Milí řešitelé!

Jubilejní 20. ročník KSP je úspěšně za námi a ti nejlepší z vás se mohou těšit na závěrečné soustředění. Než se ale pustíme do rozebírání úloh poslední série, připravili jsme si pro vás malé překvapení. Poprosili jsme našeho Hrošíka, aby nám věnoval pár minut ze svého nabitého programu (to víte, šéfuje celému KSP a přitom musí alespoň 22 hodin denně spát) a poskytl nám exkluzivní interview.

Dobrý den.

Huiii.

Rádi bychom vám položili několik otázek týkajících se vedení KSP, máte chvilku...

Ale opravdu jen chvilku – za pět minut mi začíná druhý oběd.

Naše řešitele a řešitelky by zajímalo, co je na vedení KSP nejtěžší.

Někdo by myslel, že nejtěžší je vymýšlení nebo opravování úloh. Bohužel to není pravda. Nejtěžší je ukočírovat tu nezodpovědnou bandu organizátorů. Sehnat dneska dobrý materiál je opravdu těžké, a tak se musíme spokojit s tím, co máme.

To je zajímavé. A pomáhá vám... váš živočišný druh – tedy myslím jako to, že jste hroch – ve vaší těžké funkci?

Myslím, že trochu ano. Víte, tlustokožci budí přirozený respekt, takže si nikdo nedovolí odporovat mi. Také mám velkou tlamu, takže můžu kohokoli bez potíží seřvat – ehumpf, to tam nepište. Napište tam, že si můžu s každým cokoli vyříkat, to bude znít lépe.

To musí být praktické, ale nepovažoval jste to – ehm, bytí hrochem – také někdy za nevýhodu?

Občas. Víte, už nejsem tak mrštný, jako kdysi. Taky musím hodně jíst a spát, znáte to...

Obraťme list, co považujete za největší pozitivum KSP?

Myslím, že nejlepší na KSP je, co všechno dělá pro mládež. Je mnohem lepší, aby středoškoláci seděli doma u počítače a programovali, než aby se flákali někde po různých sportech nebo dokonce běhali za holkama.

Nyní se dotknu citlivějšího tématu. Myslíte si, že je KSP lepší než ostatní semináře?

Víte, takovéhle srovnávání seminářů není úplně korektní, protože ostatní semináře se zabývají jinými obory, takže neexistuje objektivní metrika (promiňte mi ten matfyzácký výraz), pomocí které bychom je mohli porovnávat. Nicméně pokud otázku přeformulujeme jinak – například zda je informatika lepší než řekněme fyzika – tak je odpověď jasná: Informatika je lepší než všechny ostatní vědy.

To je dost silný názor... nemyslíte si, že by s ním třeba fyzikové nesouhlasili?

To je možné. Můžou s ním nesouhlasit, můžou proti němu i protestovat, ale to je asi tak všechno, co se s tím dá dělat.

A víte, že v řadách organizátorů je i jeden fyzik?

Vím. A také dva matematici, ale co se dá dělat. Jak jsem již řekl na začátku, musíme brát to, co je.

Raději přejdeme na jiné téma. Co děláte pro zábavu? Sledujete třeba Euro?

No, občas se podívám... naposledy byl kurz ke koruně okolo 25, –, ale nevím, co na tom vidíte zábavného.

Á-ha – tak snad už jen poslední otázku... jaké jsou plány KSP na příští rok?

Příští rok chceme pokračovat v rozjeté tradici a neplánujeme žádné velké změny. Detaily ohledně připravovaného seriálu a příběhu k úložkám raději vynechám, abych nezkazil překvapení. Áá – už mi vezou oběd...

Naše interview je u konce. Přeji vám za všechny organizátory pěkné prázdniny a hodně úspěchů v dalších ročnících.

Ještě bychom Vás chtěli, jako už několikrát, poprosit: myslíme si, že je škoda, že se našeho semináře účastní tak málo řešitelů. Budeme rádi, když nám na začátku příštího roku pomůžete s „reklamou“ – například pověšením zadání první série na školní nástěnku, ...

Aktuální informace o KSP naleznete na stránkách *http://ksp.mff.cuni.cz/*. Dotazy ohledně zadání můžete posílat na adresu *ksp@mff.cuni.cz*, nebo se ptát přímo na diskusním fóru KSP (*http://ksp.mff.cuni.cz/forum/*).

Vzorová řešení páté série dvacátého ročníku KSP

20-5-1 Dračí cestování

Protože drak je velmi inteligentní zvíře, před cestou si dobře rozmyslel, že každá cesta z překladiště do překladiště ho stojí drahocennou síru, takže je určitě dobré počet cest minimalizovat a nosit vždy co nejvíc jde. Náklad 14000 kg se dá rozdělit na 3 náklady po 4200 kg a jeden 1400 kg těžký. Tím pádem musí absolvovat celkem 7 cest (4 cesty tam a 3 zpátky) na přenesení celého nákladu na první překladiště. Kdyby mu ale v nějakém momentu zbylo jenom 12600 ($3 \cdot 4200$) kg síry, dokázal by ji odnést na 5 cest. Pro 8400 kg by se musel vracet jenom jednou, což dělá jenom 3 cesty a

4200 kg unese najednou. Drak si okamžitě uvědomil, že překladiště se mu nejvíc vyplatí dělat tak, aby mu v něm zbylo o jednu várku síry míň než právě má, aby to dokázal odnést na méněkrát. Proč je každé jiné rozložení překladišť horší nebo nanejvýš stejně dobré? Zaveďme si značení: Nechť $S(A)$ je množství síry v překladišti A, N nosnost draka, SP jeho spotřeba na kilometr, $V(A,B)$ vzdálenost mezi překladišti A a B a $M(A,B)$ množství síry, které je drak schopný přenést z překladiště A do překladiště B. Je celkem snadné vidět, že počet cest, které musí drak uletět na přenesení nákladu z překladiště A do překladiště B je roven horní celé části z $S(A)/N$ vynásobené dvěma (drak se musí vracet) mínus jedna (poslední várku se už nevrací). Tím pádem $M(A,B) = S(A) - \textit{počet cest} \cdot V(A,B) \cdot SP$ (tolik toho drak sní po cestě). Z toho ale vyplývá, že kdyby sme nějaké překladiště posunuli dál, pak by drak letěl zbytečně kousek trasy o dvě cesty víc (tam a zpátky), i kdyby už nemusel, čím spotřebuje víc síry. Naopak, kdybychom posunuli nějaké překladiště o kousek blíž, zbylo by na tom překladišti víc síry a drak by musel k dalšímu stanovisti letět zase o dvě cesty víc (protože by ji neunesl na míň cest, kousek by mu tam zbyl). Taky je jednoduché vidět, že dělat si ještě nějaké překladiště navíc nemá smysl, protože jestli projde část trasy na n-krát a pak druhou část trasy taky na n-krát, spotřebuje přitom stejné množství síry jako kdyby letěl celou trasu na n-krát ($\textit{počet cest} \cdot V1 \cdot SP + \textit{počet cest} \cdot V2 \cdot SP = \textit{počet cest} \cdot (V1 + V2) \cdot SP$) a tím taky stejné množství síry přenese. Když jsme si tedy dokázali optimální rozložení překladišť, snadno podle toho spočítáme maximální množství síry, které je drak schopný přenést na 4200 kilometrovou vzdálenost. Drak si dělá stanoviště tak, aby na dalším stanovišti zbylo o jednu várku míň než má, tj. v takové vzdálenosti, aby při přenosu spotřeboval právě jednu várku. První překladiště si tedy drak zřídí $1400/7 = 200$ km od startu. Tím se zbaví jedné várky. Další várky se zbaví za $4200/5 = 840$ km, další za $4200/3 = 1400$ km. Nyní je drak 1760 km od cíle a na překladišti má už jenom jednu 4200 kilogramovou várku. Tu si naloží na záda a domů se vrátí s 2440 kg síry.

Mária Vámošová

20-5-2 Piškorcův deratizátor

A nejhorší ze všeho jsou trpaslíci, ty potvory vám vlezou všude a strašně rychle se množí. Já je chytám a většinou je zahazuju, ale mám švagra a ten si je nechává na chov...

Protože Temný mág není zdaleka jediný, kdo má se skřítky problémy, pojďme si říct, jak na ně.

Je jednoduché si uvědomit, že čarodějnice může každý den kouzlit vždy jako první. Pokud by v optimálním plánu kouzlila čarodějnice až po kouzelníkovi, mohla klidně kouzlit i před ním. Dále zajisté platí, že pokud může poté kouzelník provést deratizaci, musí ji provést ihned. Pokud by totiž čekal alespoň den, ztrojnásobil by se počet skřítků a on by místo jedné deratizace musel provádět tři.

Jeden den deratizace probíhá tak, že čarodějnice se rozhodne, jak změní populaci skřítků. Poté kouzelník sesílá deratizační kouzlo, dokud není skřítků méně než N. Tedy pouze čarodějnice má možnost volby. A pokud deratizace neskončila (nějací skřítci zůstali), jejich počet skřítků se ztrojnásobí a deratizace pokračuje.

Problém převedeme na hledání nejkratší cesty v grafu. Uvažujme graf s vrcholy očíslovanými 0 až $N-1$. Hodnota vrcholu odpovídá počtu skřítků na konci dne (po kouzlení, ale před trojnásobením). Hrana z vrcholu i do vrcholu j znamená, že se po jednom dni deratizace změní počet skřítků z i na j. Z každého vrcholu vedou nejvýš tři hrany, protože čarodějnice si může vybrat ze tří možností. Hrany v tomto grafu budou ohodnocené a ohodnocení hrany bude počet mágem seslaných deratizačních kouzel, tedy číslo 0, 1 nebo 2.

Nejkratší cesta v tomto grafu z vrcholu $K-1$, K či $K+1$ do vrcholu 0 je určitě řešením naší úlohu. Jak nejrychleji takovou hranu nalézt? Pokud by hrany nebyly ohodnocené, stačilo by použít prohledávání do šířky (pokud nevíte, co to je, nastal dobrý čas pro přečtení kuchařky třetí série 20. ročníku KSP). Protože jsou ale hrany ohodnocené jenom hodnotami 0, 1 a 2, můžeme upravit prohledávání do šířky následovně: nebudeme mít jednu, ale tři fronty F_0, F_1 a F_2. Ve frontě F_i budeme mít vrcholy, do kterých se dostaneme po provedení i deratizačních kouzel. Vrcholy budeme vybírat jenom z F_0 a jejich nenavštívené sousedy, do kterých vede hrana s ohodnocením j, budeme ukládat do fronty F_j. Když nám fronta F_0 dojde, uděláme z F_1 novou frontu F_0, z F_2 novou frontu F_1 a nová F_2 bude prázdná. Tak zaručíme, že vrcholy budeme zpracovávat v pořadí rostoucího počtu deratizací, které potřebujeme, abychom se do těchto vrcholů dostali.

Časová složitost tohoto postupu je $\mathcal{O}(N)$, protože graf se skládá z N vrcholů a $3N$ hran, při upraveném procházení navštívíme každý vrchol nejvýš jednou a každého souseda umíme zpracovat v konstantním čase. Paměťová složitost je zřejmě také lineární.

... to přece musí jít lépe! A taky že jo. Je možné dokázat, že stačí najít postup, který zabere co nejméně dní, protože takový postup použije nejmenší možný počet deratizačních kouzel. (Není to zas tak těžké, zkuste si to!) Poté je už řešení v čase $\mathcal{O}(\log N)$ nasnadě: na konci nultého dne jsou možné počty skřítků $K-1$, K a $K+1$. Na konci prvního $3K-4, \ldots, 3K+4$, konci i-tého $3^iK-(3^{i+1}-1)/2$ až $3^iK+(3^{i+1}-1)/2$, přičemž jde dosáhnout každého počtu mezi. Stačí tedy najít první interval takového tvaru, který obsahuje násobek N-ka.

Tereza Klimošová & Milan Straka & Tomáš Gavenčiak

20-5-3 Obrana před draky

Udatných drakobijců bylo mezi našimi řešiteli pomálu, takže většinu stačil drak skolit dříve, než stačili mrknout. Tak alespoň posmrtně si můžeme předvést, jak hledat místo pro vypuštění mocného protidračího kouzla.

Nejdřív si uvědomíme, že pokud máme kružnici, která obsahuje optimální počet temných kamenů, můžeme ji vždy posunout tak, aby na její hranici byly alespoň dva zadané body (alias temné kameny).

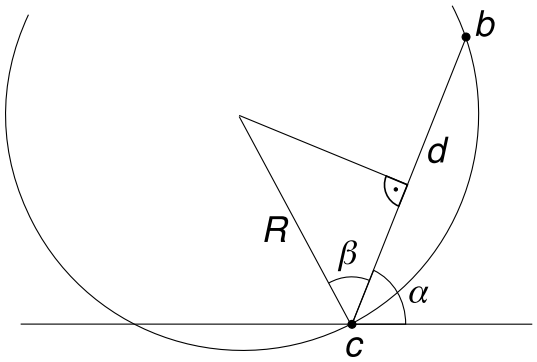
Jednoduchý algoritmus je nyní nasnadě: pro každé dva body najdeme kružnice, na kterých tyto dva body leží. Protože poloměr kružnice je pevně dané R, existují takové kružnice nejvýš dvě. Celkem tedy máme $\mathcal{O}(N^2)$ kružnic a víme, že jedna z nich je optimální. Stačí tedy pro každou z těchto kružnic zjistit, kolik je v ní zadaných bodů, což můžeme udělat jednoduše v lineárním čase. Tak získáme řešení s časovou složitostí $\mathcal{O}(N^3)$ a lineární paměťovou složitostí.

Je ale zřejmé, že takové řešení třináct bodů nezíská. Nyní si popíšeme řešení se složitostí $\mathcal{O}(N^2 \log N)$. Nebudeme zkoumat kružnice pro každé dva body, ale budeme zkoumat vždy všechny kružnice, na jejichž hranici je daný bod.

Vybereme tedy jeden ze zadaných bodů, kterému budeme říkat centrální, a budeme uvažovat kružnici, jejíž střed je o R nad daným bodem (kružnice tedy „stojí“ na tomto bodu). Pokud touto kružnicí budeme kolem daného bodu otáček tak, aby byl pořád na jejím okraji, budou ostatní body vstupovat a vystupovat z této kružnice. Stačí sledovat vstupující a vystupující body, upravovat si počet bodů v kružnici a po jedné otáčce kružnice kolem bodu si vybrat kružnici s maximálním počtem bodů uvnitř. Když tento postup provedeme pro všechny zadané body, získáme zajisté kružnici s největším počtem bodů uvnitř.

Mějme nějaký centrální bod. Polohu kružnice, která se ho dotýká, budeme určovat úhlem, který svírá spojnice středu kružnice a centrálního bodu s osou x. Řekněme, že pro daný centrální bod a další bod dokážeme zjistit úhly, kdy tento další bod vstoupí do kružnice, kterou rotujeme kolem centrálního bodu, a kdy z ní vystoupí. Pro všechny necentrální body získáme $\mathcal{O}(N)$ úhlů. Pokud tyto úhly setřídíme, můžeme pak v lineárním čase provést otáčení kružnice kolem centrálního bodu a najít tak kružnici s největším počtem bodů uvnitř. Pro jeden centrální bod bude tento postup trvat $\mathcal{O}(N \log N)$ času kvůli třídění úhlů. Celkem tak získáme řešení v čase $\mathcal{O}(N^2 \log N)$ s lineární paměťovou složitostí.

Zbývá vyřešit několik detailů. Za prvé, pro jeden úhel je třeba zpracovat nejprve všechny vstupy a pak až výstupy bodů. Za druhé, v počáteční pozici kružnice musíme spočítat, které body jsou už uvnitř. Můžeme to provést triviálně v lineárním čase, protože to provádíme pro každý centrální bod jenom jednou. A za třetí, musíme umět spočítat úhel vstupu a výstupu bodu z kružnice. Mějme centrální bod c a jiný bod b, který je od něj vzdálený $d \leq 2R$.



Úhel α je zřejmě $\operatorname{atan}[(b.y - c.y)/(b.x - c.x)]$, což můžeme v jazyce C zapsat jako $\operatorname{atan2}(b.y - c.y, b.x - c.x)$, a úhel β je $\operatorname{acos}\left(\frac{d}{2}/R\right)$. Úhel vstupu pak získáme jako $\alpha + \beta$ a úhel výstupu jako $\alpha - \beta$.

Program implementuje popsaný algoritmus s tím rozdílem, že kružnicí otáčí proti směru hodinových ručiček a body, které jsou na začátku v kružnici, poznává tak, že jejich úhel vstupu je větší než úhel výstupu.

Milan Straka

20-5-4 Dračí chodbičky

Napřed, jak bude algoritmus fungovat. Nejdříve bude ignorovat veškeré jeskyně s pokladem a na tom zbytku spočítá minimální kostru, například algoritmem popsaným v kuchařce 18-3. Poté vezme každou jeskyni s pokladem a připojí ji k nejbližší jeskyni bez pokladu. Jediné na co si je třeba dát pozor je speciální případ, pouze dvě jeskyně, obě s pokladem.

Proč to funguje? Kdyby byly dvě jeskyně s pokladem spojeny napřímo a žádná další cesta z nich nevedla, pak utvoří zcela samostatnou komponentu. Tedy, každá taková musí být připojená k některé bez pokladu. Je jedno, ke které, neboť zbylé jeskyně musí být navzájem propojené (a lze nahlédnout, že přes jeskyně s pokladem to nelze). Tedy vybereme si tu, ke které vede nejkratší chodba.

Zbylý kus musí být navzájem propojený a mít minimální možný součet hran. Toto přesně počítá algoritmus minimální kostry a jeho zdůvodnění správnosti lze nalézt ve zmíněné kuchařce.

Zbývá ještě časová a paměťová složitost. Paměťová je jednoduchá, pamatujeme si každý vrchol (jeskyni) a hranu (chodbu), tedy $\mathcal{O}(N+M)$. V časové bude jednak figurovat tvorba minimální kostry, který je $\mathcal{O}(N+M \cdot \log M)$. Při připojování pokladů projdeme každý vrchol a každou hranu nejvýše jednou, tedy zde je složitost $\mathcal{O}(N+M)$. Celková bude tedy $\mathcal{O}(N + M \cdot \log M)$.

A jedna implementační poznámka na závěr. Obě fáze jsou na sobě zcela nezávislé. Proto je možné tyto dvě fáze prolnout a udělat je obě na jeden průchod seřazenými hranami, jen si u hran dáme pozor, aby maximálně jeden z konců byl s pokladem a nebyl již připojen jinam.

Michal „vorner“ Vaner

20-5-5 Roztržitý matematik

Milí řešitelé a řešitelky, podle výsledků praktické úložky, které právě nemohu najít, jste si všichni vedli skvěle. Nebo si alespoň myslím, že jste si vedli skvěle ... tedy určitě alespoň většina z vás ...

Každopádně jsem si tu pro vás připravil nástin řešení, abyste si udělali alespoň hrubou představu o tom, jak to u nás chodí a na co si dávat pozor. Na úvod bych rád zdůraznil, že s papíry se to nemá tak jednoduše, jak by se mohlo na první pohled zdát. Jakmile na papír cokoli napíšete, začne žít vlastním životem a sám od sebe se přesunuje. Má tendenci se schovávat pod jiné papíry, když ho právě potřebujete, a naopak ležet na vrchu a překážet, pokud zrovna hledáte něco jiného.

Ale to jsem trochu odbočil... ach ano – to řešení. Někde jsem ho tu měl připravené. Kam se asi mohlo schovat? V zásadě teď může být kdekoli. Věřili byste, že jsem jednou našel svůj článek dokonce až pod automatem na kávu? Opravdu netuším, jak se tam dostal, protože automat je na chodbě poměrně daleko od mého kabinetu ...

Ale abych se vrátil – problém, se kterým se každý den potýkám, se nazývá move-to-front transformace. Můj kolega z informatiky tvrdí, že se používá také při kompresi, ale to mi příliš nepomůže. Jádro problému spočívá v rychlém nalezení a odebrání i-tého papíru v pořadí a jeho vložení na začátek tak, aby se správně posunuly ostatní papíry.

Půjdeme-li na to přímo, nenarazíme na žádné potíže. Všechny papíry si uložíme do pole tak, že i-tý papír se nachází na indexu i. Nalezení papíru máme zadarmo v konstantním čase. Papír odebereme a všechny papíry, které jsou před ním, posuneme o jednu pozici. Tím se nám vzniklá díra zaplní a naopak vytvoříme díru na první pozici. Nyní na začátek vložíme odebraný papír a máme hotovo.

Tohle řešení má lineární časovou složitost na každou operaci (tzn. celkem $\mathcal{O}(N \cdot k)$, kde N je počet papírů a k počet operací), takže se hodí k přerovnávání několika papírků na stole mého pořádkumilovného kolegy, ale prohledání celého mého kabinetu by zabralo věčnost ...

Dlouho jsem si s tím lámal hlavu, až mi kolega informatik poradil lepší řešení. Jak jsem se dozvěděl, klíčem jsou stromy – tím nemyslím to, co mi roste pod okny, ale binární stromy. Je vhodné použít nějakou variantu vyvažovacích stromů (AVL, Červeno-černé, ...), protože jinak vaše řešení rychle zdegeneruje na lineární spojový seznam. Sám se ve stromech příliš nevyznám, takže pokud vás zajímají detaily, nahlédněte do kuchařky.

V každém vrcholu u bude uložen počet prvků (označme jej $c(u)$) v podstromě, který má u jako kořen, a také číslo papíru, který je v tomto vrcholu uložen.

Takový strom postavíme jednoduše. Na začátku víme, že papíry jsou seřazeny od 1 do N. Kořen našeho stromu bude reprezentovat prostřední papír z daného intervalu. Levý a pravý podstrom pak vygenerujeme rekurzivně. Počet prvků v každém podstromě spočítáme také snadno: stačí v každém vrcholu sečíst:

$$c(\textit{levého podstromu}) + c(\textit{pravého podstromu}) + 1.$$

Nyní se podívejme, jak rychle nalézt, co hledáme. Řekněme, že jsme ve vrcholu u a pátráme po i-tém papíru (oproti zadání je budeme číslovat od nuly, to vyjde elegantněji). Podíváme se na počet prvků v levém podstromě $\ell = c(\textit{levý syn } u)$. Pokud je $i < \ell$, víme, že se hledaný prvek nachází v levém podstromu, je-li $i = \ell$, hledaným prvkem je u sám, a konečně v posledním případě ($i > \ell$) se hledaný papír nachází v pravém stromu. Samozřejmě si musíme dát pozor, když přecházíme do pravého podstromu. Tam už nehledáme i-tý papír, ale papír s indexem $i - \ell - 1$.

Odebrání samotného papíru pak probíhá podle pravidel mazaní z binárního vyhledávacího stromu (viz kuchařka a též níže). Stejně tak musíme po mazaní provést vyvážení stromu, které závisí na tom, jaký typ stromu jsme použili (opět viz kuchařka). Po mazání je nezbytné ještě opravit všechny hodnoty $c(u)$ ve vrcholech, které ležely po cestě k hledanému papíru.

Odebraný papír vložíme do stromu na nejlevější pozici (tedy na první místo). Opět dodržíme pravidla pro vkládání do stromu, opravíme všechny hodnoty $c(u)$ po cestě a provedeme vyvážení.

Nakonec potřebujeme ještě vypsat konečnou permutaci dokumentů. Stačí pouze projít a vypsat náš strom v pořadí in-order.

Časová složitost uvedeného algoritmu je $\mathcal{O}(\log N)$ na jednu operaci, protože hledání, mazání i vkládání trvá u vyváženého binárního stromu logaritmicky dlouho. Paměťová složitost se nám přitom nezhoršila. Sice spotřebujeme několikrát víc paměti, ale asymptoticky zůstáváme stále na příjemné složitosti $\mathcal{O}(N)$.

Jeden student mi ještě tvrdil, že zná řešení v čase $\mathcal{O}(k\sqrt{N})$, ale vůbec si nejsem jistý, jak by takové řešení mělo fungovat, takže si můžete zkusit takové řešení napsat za domácí cvičení.

Náš čas na konzultaci bohužel vypršel a já se s vámi musím rozloučit. Někde jsem tu měl papír se seznamem dalších schůzek – ale kam jsem si ho sakra založil ...?

Martin „Bobřík“ Kruliš

V programu si vyzkoušíme jednu méně tradiční, ale moc pěknou odrůdu stromů, totiž BB-α stromy již letmo zmíněné v kuchařce. Místo podle hloubky budou vyvažovány podle váhy, čili počtu vrcholů v podstromech. V dokonale vyváženém stromu platí, že se váha levého a pravého podstromu každého vrcholu liší nejvýše o jedničku. My nebudeme tak přísní: povolíme, aby jeden podstrom měl až dvakrát větší váhu než druhý.

Všimněte si, že toto pravidlo nám stále zaručuje logaritmickou hloubku: oba synové vrcholu s váhou w mají váhy nejvýše $(2/3)w$, jejich synové nejvýše $(2/3)^2 w$ atd., takže váhy stále klesají exponenciálně. Hloubka stromu tedy bude přibližně $\log_{3/2} n$.

Když do stromu přidáme nový vrchol, a nebo naopak nějaký smažeme, přepočítáme všechny váhy na cestě ze změněného vrcholu do kořene (všimněte si, že to tak jako tak v naší úloze potřebujeme). Přitom budeme kontrolovat, jestli váhy stále splňují vyvažovací podmínku. Jakmile objevíme vrchol, ve kterém podmínka neplatí, nebudeme se snažit vyvážení obnovit rotacemi (což by také šlo), ale podnikneme daleko razantnější krok: celý podstrom rozebereme a předěláme na dokonale vyvážený.

Přeskládání celého podstromu je samozřejmě časově náročné: trvá $\mathcal{O}(\textit{váha podstromu})$ – inu, když se kácí les, létají třísky –, ale ukážeme, že nebudeme „kácet“ příliš často, takže se nám složitost nepokazí.

Představme si na chvilku, že do stromu jenom vkládáme. Vyberme si nějaký podstrom a sledujme, co se s ním bude dít. Nejprve je během některého kácení postaven jako dokonale vyvážený a tehdy se váha jeho levého a pravého syna rovnají (± 1) nějakému číslu w. Pak do podstromu přibývají nové prvky a vyvážení se postupně zhoršuje. Nakonec se situace stane příliš nahnutou, a tak podstrom pokácíme. Všimněte si, že mezi vytvořením podstromu a jeho pokácením se musel jeden syn stát dvakrát těžším než druhý, takže se muselo objevit alespoň w nových prvků. Přebudování samo trvá $\mathcal{O}(w)$, tudíž stačí, aby na něj každý z w nově vložených prvků přispěl časem $\mathcal{O}(1)$. Každý prvek si takto předplatí všechna pokácení na cestě z něj do kořene (přesně těm přispívá), a tak celkově přispěje časem $\mathcal{O}(\log n)$.

Mazání nám tuto analýzu trochu zkomplikuje, ovšem snadno nahlédneme, že když je povoleno jak vkládat, tak mazat, nejrychlejší způsob, jak pokácet strom váhy $2w$, je smazat z jednoho jeho podstromu $w/2$ vrcholů a druhý podstrom nechat na pokoji. (Na počátku měly podstromy váhy w, na konci má jeden $w/2$ a druhý stále w.) Opět na to potřebujeme řádově w operací, takže původní úvaha s příspěvkem $\mathcal{O}(\log n)$ na operaci stále funguje.

Kolik času tedy spotřebujeme na jednu operaci? Inu, když máme smůlu a zrovna jsme způsobili jedno nebo více kácení, operace může trvat až n kroků. Ovšem díky našemu principu „předplácení si“ stále platí, že libovolných k po sobě jdoucích operací trvá $\mathcal{O}(k \log n)$, což nám pro řešení úlohy úplně stačí. Tomu se obvykle říká, že každá operace má *amortizovanou časovou složitost* $\mathcal{O}(\log n)$.

Stálo to trochu počítání, ale zato jsme si ušetřili spoustu programování. Inu, i informatici jsou líní a někdy i roztržití.

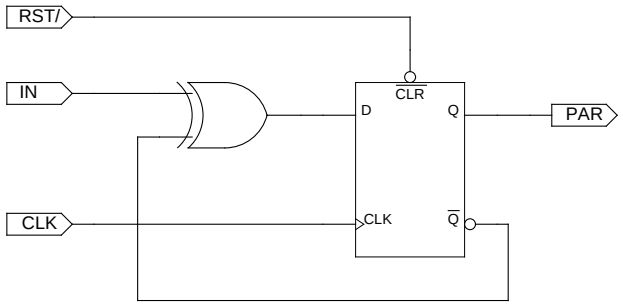
Martin Mareš

20-5-6 Hrady, hrádky, hradla

a) Jak se průběžná parita má chovat? Inu, pokud na vstupu přijde nula, parita se nijak nemění. Pokud naopak přijde jednička, parita se zneguje. Jinými slovy nová parita je XOR staré parity s bitem na vstupu. Pak už stačí přidat klopný obvod D, aby si paritu pamatoval a novou zapsal při náběžné hraně hodinového signálu.

Ještě bychom měli domyslet, jakou hodnotu má mít parita na začátku výpočtu. Za tímto účelem doplníme klopný obvod D o vstup CLR (clear), který způsobí jeho vynulování, a necháme na uživateli, ať na začátku „zatahá za reset“ a obvod zinicializuje. (Takový vstup mají i opravdové klopné obvody.)

Celý obvod bude tedy vypadat takto:



b) Ani čítač nebude složitý. Nejnižší bit dvojkového čítače při každém „tiku“ hodin svou hodnotu zneguje, jinými slovy na jeho spočítání stačí dělička frekvence zmíněná v zadání. Další bit v pořadí se změní právě tehdy, když nejnižší bit přejde z jedničky do nuly, čili když nastane náběžná hrana na negovaném výstupu zmíněné děličky. Takže stačí na tento negovaný výstup napojit další děličku, a tak dále.

Obvod pro N-bitový čítač tedy bude sestávat z N klopných obvodů D zapojených jako děličky. Schéma pro $N = 4$ najdete níže, sem do úzkého sloupečku se nevešlo.

Zbývá dodat snad jen to, že i u obvodů tohoto druhu má smysl zkoumat časovou složitost – v tomto případě dobu, za kterou se obvod po tiku hodinového signálu ustálí a vydá stabilní výstup. U našeho čítače je to $\mathcal{O}(N)$ kroků (změna musí „probublat“ až N D-čky). Šel by ale postavit i tak, aby reagoval už za $\mathcal{O}(\log N)$ kroků, zkuste si to třeba o prázdninách vymyslet. S páječkou v ruce se s vámi loučí

Martin Mareš

Úloha 20-5-2 – Piškorcův deratizátor – program

```
program deratizace;
var
  N, K    : longint;                    {hodnoty ze zadání                                          }
  kouzel  : array [0..N-1] of longint; {počet kouzel potřebný k dosažení těchto stavů na konci dne}
  odkud   : array [0..N-1] of longint; {jaký byl ideální včerejší počet, abychom se dostali sem   }
  jakpak  : array [0..N-1] of longint; {jak čarodějka změnila včera počet, abychom se dostali sem }
  skupiny : array [0..2,0..N-1] of longint; {skupiny počtů k prozkoumání ...                       }
  plnost  : array [0..2] of longint;   {... jejich plnost ...                                      }
  vybirana: longint;                    {... a právě vybíraná skupina                               }

procedure vloz(skupina,copak:longint);
begin
  skupiny[skupina,plnost[skupina]]:=copak;
  inc(plnost[skupina]);
end;

procedure zkus_pridat(co,jak,nakouzel:longint); {Je tenhle tah zlepšení?}
var zitra,deratu,skupina:longint;
begin
  zitra:=(co*3+jak) mod N;                      {kolik jich bude zítra večer?}
  deratu:=(co*3+jak) div N;                     {kolik D-kouzel bude další den?}
  if kouzel[zitra]<=nakouzel+deratu then exit;
  kouzel[zitra]:=nakouzel+deratu;               {ulož tento postup ...}
  odkud[zitra]:=co;
  jakpak[zitra]:=jak;
  vloz((vybirana+deratu) mod 3,zitra);          {...a vlož ho do správné skupiny}
end;

var i,j:longint;

begin
  read(N, K);
  for i:=0 to N-1 do kouzel[i]:=K+1;            {zatím nedosažené stavy}
  vybirana:=0;
  for i:=-1 to +1 do begin
    j:=(K+i) mod N;
    kouzel[j]:=(K+i) div N;
    odkud[j]:=K; jakpak[j]:=i;
    vloz((vybirana+kouzel[j]) mod 3,j);
  end;
  while (plnost[0]+plnost[1]+plnost[2]>0) do begin
    if plnost[vybirana]=0 then vybirana:=(vybirana+1) mod 3; {cyklit skupiny}
    if plnost[vybirana]=0 then vybirana:=(vybirana+1) mod 3; {cyklit skupiny}
    i:=skupiny[vybirana,plnost[vybirana]-1];    {vybrat prvek}
    dec(plnost[vybirana]);
    if i=0 then                                 {deratizováno!}
    begin
      writeln('Postup pozpatku:');
      while (odkud[i]<>K) do begin
        j:=3*odkud[i]+jakpak[i];
        writeln('rano ',3*odkud[i],', uprava ',jakpak[i],', deratizaci ',j div N,', vecer ',i);
        i:=odkud[i];
      end;
      writeln('rano ',K,', uprava ',jakpak[i],', deratizaci ',(K+jakpak[i]) div N,', vecer ',i);
```

```
      exit;
    end;
    for j:=-1 to 1 do zkus_pridat(i,j,kouzel[i]);
  end;
end.
```

Úloha 20-5-3 – Obrana před draky – program

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#define MAX_B      10000
#define DIST2(a,b) (((a).x-(b).x)*((a).x-(b).x)+((a).y-(b).y)*((a).y-(b).y))
#define EPS        1e-10

double N;
struct bod     { double x;   double y; } body[MAX_B]; int B;
struct udalost { double uhel; int bod; } u[ 2*MAX_B]; int U;

int udalost_cmp_uhel(const void* a, const void* b) {
  if (((struct udalost*)a)->uhel == ((struct udalost*)b)->uhel) return 0;
  return ((struct udalost*)a)->uhel < ((struct udalost*)b)->uhel ? -1 : 1;
}

int main(void) {
  scanf("%d%lf", &B, &N); assert(B>=1);
  for (int i=0;i<B;i++) scanf("%lf%lf", &body[i].x, &body[i].y);

  int best=1; struct bod best_pos=body[0]; double best_uhel=0;
  int body_uvnitr[2*B];
  for (int i=0;U=0,i<B;i++) {                          // každý bod může být na hranici
    int act=1;
    for (int j=0;j==i && j++,j<B;j++) {                // spočítej události ostatních bodů
        double d=DIST2(body[i],body[j]); if (d>4*N*N) continue;
        double uhel_zaklad=atan2(body[j].y-body[i].y,body[j].x-body[i].x);
        if (uhel_zaklad<0) uhel_zaklad+=2*M_PI;
        double uhel_zmena=acos(sqrt(d)/(2.0*N));
        u[U].bod=j; u[U++].uhel=uhel_zaklad-uhel_zmena-EPS; // dvě události, první vstup
        u[U].bod=j; u[U++].uhel=uhel_zaklad+uhel_zmena+EPS; // a druhá výstup z kružnice
        if (u[U-2].uhel< 0     ) u[U-2].uhel+=2*M_PI;      // normalizace
        if (u[U-1].uhel>=2*M_PI) u[U-1].uhel-=2*M_PI;

        act+=body_uvnitr[j]=u[U-1].uhel<u[U-2].uhel;       // je bod uvnitř ve výchozí pozici?
    }

    qsort(u, U, sizeof(struct udalost), udalost_cmp_uhel);// setřiď události podle úhlu

    for (int j=0;j<U;j++) {                            // projdi události
      act+=body_uvnitr[u[j].bod]?-1:1;
      body_uvnitr[u[j].bod]^=1;
      if (act>best) best=act, best_pos=body[i], best_uhel=u[j].uhel; //aktualizuj maximum
    }
  }

  printf("Temný mák získá %d temných kamenů, pokud zakouzlí v (%g,%g)\n",
         best, best_pos.x+cos(best_uhel)*N, best_pos.y+sin(best_uhel)*N);
  return 0;
}
```

Úloha 20-5-4 – Dračí chodbičky – program

```
#include <stdio.h>
#include <stdlib.h>

struct vrchol_t {
  //Jak bohatá je to jeskyně?
  int poklad;
  //Už jsem to s něčím propojil?
  int spojeno;
  //V tomto postavíme dfu
  int dfu_otec;
  int dfu_vaha;
};

struct hrana_t {
  //Indexy obou konců
  int vrcholy[2];
  int delka;
};

static int mensi_hrana(const void *_1, const void *_2) {
```

```
  return ((struct hrana_t *) _1)->delka - ((struct hrana_t *) _2)->delka;
}

static int dfu_find(struct vrchol_t vrcholy[], int v) {
  if(vrcholy[v].dfu_otec == v)
    return v;
  else {
    int result = dfu_find(vrcholy, vrcholy[v].dfu_otec);
    vrcholy[v].dfu_otec = result;//Zkomprimuj cestu
    return result;
  }
}

static void dfu_union(struct vrchol_t vrcholy[], int v1, int v2) {
  if(vrcholy[v1].dfu_vaha > vrcholy[v2].dfu_vaha)
    dfu_union(vrcholy, v2, v1);//Zapojovat spravnym smerem
  else {
    vrcholy[v1].dfu_otec = v2;
    vrcholy[v2].dfu_vaha += vrcholy[v1].dfu_vaha;
  }
}

int main(void) {
  //Nějaké to načtení
  int vrcholu, hran;
  scanf("%d%d", &vrcholu, &hran);
  struct vrchol_t vrcholy[vrcholu];
  for(int i = 0; i < vrcholu; i++) {
    scanf("%d", &vrcholy[i].poklad);
    vrcholy[i].dfu_otec = i;
    vrcholy[i].dfu_vaha = 1;
    vrcholy[i].spojeno = 0;
  }
  struct hrana_t hrany[hran];
  for(int i = 0; i < hran; i++) {
    scanf("%d%d%d", &hrany[i].vrcholy[0],
        &hrany[i].vrcholy[1], &hrany[i].delka);
  }
  //Hrany jsou potřeba setříděné
  qsort(hrany, hran, sizeof *hrany, mensi_hrana);
  //Bereme jednotlivé hrany a vybírame, jestli je chceme
  for(int i = 0; i < hran; i++) {
    //Nespoj dvě s pokladem, pokud to není speciální případ
    if(((vrcholy[hrany[i].vrcholy[0]].poklad
        && vrcholy[hrany[i].vrcholy[1]].poklad)
        && (vrcholu != 2))
        || (vrcholy[hrany[i].vrcholy[0]].poklad
        && vrcholy[hrany[i].vrcholy[0]].spojeno)
        || (vrcholy[hrany[i].vrcholy[1]].poklad
        && vrcholy[hrany[i].vrcholy[1]].spojeno))
      continue;
    int komponenty[] = {
        dfu_find(vrcholy, hrany[i].vrcholy[0]),
        dfu_find(vrcholy, hrany[i].vrcholy[1])};
    if(komponenty[0] != komponenty[1]) {
      printf("%d %d %d\n", hrany[i].vrcholy[0],
          hrany[i].vrcholy[1], hrany[i].delka);
      dfu_union(vrcholy, komponenty[0], komponenty[1]);
      vrcholy[hrany[i].vrcholy[0]].spojeno = 1;
      vrcholy[hrany[i].vrcholy[1]].spojeno = 1;
    }
  }
  return 0;
}
```

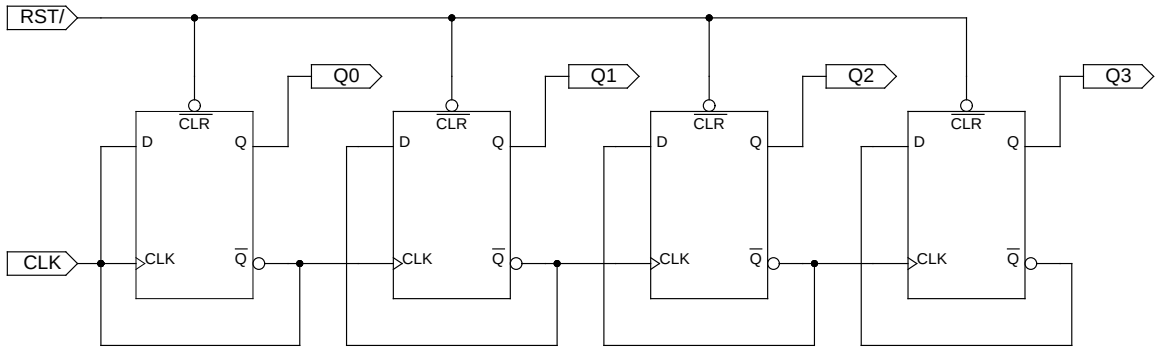
Úloha 20-5-5 – Roztržitý matematik – program

```
#include <stdio.h>
#include <stdlib.h>

struct node {                   // Vrchol stromu
  struct node *left, *right;    // Synové
  int id;                       // Číslo papíru
  int weight;                   // Velikost (váha) podstromu
};

static int weight(struct node *n)
{
  return (n ? n->weight : 0);   // Váha stromu, prázdný má 0
}
```

```
// Vytvoří ze stromu seznam, 'right' ukazuje na následníka.
struct node *tree_to_list(struct node *n, struct node *tail)
{
  if (!n)
    return tail;
  n->right = tree_to_list(n->right, tail);
  return tree_to_list(n->left, n);
}

// Odpojí 'count' prvků ze seznamu '*lp' a vytvoří z nich
// vyvážený strom. Vrátí ukazatel na kořen.
struct node *list_to_tree(struct node **lp, int count)
{
  if (!count)
    return NULL;
  int half = (count-1)/2;
  struct node *left = list_to_tree(lp, half);
  struct node *root = *lp;
  *lp = root->right;
  root->left = left;
  root->right = list_to_tree(lp, count-1-half);
  root->weight = count;
  return root;
}

// Přepočítá váhy podle synů a pokud je strom příliš
// vychýlený, přebuduje ho.
struct node *reweight(struct node *n)
{
  int lw = weight(n->left);
  int rw = weight(n->right);
  n->weight = 1 + lw + rw;
  if (lw+rw > 1 && (lw > 2*rw || rw > 2*lw))
    {
      struct node *tmp = tree_to_list(n, NULL);
      n = list_to_tree(&tmp, n->weight);
    }
  return n;
}

// Vloží vrchol do stromu před všechny ostatní.
struct node *add_start(struct node *root, struct node *new)
{
  if (!root)
    {
      root = new;
      new->left = new->right = NULL;
    }
  else
    root->left = add_start(root->left, new);
  return reweight(root);
}

// Smaže k-tý nejmenší prvek ve stromu a vrátí jak tento
// prvek (*kthp), tak nový kořen stromu.
struct node *del_kth(struct node *n, int k,
                     struct node **kthp)
{
  if (k < weight(n->left))
    {
      // k-tý nejmenší je v levém podstromu
      n->left = del_kth(n->left, k, kthp);
      return reweight(n);
    }
  k -= weight(n->left);
  if (k)
    {
      // k-tý nejmenší je v pravém podstromu
      n->right = del_kth(n->right, k-1, kthp);
      return reweight(n);
    }
  // Mám ho, pryč s ním. Má jen jednoho syna?
  *kthp = n;
  if (!n->left)
    return n->right;
  if (!n->right)
    return n->left;
  // Má dva => prohodím s maximem levého podstromu.
  n->left = del_kth(n->left, n->left->weight-1, kthp);
  int id = n->id;
  n->id = (*kthp)->id;
```

```
  (*kthp)->id = id;
  return reweight(n);
}

// Vypíše všechny hodnoty ve stromu.
void dump_tree(FILE *fo, struct node *root)
{
  if (!root)
    return;
  dump_tree(fo, root->left);
  fprintf(fo, "%d ", root->id);
  dump_tree(fo, root->right);
}

int main(void)
{
  FILE *fi = fopen("papiry.in", "r");
  FILE *fo = fopen("papiry.out", "w");
  int N, K, op;
  fscanf(fi, "%d%d", &N, &K);

  // Vložíme všechna lejstra do stromu.
  struct node *root = NULL;
  for (int i=N; i>0; i--)
    {
      struct node *n = malloc(sizeof(*n));
      n->id = i;
      root = add_start(root, n);
    }

  // Vykonáváme příkazy ze vstupu.
  for (int i=1; i<=K; i++)
    {
      fscanf(fi, "%d", &op);
      struct node *n;
      root = del_kth(root, op-1, &n);
      root = add_start(root, n);
    }

  // Vypíšeme, jak to dopadlo.
  dump_tree(fo, root);
  fputc('\n', fo);
  fclose(fo);
  fclose(fi);
  return 0;
}
```

Úloha 20-5-6 – Hrady, hrádky, hradla – schéma čítače



Konečná výsledková listina dvacátého ročníku KSP

		škola	*ročník*	*sérií*	*2051*	*2052*	*2053*	*2054*	*2055*	*2056*	*série*	*celkem*
1.	Peter Ondrúška	SPŠDubnica	4	5			13	11	15	7	48,2	224,4
2.	Filip Hlásek	GMikulášPL	1	5	8	8			12	8	40,6	173,5
3.	Vlastimil Dort	GŠpitálsPH	2	10		8	9	11	0	12	40,6	146,5
4.	Jan Michelfeit	G HBrod	4	9							0,0	144,3
5.	Filip Štědronský	GMikulášPL	1	5				11	15	12	38,0	138,1
6.	Alena Skálová	GNaVPláni	4	5							0,0	134,8
7.	Petr Malý	GSladNámPH	4	5				7	6		17,7	133,6
8.	Libor Plucnar	GPBezruče	3	9		5		9	6		22,1	119,9
9.	Štěpán Weber	GBuďánkaPH	3	4				7			9,1	113,4
10.	Pavel Veselý	G Strakon	3	12	8						8,0	94,9
11.	Trung Ha duc	GMasarykPL	2	9							0,0	93,0
12.	Tomáš Toufar	G Bílovec	4	3							0,0	83,5
13.	Vojtěch Tůma	G Jihlava	4	7							0,0	81,3
14.	Petr Babička	G SvětláNS	3	5					6		8,9	64,2
15.	Stanislav Fořt	G Tábor	0	7							0,0	60,2
16.	Jan Škoda	GMikulášPL	1	3							0,0	59,1
17.	Jitka Novotná	G Bílovec	3	2							0,0	58,6
18.	Lukáš Kripner	G Litvínov	2	6							0,0	57,7
19.	Jakub Hrnčíř	GFXŠaldyLI	1	3							0,0	55,9
20.	Radim Cajzl	G NMnMor	1	15	1					9	8,3	54,9
21.	Pavel Kratochvíl	ZŠSvětlá	0	4							0,0	51,0
22.	David Marek	SPŠ Zlín	4	4							0,0	49,2
23.	Tomáš Jakl	G MTřebová	4	5	5	2		7	5		26,5	45,9
24.	Adam Streck	G Hořice	4	2	8						8,0	35,6
25.	Jan Matějka	GJírovcoČB	3	4							0,0	35,4
26.	Milan Rybář	GJungmanLT	3	2							0,0	35,0
27.	Jakub Kaplan	GJKTylaHK	4	16							0,0	34,4
28.	Jiří Setnička	G25březnPH	1	2	1	0			6		12,5	34,3
29.	Roman Smrž	GOhradníPH	4	2							0,0	34,0
30.	Jakub Suchý	GMikulášPL	1	3					6		9,8	33,9
31.	Pavel Taufer	GArcibisPH	2	2	6						7,4	33,1
32.	Jiří Zárevúcky	SŠInformFM	3	1							0,0	32,5
33.	Tomáš Sýkora	G VKlobou	4	11							0,0	30,8
34.	David Brázdil	G Zlín	3	6							0,0	29,8
35.	Martin Vlach	G Jihlava	4	1							0,0	29,7
36.	Karel Tesař	SPŠEPlzeň	2	2							0,0	28,4
37. – 38.	Jiří Keresteš	SPŠEPlzeň	2	4							0,0	28,2
	Petr Sokola	SPŠ Zlín	4	2							0,0	28,2
39.	Vojtěch Kolář	G Neratov	3	1							0,0	27,3
40.	Dominik Smrž	GOhradníPH	0	2		6			2		13,0	27,1
41.	Jakub Červenka	GŠpitálsPH	2	4							0,0	26,0
42.	Martin Patera	GArabská	2	1							0,0	25,5
43.	Jan Žák G HBrod	3	6							0,0	24,1	
44.	Alžběta Pechová	SPŠSVsetín	3	4							0,0	22,6
45.	Miroslav Klimoš	G Bílovec	3	21							0,0	19,4
46.	Jan Vaňhara	G Holešov	3	1							0,0	17,8
47.	Marek Nečada	G Jihlava	4	1					6		10,6	10,6
48.	Petr Holášek	G Příbor	4	5							0,0	10,5
49.	Nikolas Zigmund	ZŠHavířov	1	1							0,0	8,9
50.	Lukáš Timko	G Tábor	0	2							0,0	8,6
51.	Peter Uhnák	GBBolzana	2	1							0,0	7,4
52.	Peter Smatana	EkoGLabsBO	3	1							0,0	6,4