pamatovat pouze hodnotu maxima, ale i index dne, ve kterém se hodnota nabývá. Všimněte si, že je to vlastně analogický postup hledání jednoho maxima pro danou množinu prvků – pouze si průběžné výsledky zapisujeme do pole. Zkonstruované datové struktuře se říká *suffixová maxima*.

Díky suffixovým maximům pro každou pozici známe maximální cenu uhlí pro prodej a den, kdy jí nabývá, v konstantním čase. Teď nám stačí jednoduše projít všechny dny, zkusit kolik bychom maximálně vydělali, pokud bychom ten den uhlí nakoupili a pamatovat si maximální takovou dvojici dní. Specificky pro všechny i spočítáme funkci $g(i) = \lfloor K/p[i] \rfloor \cdot (M[i] - p[i])$ a pamatujeme si maximální dvojici i, j. Časová složitost je $\mathcal{O}(N)$, jelikož nejdříve postavíme suffixová maxima v lineárním čase a poté pro všechny prvky spočítáme nějakou funkci v konstantním čase a vrátíme maximální výsledek.

Program (C++):
`http://ksp.mff.cuni.cz/viz/38-1-1-tp.cpp`



Ještě lépe?

Pojďme si ještě ukázat jeden algoritmus, který běží také v $\mathcal{O}(N)$, ale má k tomu další příjemné vlastnosti. První pojďme obrátit náš přístup k nakupování ve fixní den a zjištění nejlepšího dne, kdy prodat. Co kdybychom pro fixní index j hledali den, kdy se nejvíc vyplatilo nakoupit, abychom v den j prodali? Teď bychom zase využili *prefixová minima* – analogicky k suffixovým maximům si akorát budeme pamatovat minimum z předchozích prvků místo maximum z nadcházejících, pojmenujme je m; Zkonstruujeme je v podstatě stejně – pro první prvek není žádný předchozí menší, pro druhý je menší ten první a pro každý další je hledané minimum to menší z $m[i-1]$ a $p[i-1]$.

Teď si všimněme, že tímto způsobem si nemusíme prvky ukládat do pole. V každém kroku nás zajímají akorát 3 věci: aktuální prvek, minimum z předchozích prvků a průběžný maximální interval pro nákup a prodej. Tedy si prefixová minima ani nemusíme ukládat – stačí si pamatovat to jedno průběžné minimum z předchozích prvků. Postupně čteme nové prvky a zároveň pro ně jak počítáme výdělek, kdybychom nakoupili v minimu a prodali právě teď, a zároveň průběžně přehodnocujeme dosavadní minimum.

Tomu se říká, že je algoritmus *online* – s tím jak dostáváme vstup, tak ho průběžně chroustáme a přepočítáváme výsledek, dokud se nedostaneme na konec, kde výsledek zase vyplivneme. Také to znamená, že jsme celý vstup prošli pouze jednou, což je dvakrát méně, než dvakrát! Zároveň si v průběhu celého algoritmu pamatujeme pouze průběžné minimum ze dní, maximum z dvojic (nákup, prodej) a aktuální prvek – to je konstantně mnoho paměti.

Program (C++):
`http://ksp.mff.cuni.cz/viz/38-1-1.cpp`

Úlohu připravili: Kristýna Petrlíková, Matúš Púll

38-1-2 Bambulus

Ukážeme si dvě řešení: jedno se ztrátou $\Theta(\sqrt{N})$, jedno s (optimální) ztrátou $\Theta(\log N)$. Pro jednoduchost si jednotlivé pilíře obarvíme a budeme místo o divácích vyznávajících první/druhý/třetí pilíř mluvit o červených, zelených a modrých divácích.

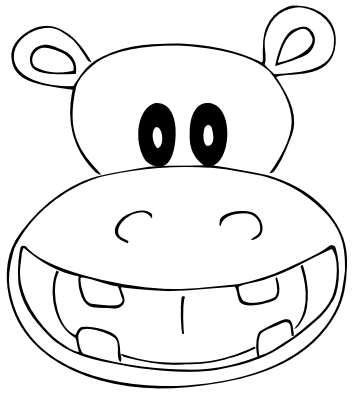
Rozdělujeme na bloky

Začneme tím, že si zvolíme parametr K a rozdělíme hlediště na *bloky* velikosti K. Každý blok bude vyhrazen pouze pro diváky jedné barvy a budeme ho plnit postupně zleva doprava. Poslední políčko každého bloku zůstává vždy volné, to aby spolu mohly libovolné dva bloky sousedit nezávisle na tom, jak vypadají uvnitř.

Algoritmus si bude v každém okamžiku udržovat seznam ještě nezaplněných bloků – nejvýše jeden pro každou barvu. Přijde-li nový divák, umístíme ho do nezaplněného bloku příslušné barvy. Pokud takový blok neexistuje (protože danou barvu vidíme poprvé, nebo protože minulý divák dokončil blok), tak si otevřeme nový. Pokud už nám došly volné bloky, končíme.

Jaký má tento algoritmus ztrátu? V okamžiku, kdy skončíme, existují nejvýše dva nezaplněné bloky, každý obsahuje nejvýše K volných míst. V každém z $\mathcal{O}(N/K)$ zaplněných bloků je vždy právě jedno volné místo. Celkem tedy máme $\mathcal{O}(K + N/K)$ volných míst.

Teď stačí zvolit takové K, aby tato hodnota byla co nejmenší. Optimální je mít K a N/K zhruba stejně velké, zvolíme tedy $K = \sqrt{N}$. Tím dostáváme, že volných míst je $\mathcal{O}(\sqrt{N})$.



Optimální řešení

Začneme s jednoduchým hladovým algoritmem, který diváky rozesazuje do hezky souvislých úseků a pak si rozmyslíme, jak ho volat opakovaně na postupně se zaplňujícím hledišti.

Náš hladový algoritmus funguje jednoduše: červené diváky umisťuje do hlediště zleva, modré zprava, a zelené zprostředka (prvního zeleného diváka umístí na pozici $\lfloor N/2 \rfloor$ a pak zelenou oblast rozšiřuje střídavě zleva a zprava). Těsně předtím, než se potkají oblasti různých barev, skončí.

Rozmysleme si, jak v tomto okamžiku vypadá hlediště. Jako D označíme počet umístěných diváků (tedy $N - D$ je počet volných míst). Bez újmy na obecnosti můžeme předpokládat, že se téměř potkali červení a zelení diváci. Jak vypadá hlediště? Zleva se skládá z nějak dlouhého červeného úseku, pak jednoho volného místa, pak zeleného úseku, pak volného úseku a nakonec z modrého úseku.

Nyní uvažme volný úsek mezi zelenými a modrými diváky. Odstraníme-li z něj obě nejkrajnější místa, zbyde nám úsek velikosti $N - D - 3$, který nesousedí s žádnými diváky. Teď přichází klíčové pozorování: takový volný úsek je vlastně takové malé hlediště velikosti $N - D - 3$. Můžeme tedy zapomenout zbytek hlediště a pustit na toto menší hlediště ten samý algoritmus! Ten opět po nějakém čase skončí a zbyde mu o něco menší volný úsek, ten zase zbavíme okrajů a prohlásíme za ještě menší jeviště, a tak dále, dokud velikost nového jeviště neklesne na nulu.

Zbývá zanalyzovat ztrátu tohoto algoritmu. Nejprve zříme, že jedna iterace algoritmu zmenší velikost hlediště aspoň na polovinu, konkrétně že $D \geq N/2-1$. To proto, že algoritmus skončí až teprve, když se dva úseky potkají, a to dříve než při $N/2 - 1$ divácích nemůže nastat. Proto bude mít menší hlediště velikost nejvýše $N - D - 3 \leq N/2 - 2 \leq N/2$. To znamená, že celkově bude nejvýše tolik iterací, kolikrát můžeme opakovaně půlit N, než se dostaneme na 1. Tedy iterací bude nejvýše $\log_2 N$.

Každá iterace algoritmu přispívá třemi místy k celkové ztrátě. Celková ztráta je tedy nejvýše $3 \log_2 N = \mathcal{O}(\log N)$.

Dvě poznámky na závěr:

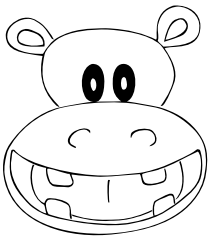
- V řešení jsme ignorovali okrajové případy, kdy má nějaký z úseků velikost nula. Dá se snadno rozmyslet, že to nevadí (ba dokonce v takových případech je pro nás situace příznivější).
- Náš algoritmus se dá upravit, aby místo tří políček ztrácel na každé hladině rekurze jen jedno. Asymptotická ztráta nicméně zůstává $\mathcal{O}(\log N)$.

Úlohu připravil: Říša Hladík

38-1-3 Mediánový poddaný

První myšlenka by byla třeba poslat posla do každého kraje pro pracovitost každého poddaného. Tedy do každého z K krajů udělat N požadavků pro poddané. Ze seznamu pracovitosti všech poddaných už zjistíme medián například seřazením, ale už neposíláme posla. Posla jsme přitom poslali $\Theta(N \cdot K)$-krát. To ale určitě zvládneme rychleji.

Intuitivní pohled na to by byl, že jakýkoliv výpočet máme zadarmo, ale transport dat nás něco stojí, takže se chceme vyhnout transportu všech dat. Pojďme si třeba říci, že si kraje své záznamy o poddaných setřídí. Teď budeme posílat dotazy na kraje a ptát se jich, kolik poddaných mají, kteří mají menší pracovitost než daný dotaz. Pokud se krajů zeptáme na nějakou pracovitost a jejich odpovědi se sečtou na $\lfloor N \cdot K/2 \rfloor$ a existuje poddaný s danou pracovitostí, tak jsme našli mediánového poddaného a jsme vysmátí.



Zbývá nám vymyslet, jak zjistit takovou prostřední pracovitost. V principu na to použijeme binární vyhledávání (o kterém máme speciální kuchařku),[1] konkrétně jeho pokročilejší verzi, kde máme nějaký predikát $P(x)$ a hledáme největší x, které ještě splňuje predikát. Uvažme pohled ze zmíněné kuchařky – za náš predikát $P(x)$ vezmeme, zda je v království počet poddaných s pracovitostí nejvýše x menší než $\lfloor N \cdot K/2 \rfloor$. Teď stačí s tímto predikátem binárně vyhledat nejvyšší hodnotu pracovitosti, která ještě splňuje predikát.

Zjevně taková hodnota musí být o 1 menší, než hodnota pracovitosti nějakého poddaného – kdyby nebyla, tak predikát splňuje ještě nějaká vyšší hodnota. Kdybychom se ale dostali až na toho poddaného napravo, tak by se počet poddaných s menší pracovitostí zvýšil o 1 a už nesplňujeme predikát. Za hledaného mediánového poddaného tedy prohlásíme poddaného s pracovitostí o 1 vyšší, než je naše nalezená pracovitost.

Binárně vyhledáváme v rozmezí 0 až P_{max}, kde P_{max} je maximální hodnota pracovitosti přes všechny poddané. K té se můžeme dostat tak, že se na začátku zeptáme všech krajů na nejpracovitějšího poddaného a z nich vezmeme maximum. Provedeme tedy nejvýše $\log_2 P_{max} + 1$ zeptání všech K krajů, tedy $\mathcal{O}(K \log P_{max})$ poslání poslů.

Úlohu připravili: Matúš Púll, Dan Skýpala

[1] `http://ksp.mff.cuni.cz/viz/kucharky/binarni-vyhledavani`

38-1-4 Jedno slovo

Keď sa pokúsime riešiť jednotlivé vstupy, rýchlo si všimneme, že úloha má vlastne osem podúloh, ktoré vyzerajú nazvájom dosť rozdielne. Spoločné majú len to, že v každom sa vyskytujú len riadky s rôzne sa opakujúcimi slovami "KSP" nalepenými za seba.

Zo zadania vieme, že máme pre každú správu nájsť riešenie. Správnym dekódovaním každého vstupu teda dostaneme správu, ktorá nam povie, čo je riešením – bude to nejaké slovo, ktoré odovzdáme. Občas sa tam vyskytnú aj zbytočné vety, ktoré treba ignorovať, tie nám však svoju zbytočnosť naznačia.

Podúlohy sú zoradené podľa približnej obtiažnosti, skúsme ich teda riešiť postupne:

1. V prvej úlohe máme riadky, na ktorých sú nejaké počty KSP, občas sú riadky prázdne. Skúsme teda jednotlivé počty KSP v riadkoch spočítať. Dostaneme čísla od 1 do 26, ktoré môžeme rovno zmeniť na písmená podľa ich poradia v abecede. Prázdny riadok značí medzeru medzi slovami.

2. V ďalšej podúlohe je podozrivo veľa riadkov len s jedným KSP, občas ale nejaký má KSP dve. Keď si ich skúsime spočítať, zistíme, že riadkov s dvomi KSP je cca 25-krát menej. A ak si rozdelíme riadky na 26-tice, ukáže sa, že vo väčšine z nich je práve jeden riadok s dvomi KSP. Z jeho pozície v 26-tici teda vezmeme písmeno. Ak majú všetky riadky len jedno KSP, vezmeme medzeru.

3. Táto podúloha vyzerá podobne ako prvá, ale v každom riadku je aspoň jedno KSP… vlastne všade je KSP dosť veľa. A keď si spočítame štatistiku pre rôzne počty, všimneme si, že nejmenší počet je 32 KSP v riadku a tiež sú tam počty od 62 až po 90. To sú ASCII čísla medzery a veľkých písmen abecedy, stačí teda použiť funkciu `chr` v Pythone alebo jej ekvivalent na získanie písmen z riadkov.

4. Vo štvrtej podúlohe máme zase menšie počty KSP na každom riadku a aj prázdne riadky. Veľa počtov nám ale chýba, napríklad 1, všetky násobky 2 okrem dvojky samotnej, 9… Vlastne máme iba prvočíselné počty. Stačí teda zobrať pre každé prvočíslo toľké písmeno, koľké prvočíslo to je (napríklad 2 bude A, 3 B, 5 C…). Prázdny riadok bude zrejme opäť medzera.

5. V tejto podúlohe si môžeme rýchlo všimnúť, že každý tretí riadok je prázdny. Po prázdnom riadku vždy nasleduje riadok, ktorý je buď prázdny, alebo obsahuje 1 alebo 2 KSP. A ďalší riadok vždy obsahuje $0-9$ KSP. Kombinácia s dvoma nulovými riadkami sa vyskytuje často, to môže byť medzera. Keď sú na prvom riadku 2 KSP, na druhom nie je nikdy viac než 6. Je to teda desiatkový zápis, prvá číslica vyjadruje počet desiatok a druhá počet jednotiek, čísla opäť prevedieme na písmená.

6. Tu máme nejaké riadky s $0-5$ KSP, pričom si rýchlo všimneme, že počty KSP v riadkoch klesajú v maých skupinách riadkov oddelených prázdymi riadkami. Pripomína to teda tiež nejaký spôsob zápisu po cifrách, ale s rozdielom, že pre každý z počtov $5-1$ máme len dva možné stavy – buď je alebo nie je prítomný v klesajúcej postupnosti. Je to teda binárne kódovanie, kde x KSP pridáva 2^x do súčtu, výsledky prevedieme na písmená. Medzeru teda spoznáme podľa prázdnej postupnosti, teda dvoch prázdnych riadkov po sebe.

7. Táto podúloha vyzerá presne ako prvá podúloha, ale podozrivo sa v nej opakuje niekoľko počtov. Keď ju skúsime vyriešiť pomocou riešenia prvej podúlohy, dostaneme len ďalší text obsahujúci veľa slov KSP a občas nejaké slovo "return". To ale znamená, že môžeme nahradiť "return" za symbol nového riadku a dostať valídne zadanie pre prvú podúlohu! Keď ho znova dešifrujeme, dostaneme riešenie.

8. Táto poúloha zase vyzerá presne ako druhá podúloha. Keď ju dešifrujeme, dostaneme opäť text s KSP a slovami "return", ktoré nahradíme novými riadkami. Dostaneme niečo, čo sa nápadne podobá na zadanie piatej podúlohy – vyriešime teda aj tú. Dostaneme ešte jeden text zložený z KSP a "return", ktorý nám tento raz pripomína šiestu podúlohu. Keď vyrišime aj tú, dostaneme krátku správu.

Úlohu připravili: Ríša Hladík, Ján „Janči“ Plachý

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*