Milí řešitelé a řešitelky!

Blíží se konec 22. ročníku, ale ještě než odjedete k moři, do hor, na vandr či na jinou skvělou letní akci, přilétá k vám na křídlech větru poslední série sepsaná na starý pergamen Pavlem „Pauliem“ Veselým spolu s druhým dílem seriálu o APL od Martina Mareše.

Termín jejího odevzdání je stanoven na pondělí 14. června v 8:00 SELČ, což znamená, že papírové řešení byste měli podat na poštu do středy 9. června.

Nezapomeňte, že nyní máte poslední šanci, jak si zajistit účast na našem úžasném podzimním soustředění! A ti, kdo mají již spoustu bodů, mohou bojovat o titul *Král KSP* a získat tak jedno království a půl princezny (více informací již brzy na našich webových stránkách).

Elektronická řešení přijímáme na obvyklé stránce *https://ksp.mff.cuni.cz/submit/*. Náš certifikát sice není podepsán žádnou komerční autoritou, nicméně pro jeho ověření zde uvádíme jeho SHA1 hash:

`10:3B:C1:E2:4F:9A:01:98:DA:DB:5D:A0:D5:5C:80:57:DE:AD:28:C3`

Papírová řešení nám pak můžete zasílat klasickou poštou na adresu

Korespondenční seminář z programování
KSVI MFF UK
Malostranské náměstí 25
118 00 Praha 1

Pátá série dvaadvacátého ročníku KSP

„Je rozhodnuto, vyrazíš hned,“ zahřměl otec a skály vše zopakovaly ozvěnou.

„Ale . . . “

„Žádné ale!“ zaburácel, až se země zatřásla.

„. . . když já jsem ještě moc malý,“ pípl jsem. „Tys byl určitě větší, než jsi odletěl.“

„Řekla to veliká pramáti a té nelze odporovat!“

„Jenže ona je netrpělivá a sklerotická. Chce mít všechno hned a zapomněla, že jsem příliš mladý.“

„Neurážej pramáti! A teď upaluj pro princeznu.“

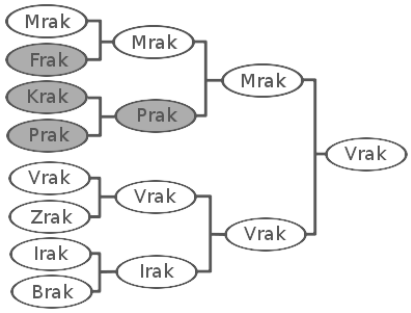
Jestli si myslíte, že draci mají život lehký, když umí chrlit oheň, přetrvají věky a jsou velcí, šeredně se pletete. Téměř nikdo nás sice neohrožuje, až na lidi, a o ty právě jde. Jsou sice drobní, jenže je jich jak mravenců. A tak se musíme skrývat v pustých horách mezi skalami, lovit zvěř a doufat, že na nás nepřijde armáda lučištníků.

Z bezpečí mezi štíty, tyčícími se vysoko do nebe, vycházíme jen z jediného důvodu: ukrást si princeznu, v případě dračic samozřejmě prince. Pramáti pak člověka očaruje tak, aby věrně sloužil svému drakovi. Princ či princezna je vlastně celkem užitečná věc, jelikož má šikovné prťavé ručičky, což nám drakům chybí. Navíc se jedná o tradici a zároveň rituál vstupu mezi velké draky.

22-5-1 Turnaj — 10 bodů

Ještě před odletem jsem musel dohrát turnaj v ohnivém zápase, což je bezkontaktní souboj, v němž se postaví dva draci proti sobě a chrlí na sebe oheň, dokud jednomu nezačne hořet tráva nandaná na čenich.

Už před pár dny jsme sehráli celého pavouka vyřazovacího turnaje (dva draci se utkají, do dalšího kola postupuje jen jeden, jenž se utká s vítězem jiného souboje . . .), takže známe a obdivujeme vítěze, jenže se neumíme dohodnout, kdo je na druhém místě. Občas totiž nelze určit, jak by dopadl souboj dvou draků, když se spolu neutkali, nemají společného soupeře, ani jejich soupeři spolu nebojovali a nemají společného protivníka, s nímž se utkali . . . Platí však alespoň tranzitivita: jestli drak A porazil draka B a drak B draka C, porazil by i drak A draka C.



Příklad pavouka hry. Šedě jsou zvýrazněni ti, které by určitě porazil drak Mrak.

Potřebujeme zjistit, jací jsou kandidáti na druhé místo, kolik jich je a jak mezi nimi co nejefektivněji vybrat, tedy kolik zápasů je třeba navíc ještě sehrát. Tato úloha je logická, takže nemusíte psát program (draci stejně nedisponují počítači), stačí popsat a zdůvodnit postup určení druhého nejlepšího draka, který by určitě porazil všechny ostatní kromě prvního.

Po zdárném obhájení druhého místa jsem vyrazil. Tížil mě jen pocit, že nevím, jestli za tohle dobrodružství nějaká princezna stojí. Ta poslední, již na svých zádech před 100 lety přitáhl bratranec, byla podle otce nějaká podivná. Možná si jenom na princeznu hrála. Uvidíme, jakou seženu teď.

22-5-2 Stráže údolí — 10 bodů

Nejprve jsem ovšem musel prosvištět kolem stráží hlídajících naše údolí. Dělají nám je havrani rozmístění poblíž ústí do jiného údolí tak, aby byli na přímce. Jenže někteří jsou moc blízko u sebe a mají tendenci se místo hlídání vybavovat, takže jsme se rozhodli počet stráží zredukovat. Nevíme však, které propustit.

Známe polohu všech N havranů na přímce, zadanou celočíselnými mezerami mezi nimi, a chceme jich vyřadit K, aby byli dva nejbližší havrani od sebe co nejdále. Potřebujeme tedy maximalizovat minimální vzdálenost mezi nimi. Dokážete pro nás rychle najít K havranů, jež pošleme do výslužby?

Například pro $N = 6$, $K = 3$ a mezery mezi havrany `4, 6, 2, 5, 7` je správným řešení propustit 2., 3. a 5. havrana (bráno zleva), takže zůstanou mezery `12, 12`. Pro $N = 14$, $K = 7$, mezery `5, 12, 6, 3, 8, 1, 4, 1, 1, 9, 15, 1, 16` vyhodíme 2., 4., 6., 7., 8., 9. a 12. (možností je tentokrát více).

Hned, jak jsem přeletěl lesy, mi něco nepřišlo úplně v pořádku. Žádný drak se totiž nikdy ve svém vyprávění nezmínil o dlouhých černých liánách vedoucích mezi roztodivnými stromy s mnohými tenkými kmeny. Raději jsem je nadletěl.

Jakmile jsem uviděl první lidské osídlení, vlétl jsem do něj jako bouře. Tedy, za pár desítek let už budu dost velký, abych tam vletěl jako bouře. Nicméně jsem způsobil nemalé pozdvižení. Několik lidí sedících u stolu venku všechno převrhlo a dalo se na zmatený únik do domu, jedna ženská zavřeštěla „Zavolejte policii!“ (kdo je to k čertu ta policie?) a mladý pár jdoucí po ulici se pokusil vzít přede mnou nohy na ramena.

Rozhodl jsem se dohonit prchající pár, chytl do svých spárů mladíka a pak ho shodil na zem, aby mi neutekl. Jekot ostatních lidí nabíral na intenzitě.

„Kde najdu krále?“ zařval jsem lidským jazykem.

Kluk měl v obličeji barvu vápence, chvíli vypadal, že strachy zapomněl mluvit. Poté, co jsem se pochlubil malým plamínkem, dokázal ze sebe vyloudit „Co ... Cože? My nemáme krále.“

„Nelži! Kde přebývá princezna?“ Nenechám se přece oblafnout nějakým zbabělcem.

Mladík na mě ještě okamžik zíral a pak ukázal na sever. „Tam ... Támhle. Velké město.“

Ještě jsem mu předvedl, že pro mě není problém zapálit celou vesnici, a rozletěl jsem se.

Než jsem dorazil k tomu městu, setmělo se. Lidé se odjakživa báli tmy, ale netušil jsem, že kvůli tomu rozsvěcují tolik světel. Dokonce i po cestách se pohybovaly zářivé body, asi už dávají louče i na své koně. Mě však víc zaujala zvláštní síť ulic.

22-5-3 Zrcadla — 10 bodů

V téhle okrajové části města byly dost podivné cesty. Tvořily totiž rozlehlou čtvercovou síť, ale domy byly postaveny jen někde. V jednom místě se nacházel velmi silný světelný zdroj záhadného původu svítící pouze jedním směrem.

Vzpomněl jsem si, jak mi strýc vyprávěl o zrcadlech, a napadlo mě, jestli by se nedala využít spolu se zdrojem silného záření na osvětlení nějakého konkrétního domu.

Je tedy dána čtvercová síť o rozměrech $M \times N$, v níž se nachází K domů a jeden další, na který chceme z libovolné strany dosvítit. V jednom bodě máme světelný zdroj, který může svítit pouze vertikálně nebo horizontálně (ve smyslu čtvercové sítě), přičemž směr si můžeme vybrat.

Ptáme se, jestli lze do čtvercové sítě rozmístit na políčka zrcadla tak, aby byl jeden daný dům osvícen, a pokud ano, kolik nejméně jich je potřeba. Domy jsou neprůsvitné a zrcadla umísťována do políček diagonálně (světlo se tedy šíří po čtvercové síti vždy jen horizontálně nebo vertikálně).

Pokud například máme čtvercovou síť o rozměrech 3×3, světlo na souřadnicích `[2, 1]`, dům, jenž chceme osvítit, na `[2, 3]` a další domy na `[1, 2]` a `[2, 2]`, je správným řešením, že stačí umístit dvě zrcadla (na políčka `[3, 1]` a `[3, 3]`).

Stačila chvíle a vlétl jsem přímo nad moře světel. A jak byla některá silná! To musí být ale oheň, který dokáže takhle zářit.

A ta obydlí ... Mnohdy byla velmi vysoká, až se mi nechtělo letět nad ně.

I když je noc, můžou mě takhle klidně zahlédnout, říkal jsem si. Obzvlášť v takovém velkém množství, v jakém jsou dole na náměstí.

22-5-4 Davy lidí — 12 bodů

Jak jsem se díval na stovky lidiček pod sebou, přišlo mi, že část věnuje pozornost soše a druhá část fontáně. Navíc spousta z nich měla za objektem svého zájmu jiného člověka tak, že socha či fontána ležela ve středu úsečky tvořené těmito lidmi.

Mě by zajímalo, jestli se dá dav N lidí, u nichž známe jejich souřadnice v rovině, rozdělit na dvě části, které jsou obě středově souměrné. Střed souměrnosti jedné části tvoří fontána a střed druhé socha, ale jejich souřadnice nejsou zadané. Někdy se čirou náhodou na fontáně či na soše může vyskytovat člověk. Z pohledu draka jsou všichni lidé stejně body, takže jejich rozměry zanedbejte.

Například pro 8 lidí se souřadnicemi `[6, 2]`, `[6, 4]`, `[1, 3]`, `[-4, -2]`, `[-4, 2]`, `[1, -3]`, `[6, -2]`, `[6, -4]` rozdělení na dvě středově souměrné části existuje (1., 3., 4., 5., 6. a 8. člověk v jedné části, 2. a 7. v druhé), ale 6 lidí a souřadnice `[4, 0]`, `[0, 4]`, `[0, 2]`, `[4, 4]`, `[2, 3]`, `[2, 0]` už rozdělit nelze.

Let nad městem mě už začínal docela unavovat, když jsem konečně uviděl hrad, nebo alespoň něco jemu podobného. Každopádně jsem žádné lepší místo, kde hledat princeznu, v okolí neviděl.

S žuchnutím jsem přistál na dlažbě a protáhl si křídla, ve vzduchu jsem byl přeci jenom docela dlouho. Na nádvoří nikdo nebyl, takže jsem prošel branou na další. A tam jsem ji spatřil.

Seděla na lavičce sklopená nad něčím hnědým rozlámaným na kousky a tvářila se nešťastně. Na sobě měla dlouhý černý kabát, vysoké černé lesklé boty, což ladilo k jejím rozpuštěným vlasům barvy temné noci s jediným červeným pruhem.

Sice prý princezny běžně vypadají jinak, ale za 100 let se lidská móda může radikálně změnit. Popošel jsem blíže a oslovil ji.

„Milá princezno, mohu vám s něčím pomoci?“ zeptal jsem se, jak nejgalantněji jsem uměl.

V odpověď jsem dostal škytnutí a smutný pohled. Pak vstala, popošla pár kroků směrem ke mně, ale trochu se motala, takže se jí rozsypaly ty hnědé věci po zemi. Zjevně jí to však nevadilo.

„Potřebuju vyřešit jeden příklad,“ říkala ztěžka. „Na zítra do školy. Jinak ten matfyz neudělam.“

„A když ti pomůžu, poletíš na mých zádech ke mně domů?“

Podívala se na mě a zasmála se. „Jasně. Mimochodem, hustej převlek.“

„Tak povídej.“

22-5-5 Čokolámání — 10 bodů

Princezna dostala něco, čemu se říká čokoláda. Má to obdélníkový tvar a $M \times N$ dílků, které se dají lámat podle os mezi nimi. Ke každé této ose (horizontální i vertikální) měla napsané číslo, totiž cenu zlomu.

Její úkol spočíval v nalezení postupu, jak co nejrychleji zjistit, za jakou nejnižší cenu lze rozlámat čokoládu na jednotlivé dílky. Cena každého zlomu může být započítána vícekrát bez závislosti na délce zlomu. Například rozdělíme-li čokoládu na řádky a ty pak budeme lámat, započítá se cena každého vertikálního zlomu M krát, kdežto každého horizontálního jen jednou.

Tato úloha je praktická a řeší se ve vyhodnocovacím systému CodEx: *http://codex2.ms.mff.cuni.cz/ksp*. Pokud jste zatím žádnou praktickou úlohu neřešili, přečtěte si stručný úvod: *http://ksp.mff.cuni.cz/about/codex.html*.

Chviličku mi úloha zabrala, ale už jsem viděl mnohem těžší. Princeznu moje řešení zjevně nadchlo, vyskočila radostí a pak bez váhání vylezla na můj hřbet.

„Jakého království jsi ty vlastně princezna?“ osmělil jsem se zeptat, když přestala nadšeně vykřikovat nad úžasnou podívanou na noční Prahu, jak město nazvala.

„Já jsem princezna . . . princezna . . . matfyzu! A tohle celé město patří jenom nám! Podívej, támhle v těch vysokých zelených budovách sídlíme.“

Zadíval jsem se na novodobé zelené hrady a přemýšlel, jak radikálně se změnila architektura, když tu náhle se vpravo ozval hukot. Po pár sekundách i za mnou. A vlevo taky.

„Leť, ty můj draku, leť!“ křičela z plna hrdla moje princezna. Strach v jejím hlase jsem však nepostřehl.

Ohlédl jsem se a uviděl spoustu zvláštních ptáků bez křídel. Všichni vyluzovali monotónní hluk. Že by princezna měla až takovouhle ochranu proti drakům?

22-5-6 Hlídači princezny — 13 bodů

Princezna je v království vždy velmi ceněna, a tak má spoustu bodyguardů, strážců a hlídačů. Kdyby se všichni kryli navzájem, vznikl by akorát zmatek, takže každý hlídač kryje právě jednoho jiného.

Do útoku proti nepříteli se vždy posílá co nejvíce hlídačů, ale každý z nich musí být kryt někým, kdo nejde do útoku. Jak najít takové hlídače?

Úlohu si můžete představit jako orientovaný graf, kde z každého vrcholu vede jen jedna hrana (do něj však může vést více hran, nebo žádná). Chceme najít největší množinu vrcholů takovou, že do každého vrcholu z této množiny vede alespoň jedna hrana z vrcholu mimo tuto množinu.

Pokud máme 5 hlídačů (očíslovaných od 1 do 5) a `1 ->` (hlídá) `3`, `2 -> 3`, `3 -> 4`, `4 -> 5` a `5 -> 3`, pak vybereme do útoku hlídače 3 a 5.

„Vyzýváme vás, abyste okamžitě sletěl dolů, jinak bude zahájena varovná palba,“ ozvalo se silným, avšak podivně plochým hlasem zezadu.

„Vyprdni se na ně a ukaž jim, zač je toho loket!“ dodala mi odvahu moje princezna.

„Tak se pořádně drž!“

Prudce jsem slétl dolů. Přímo mezi domy. Neztrácel jsem čas obdivováním výšky zdejších budov a rozletěl se ulicí. Za mnou se však ozval svist hlídačů. Ihned jsem zabočil a uslyšel obrovskou ránu. Jednomu se manévr nepodařil.

Máchal jsem křídly jako nikdy v životě, jenže oni byli pořád za mnou. Drželi se jak klíšťata. Ještě jsem neslyšel, že by něco tak velkého dokázalo stíhat draka.

„Prásk! Prásk! Prásk!“ ozvalo se a já pocítil bolest na levém křídle. Provedl jsem bleskový úhyb.

„Kličkuj!“ zakřičela princezna zděšeně z hřbetu.

Nebylo třeba mě pobízet. O lučištnících jsem slyšel. Jenže tihle mají sakra rychlé šípy.

Rychle jsem zabočil do jedné úzké uličky. „Schovej se támhle,“ zařvala ihned. Neváhal jsem a vletěl do velkých otevřených vrat, až jsem se bouchl jedním křídlem. Po pár sekundách kolem s hukotem prolétlo pár princezniných strážců.

„Tak tady chvíli zůstaneme a pak rychle vyletíme.“

Byl jsem tak zadýchaný, že jsem jenom zakýval hlavou. Na vymýšlení lepšího plánu nebyl čas.

Po několika minutách odpočinku jsme se odvážili vyjít z úkrytu. Ihned jsem vyletěl, jak nejvýše to šlo. Naštěstí nás nezpozorovali, takže jsme se mohli v klidu vzdálit od města a tam si odpočinout. I ona toho měla dost, hned usnula pod mým ocasem suplujícím peřinu.

Další den jsme už naštěstí bez nepříjemných příhod dolétli domů. S princeznou jsem nyní velmi spokojen, je chytrá, akční a šikovná. Taková dračice . . .

22-5-7 ArcheoPaleoLingua — 14 bodů

V této sérii si ještě jednou vyzkoušíme APL, programovací jazyk z dob našich prababiček, pradědečků a pralidí vůbec. K základním operacím a operátorům, které jsme si zavedli v úloze 22-4-7, doplníme pár dalších a naprogramujeme něco krapet složitějšího. A opět bez namáčení, . . . totiž bez cyklů.

Dyadický operátor *replikace* `x/y` dostane dva vektory stejné délky a vytvoří vektor, ve kterém se nejprve vyskytuje `x[0]` kopií prvku `y[0]`, pak `x[1]` kopií prvku `y[1]`, atd. Tedy například `3 0 2/4 5 6 → 4 4 4 6 6`. Pokud se ve vektoru x vyskytují jen nuly a jedničky, replikace vlastně jen vybere ty prvky z y, na jejichž pozici se v x vyskytuje jednička, tedy provede jakousi *kompresi* vektoru y.

Inverzní operací k této kompresi je dyadický operátor *expanze* `x\y`. Ten na vstupu potřebuje vektor x složený z nul a jedniček a vektor y, jehož délka je rovna počtu jedniček v x. Výsledkem je pak vektor vzniklý z x nahrazením každé jedničky odpovídajícím prvkem z y. Kupříkladu tedy platí `0 1 1 0 0 1\4 5 6 → 0 4 5 0 0 6` a také `x/x\y → y`.

Konečně přidáme operátor *scanování* `f\x`, kde f je dyadická operace a x vektor. Dá nám vektor, jehož i-tá složka je redukce `f/` prvních $i+1$ složek vektoru x. Nechme raději hovořit příklad: `+\1 2 3 → 1 (1+2) (1+2+3) → 1 3 6`.

A nyní slíbené úkoly:

Úkol 1 (5 bodů): Napište funkci, která najde všechna prvočísla menší než dané N.

Úkol 2 (4 body): Jak v APL uspořádat daný vektor čísel vzestupně? Slibujeme, že se žádné číslo nebude opakovat. (Zde prosíme používejte pouze podmnožinu APL, kterou jsme nadefinovali v našem seriálu. Operátory ⍒ a ⍋, jakkoliv krásné, se nepočítají.)

Úkol 3 (5 bodů): Je dán vektor nul a jedniček. Jak zjistit délku nejdelšího souvislého úseku tvořeného jedničkami?

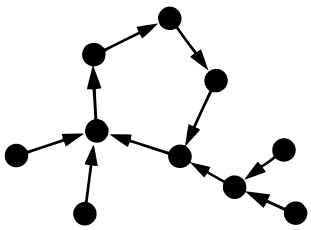
Připomínáme, že u této úlohy na časové ani paměťové složitosti nezáleží, rozhodující je krátkost a elegance vašeho programu. Archeologii zdar.

22-4-1 Zaheslované stránky

Ano, je to úloha na grafy, dokonce orientované![1] Vrcholy jsou stránky, hrany si nastavíme, aby vedly z odemknutelného dokumentu na dokument s příslušným heslem. Potom má graf tu zvláštní vlastnost, že počet hran, které vychází z libovolného vrcholu, je roven 0, nebo 1.

To se bude jednak příjemně implementovat, druhak to silně omezuje strukturu takových grafů. Ptejme se na počet hran u každé komponenty slabé souvislosti (to je taková, která nehledí na orientaci hran) v závislosti na počtu vrcholů komponenty (n).

1) Nejméně to může být $n-1$: v případě, že jde o strom. Odebrání libovolné hrany způsobí rozpad komponenty.
2) Nejvíce to může být n, protože každému vrcholu přísluší nejvýše jeden výstupní konec hrany. Taková komponenta může být tvořena orientovanou kružnicí, vlastnost ale neporušíme ani tím, přilepíme-li k takové kružnici cestu vedoucí do nějakého jejího vrcholu. Situace dokonce může vypadat jako na obrázku:



V obou případech stačí prolomit jedinou stránku. Našemu algoritmu proto stačí spočítat komponenty slabé souvislosti daného orientovaného grafu a určitě bychom to mohli v lineárním čase a prostoru stihli spočítat tak, že bychom si situaci odorientovali a spouštěli do nových a nových vrcholů průchody do hloubky / šířky.

Můžeme to ale udělat jednodušeji. Uložíme-li graf do pole délky n, které na i-tém místě uložíme, do kolikátého vrcholu míří hrana vycházející z i-tého vrcholu (nulu, pokud žádná nevychází), máme k dispozici šikovnou reprezentaci, kterou se snadno projdeme bez použití front a zásobníků: stačí si prstem (proměnnou) ukazovat, kde právě jsme, a jít „rovnou za nosem“.

Začneme-li si ukazovat v libovolném vrcholu komponenty prvního druhu, dostaneme se do kořene stromu, v komponentě druhého druhu se zacyklíme v jejím cyklu. Pokud se nám tyto případy podaří detekovat a zaznamenat do kořene / celé kružnice, že už jsme tam byli, můžeme při takové příležitosti přičíst k počtu nutně prolomených stránek jedničku.

Toto poznamenávání může probíhat ve zvláštním poli, kde si budeme při průchodu přepisovat tři značky: *prolomeno*, *právě lámáno* a *netknuto*. Na počátku je všude poznamenáno *netknuto*, při procházení za sebou pokládáme *právě lámáno*, což po dokončení průchodu zaměníme za *prolomeno*. Cyklus pak snadno detekujeme tím, že narazíme na značku *právě lámáno*, dojdeme-li někdy do situace *prolomeno*, můžeme skončit bez navýšení počtu nutných prolomení.

Ještě jednoduší je přepisovat uložený graf. Pěknou implementaci na tomto základě napsal Dominik Smrž – autorské řešení využívá jeho nápadu se zápornými čísly vrcholů coby identifikátory průchodů. Určitě si ho projděte, je krátké.

Lukáš Lánský

22-4-2 Rozvoz zásilek

Nejprve pár slov o tom, jak úlohu neřešit. Ne vždy je dobrý nápad zkontrolovat úsek, kterým prochází největší počet dosud nezkontrolovaného zboží – mohlo by se to totiž v budoucnu vymstít. Ukažme si to na protipříkladu (ve formátu zadání úlohy): `1:1->2, 2:1->4, 2:3->6, 1:5->6` se dvěma skenery k dispozici. V prvním kroku je sice nejvýnosnější zkontrolovat úsek `3-4`, ale tím dostanu řešení o nejvýše 5 zkontrolovaných kusech, zatímco optimální řešení kontrolující úseky `1-2` a `5-6` dodá luxusních 6. Aneb důkazy nejsou pro blaho opravovatelů, ale pro to, abyste si mohli být jistí funkčností Vašich řešení.

Když tedy na první pohled nevíme, které řešení je nejlepší, zkusíme preventivně všechny. A jak říká známe přísloví známé už z dob počítání na prstech, „od backtracku k dynamice krůček.“ Budeme si tedy počítat, kolik nejvýše zásilek zvládneme zkontrolovat pomocí `k` skenerů na stanicích `1..i`, a to v (dvourozměrném) poli `soucet`; pak optimální řešení úlohy zkontroluje právě `soucet[N][S]` kusů zboží.

Jak spočísti hodnotu `soucet[i][j]`? Inu, pro `i=1`, tedy jeden skener, je otázka jednoduchá, prohlédneme všechny stanice v intervalu `1..j` a podíváme se, ve které užitím skeneru zkontrolujeme nejvíce zboží – přesně to je hodnota `soucet[1][j]`. Pro více skenerů už to ale takto jednoduše nejde, viz první odstavec. Každopádně si můžeme říci – buďto oskenujeme úsek `j-1:j` (poslední úsek intervalu), nebo ne; spočítáme nejlepší řešení pro obě situace a pamatujeme si jen to lepší. V druhém případě vlastně skenujeme jen na intervalech `1..j-1`, tedy počet kusů je `soucet[i][j-1]`. Prvý případ je drobet složitější, na počtu zboží, které oskenujeme v `j`-té stanici, se totiž promítne, jaké předchozí stanice jsme již oskenovali. Každopádně to opět můžeme vzít hrubou silou, tedy postupně uvážit, že před `j`-tou stanicí jsme jako poslední oskenovali `k`-tou pro všechna `k` menší než `j`, a z těchto si vzít nejlepší řešení.

Nechť `cena[k][j]` říká, kolik zboží se proskenuje úsekem `j-1:j`, byl-li nejlevější proskenovaný úsek `k-1:k`; pak by předchozí myšlenka vypadala v pseudokódu následovně:

```
l = 0;
for(k=0;k<j;k++)
  l = max(l,soucet[i-1][k] + cena[k][j]);
soucet[i][j] = max(soucet[i-1][j],l);
```

Bystrý čtenář si jistě klade otázku, jak víme, že v optimálním řešení pro `i-1` skenerů na stanicích `1..k` bude poslední skener právě na pozici `k-1:k`, to jest, že můžeme počítat `cena[k][j]` a ne nějakou nižší. Inu, my to nevíme, ale je nám to jedno. Ono se bude stávat, že skener nebude na poslední pozici a my tak vlastně v tomto kroku dostaneme horší řešení; jenže z nějakého předchozího kroku už máme lepší.

Zbývá vysvětlit, jak efektivně spočteme `cena[i][j]`. Dělat to v průběhu počítání není ono, protože spoustu věcí

[1] Kuchařka o grafech: *http://ksp.mff.cuni.cz/tasks/20/cook3.html*

bychom dělali mockrát. Raději si hodnoty spočítáme dopředu, všechny najednou. Nejprve si intervaly dopravy zboží setřídíme vzestupně dle jejich koncových zastávek. Pak postupně pro všechny úseky `j-1:j` provedeme následující: do fronty si naházíme všechny intervaly dopravy zboží, které procházejí přes `j-1:j`, a při tom si spočteme celkový počet zboží `m` v těchto intervalech. Pak budeme postupně brát úseky `i-1:i > j-1:j` a spočteme `cena[j][i]`. Z fronty vyházíme ty intervaly, které už do `i-1:i` nezasáhnou (takové se nacházejí jen a pouze na začátku fronty), a o počet zboží v nich snížíme `m`, `cena[j][i]` je pak celková suma zboží v intervalech procházejících přes `i-1:i` minus `m`. Pro podrobnosti viz zdrojový kód.

Jak se přesvědčit, že algoritmus je správně? Inu, indukcí podle počtu skenerů a zastávek. Pro jeden skener a libovolně zastávek náš algoritmus určitě funguje. Udělejme tedy indukční krok pro `N` skenerů a `S` zastávek, za předpokladu, že pro menší počet skenerů a zastávek algoritmus funguje. Pokud optimální řešení neskenovalo poslední zastávku, tak tak neučiní ani náš algoritmus, neboť si všimne, že `soucet[N][S-1]` je větší než libovolné řešení skenující poslední zastávku (z indukčního předpokladu), a vítězíme. Pokud optimální řešení poslední zastávku skenovalo, podíváme se, jakou zastávku skenovalo jako předposlední, nechť je to `k`-tá. Pak ono optimální řešení zkontroluje nejvýše `soucet[N-1][k] + cena[k][j]` zásilek (z indukčního předpokladu), ale přesně tohle náš algoritmus vezme v úvahu. Celý důkaz je v podstatě jen přečtením toho, co algoritmus dělá, tak už to u (fungujících) dynamik bývá :).

Ještě se stručně zamysleme nad tolik omílanou časovou složitostí. Předpočet pole `cena` nás stojí $\mathcal{O}(S \cdot (S+P))$ – kde S je počet stanic a P počet zásilek zboží – neboť pro každý úsek `j-1:j` postupně do fronty naházím a vyházím až P prvků a při tom kouknu na nejvýše S zastávek. A počítání pole `soucet` je s časem na $\mathcal{O}(N \cdot S^2)$, neboť pro každý počet skenerů počítám každý úsek `i-1:i` kouknutím na až S předchozích úseků. S pamětí se vejdeme do sympatických $\mathcal{O}(S^2+P)$. Za domácí úkol si zkuste řešení upravit tak, aby dělalo to co má, tedy neodpovídalo jen váhou optimálního řešení, ale ono řešení přímo vypsalo.

Ať Vás provází dynamika.
Vojta Tůma

22-4-3 Muzeum

Úloha se dala řešit vylepšeným algoritmem na hledání mostů z grafové kuchařky. Tady si ukážeme jeden trošku elegantnější postup.

Všimneme si, že centrální kamera nemůže ležet na kružnici. To je vidět z toho, že má stupeň rovný počtu komponent a mezi komponentami nemůže vést hrana. Další pozorovací cvičení: Když se jedná o biologické oddělení, tak nám stačí znát kostru grafu (na hranách, které tvoří kružnice, nezáleží, protože ty jsou jen uvnitř komponent). Do třetice si všimneme, že jen z centrálního vrcholu můžeme prohledat $(N-1/K)\cdot(K-1)+1$ kamer[2], a to tehdy, když tam nezačínáme s hledáním.

Při řešení budeme nejprve předpokládat, že jsme správně u biologů, pak najdeme centrální kameru a nakonec ověříme, že jsme tam skutečně správně byli. Začneme tedy hledat třeba do hloubky a při návratu počítat navštívené vrcholy. Pokud narazíme na takový, který je podezřelý z centrálnosti, poznamenáme si ho. Teď už jen ověření. Kupodivu stačí to stejné prohledávání. Centrální vrchol si označíme za navštívený a pro každý z jeho sousedů zkontrolujeme, že z něj jde prohledat právě tolik vrcholů, kolik se na slušnou komponentu patří.

Jitka Novotná

22-4-4 Ořez zárodků

Označme si mateřský strom A, odvozený B. Začneme drobným pozorováním: Pokud ve stromě A najdeme posloupnost bratrských podstromů, která odpovídá podstromům synů kořene B, potvrdili jsme odvození B od A. Je-li x kořenem stromu X, jeho bratrským podstromem přirozeně rozumíme podstrom s kořenem y, kde y je bratrem x. Jaký strom zvolit jako mateřský? Zřejmě ten, který obsahuje více vrcholů. Každé „osekání“ pouze vrcholy odebírá. Pokud jich mají po „osekání“ stejně, musí být stromy identické a uvedené pozorování nadále platí.

Jak efektivně hledat posloupnost podstromů synů kořene B v A? Uděláme cimrmanovský krok stranou, vyhneme se znovuobjevování kola a převedeme problém na hledání podřetězce v řetězci. Ano, kuchařku jste si měli přečíst ...

Zbývá najít vhodnou reprezentaci stromu pomocí řetězce. Odpověď je triviální – použijeme uzávorkované výrazy. List je reprezentovaný pomocí `()` Každý jiný vrchol (včetně kořene) pak jako `(` *=reprezentace 1. syna= =reprezentace 2. syna= ... =reprezentace posledního syna=* `)`. Dva malé stromečky ze zadání této úlohy jsou pak reprezentovány například takto: `(((()())()))` a `((()()()))`.

Zřejmě každý strom má nějakou reprezentaci. Platí také, že je reprezentací strom jednoznačně určen? To snadno dokážete pomocí indukce. Pro list to platí a dále postupně podle složitosti vrcholu ... Zkuste si to rozmyslet. Také platí, že každý správně uzávorkovaný výraz (v běžném slova smyslu) reprezentuje nějaký strom. Pokud tedy vezmeme několik správně uzávorkovaných výrazů a „slepíme“ je za sebe do řetězce q, reprezentují posloupnost nějakých stromů $Y_1, Y_2, \ldots, Y_n$. Pokud se navíc q vyskytuje v reprezentaci nějakého stromu X, našli jsme uvnitř X interval sousedících bratrských podstromů $Y_1, \ldots, Y_n$.

Ať to tedy uzavřeme: Vezměme reprezentaci B a odštípněme vnější závorky (tj. získáme „slepenec“ reprezentací podstromů jeho synů), označme jako q. Pokud nalezneme q v reprezentaci stromu A, platí, že B je odvozený od A, v opačném případě nemůže být B od A odvozen.

Cože? Ještě jste si tu kuchařku nepřečetli a nevíte jak najít q v reprezentaci A? Přece pomocí vynálezu pánů Knutha, Morrise a Pratta ... algoritmem KMP.

Čas, paměť? Trvání výroby řetězcové reprezentace stromu a její velikost jsou lineární vzhledem k počtu vrcholů stromu. KMP běží v lineárním čase se součtem délek řetězců (jehly i kupky sena :o)). Časová i prostorová složitost algoritmu je tedy $\mathcal{O}(N)$, kde N budiž součtem počtu vrcholů obou stromů.

Pepa Pihera

22-4-5 Energetické články

Úloha měla přísné limity v zadání, ale samotná data byla většinou přísná jen v jednom parametru, a tak jistě prošla i asymptoticky ne tak dobrá, leč vynalézavá řešení. A tak to u praktických úloh má být!

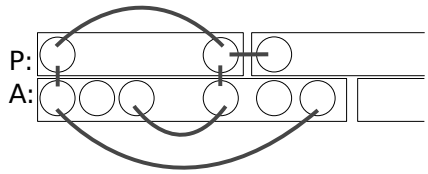
[2] to je počet všech kamer bez jedné komponenty

Pojďme na řešení: nejprve bylo důležité si všimnout, že pokud zřetězíme dva palindromy za sebe a vznikne další palindrom, musí se stát, že oba palindromy mají společný „základ“. Pokud je kratším palindromem `ABA`, pak delší palindrom (aby bylo zřetězení palindromem) musí začínat `ABA` (má tvar `ABAx`), a protože delší palindrom je také palindrom, bude i končit `ABA` (tvar `ABAxABA`) a dále pokračujeme stejným argumentem (indukcí). Není pravda, že jeden je mocninou druhého (například pro `ABAABA` a `ABAABAABA`), ale společný kořen jistě mají.

Hodilo by se nám tedy počítat **kořen** palindromu, tedy nejkratší řetězec takový, že jeho vhodným umocněním získám celý palindrom. Takový kořen je sám palindromem. Umíme-li kořen najít, pak stačí pro vstupní palindromy vždy najít kořeny, tyto kořeny jednoduše zpracovat (ať už pomocí hashování nebo pomocí setřízení, jako je to předvedeno ve vzorovém řešení) a správně napočítat.

Teď na tu pořádnou práci: jak co nejrychleji spočítat kořen palindromu? Půjdeme na to stejně, jak popisoval Vojta na KSPáckém fóru: budeme postupně procházet řetězec a průběžně si přepočítávat, jak by takový kořen mohl být dlouhý. Zpět se už ohlížet nebudeme – čteme-li Z-té písmenko a myslíme si, že kořen je P znaků dlouhý, podíváme na (Z mod P)-té písmenko a porovnáme. (Vězte, že je to takové písmeno, jež musí být stejné se Z-tým písmenkem, aby opravdu platilo, že kořen má délku P.)

Když se písmenka neshodují, můžeme prodloužit periodu P na $Z-1$ nebo Z. Nemůže se nám stát, že se nám schovává perioda někde mezi P a $Z-1$. To se nahlédne podobně jako první pozorování: byla-li by tam taková perioda A, pak v periodě A je 1., A-té, $A+1$-ní a P-té písmeno shodné. Protože A je menší než $Z-1$, víme, že A-té písmenko není P-té ani 1., a to nám dá další shodnou dvojici ... a nakonec dostaneme, že by všechna písmenka musela být shodná.



Co se týče časové složitosti, tak pokud využijeme chytrou hashovací funkci se zhruba konstantní časovou složitostí na přístup (a pamatujeme si neprázdná políčka), pak se (nikoli v nejhorším případě) dostaneme k $\mathcal{O}(V+N)$, kde N je počet řetězců a V je velikost vstupu (sice mohla být až $N \cdot K$, ale rychleji by to stejně nešlo). Vzorové řešení používá mnoho asymptoticky záludných situací (například porovnávání řetězců nebo jejich třídění zabudovanou knihovnou), ale přesto jsme se jej rozhodli takto zveřejnit. Ukazuje totiž vlastnost mnoha praktických a soutěžních úloh – občas si můžete dovolit zkusit vyměnit nejlepší konstantu či asymptoticky horší složitost za čitelnost a jednoduchost kódu, pokud je hlavní podmínkou „vejít se“ do časového limitu.

Martin Böhm & CodEx

22-4-6 Umisťování panelů

Najskôr sa zamyslime nad tým, ako vyzerajú jednotlivé obdĺžniky, ktoré vystupujú v nejakom riešení, a ako ich rýchlo všetkých nájsť.

Tieto obdĺžniky sú najmenšie obdĺžniky, ktoré obsahujú množinu nejakých K bodov. Teda určite platí, že ohraničujúce x-ové a y-ové súradnice každého obdĺžnika sú rovnaké ako príslušné súradnice nejakých bodov, ktoré daný obdĺžnik obsahuje. Pretože inak by sme mohli jednoducho obdĺžnik v príslušnom smere zmenšiť a dostali by sme menší obdĺžnik, obsahujúci tie isté body.

Teda všetkých obdĺžnikov je maximálne $\mathcal{O}(N^4)$, pretože toľko je kombinácií, ako vybrať ohraničujúce body obdĺžnika. A pre každý obdĺžnik vieme v čase $\mathcal{O}(N)$ skontrolovať, či obsahuje práve K bodov.

V skutočnosti však počet všetkých možných obdĺžnikov obsahujúcich K bodov je rádovo menší. Ak totiž máme danú spodnú a hornú y-ovú súradnicu obdĺžnika a tiež ľavú x-ovú súradnicu obdĺžnika, potom existuje nanajvýš jedna možná hodnota x' pre pravú x-ovú súradnicu obdĺžnika tak, že obsahuje K bodov. Pretože obdĺžniky s menšou pravou x-ovou súradnicou ako x' obsahujú menej ako K bodov a obdĺžniky s väčšou pravou x-ovou súradnicou ako x' obsahujú viac bodov ako K.

Teda počet všetkých korektných obdĺžnikov je len $\mathcal{O}(N^3)$. Ak si naviac na začiatku usporiadame body podľa x-ovej súradnice, tak pre každú kombináciu hornej a dolnej súradnice y a ľavej súradnice x obdĺžnika vieme nájsť odpovedajúcu pravú stranu obdĺžnika v čase $\mathcal{O}(N)$. To spravíme jednoducho tak, že budeme postupne pridávať body do obdĺžnika, pokiaľ nedosiahneme počet K.

Ku vzorovému riešeniu však potrebujeme nájsť všetky korektné obdĺžniky rýchlejšie, a to v čase $\mathcal{O}(N^3)$. To vyriešime metódou „okienka“ (sliding window) nasledovne:

Najskôr si body znova usporiadame podľa x-ovej súradnice. Zafixujeme si hornú a dolnú súradnicu y a následne jedným prechodom nájdeme pravý okraj ku všetkým obdĺžnikom, ktoré majú príslušnú hornú a dolnú y-ovú súradnicu a ďalej budeme teda pracovať len s bodmi ležiacimi v tomto páse.

Najskôr nájdeme štandardným spôsobom (pridávaním bodov, pokiaľ nedosiahneme K) pravý okraj k obdĺžniku, ktorý má ľavý okraj na prvej x-ovej súradnici. Následne sa presunieme na ďalšiu ľavú x-ovú súradnicu a vieme, že príslušný pravý okraj leží napravo od pravého okraja posledného nájdeného obdĺžnika. Teda ho nájdeme znova pridávaním bodov, ale začíname od posledného nájdeného okraju. Všimnime si, že súradnice všetkých ľavých a pravých okrajov nájdených obdĺžnikov navzájom tvoria neklesajúcu postupnosť, a teda pri tomto hľadaní spravíme spolu lineárne veľa práce.

Teraz môžeme pristúpiť k vyriešeniu celej úlohy a zamyslieť sa, čo to znamená nájsť dva disjunktné osovo-paralelné obdĺžniky, pričom každý obsahuje práve K bodov.

V podstate to znamená to, že v každom takomto riešení existuje deliaca horizontálna alebo vertikálna priamka taká, že jeden z obdĺžnikov je na jednej strane tejto priamky a druhý je na opačnej strane priamky.

Bez ujmy na všeobecnosti teda predpokladajme, že pre optimálne riešenie je táto priamka horizontálna. (Vyriešením rovnakej úlohy pre vertikálnu priamku a vybratím lepšieho riešenia dostaneme celkové optimum.)

Teda pre hľadané dva obdĺžniky musí platiť, že horná y-ová súradnica jedného je menšia ako dolná y-ová súradnica druhého.

Pre každú y-ovú súradnicu si teda spočítame najmenší obdĺžnik, ktorý má na tejto pozícii dolný okraj, a taktiež najmenší obdĺžnik majúci na tejto súradnici horný okraj. Následným vyskúšaním všetkých dvojíc dolného a horného

okraju nájdeme optimálne riešenie.

Časová zložitosť spočíva z $\mathcal{O}(N \log N)$ na utriedenie bodov na začiatku, $O(N^3)$ na nájdenie všetkých korektných obdĺžnikov a nakoniec $O(N^2)$ na vyskúšanie všetkých kombinácií dolného a horného okraja dvoch obdĺžnikov. Teda spolu $O(N^3)$.

Pamäťová zložitosť nám vystačí lineárna, ak si pri hľadaní obdĺžnikov súčasne vytvárame aj tabuľku obvodu najmenšieho obdĺžnika majúceho daný horný a dolný okraj, pričom túto tabuľku ihneď po nájdení obdĺžnika zaktualizujeme a samotné obdĺžniky si teda nemusíme pamätať.

Peter Ondrúška

22-4-7 Pozdrav z pravěku

Úkol 1: Nabízí se `x÷|x`, tedy dělit číslo jeho absolutní hodnotou, ale to selže pro $x = 0$. Lepší je využít toho, že porovnávání vrací 0 nebo 1 a použít `(x>0)-(x<0)`.

Úkol 2: Jelikož APL vyhodnocuje zprava doleva, `⍳5+1` je totéž co `⍳6`, tedy vektor `0 1 2 3 4 5`. Naproti tomu `1+⍳5` je totéž co `1+0 1 2 3 4`, což je podle pravidel o počítání s vektory `1 2 3 4 5`.

Úkol 3: `⍳(2+⍳3) → ⍳(2+0 1 2) → ⍳(2 3 4)`, což vytvoří trojrozměrné pole tvaru $2 \times 3 \times 4$ vyplněné čísly od 0 do 23.

Úkol 4: Stačí `/⌈/⌈x` – nejdříve nám `/⌈x` dá vektor maxim sloupečků a z něj si pak druhou redukcí vybereme maximum.

Úkol 5: Všimněme si, že jednička má být právě tam, kde je první souřadnice (číslo řádku) menší než druhá. Stačí tedy použít direktní součin `(⍳n)∘.<(⍳n)`.

Úkol 6: Nejprve vytvoříme matici

$$\begin{pmatrix} 0 & 1 & 2 & 3 & \dots & n-2 & n-1 \\ n-1 & n-2 & n-3 & n-4 & \dots & 1 & 0 \end{pmatrix}$$

laminací vektorů `⍳n` a `(n-1)-⍳n`. Pak ji stačí transponovat a operátorem `⍴` přeformátovat na vektor délky $2n$. Celý program tedy zní `(2× n)⍴⍉(⍳n)⍪((n-1)-⍳n)`.

Úkol 7: Nebudeme troškaři, najdeme rovnou všechny společné dělitele zadaných čísel x, y a pak z nich vybereme toho největšího:

```
a ← 1+⍳n
p ← 0=a|x
q ← 0=a|y
r ← a×p×q
d ← /⌈r
```

Jak to funguje? Nejprve sestrojíme vektor a obsahující čísla $1, \dots, n$. Další vektor p obsahuje jen nuly a jedničky, přičemž jedničky jsou přesně na místech dělitelů čísla x (spočítáme zbytky a porovnáme je s nulou). Podobně q indikuje dělitele čísla y. A vektor r vznikne z a vynulováním těch čísel, která nejsou společnými děliteli x a y, takže už stačí najít maximum z jeho prvků.

Program ještě můžeme trochu zkrátit:

```
/⌈(0=a|x)×(0=a|y)×a←1+⍳n.
```

Filip Hlásek vymyslel ještě magičtější řešení:

```
+/0=y|x×1+⍳y,
```

zkuste přijít na to, jak funguje. Poradíme vám, že se k tomu hodí rovnost $xy = nd$, kde d je největší společný dělitel a n nejmenší společný násobek.

Martin Mareš

Vzorové programy

22-4-1 Zaheslované stránky C

```
#include <stdio.h>
int main()
{
  int tmp;

  printf("Dokumentů?\n");
  int n; scanf("%d", &n);
  int D[n+1];
  for (int i=1; i<=n; i++)
    D[i] = 0;

  for (int i=1; i<=n; i++)
  {
    printf("Dokument %d\n -- kolik hesel obsahuje?\n", i);
    int h; scanf("%d", &h);
    printf(" -- které dokumenty tato hesla otevírají?\n");
    for (int j=0; j<h; j++)
    {
      scanf("%d", &tmp);
      D[tmp] = i;
    }
  }

  int pruniku = 0;
  for (int i=1; i<=n; i++)
    if (D[i] >= 0) // pokud jsme tu ještě nebyli
    {
      int j = i; // projdeme odsud graf
      while (D[j] > 0)
      {
        tmp = j;
        j = D[j];
        D[tmp] = -i;
      }

      if (D[j] == -i) pruniku++; // našli jsme... kružnici!
      if (D[j] == 0 ) { // ... vrchol s výstupním stupněm 0!
        pruniku++;
        D[j] = -i; }
    }

  printf("Je potřeba %d průniků.\n", pruniku);
  return 0;
}
```

22-4-2 Rozvoz zásilek C

```
#include <stdio.h>
#include <stdlib.h>
#define MAXN 1000
#define MAXP 1000
#define MAXS 1000

/* soucet[i][j] = nejlepší součet zkontrolovaného
 * zboží v intervalu stanic 1..j pomocí i scanneru */
int soucet[MAXN][MAXS];

/* cena[i][j] = kolik zásilek by zkontroloval
 * scanner na úseku j-1:j, pokud nejbližší scanner
 * před ním je na posici i-1:i;
 * cena[i][i] = kolik zásilek by zkontroloval
 * scanner na posici i-1:i, pokud před ním žádný jiný
 * scanner není */
int cena[MAXS][MAXS];

int zas_odkud[MAXP], zas_kam[MAXP], zas_kolik[MAXP];
int tmp[MAXP];

int main(){
  int i,j,k,l,m;
  int N, S, P; /* počet scannerů, stanic, zásilek */
  scanf("%d %d %d",&N,&S,&P);

  for (i=0; i<P; i++)
    scanf("%d %d %d",
        &zas_kolik[i], &zas_odkud[i], &zas_kam[i]);
  /* nechť jsou zastávky vzestupně setříděny dle zas_kam */

  /* předpočet pole cena */
  /* nejprve členy na diagonále, tj. cena[i][i] */
  for (i=0;i<S;i++) {
    cena[i][i] = 0;
    for (j=0; zas_kam[j]<i; j++);
    for (; j<P; j++)
      if (zas_odkud[j]<i) cena[i][i] += zas_kolik[j];
  }

  /* nyní spočteme zbytek pole cena */
  for (i=0; i<S-1; i++) {
    k = 0;
    for (j=0; zas_kam[j]<i; j++);
    for(; j<P; j++) if (zas_odkud[j] < i) tmp[k++] = j;
    m = cena[i][i];

    for(l=0, j=i+1; j<S; j++) {
      /* z fronty vyhodíme nesmyslné */
      for(; zas_kam[tmp[l]]<j && l<k; l++)
        m -= zas_kolik[tmp[l]];
      cena[i][j] = cena[j][j] - m;
    }
  }

  soucet[1][0] = cena[0][0];
  for(i=1; i<S; i++)
    soucet[1][i] = (soucet[1][i-1] > cena[i][i] ) ?
                            soucet[1][i-1] : cena[i][i];

  for(i=2; i<=N; i++) {
    /* ať v příštím kroku nečteme nedefinované */
    soucet[i][i-2] = 0;

    for(j=i-1; j<S; j++){
      soucet[i][j] = soucet[i][j-1];
      for(k=0; k<j; k++){
        l = soucet[i-1][k] + cena[k][j];
        soucet[i][j] = soucet[i][j]>l ? soucet[i][j] : l;
      }
    }
  }
  printf("%d\n", soucet[N][S-1]);
}
```

22-4-3 Muzeum C

```
#include <stdio.h>
#define N_Max 20
#define M_Max 30
int N,K;
int sousedi[2*M_Max];
int zacatky[N_Max];
int navstiveno[N_Max];
int podezrely=0;

int prohledej(int poc) {
  int i;
  int prohledano=1;

  navstiveno[poc]=1;
  for (i=zacatky[poc]; i<zacatky[poc+1]; i++)
    if (!navstiveno[sousedi[i]])
      prohledano += prohledej(sousedi[i]);
  if ( prohledano == (N-1)/K*(K-1)+1 )
    podezrely = poc;
  return prohledano;
}

int main() {
  int vrchol,pocet,zacatek,i,j;
  //načtení seznamu sousedů
  //vrcholy jsou číslované od 0
  //první číslo na řádku je počet
  scanf("%d %d",&N,&K);
  zacatek=0;
  for (vrchol=0,zacatek=0; vrchol<N; vrchol++) {
    zacatky[vrchol]=zacatek;
    scanf("%d",&pocet);
    for (i=0; i<pocet; i++) {
      scanf("%d",&sousedi[zacatek]);
      zacatek++;
    }
  }
  zacatky[N]=zacatek;
  //kontrolní výpis
  for (i=0;i<N;i++) {
    printf("%d: ",i);
    for(j=zacatky[i]; j<zacatky[i+1]; j++)
      printf("%d ",sousedi[j]);
    printf("\n");
  }

  //najdeme vrchol podezřelý z toho, že je centralní
  //pokud to byla 0, tak tam zůstane
  for (i=0; i<N; i++) navstiveno[i]=0;
  prohledej(0);
  printf("%d\n",podezrely);

  //ověříme, zda je podezřelý skutečně centrální
  for (i=0; i<N; i++) navstiveno[i]=0;
  navstiveno[podezrely]=1;
  for (i=sousedi[podezrely]; i<sousedi[podezrely+1]; i++)
    if ( prohledej(i) != (N-1)/K ) {
      printf("Toto není bio oddělení.\n");
      return 0;
    }
  printf("Toto je bio oddělení s centralní kamerou %d.\n",
                        podezrely);
  return 0;
}
```

22-4-5 Energetické články C++

```
// Řešení využívá datové struktury a algoritmy knihovny
// STL v C++, často povolená a užitečná na soutěžích

#include <cstdio>
#include <algorithm>
#include <vector> // dynamické pole
#include <string> // řetězce s proměnnou délkou
#include <cstring>

#define MAX 1000001

using namespace std;

int koren(const char* slovo, int delka)
{
    int p = 1, i = 0, d = delka / 2;

    for (int z = 1; z < delka; z++)
    {
        if (slovo[z] == slovo[i % p]) // perioda funguje dále
            i++;
        else if (slovo[z] == slovo[0])
        {
            p = z;
            i = 1;
        } else {
            p = z + 1;
            i = 0;
        }

        if (p > d)
            return delka;
    }

    return p;
}

int main(void)
{
  int n = 0, dl = 0, soucet = 0, souv = 0;
  FILE *fin, *fout;
  vector<string> koreny; // pole řetězců == kořenů
  char slovo[MAX];

  fin = fopen("clanky.in", "r");
  fscanf(fin, "%d", &n);
  for (int i = 0; i < n; i++)
  {
    fscanf(fin, "%d %s\n", &dl, slovo);
    slovo[koren(slovo,dl)] = 0; // uřízne zbytek slova po kořenu
    koreny.push_back(slovo); // vloží kořen do pole řetězců
    // nepamatujeme si celý vstup, ale pouze kořeny
  }
  fclose(fin);
  sort(koreny.begin(), koreny.end()); // setřídí kořeny

  soucet = n;
  for (int i = 1; i < n; i++)
  {
    if (koreny[i-1] == koreny[i])
    {
        souv++;
        soucet += 2 * souv; // přičti 1 za každý stejný kořen dříve
    }
    else souv = 0;
  }

  fout = fopen("pocet.out", "w");
  fprintf(fout, "%d\n", soucet);
  fclose(fout);

  return 0;
}
```

```
program OrezZarodku;
const MaxN = 1000;
      Open = true; Close = false; {otevřená a zavřená závorka}

var t:array[1..2, 1..MaxN * 2] of boolean; {reprezentace 2 stromů, max 2N závorek}
    tlen : array[1..2] of Integer; {délky reprezentací}

procedure ReadTree(tIndex: Integer);
var n: Integer; {vrcholů}
    indices: array[1..MaxN+1] of integer; {začátky seznamů v edge}
    edge: array[1..MaxN] of integer; {seznam hran}
    v: Integer; {vrchol}
    index: Integer;
    stackE, stackV : array[1..MaxN] of integer; {zásobník hran a vrcholů}
    stackDepth : Integer; {hloubka zásobníku}
begin
  {načteme seznamy potomků}
  write('Vrcholů: '); readln(n);
  index := 1;
  for v:= 1 to n do begin
    write(v,': ');
    indices[v] := index;
    while not eoln do begin
      read(edge[index]);
      index := index + 1;
    end;
    readln;
  end;
  indices[n+1] := index;

  {pomocí zásobníku převedeme na závorkovou reprezentaci}
  stackDepth := 1; {na počátku je v zásboníku jeden vrchol}
  stackV[1] := 1; {je to kořen = 1}
  stackE[1] := 1; {a máme se vydat jeho prvním potomkem}
  t[tIndex, 1] := Open; {zapíšeme si za něj otevírací závorku}
  index := 2; {další závorky píšeme od pozice 2}
  while stackDepth > 0 do begin
    v := stackV[stackDepth]; {vrchol na vrcholu zásobníku :-) }
    if stackE[stackDepth] <= indices[v + 1] - indices[v] then begin
                {ještě jsme neprošli nějakou hranu z vrcholu}
      t[tIndex, index] := Open; {zapíšeme otevírací závorku}
      index := index + 1;

      stackV[stackDepth + 1] := edge[indices[v] + stackE[stackDepth] - 1];
      stackE[stackDepth + 1] := 1; {nový vrchol má zpracovat prvního syna}
      stackE[stackDepth] := stackE[stackDepth] + 1; {další hrana hotová}
      stackDepth := stackDepth + 1;
    end else begin
                        {opouštíme vrchol}
      t[tIndex, index] := Close;
      index := index + 1;
      stackDepth := stackDepth - 1;
    end;
  end;

  tlen[tIndex] := n * 2; {délka reprezentace}
end;

var A, B: Integer;
    KMP : array[1..2 * MaxN] of integer;
    i, j : Integer;

begin
  ReadTree(1);
  ReadTree(2);
  if tlen[1] >= tlen[2] then begin {vetší strom bude rodič}
      A := 1;
      B := 2;
    end else begin
      A := 2;
      B := 1;
    end;

  {ořízneme vnější pár závorek (jde to bez toho, ale didaktické účely...)}
  for i := 2 to tlen[B] - 1 do
   KMP[i - 1] := KMP[i];
  tlen[B] := tlen[B] - 2;

  if tlen[B] = 0 then begin {Ha, test case!}
```

```
    writeln('Strom ', B,' je výsledkem ořezu stromu ', A,'.');
    exit;
   end;

  {Příprava KMP <- hledáme strom B, stavíme záchranou funkci}
  KMP[1] := 0;
  for i := 2 to tlen[B] do
   if t[B, i] = t[B, KMP[i - 1] + 1] then
      KMP[i] := KMP[i - 1] + 1 else KMP[i] := 0;

  {Hledani, j = pozice v B, i = pozice v A}
  j := 0;
  for i := 2 to tlen[A] - 1 do {první a poslední závorku lze vynechat}
   if t[A, i] = t[B, j + 1] then begin
     j := j + 1;
     if j = tlen[B] then begin
       writeln('Strom ', B,' je výsledkem ořezu stromu ', A,'.');
       exit;
     end;
   end else
   while (j > 0) and (t[A, i] <> t[B, j + 1]) do
     j := KMP[j];

   {sem se dostane, jen když B v A nenajdeme}
   writeln('Nejedna se o předka a potomka ve smyslu ořezávání.');
end.
```

22-4-6 Umisťování panelů — C++

```
#include <iostream>
#include <algorithm>
using namespace std;
#define INF 1000000000
#define MAXN 300
int N, K, sirka, vyska;
pair<int,int> body[MAXN];
// najmenšie obdĺžniky, ktoré majú hornú
// resp. dolnú hranicu na danej súradnici
int minNad[MAXN];
int minPod[MAXN];

int ries() {
  // usporiadame body podľa x-ovej súradnice
  sort(body,body+N);

  for (int i = 0; i < N; i++) minNad[i] = minPod[i] = INF;

  for (int a = 0; a < N; a++) {
           // dolná hranica okienka pre y
    for (int b = 0; b < N; b++) {
             // horná hranica okienka pre y
      int y1 = body[a].second;
      int y2 = body[b].second;

      // vyfiltrujeme si body, ktoré su medzi hranicami
      pair<int,int> bodyMedzi[MAXN];
      int M = 0;
      for (int i = 0; i < N; i++) {
        if (y1 <= body[i].second && body[i].second <= y2)
          bodyMedzi[M++] = body[i];
      }

      // vybrané body prejdeme metódou okienka
      int i = 0, j = 0; // ľavá a pravá hranica okienka
      while (i < M) {
        int x1 = bodyMedzi[i].first;
        int x2 = bodyMedzi[j].first;

        // ak je v okienku menej ako K bodov,
        // posunieme pravý okraj okienka a započítame
        // všetky nové body, ktoré doň vstúpia
        while (j < M && (j-i) < K) {
          x2 = bodyMedzi[j].first;
          while (j < M && bodyMedzi[j].first == x2)
            j++;
        }

        // ak máme v okienku práve K bodov,
        // započítame konfiguráciu
        if ((j-i) == K) {
          minPod[b] = min(minPod[b],y2-y1+x2-x1);
          minNad[a] = min(minNad[a],y2-y1+x2-x1);
        }

        // posunieme ľavý okraj okienka a odpočítame body,
        // ktoré z okienka vypadnú
        while (i < M && bodyMedzi[i].first == x1) {
          i++;
        }
      }

    }
  }

  int minSuma = INF;
  // vyskúšať všetky kombinácie dvoch disjunktných riešení
  for (int y1 = 0; y1 < N; y1++) {
    for (int y2 = 0; y2 < N; y2++) {
      if (body[y1].second < body[y2].second)
        minSuma = min(minSuma,minPod[y1]+minNad[y2]);
    }
  }

  return minSuma;
}

int main() {
  cin >> sirka >> vyska >> N >> K;
  for (int i = 0; i < N; i++)
    cin >> body[i].first >> body[i].second;
  // vyriešime optimum pre horizontálnu čiaru
  int minimum1 = ries();
  // otočíme okolo diagonály
  for (int i = 0; i < N; i++)
    swap(body[i].first, body[i].second);
  // vyriešime optimum pre vertikálnu čiaru
  int minimum2 = ries();
  // skontrolujeme, či existuje riešenie
  if (minimum1 == INF && minimum2 == INF)
    cout << "riesenie neexistuje" << endl;
  else
    cout << 2*min(minimum1,minimum2) << endl;
  return 0;
}
```

Výsledková listina dvacátého druhého ročníku KSP po čtvrté sérii

		Škola	*ročník*	*sérií*	*2241*	*2242*	*2243*	*2244*	*2245*	*2246*	*2247*	*série*	*celkem*
1.	Jiří Eichler	SlovanG_OL	2	4	10	4	9	10	10	13	12	45,0	178,1
2.	Vojtěch Kolář	G_Neratov	4	15	10		10	10	10		10	40,0	152,9
3.	Vojtěch Hlávka	GŠlapanice	1	4	5	1	2	6	10	12	11	41,6	140,7
4.	Filip Hlásek	GMikul23PL	3	14	10				2		12	23,4	130,2
5.	Pavol Rohár	GMRŠKošice	4	5								0,0	113,4
6.	Vlastimil Dort	GŠpitálsPH	4	19				9			8,5	13,5	107,6
7.	Miroslav Olšák	GBuďánkaPH	4	3					8			9,3	98,4
8.	Štěpán Šimsa	GJungmanLT	1	8	10				5			16,0	97,3
9.	Ondřej Hübsch	ZŠJílov_PH	0	4	9				3			14,7	84,6
10.	Karel Tesař	SPŠE_Plzeň	4	11								0,0	80,1
11.	Karel Král	G_Most	4	9	10	1	0				4	16,1	61,3
12.	Jiří Setnička	G25březnPH	3	9		1					12	13,3	54,2
13.	Daniel Stahr	GJungmanLT	3	2	7		3		2		2,5	24,1	50,2
14.	Petr Hudeček	GCoubTábor	2	2								0,0	48,5
15.	Pavel Taufer	ArcibisGPH	4	11	7			3	2		3	15,0	48,3
16.	Petr Pecha	SPŠsVsetín	3	9								0,0	48,1
17.	Ondřej Mička	G_Jírov_ČB	1	3	8		3					14,7	37,4
18.	Martin Zikmund	G_Turnov	2	6								0,0	37,1
19.	Petr Čermák	GEBenešeKL	4	6								0,0	34,8
20.	Jakub Diatel	G_Slavičín	2	4	5						2,5	11,8	34,6
21.	Filip Štědronský	GMikul23PL	3	11								0,0	34,3
22.	Ondřej Cífka	GNadAlejPH	1	1	10	1	4				12	31,6	31,6
23.	Martin Holec	G_Slavičín	3	7							5,5	7,1	29,3
24.	Daniel Šafka	GKepleraPH	3	1								0,0	25,1
25.	Petra Vahalová	G_Plasy	4	1								0,0	24,8
26.	Tomáš Novella	GAlejKošic	4	1								0,0	23,9
27.	Kateřina Lorenzová	G_Česká_ČB	3	8							6	7,2	22,9
28.	Mária Mrocková	GJHroncaBA	3	1								0,0	21,8
29.	Jonatan Matějka	G_Jírov_ČB	0	2								0,0	20,2
30.	Radim Cajzl	GNoMěsNMor	3	22								0,0	18,6
31.	Petr Zvoníček	G_Slavičín	4	7								0,0	18,2
32.	Martin Mach	G_Jírov_ČB	2	2	8							9,4	17,3
33.	Matěj Kocián	GLesníZlín	3	1								0,0	17,1
34.	Dominik Smrž	GOhradníPH	0	4	10						3,5	16,0	16,0
35.–36.	Dana Marečková	GPatočkyPH	4	1								0,0	12,8
	Filip Matzner	GJirsíkaČB	3	1								0,0	12,8
37.	Jakub Červenka	GŠpitálsPH	4	5								0,0	12,0
38.	Tomáš Masák	GJirsíkaČB	3	1								0,0	10,7
39.	Hynek Jemelík	GJarošeBO	3	5								0,0	10,0
40.	Tomáš Maleček	GEBenešeKL	4	1								0,0	9,1
41.	Pavel Kratochvíl	VOŠGSvětlá	2	8								0,0	7,8
42.	Michal Bilanský	GLepařovJČ	4	6								0,0	6,6
43.	Karel Hulec	GJirsíkaČB	3	1								0,0	6,0