Korespondenční Seminář z Programování

33. ročník | KSP | Listopad 2020

Vzorová řešení první série třicátého třetího ročníku KSP

33-1-1 Prosperující provincie

Nejdřív si setřídíme události podle roku, kdy se staly.

Teď bychom je mohli procházet, zaznamenávat si osídlení a katastrofy a udržovat si vztahy všech říší a konce civilizací v jejich provinciích. Jde to ale jednodušeji.

Bez katastrof

Víme, že události se v každé provincii dějí v konkrétním pořadí. Provincie je nejprve osídlena, pak může být její říše podrobena (i několikrát), a nakonec její říše zanikne. Pouze katastrofa může nastat kdykoli. Předpokládejme tedy na chvíli, že katastrofy nikdy nenastanou.

To, na které říši civilizace v dané provincii závisí, se časem může několikrát změnit a informace o konci civilizace je k nalezení až v říši, jejíž správa dané province je v čase poslední. Proto budeme procházet události od konce a tuto informaci si ve směru k osídlení provincie předávat. Ke každé říši si budeme ukládat pouze rok, ve kterém byla zničena poslední z řetězce postupně se podrobujících říší (= rok, ve kterém skončí civilizace v provinciích, které získala zničená říše taky tím, že si jejich předchozí říše podrobila).

Pro každou událost se rozhodneme podle jejího druhu:

- *zničení říše:* uložíme si k ní rok jejího zničení, tedy rok zániku civilizace v jejích provinciích.
- *podrobení říše:* uložíme si k podrobené říši rok, který máme uložený u říše, která si ji podrobila. (Ten už určitě známe, je to rok, ve kterém zanikla poslední z řetězce říší, které si postupně nárokovaly provincie této říše, a ten určitě nastane později, a tedy jsme ho už prošli.)
- *osídlení provincie:* pak známe rok osídlení provincie a u říše, která ji právě kolonizovala, najdeme rok, kdy v provincii skončí civilizovaný život. Můžeme tedy dopočítat, jak dlouho se zde dá civilizovaně žít.

Potom už jen najít maximum z dob civilizovaného žití a jeho provincii a máme vyřešeno.

Katastrofy nastávají

Při zpracování události osídlení provincie si zapamatujeme rok počátku a zániku civilizace v ní. Pak projdeme ještě jednou všechny události a když narazíme na katastrofu, zánik civilizace v dané provincii uspíšíme na rok katastrofy (katastrofou nejde civilizovanou dobu prodloužit, jen zkrátit).

Nakonec projdeme přes všechny provincie začátek a konec civilizace, odečteme začátek od konce, a tím získáme dobu civilizovaného žití. Najdeme maximum a provincii, ke které patří, a máme vyřešeno, teď už úplně.

Časová složitost je lineární k počtu událostí (třídit můžeme přihrádkově, každou událost procházíme konstanta-krát). Prostorová složitost je také lineární k počtu událostí.

Program (C++):
`http://ksp.mff.cuni.cz/viz/33-1-1.cpp`

Terka Hrochová

33-1-2 Kuchyňská prkénka

Pomalé řešení

Úlohu můžeme snadno vyřešit pomocí dynamického programování. Ti z vás, kteří se s touto metodou ještě nesetkali, si o ní mohou něco přečíst například v naší kuchařce.[1]

Každý centimetr desky si můžeme představit jako políčko. Políčko pak může být buď nevyužité, nebo může být využité jako i-tý centimetr požadované destičky. Těmto možným stavům říkejme *obsazení*. Možných obsazení je tedy $M+1$ (M různých poloh destičky a jedno jako nevyužitý kousek).

Pro každé políčko a každé jeho možné obsazení si spočítáme maximální hodnotu za předešlá políčka. Budeme počítat jen již ukončené destičky.

Pro první dílek je možné pouze to, aby zde bylo prázdné obsazení za nulový výdělek nebo začátek destičky za výdělek mínus cena za opravení daného dílku. Ostatním obsazením pro jejich nemožnost určíme výdělek $-\infty$.

Každé další políčko budeme počítat z toho předchozího. Výdělek za prázdné políčko je pouze maximum z výdělku za předchozí políčko v případě, že je prázdné, nebo je tam konec destičky (prostě si z těchto variant vybereme tu lepší). Začátek nové destičky také může následovat po konci destičky nebo po prázdnému políčku. Od maxima z těchto hodnot tedy jen odečteme cenu na opravu daného políčka.

Ostatní obsazení jsou jednoduchá. Evidentně i-té políčko destičky musí následovat za $i-1$ políčkem, tedy od ceny $i-1$ políčka destičky na předešlém poličku odečteme náklady na opravu. K celé destičce ještě musíme přičíst výdělek za ni.

Celkový výsledek pak jen přečteme jako maximum z konce destičky a ničeho na posledním políčku.

Toto řešení poběží v čase $\mathcal{O}(NM)$.

Rychlé řešení

Dále toto řešení vylepšíme.

Nejprve zadání mírně přeformulujeme. Políčka již nebudeme opravovat. Na políčka místo toho budeme umísťovat buď destičku délky M za výdělek K, anebo kostičku za výdělek stejné hodnoty jako původní cena za opravu. Stále nás zajímá maximální cena, co si můžeme vydělat.

Výsledek takto upravené úlohy bude právě o součet všech cen za opravy větší než výsledek původní úlohy.

Proč? Inu proto, že v původní úloze jsme odečetli cenu všech oprav na využitých políčkách a v nové úloze naopak přičteme cenu všech oprav na nevyužitých políčkách. Evidentně totiž na každé políčko, kde není destička, můžeme umístit kostičku a řešení se tím nezhorší. Každé políčko tedy do rozdílu úloh přispěje právě jednou cenou své opravy.

Na tuto úlohu můžeme využít jednodušší dynamické programování. Pro každou souvislou posloupnost si spočítáme maximální cenu, kterou lze na těchto políčkách vydělat.

Prázdná posloupnost má evidentně nulový výdělek.

[1] `http://ksp.mff.cuni.cz/viz/kucharky/dynamicke-programovani`

Pro všechny ostatní posloupnosti uvážíme dvě varianty:

- Na posledním políčku by mohla být kostička. V tomto případě výdělek odpovídá výdělku za o jedna kratší posloupnost zvýšenému o cenu dané kostičky.
- Na posledním políčku by také mohlo být poslední políčko destičky. V tomto případě se podíváme na zisk u o M kratší posloupnosti a k němu přičteme K. Když o M kratší posloupnost neexistuje, tak tato varianta nemůže nastat.

Z těchto dvou variant pak jen vybereme maximum.

Finální výsledek je maximální výdělek za posloupnost všech políček. Abychom získali odpověď na původní úlohu, tak od něj ještě odečteme součet všech oprav.

Časová složitost tohoto algoritmu je $\mathcal{O}(N)$. V případě, že budeme zapomínat mezivýsledky, jakmile nebudou potřeba, může paměťová složitost klesnout na pouhých $\mathcal{O}(M)$. Vstup budeme načítat, průběžně zpracovávat a při tom ho průběžně sčítat.

Jiří Kalvoda

33-1-3 Laser v bludišti

Shrňme si úlohu: máme mapu, na které se vyskytují různé objekty jako zrcadla, zdi, zdroj laseru a cílové místo, kam chceme laser dostat. Co nás může prvně napadnout, je to, že si celou mapu uložíme a pak s ní budeme dále pracovat. Ale ouha, když si necháme vygenerovat větší vstup od odevzdávátka, tak náš milý program spadne, že nemá dost paměti a i kdyby jsme měli dost paměti, tak náš program by nejspíše nikdy nedoběhl.

Zkusme si úlohu přeformulovat. Máme graf a v něm vedou cesty mezi nejbližšími vrcholy, které mají stejnou x-ovou nebo y-ovou souřadnici. V tomto grafu jsou neorientované hrany a nejsou ohodnoceny. Máme 4 typy vrcholů. Vrchol *zeď* říká, že tady cesta nevede a musíme hledat jinde. Vrchol *zdroj laseru* říká, zde začíná cesta a můžeme jít jen směrem nahoru. Dále vrchol *cíl,* kde má laser skončit a nakonec vrchol *zrcadlo.*

Vrcholu typu zrcadlo je trochu speciální, a to tím, že je ohodnocen, a to ještě trochu zvláštně. Neobsahují totiž informaci, že do tohoto vrcholu se dostaneme za tolik energie, ale říkají, že pokud půjdeme od nějakého vrcholu, tak jedna či žádná cesta daným směrem (danou hranou) je zadarmo a za ostatní si musíme připlatit. Konkrétně jednou jednotkou energie. Jak a kdy musíme zaplatit energií, je dáno natočením zrcadla.

V tomto grafu (popsaném výše) hledáme cestu od zdroje do cíle, která stojí nejméně energie.

Jak cestu najdeme? Možná se vám to nebude z počátku zdát, ale můžeme použít s malou modifikací známý algoritmus na hledání cest: Dijkstrův algoritmus. První modifikace je vcelku zjevná: ve vrcholu budeme muset zjišťovat, jestli daná hrana bude stát jednotkovou energii či nulovou. Druhá modifikace je, že Dijkstrův algoritmus se nevrací do vrcholů, které již zpracoval. Důvod je ten, že když se v obecném grafu dostaneme znovu do daného vrcholu za cenu 0, tak nás to nezajímá, protože je nám jedno z jakého vrcholu jsme přišli.

U této úlohy nás to zajímá, protože můžeme přijít k zrcadlu z více směrů za stejnou potřebnou energii, ale z jednoho směru může být výhodnější pokračovat dál (můžeme jít zadarmo). Takže by se mohlo stát, že kdybychom zpracovali vrchol jen jednou, tak bychom nenašli nejlevnější cestu.

Nabízí se jednoduché řešení, že se u vrcholů nebudeme vracet ke směru, se kterým jsme již dříve pracovali. Čili ke každému vrcholu se budeme vracet maximálně 4-krát (pro každý směr).

Nakonec vyřešíme problém, který jsme opomněli zmínit: totiž jak si graf vyrobíme. Odpověď je vcelku jednoduchá: Všechny vrcholy si nejdříve setřídíme primárně podle x-ové osy a sekundárně podle y-ové osy. Když budeme číst setřízenou posloupnost vrcholů, tak všechny vrcholy v daném sloupci budou za sebou. Dále dva vrcholy vedle sebe v posloupnosti se stejnou x-ovou souřadnicí jsou nejbližšími sousedy kvůli tomu, že jsme třídili sekundárně podle druhé osy. Mezi těmito vrcholy povede vertikální hrana v grafu. Nyní jsme vyřešili vertikální hrany a horizontální uděláme obdobně: budeme třídit primárně podle y-ové osy a sekundárně podle x-ové.

Časová složitost třídění vrcholů je $\mathcal{O}(n \log n)$. Dijkstrův algoritmus pak bude trvat $\mathcal{O}(n + n \log n)$, protože hran je řádově stejně jako vrcholů. Pokud využijeme toho, že hrany stojí 0 nebo 1 jednotek energie, tak složitost můžeme stáhnout na $\mathcal{O}(n)$ – můžeme si vyrobit frontu, do které půjde přidávat na oba konce. Hrany za 0 jednotek energie budeme dávat na začátek fronty (mají přednost) a hrany za 1 jednotku energie budeme přidávat na konec fronty. Tímto jsme ze zbavili potřeby mít haldu a zrychlili tím algoritmus, nicméně musíme pořád třídit vrcholy, takže to celkovou časovou složitost neovlivní.

Celková časová složitost je $\mathcal{O}(n \log n)$.

Paměťová složitost je $\mathcal{O}(n)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-1-3.py`

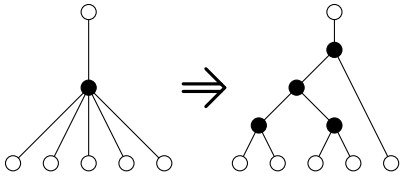
Michal Kodad

33-1-4 Sbírání bonbónů

Strom si zakořeníme ve vrcholu, ve kterém dle zadání začínáme.

Dále si zadání mírně poupravíme: V grafu povolíme různě dlouhé hrany, ale na úkor toho omezíme počet potomků každého vrcholu na maximálně dva.

Dá se ukázat, že původní úlohu lze na tuto snadno přepracovat. Všem původním hranám jednoduše určíme délku jedna. Poté každý vrchol, který má $P > 2$ potomků nahradíme $P - 1$ vrcholy, které libovolné propojíme do stromu pomocí hran nulové délky. Na kořen tohoto stromečku připojíme rodiče nahazovaného vrcholu a na volné vrcholy pak jeho potomky. Jestliže v nahrazovaném vrcholu je bonbón, tak ho umístíme do kořene přidaného stromečku.



Úlohu vyřešíme pomocí dynamického programování. Pro každou kombinaci vrchol a počet bonbónů si spočítáme, jaký je minimální počet kroků, abychom (když začneme v daném vrcholu) v jeho podstromu sesbírali příslušný počet bonbónů. Z hodnot pro kořen pak víme minimální počet kroků v naší úloze.

Jak ale lze tyto hodnoty spočítat? Můžeme je počítat od listů:

Pro list bez bonbónu evidentně jsou všechny nenulové počty nedosažitelné (což můžeme reprezentovat hodnotou $+\infty$. Když se ve vrcholu (nejen tedy v listu) nachází bonbón, tak se nám ho evidentně vyplatí sebrat (protože nemusíme udělat žádné kroky). Tedy oproti stavu bez bonbónu stačí posunout všechny hodnoty pro daný vrchol o jeden index výše a pro jeden bonbón říct, že požadovaný počet kroků je nula.

Pro vrchol s jedním potomkem bez bonbónu pro všechny nenulové počty bonbónů musíme pokračovat do potomka. Hodnoty tedy můžeme převzít z potomka s tím, že ke každé z nich přičtu dvojnásobek délky hrany mezi nimi (musíme ji projít v obou směrech).

Zbývá nám tedy uvážit nejsložitější případ – vrchol bez bonbónu s dvěma potomky. Pro každý nenulový počet bonbónů máme několik možností, z niž vybereme tu nejlepší.

Buď můžeme všechny bonbóny vzít z jednoho podstromu. Pro každý podstrom se tedy stačí podívat na příslušnou hodnotu a zvýšit ji o dvojnásobek hrany do něj. Nebo můžeme vzít z každého potomka část. Vyzkoušíme tedy všechny možnosti rozložení (na to lze použít cyklus od jedné do požadovaného počtu bez jedné, který bude určovat počet v prvním potomkovi a po druhém budeme chtít zbytek). Pro každou z těchto variant pak jednoduše vezmeme součet příslušných hodnot potomků zvýšené o dvojnásobek obou hran. Ze všech těchto variant pak jednoduše vezmeme minimum.

Pro snazší rekonstrukci řešení si ke každé hodnotě můžeme psát, jak vznikla. Tedy například kolik bonbónů mělo být odebráno z levého podstromu. Rekonstrukci lze pak provézt jediným rekurzivním průchodem do hloubky. Bude se jednat o funkci dvou parametrů – vrchol a počet bonbónů z daného podstromu. Ta nejprve do výstupu vypíše číslo daného vrcholu. Poté se rekurzivně zavolá na každého svého potomka, z jehož podstromu se má získat nenulový počet vrcholů. Po každé rekurzi pak znovu vypíše číslo daného vrcholu.

Aby fungovala úprava zadání, kde jsme některé vrcholy nahradili množinou vrcholů, všem vrcholům z množiny určíme stejné id. Pak jen ve výstupu nahradíme více stejných hodnot po sobě za jednu.

Složitost

Nejprve nahlédneme, že provedenou úpravou grafu se asymptoticky nezmění počet vrcholů. Počet vrcholů, co každým nahrazením přibude je počet potomků bez dvou. Tedy počet vrcholů, co celkem přibude, lze omezit celkovým počtem potomků v grafu. Jelikož každý vrchol je nejvýše jedenkrát potomkem jiného vrcholu, tak přibude nejvýše N vrcholů. Celkový počet tedy bude maximálně $2N$, což je stále $\mathcal{O}(N)$.

Pro každý vrchol pak počítáme K hodnot, každou z nich cyklem délky až K. Celková časová složitost je $\mathcal{O}(NK^2)$. Paměťová pouze $\mathcal{O}(NK)$.

Zrychlení řešení

Jelikož K může být veliké až jako N, tak předchozí odhad za použití pouze N určuje složitost $\mathcal{O}(N^3)$. Pojďme to trošku zrychlit.

Pro každý vrchol můžeme počítat pouze hodnoty dynamického programování, které jsou dosažitelné. Tedy maximálně tolik, jaká je velikost jeho podstromu.

Indukcí pak nahlédneme, že podstrom velikosti X zvládneme celý spočítat v čase $\mathcal{O}(X^2)$: Listy spočítáme v konstantním čase. Nechť k je konstanta asymptotické složitosti. Vrcholy s jedním potomkem vyřešíme v čase potřebném pro potomka, tedy $k(X-1)^2$ zvýšeném $\mathcal{O}(X)$, což evidentně náleží $\mathcal{O}(X^2)$.

Pro vrchol se dvěma potomky velikosti A a B nejprve vyřešíme oba v čase $k(A^2+B^2)$. Při počítání minim pak uděláme $k(AB)$ operací, protože každou dvojici hodnot z potomků použijeme právě jednou. Celkem tedy $k(A^2+AB+B^2) \leq k(A+B)^2$.

Časová složitost celého algoritmu tedy je $\mathcal{O}(N^2)$.

Lepší odhad složitosti

Obarvíme si všechny vrcholy, jejichž velikost podstromu je větší než K. Když jdeme od listu ke kořeni, velikost podstromu narůstá. Díky tomu vybarvené vrcholy tvoří souvislou komponentu obsahující kořen. Neobarvené vrcholy tvoří několik podstromů.

Každý neobarvený podstrom evidentně zvládneme spočítat v kvadratickém čase vzhledem k počtu vrcholů. Ovšem počet vrcholů v každém z nich je nejvýše K, tedy ze součtu složitostí mohu vytknout K a zbude mi počet obarvených vrcholů. Z toho plyne, že celková časová složitost na vypočtení obarvených vrcholů je $\mathcal{O}(NK)$.

Z obarvených vrcholů zvýrazníme všechny ty, které mají dva obarvené potomky. Každý vrchol, který obsahuje jenom jednoho potomka, evidentně zvládneme vyřešit v $\mathcal{O}(K)$. K ostatním nezvýrazněným vrcholům lze přiřadit alespoň jeden nevybarvený podstrom, jehož kořen je jeho potomkem. Žádný podstrom nebude přiřazen dvakrát.

Časová složitost na výpočet každého z těchto nezvýrazněných vrcholů tedy lze určit jako $\mathcal{O}(Kx)$, kde x je velikost onoho nevybarveného podstromu. Tuto časovou složitost můžeme rozložit na těchto x přiřazených vrcholů v podstromu. Amortizovaně je tedy časová složitost každého vrcholu $\mathcal{O}(K)$. Celkem tedy také $\mathcal{O}(NK)$.

Zbývá dořešit zvýrazněné vrcholy. Uvažme pouze strom z vybarvených vrcholů. Zvýrazněné vrcholy v něm tvoří jediné vrcholy, které obsahují (alespoň) dva potomky. Takovýchto vrcholů je ale v tomto grafu o jedna méně než listů. Ovšem každý list je v původním grafu kořenem podstromu velikosti alespoň K, díky tomu listů a tedy i nezvýrazněných vrcholů může být nejvýše $\mathcal{O}\left(\frac{N}{K}\right)$. I když tedy každý takových vrchol vyřešíme v $\mathcal{O}(K^2)$, celková časová složitost je $\mathcal{O}(NK)$.

Časová složitost celého algoritmu se tedy sečte na $\mathcal{O}(NK)$.

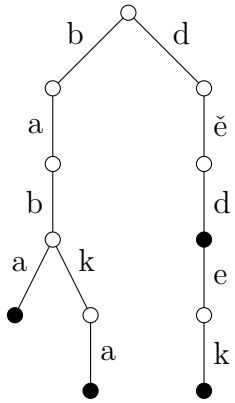
Jiří Kalvoda

33-1-X1 Krycí jména

Jednoslovná jména

Nejprve uvažujme agenty pojmenované jednoslovným jménem. Tehdy postačí naplnit jejich jmény písmenkový strom neboli trii. To je strom, který se v kořeni rozhoduje podle prvního písmene jména, o hladinu níže podle druhého písmene atd. Navíc si označíme vrcholy, ve kterých končí nějaké jméno.

Například pro příjmení Baba, Babka, Děd a Dědek vytvoříme tento strom:



(Všimněte si, že ne všechny označené vrcholy jsou listy stromu.)

Když chceme rozluštit nějakou zkratku, vyrazíme od kořene po písmenech zkratky. Pokud bude některé písmeno ve stromu chybět (třeba pro dotaz `Bač.` chybí hrana pro `č`), neodpovídá zkratka žádnému jménu.

V opačném případě se podíváme na podstrom ležící pod vrcholem, kde zkratka skončila. Označené vrcholy v tomto podstromu odpovídají přesně těm jménům, která vyhovují dané zkratce. Pokud je takový vrchol jediný, vypíšeme ho jako jednoznačné řešení (buďto projdeme strom od označeného vrcholu ke kořeni a posbíráme písmenka, nebo si u označených vrcholů pamatujeme, jakému jménu odpovídají). Pokud je označených vrcholů v podstromu více, zkratka není jednoznačná.

Aby se nám tyto případy lépe rozlišovaly, předpočítáme si pro každý vrchol stromu, kolik označených vrcholů je v podstromu pod ním. Tím pádem jakmile zkratka skončí, poznáme podle čísla ve vrcholu, je-li jednoznačná. A pokud je, můžeme si být jistí, že pod vrcholem leží už pouze cesta do označeného vrcholu (listu). Takže stačí jít pořád dolů a najdeme ho.

Jakou časovou složitost toho řešení má? Písmenkový strom postavíme v lineárním čase s velikostí seznamu jmen (velikost seznamu budeme značit S a myslíme tím celkový počet písmen ve jménech). Složitost dotazu můžeme omezit délkou nejdelšího jména L.

Dvojslovná jména

V naší úloze se ovšem jména skládají ze dvou slov. To situaci poněkud komplikuje: Můžeme si postavit zvlášť strom pro křestní jména a zvlášť pro příjmení. Ale pak narážíme na situace, kdy zkratce vyhovuje mnoho křestních jmen a mnoho příjmení, ale jen jedna z jejich kombinací je krycím jménem agenta.

Jak se vyhnout zkoušení všech kombinací? Pomůžeme si trikem ze seriálu o stromech (úloha 29-2-7). Opět se soustředíme na jeden písmenkový strom. Jakmile do něj vložíme všechna slova, projdeme ho do hloubky. Při tom budeme měřit čas (počítat kroky prohledávání) a ke každému vrcholu si poznamenáme, v jakém čase jsme do něj poprvé vstoupili a v jakém jsme ho naposledy opustili. Všimněte si, že interval mezi těmito dvěma časy jsme strávili prohledáváním podstromu pod daným vrcholem.

Každá zkratka odpovídá podstromu, tedy intervalu časů. Takže stačí vyhledat všechna jména, jejichž čas leží v daném intervalu. Který čas, když si pamatujeme jak příchod, tak odchod? Je to jedno, buď v intervalu leží oba, nebo ani jeden.

Pro dvojslovná jména tento trik aplikujeme zvlášť pro křestní jména a zvlášť pro příjmení. Krycí jméno agenta tedy popíšeme uspořádanou dvojicí časů (i, j). Zkratka křestního jména nám omezí i na nějaký interval $\langle i_1, i_2\rangle$. Zkratka příjmení omezí j na $\langle j_1, j_2\rangle$.

Můžeme si tedy představit, že krycí jména odpovídají bodům v rovině o souřadnicích (i, j). Každá zkratka v rovině vymezí nějaký obdélník $\langle i_1, i_2\rangle \times \langle j_1, j_2\rangle$, ve kterém chceme najít všechny body. Přesněji řečeno chceme zjistit, kolik bodů v něm leží, a pokud právě jeden, tak chceme vědět, který to je.

To přímo volá po nějaké dvojrozměrné datové struktuře, třeba 2D intervalovém stromu. Stačí nám statický: jednou si ho pro dané body postavíme a pak už mu jenom klademe dotazy. Jak se takový strom buduje, najdete například v řešení úlohy 24-4-7, a to včetně urychlovacího triku zvaného *zlomkové kaskádování* (fractional cascading). S tímto trikem dokážeme sestrojit strom pro n bodů v čase $\mathcal{O}(n \log n)$ a odpovídat na „počítací“ dotazy v čase $\mathcal{O}(\log n)$. My sice potřebujeme kromě počtu znát i jeden konkrétní bod, ale tomu úlohu snadno přizpůsobíme: kdykoliv si struktura pamatuje počet bodů v nějaké části roviny a tento počet je nenulový, uložíme k tomu souřadnice jednoho z bodů.

Co se časové složitosti týče: Připomeňme, že S značíme velikost seznamu jmen a L maximální délku jména. Nejprve strávíme čas $\mathcal{O}(S)$ vytvořením obou písmenkových stromů, jejich projitím do hloubky a spočítáním času příchodu a odchodu pro všechny vrcholy. Pak přeložíme jména agentů na body v rovině a v čase $\mathcal{O}(S \log S)$ postavíme 2D intervalový strom. Vyhodnocování zkratky pak najde příslušné podstromy písmenkových stromů (to trvá $\mathcal{O}(L)$), přeloží je na intervaly a položí obdélníkový dotaz intervalovému stromu. Ten v čase $\mathcal{O}(\log S)$ ověří, zda je v obdélníku právě jeden bod a pokud ano, najde ho. Bod pak v čase $\mathcal{O}(L)$ přeložíme na jméno a to vypíšeme.

Celkem tedy předvýpočet trvá $\mathcal{O}(S \log S)$ a odpověď na dotaz $\mathcal{O}(L + \log S)$.

Persistentní stromy

Řešení jsme tedy poskládali z několika standardních kostiček: trií, DFS očíslování stromů a 2D intervalových stromů. Vy jste se ale chtěli naučit něco nového, viďte? Tak vám přeci jenom něco ukážeme: hezký způsob, jak 2D vyhledávací strom nahradit strukturou, která se příjemněji programuje.

Nejprve definujeme *persistentní vyhledávací strom.* Ten umí totéž jako obyčejný vyhledávací strom, jen si navíc pamatuje svou historii. Kdykoliv vkládáme nebo mažeme prvek, vznikne tím nová verze stromu. A kdykoliv se stromu na něco ptáme, uvedeme, v jaké verzi hledáme. Verze můžeme například číslovat přirozenými čísly.

Ukážeme si jednoduchou konstrukci persistentního stromu. Začneme s obyčejným stromem, ale prohlásíme, že jakmile jednou nějaký vrchol vznikne, už ho nikdy nebudeme měnit. Kdykoliv v nějakém vrcholu budeme chtít změnit klíč nebo některý z ukazatelů na syny, prostě vrchol zkopírujeme a změníme kopii. Tím pádem ovšem musíme přesměrovat ukazatel v otci na tuto kopii, což vynutí zkopírování otce a tak dále. Při přidáni listu tedy například zkopírujeme celou cestu až do kořene, čímž vznikne nový kořen, který přiřadíme k nové verzi. Všimněte si, že nám vznikl nový strom, který má některé vrcholy nové, ale na zbývající podstromy se odkazujeme do původního stromu. To nevadí, jelikož staré vrcholy se už nikdy nezmění.

Jak rychlé to je? Strom na n vrcholech je vysoký $\mathcal{O}(\log n)$, takže vložení nového prvku trvá $\mathcal{O}(\log n)$ a způsobí zkopírování $\mathcal{O}(\log n)$ vrcholů na cestě mezi novým listem a kořenem. Časovou složitost jsme tedy oproti klasickému stromu nezhoršili, ale navíc spotřebujeme $\mathcal{O}(\log n)$ paměti na uložení změny do historie. Mazání ze stromu bychom provedli obdobně.

Málem bychom zapomněli, že stromy je potřeba vyvažovat. To překvapivě není takový problém: stačí si uvědomit, že běžné způsoby vyvažování stromů (třeba AVL nebo červeno-černé stromy) jsou lokální – tím myslíme, že veškeré změny se odehrávají jen v blízkém okolí cesty mezi přidaným/smazaným vrcholem a kořenem. Takže stále stačí zkopírovat jen $\mathcal{O}(\log n)$ vrcholů. Pomocné informace, jako je třeba znaménko u AVL stromů nebo barva u červeno-černých, není ani potřeba verzovat, protože nejsou při vyhledávání potřeba a modifikace se týkají jen poslední verze.

Vyvažovat stromy se nám jistě nechce. Můžeme si pořídit *persistentní intervalový strom* vybudovaný nad předem známou množinou klíčů. Bude fungovat podobně jako běžný intervalový strom vybudovaný nad polem (listy jsou prvky pole indexované klíči, obsahující 0 nebo 1 podle toho, je-li klíč zrovna přítomen, vnitřní vrcholy obsahují součty listů v podstromu pod nimi). Ovšem aby fungovalo kopírování, musíme se mezi vrcholy odkazovat pointery, nestačí strom očíslovat jako haldu a uložit do pole. Také všechny operace musíme provádět shora dolů (protože verze je identifikovaná kořenem). Každopádně operace s klíči i dotazy budou nadále trvat $\mathcal{O}(\log n)$ a jedna verze spotřebuje $\mathcal{O}(\log n)$ paměti.

Teď se ale vraťme k původnímu 2D problému. Dostaneme nějaké body v rovině a chceme odpovídat na obdélníkové dotazy. Prozatím předpokládejme, že žádné dva body nemají tutéž y-ovou souřadnici.

Předvýpočet proběhne takto: setřídíme body podle y shora dolů a v tomto pořadí je zameteme vodorovnou přímkou. Přitom si budeme udržovat persistentní intervalový strom indexovaný x-ovými souřadnicemi (ty si také na začátku setřídíme) a pro každou si budeme pamatovat, kolik bodů ji už mělo. Kdykoliv tedy zameteme nějaký bod, zvýšíme počítadlo v příslušném listu intervalového stromu a vytvoříme novou verzi stromu. To vše pro n bodů zvládneme v čase $\mathcal{O}(n \log n)$ a spotřebujeme při tom $\mathcal{O}(n \log n)$ paměti.

Teď si rozmysleme, co nám umí říci 1D intervalový dotaz položený nějaké konkrétní verzi intervalového stromu. Každý interval odpovídá nějakému rozsahu x-ových souřadnic $\langle x_1, x_2 \rangle$, verze odpovídá poloze zametací přímky y. Odpovědí je počet bodů, které leží v nekonečném pásu nad vodorovnou úsečkou $[(x_1, y), (x_2, y)]$.

Tyto hodnoty můžeme použít podobným způsobem jako prefixové součty. Nechť chceme spočítat body v nějakém obdélníku $\langle x_1, x_2 \rangle \times \langle y_1, y_2 \rangle$. Pak vyhledáme verze příslušné k polohám zametací přímky y_1 a y_2 – to zvládneme binárním vyhledáním v bodech setříděných podle y. Pak se persistentního stromu zeptáme na pásy nad intervalem $\langle x_1, x_2 \rangle$ pro verze odpovídající y_1 a „těsně nad y_2“ a tyto počty odečteme. Ehjle, v čase $\mathcal{O}(\log n)$ jsme odpověděli na obdélníkový dotaz.

Zbývá dořešit opakování y-ových souřadnic. Pokud více bodů leží na jedné vodorovné přímce, stačí je zpracovat v libovolném pořadí a k poloze přímky přiřadit až poslední verzi.

Pro úlohu s krycími jmény ještě potřebujeme najít libovolný bod v obdélníku. Snadno najdeme ten nejnižší: v každém listu intervalového stromu (který odpovídá x-ové souřadnici) si budeme pamatovat odkaz na nejnižší bod s touto souřadnicí, který už jsme zametli. Vnitřní vrcholy si pak budou pamatovat minimum podle y z odkazů ve svých synech. Takže pro zadaný interval umíme najít nejnižší bod, který v něm leží. Při „prefixovém“ odčítání pásů od sebe stačí uvážit hodnotu z níže končícího pásu: pokud v obdélníku leží vůbec nějaký bod, musí ležet v nižším pásu a odčítání vyššího na tom nic nezmění.

Máme tedy jednoduchou datovou strukturu, která řeší původní problém stejně dobře jako 2D intervalové stromy.

Dodejme ještě, že svět persistentních struktur je mnohem bohatší – jednak jde v mnoha případech snižovat paměť na uložení jedné verze až na amortizovaně konstantní (byť zrovna u této úlohy ne, protože potřebujeme verzovat i součty ve vnitřních vrcholech stromu). A také se dá zařídit, aby se daly měnit i historické verze a vyrábět „alternativní historie“. Takovým strukturám se pak říká *plně persistentní* (zatímco našim přesněji *semipersistentní*). To už je ale jiný příběh a ten budeme vyprávět zase jindy.

Martin „Medvěd“ Mareš

Výsledková listina první série třicátého třetího ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*1-1*	*1-2*	*1-3*	*1-4*	*1-S*	*1-X1*	*série*	*KSP-X*	*celkem*
0.					10	11	12	12	15	10	60,0	10,0	60,0
1.	Filip Hejsek	GPísnickáPH	4	3	10	11	12	14	15		62,0	0,0	62,0
2.	Kristýna Petrlíková	SPŠJičín	3	11	10	11	12	13	15		61,0	0,0	61,0
3.	Eliška Macáková	CENADA BA	1	1	10	11	12	12	15	9	60,0	9,0	60,0
4.–5.	Jan Adámek	GKepleraPH	4	6	10	11	12	11	15	7	59,0	7,0	59,0
	Vladimír Chudý	G Chrudim	4	16	10	11	12	11	15	4,5	59,0	4,5	59,0
6.–7.	Viktor Fukala	GKepleraPH	4	5	10	11	12	10	15	9	58,0	9,0	58,0
	Jiří Kvapil	GTomkovaOL	3	15	10	11	12	10	15	2	58,0	2,0	58,0
8.–9.	Pavel Jordán	GPOA Znojmo	2	1	10	8	12	6	15		51,0	0,0	51,0
	Daniel Skýpala	GTomkovaOL	3	16	10	11	12	3	15	4	51,0	4,0	51,0
10.	Ondřej Sladký	GMikulášPL	4	8	10	11		11	15	9	47,0	9,0	47,0
11.	Vít Skalický	GPísnickáPH	3	15	10	11	12		13		46,0	0,0	46,0
12.	Ondřej Skácel	GTomkovaOL	2	1	10	2	12	6	15		45,0	0,0	45,0
13.	Robert Jaworski	GÚstavníPH	3	3	10	7	12		15	1	44,0	1,0	44,0
14.	Jan Kotovský	GPísnickáPH	2	2	10		12	2	15		39,0	0,0	39,0
15.–16.	Jakub Surga	ParkLane	3	1	10	11	2	2	13		38,0	0,0	38,0
	Lukáš Veškrna	GKepleraPH	3	1	10		12	1	15	4	38,0	4,0	38,0
17.–18.	Jakub Ondroušek	GTomkovaOL	1	1	10		12		15		37,0	0,0	37,0
	Matej Štencel	GPošKošice	4	4	10		12		15		37,0	0,0	37,0
19.	Dominik Farhan	GMikulášPL	4	5	10	11			15		36,0	0,0	36,0
20.	Janek Hlavatý	GJirsíkaČB	2	5	10	11			13		34,0	0,0	34,0
21.	Adam Kolník	SSŠVTPraha	2	1	10		4		15		29,0	0,0	29,0
22.–23.	Šimon Genčur	GBBr	1	3	10	2	1	1	14		28,0	0,0	28,0
	Martin Havelka	Gym Třeboň	3	3	10	2		1	15		28,0	0,0	28,0
24.	Prokop Randáček	GFXŠaldyLI	2	3			12		15		27,0	0,0	27,0
25.	Patrik Herman	GTomkovaOL	2	2	10	2			14		26,0	0,0	26,0
26.	Jiří Bartošík	SUHr	3	2	10				12		22,0	0,0	22,0
27.	Albert Kučera	GNadŠtolPH	4	4	10	11					21,0	0,0	21,0
28.–29.	Václav Janáček	GJarošeBO	4	4	10	10				8	20,0	8,0	20,0
	Kristýna Umlaufová	SPŠOstrov	4	2	6				14		20,0	0,0	20,0
30.	Andrej Thomas Dobrev	GJHroncaBA	4	1	10	8					18,0	0,0	18,0
31.–32.	Tomáš Kašpárek	G FrýdlNOs	3	1	10	1	1				12,0	0,0	12,0
	Jáchym Tuma	G FrýdlNOs	0	1	10		2				12,0	0,0	12,0
33.	Klára Grinerová	GZborovPH	4	1	10		1				11,0	0,0	11,0
34.–40.	Vojtěch Březina	GCoubTábor	4	4	10						10,0	0,0	10,0
	Petr Filip	GLovosice	2	1	10		0				10,0	0,0	10,0
	Vojtěch Gaďurek	PORGPha	4	1	10						10,0	0,0	10,0
	Štěpán Kovář	GNadKavaPH	4	1	10						10,0	0,0	10,0
	Michal Pavlíček	MendelGOP	3	1	10						10,0	0,0	10,0
	Daniel Šoltýs	GTřeKošice	3	2	10						10,0	0,0	10,0
	Filip Úradník	GyMimoň	4	1	10						10,0	0,0	10,0
41.	Jan Ráček	SPŠEMasLI	1	1	7						7,0	0,0	7,0
42.–43.	Bohumil Kulvejt	G Sokolov	3	1	6						6,0	0,0	6,0
	Josef Malý	GPísnickáPH	2	1	6						6,0	0,0	6,0
44.–45.	Veronika Jůzková	MensaG	3	1	2						2,0	0,0	2,0
	Matěj Strnad	ZŠRiegraSM	0	1	2						2,0	0,0	2,0

Bonusové úlohy označené „X“ mají svou vlastní výsledkovou listinu a nepočítají se do normálního bodování ročníku.

Výsledková listina KSP-X po první sérii třicátého třetího ročníku

	řešitel	*škola*	*ročník*	*sérií*	*1-X1*	*celkem*
0.					10	10,0
1.–3.	Viktor Fukala	GKepleraPH	4	5	9	9,0
	Eliška Macáková	CENADA BA	1	1	9	9,0
	Ondřej Sladký	GMikulášPL	4	8	9	9,0
4.	Václav Janáček	GJarošeBO	4	4	8	8,0
5.	Jan Adámek	GKepleraPH	4	6	7	7,0
6.	Vladimír Chudý	G Chrudim	4	16	4,5	4,5
7.–8.	Daniel Skýpala	GTomkovaOL	3	16	4	4,0
	Lukáš Veškrna	GKepleraPH	3	1	4	4,0
9.	Jiří Kvapil	GTomkovaOL	3	15	2	2,0
10.	Robert Jaworski	GÚstavníPH	3	3	1	1,0

Bonusové úlohy z jednotlivých sérií se nepočítají do bodování ročníku. Mají svou vlastní výsledkovou listinu a za jejich úspěšné vyřešení (alespoň polovina bodů za úlohu) udělujeme speciální odměny.