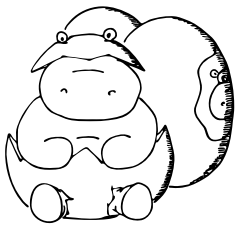
Korespondenční Seminář z Programování

32. ročník **KSP** **Duben 2019**

Milí řešitelé, řešitelky a řešitelčata!



Nelehká pandemická doba jara tohoto roku bohužel nepřála ani přípravě čtvrté série, ale již se k vám předposlední číslo jubilejního 2^5-tého ročníku KSPčka konečně dostává. Doufáme, že vás všechny zastihuje v co nejlepším zdraví a doufáme, že zdraví i nadále zůstanete a uvítáte vytržení od domácího učení další várkou zajímavých úloh.

Předchozí díl seriálu můžete odevzdávat až do konce dubna. V této sérii namísto seriálu přibalujeme **kompetitivní úlohu**, kde můžete navzájem mezi sebou poměřit schopnosti ve hledání optimálního řešení na těžkou úlohu. Zapojit se může každý, tak toho využijte! Mimo to série obsahuje další **čtyři teoretické** a **jednu praktickou úlohu**. A závěrem tohoto čísla je **kuchařka o minimálních kostrách**.

Připomínáme, že se z každé série stále **započítává 5 nejlépe vyřešených úloh** (tedy nemusíte vyřešit úplně všechny a i tak můžete dosáhnout na plný počet bodů). Také se vám body za úlohy **přepočítávají podle vašeho služebního stáří** – na přesnou definici se podívejte do pravidel na webu.

Za úspěšné řešení KSP můžete být **přijati na MFF UK bez přijímacích zkoušek**. Úspěšným řešitelem se stává ten, kdo získá za celý ročník (této kategorie) alespoň 50 % bodů. Za letošní rok půjde získat maximálně 300 bodů, takže hranice pro úspěšné řešitele je 150. Maturanti pozor, toto je poslední série, za kterou stihneme získat body, pokud potřebujete prominutí přijímaček uplatnit již letos (pokud tomu tak je, ozvěte se nám). Také každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok, placku a možná i další překvapení.

Termín série: **neděle 10. května 2020 ve 32:00 (tedy další ráno v 8:00)**

Odevzdávání: Přes web na adrese *https://ksp.mff.cuni.cz/submit/*

Značky úloh: Lehčí úloha (či její část) vhodná pro začátečníky; Open-data úloha; Úloha, u které doporučujeme začíst se do kuchařky

Odměna série: **Třem nejúspěšnějším řešitelům kompetitivní úlohy pošleme sladkou odměnu.**

Čtvrtá série třicátého druhého ročníku KSP

V minulé sérii unikla posádka Hefaista ze základny zasažené podivnou iontovou bouří, aby následně zjistila, že část ztracených kolonistů pravděpodobně v rámci tajného výzkumu iontovou bouři vytvořila a vyslala ji proti zbytkům základny s cílem zamést stopy – tedy zbylé kolonisty a také posádku hlídkové a servisní lodi Hefaistos, která přiletěla kolonistům na pomoc.

Po dramatickém úniku (přečtěte si minulý díl) a po napíchnutí místní satelitní sítě se povedlo posádce a kolonistům zjistit souřadnice utajené základny a v hlasování o pokračování jste zvolili, že se mají pokusit přepadnout druhou tajnou základnu, protože jsme přece lepší! Tak čtěte pokračování...

* * *

„Já říkám, zaútočme na ně! Tady jsme jak sedící kachny, hned jak opadne to rušení z iontové bouře, tak nás uvidí," zahřměl přes celý můstek vždy bojovný mariňák Drake.

„Ale tohle není žádná válečná loď," oponoval mu pilot, „tohle je dobrá loď na zachycení neovladatelné lodi, průlety skrz prachové ohony komet a jinou drsnou jízdu... ale probůh, vždyť nemáme žádné pořádné zbraně, jenom protimeteorické dělo."

„Kolonistů je na té základně nanejvýš devět – náklad sem sice propašovat mohli, ale další lidi by před kolonií neskryli. Nás je více a máme pořád plný set těžkých skafandrů do nebezpečného prostředí, ty něco vydrží. A dostatek nářadí na to propálit se několikametrovým pancířem hvězdné lodě, do té základny se můžeme dostat pěšky," pronesla po chvíli přemýšlení kapitánka Laren. „Navrhuji přistát vedle jejich základny a..."

Hlavní inženýr McCormack kapitánku přerušil: „Podle těhle snímků tam ale mají nějakou obrannou soustavu a pravděpodobně štítový..."

Než však stihl něco dořící, ozvalo se náhle zezadu z chodby jasné zvolání: „Tak už to hoď, vybuchne to!"

Inženýr přejel zděšeným nechápavým pohledem po ostatních a ještě než si to stihl uvědomit, už proskakoval průlezem z můstku do chodby vedoucí k prostorům pro posádku.

Vběhl do místnosti, ve které se tísnila většina shromážděných kolonistů, a pátral očima po něčem, co by mohlo vybuchnout. Výbuch na lodi ve vesmíru mohl mít fatální následky! Pak se ale zarazil, pohlédl na kolonisty v kroužku, rozchechtal se a svalil se smíchy na zem – kolonisté hráli pro ukrácení času hru!

32-4-1 Výbušné koťátko — 9 bodů

Kolonisté se pro ukrácení času rozhodli zahrát si hru – jedno z dětí z kolonie si s sebou totiž vzalo kulatou hračku v podobě koťátka, kterou si hráči hází a koťátko vždy po náhodné době „vybuchne" a postříká toho, kdo ho právě drží, obarvenou vodou. Ten ze hry vypadává a ostatní hrají dál.

Bohužel tato hračka už byla stará a náhodné odpočítávadlo přestalo být náhodné. Teď se stávalo, že koťátko vždy „vybuchne" po stejné době, která akorát stačí na to, aby si koťátko hodilo K hráčů.

Kolonistům to ale zjevně nevadí. N hráčů si stoupne do kroužku, nultý hráč koťátko spustí a hodí hráči po své levici. Ten ho opět hodí hráči po své levici a to se opakuje tak dlouho, než je koťátko hozeno K-krát – v náručí K-tého

hráče „vybuchne“ a ten tak vypadává ze hry a vystupuje z kroužku ven. Přitom ale koťátko opět spustí a hodí ho opět hráči po své levici (a po K přehozeních opět vybuchne).

Postupně vypadávají všichni hráči, až zbude samotný poslední hráč, který vyhrává. My bychom chtěli vyhrát – zajímalo by nás tak, na jakou pozici se postavit v kroužku N hráčů, když víme, že koťátko „vybuchne“ u každého K-tého hráče. A chtěli bychom to spočítat rychleji, než to odsimulovat za NK kroků.

Po tomto výbušném rozptýlení se posádka spolu se ředitelem kolonie opět vydali k plánovacímu stolu. Rychle se dohodli, že útok na cizí základnu je nezbytný. Pokusí se s Hefaistem co nejrychleji přistát, vyslat přepadové komando v těžkých skafandrech.

Základna měla obranný systém a nějaký set štítových emitorů. Hefaistos měl jen protimeteorické dělo, které se ale nabíjelo na plný výkon tak dlouho, že byl čas jen na jeden pořádný výstřel. Pilot s inženýrem připravili program, který měl při sestupu vybrat ten nejlepší cíl a vystřelit na něj.

Dva další technici, kapitánka a oba mariňáci se mezitím navlékli do těžkých skafandrů a připravili se u hangárového otvoru společně s nákladní plošinou obtěžkanou plazmovými řezáky a další technikou na vyrábění děr do dveří. Zbytek posádky se oblékl do lehčích skafandrů a i kolonisté každý dostali nouzový skafandr, kdyby se něco stalo.

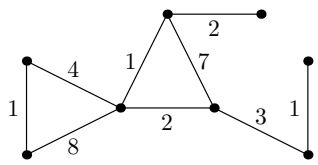
Pak se na odvrácené straně planety zažehly hlavní motory Hefaista a nasměrovaly ho na prudkou sestupovou trajektorii. Hefaistos se roztřásl, ale držel pevně – na tvrdé vstupy do atmosféry byl přeci dělaný. Bez varování se loď vyřítila z noční polokoule do oslepujícího světla místního slunce, ale nic si z toho nedělala. Senzory začaly pátrat po naprogramovaných cílech. Kolem náhle proletěl soustředěný výboj energie, ale loď minul. Senzory teď už ale věděly, po čem se dívat. S metodickou přesností zaměřovaly jednotlivé energetické stopy na povrchu. Systém věděl, že má jenom jeden výstřel, a tak stále propočítával. Kolem proletěl druhý energetický výboj, ale i ten loď na sestupové dráze minul.

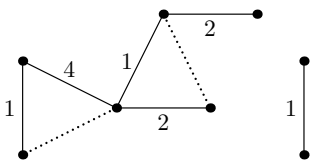
32-4-2 Jeden výstřel — 10 bodů

Senzory Hefaista zaznamenaly na povrchu u cizí základny soustavu pospojovaných energetických bodů zajišťujících obranu této základny. Body tvoří vrcholy pomyslného grafu a spoje mezi nimi pak jeho (neorientované) hrany. Soustava bodů nemusí být nutně souvislá, může tvořit více samostatných komponent.

Jednotlivé spoje (neboli hrany) přispívají do energetické soustavy různou silou. Každý spoj je ohodnocený kladným číslem a při čerpání energie ze soustavy se pak sečte energie spojů na *minimální kostře* zkonstruované podle těchto ohodnocení. Pokud soustava není souvislá, tak se spočítá minimální kostra v každé komponentě samostatně a výsledná energie je pak součet energií všech minimálních koster.

Hefaistos má protimeteorické dělo, které ale stihne vystřelit jen jednu ránu, kterou zvládne přerušit jeden spoj (jednu hranu v grafu). Vymyslete algoritmus, který určí v grafu takový spoj, jehož odstraněním se co nejvíce sníží energie celé energetické soustavy.





Například na energetické síti z levé části obrázku vychází nejlépe přerušit spoj s ohodnocením 3, výsledné kostry s celkovou energií 11 jsou pak vyznačeny v pravé části obrázku. Kdybychom namísto toho přerušili například spoj s ohodnocením 4, tak si vůbec nepomůžeme a výsledná energie zkonstruované kostry dokonce vzroste na 18.

Protimeteorické dělo na Hefaistu se zacílilo a pak lodí lehce trhlo, když vypustilo plazmový výboj na přesně vybrané místo energetické sítě u cizí základny. Spoj se s oslepujícím zábleskem rozletěl do okolí, štíty základny pohasly a Hefaistos se již obracel ke kruhovému obletu a vysouval přistávací nohy, když se proti němu rozletěl třetí energetický výboj z děla u základny. Tentokrát již neminul.

Proud energie proletěl pravou zadní částí Hefaista, utrhl část motorové sekce a prošel obálkou jeho fúzního reaktoru. Nouzové pojistky a pyropatrony, které byly hluboko do Hefaistova trupu instalovány před dlouhými lety v loděnici, v mžiku sekundy odstřelily zbylé plátování trupu a vymrštily selhávající fúzní reaktor co nejdál od lodi. Stihly to, na poslední chvíli. Reaktor se roztrhl až třicet metrů od trupu. Loď obklopil proud superpřehřáté plazmy o teplotě menšího slunce, který spaloval, co mu přišlo do cesty. Ohořelý ale stále celistvý hlavní trup Hefaista se z ohnivé koule vynořil o sekundu později a řítil se v kotrmelcích k povrchu.

Další z nouzových systémů odolné lodě zafungoval na výbornou a vnitřní prostory posádky okamžitě zaplavila extrémně expanzní vysokoporézní balistická pěna, která bleskově tuhla a chránila svůj cenný lidský náklad před nejhoršími nárazy. Hlavní počítač se po poškození svého jádra zasekl v nějaké smyčce vypisující na obrazovky na můstku sled chybových hlášení `E3, F5, E7, G6`*. Ale jeho práci převzaly záložní počítače, jejichž jedinou starostí byla stabilizace lodi před dopadem. Nějakým zázrakem se jim povedlo loď pomocí trysek přetočit opět do polohy na břicho a se zbytky kdysi devítisettunové devadesátimetrové lodě přistát klouzavým pohybem – jestli se přistáním dá nazývat prudký náraz do země s odskočením a vyrytím třísetmetrové brázdy.*

⋆⋆⋆

Balistická pěna uchránila posádku před nejhorším, i když asi nikdo nevyvázl bez alespoň lehkého zranění nebo pohmoždění. Brzy se rozsvítila nouzová světla a jednotliví lidé začali prudkými pohyby drtit pěnu, která je fixovala na jejich místech a nyní se začala rozpadat.

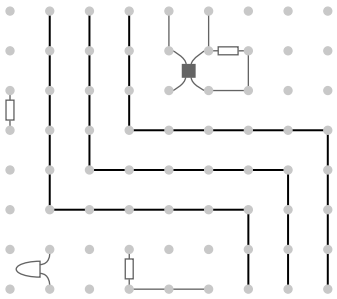
Hlavní inženýr se pokoušel rychle se probojovávat rozpadající se ztuhlou pěnou, než se dostal přes prostor osádky do hangáru, kde čekalo přepadové komando. Ti byli naštěstí uchráněni svými těžkými skafandry a všichni vypadali v pořádku, ale dveře se odmítaly otevřít. A to ani na manuální ovládání, i když do nich energie očividně šla. Hlavní inženýr rychle odtrhl krycí panel a podíval se na spálenou elektronickou destičku řídícího mechanismu. Odněkud magicky zhmotnil páječku a dal se do nouzové opravy.

32-4-3 Sběrnice na pájivém poli — 12 bodů

Hlavní inženýr potřebuje opravit řídící destičku od mechanismu otevírání dveří. Destičku si můžeme představit jako pájivé pole, což je většinou nějaká mřížka $R \times S$ kontaktů, kde můžeme pomocí pájení propojovat kontakty do cestiček.

Kdo pájivé pole neznáte, vyhledejte si na internetu obrázky. Pro naše potřeby stačí vědět, že lze na pájivém poli vytvořit cestička z „políček“ sousedících hranou (každé políčko mimo krajních tak má 4 sousedy).

Některá políčka na řídící destičce jsou již obsazena různými součástkami. Hlavní inženýr by ale potřeboval protáhnout po destičce od její vrchní hrany k její spodní hraně co nejširší *sběrnici*. Sběrnici představuje několik cestiček, které vedou po pájivém poli těsně vedle sebe a nikde se nerozcházejí, jak ukazuje například následující obrázek (tečky jsou středy políček, různé součástky představují plná políčka, tmavě je vyznačená sběrnice).



Vymyslete algoritmus, který na vstupu dostane mapu volných políček na nepájivém poli a zjistí, jakou nejširší sběrnici (kolik cestiček vedle sebe) můžeme po volných políčkách nepájivého pole protáhnout od jeho horního ke spodnímu okraji. Sběrnice široká K musí začínat na K vedle sebe ležících volných políčkách na prvním řádku a končit na K vedle sebe ležících volných políčkách na posledním řádku.

Nezapomeňte v této úloze dokázat, že vaše řešení vždy vrátí optimální výsledek (že nemůže existovat sběrnice s více cestičkami).

Hlavní inženýr odložil páječku, zasunul opravenou destičku zpět do slotu a praštil rukou do tlačítka nouzového otevírání. Těžké dveře chvíli vzdorovaly, protože vnější obšívka trupu byla výbuchem fúzního reaktoru spečena do celistvé skořápky, ale pak se skřípavým trhnutím hydraulické vzpěry skořápku prolomily a do lodi se vehnalo denní světlo.

Přepadové komando příliš nečekalo a v těžkých skafandrech se okamžitě vrhlo ven. Dva technici za sebou táhli nákladní plošinu naloženou užitečnými nástroji.

Vypadalo to, že zásah protimeteorického děla a padající trosky z Hefaista základnu celkem poničily – na hlavní budově zapuštěné do skalního masivu svítila zvenku jen nouzová světla a z některých míst základny se kouřilo. Štítový generátor i všechny obranné systémy vypadaly neaktivní – otázkou ale bylo, jak dlouho to vydrží.

V půlce cesty překvapilo komando trojici lidí v lehkých skafandrech, kteří se pravděpodobně vydali na průzkum vraku – nenapadlo je asi, že by takový pád mohl někdo přežít, nečekali, že loď bude tak odolná. Když zahlédli pětici mohutných postav, z nichž dvě svíraly nějaké zbraně, tak se na místě bez odporu vzdali. Mariňáci je jen svázali a kapitánka mezitím zavolala vysílačkou na Hefaista, aby pro ně někdo došel. Nebyl čas se zastavovat, na základně stále někde bylo šest bývalých obyvatel kolonie.

Po několikasetmetrovém přesunu dorazili k hlavním dveřím zasazeným do skalního masivu. „Prořezat tohle bude trvat aspoň hodinu, možná i víc," zhodnotil stav dveří jeden z techniků. Druhý technik, lodní expert na počítače Jason, mezitím napojil přenosný terminál na datový port u panelu dveří.

„Nejsou zakódované, jenom blokují otevírací signál z tohoto panelu", trhnul hlavou k ovládání vedle dveří. „Můžu se pokusit poslat otevírací signál smyčkou přes jiné panely... ale oni ten signál taky okamžitě blokují. Musím najít nejkratší cestu zpět."

32-4-4 Zpětný signál — 10 bodů

Pro vniknutí do základny by přepadový tým potřeboval otevřít hlavní vstupní dveře. Signál pro jejich otevření z lokálního panelu je ale blokován, je nutné signál poslat oklikou přes jiné počítače v síti.

Současně se ale osazenstvo základny brání a signály na otevření dveří blokují. Potřebovali bychom tedy najít co nejrychlejší cestu sítí zpět k našemu panelu, aby měli co nejmenší šanci ho zastavit.

Počítačová síť je tvořena jednotlivými uzly (řídící panel u vstupních dveří je jedním z nich), které jsou spojeny jednosměrnými komunikačními linkami. Navíc každá linka má jinou rychlost přenášení signálu, u každé komunikační linky tedy dostaneme číslo v milisekundách, jak dlouho trvá signálu přejít přes tuto komunikační linku do dalšího počítačového uzlu.

Naším cílem je tedy najít cestu, která začíná u našeho řídícího panelu, projde nějakou cestou sítí po komunikačních linkách a vrátí se nazpět do našeho panelu. Ze všech takových cest by měla být nejkratší.

Toto je praktická open-data úloha. V odevzdávacím systému si necháte vygenerovat vstupy a odevzdáte příslušné výstupy. Záleží jen na vás, jak výstupy vyrobíte.

Formát vstupu: Na prvním řádku vstupu dostanete dvě čísla N a M udávající počet počítačů v síti a počet jednosměrných komunikačních linek mezi nimi. Na druhém řádku pak bude jedno číslo S udávající index řídícího panelu, ze kterého chceme vyslat signál a do kterého chceme, aby se signál nakonec vrátil. Na dalších M řádcích pak budou vždy trojice čísel a, b, x udávající, že z počítače a existuje přímá komunikační linka do počítače b a přenést po ní signál trvá x milisekund. Počítače indexujeme od nuly. Zaručujeme, že vstup nebude obsahovat smyčky (tedy vždy $a \neq b$) a že všechna x budou kladná.

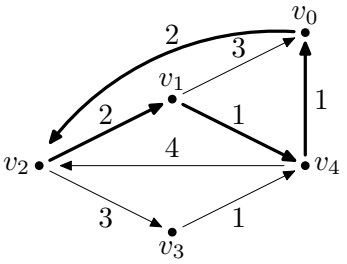
Formát výstupu: Na první řádek výstupu vypište délku (v milisekundách) vámi nalezené nejkratší cesty vedoucí z vrcholu S zpět do vrcholu S. Na druhý řádek vypište počet počítačů na této cestě (počítaje počáteční i koncový) a na další řádek vypište mezerami oddělenou posloupnost indexů počítačů na této cestě.

Ukázkový vstup:

```
5 8
2
0 2 2
1 0 3
1 4 1
2 1 2
2 3 3
3 4 1
4 0 1
4 2 4
```

Ukázkový výstup:

```
6
5
2 1 4 0 2
```



Po Jasonově konzoli běhaly proudy písmen. Technik se pod hledím těžkého skafandru potil, ale nakonec vítězoslavně zvedl hlavu a ťukl do potvrzovací klávesy. Chvíli se nic nedělo, ale pak se hlavní dveře lehce zasunuly do podlahy a odhalily vstup do přechodové komory.

Pětičlenná skupinka vstoupila dovnitř. Vnitřní dveře přechodové komory již nebyly nijak blokované a hladce se před nimi otevřely. Ocitli se v krátké chodbě s několika bočními

místnostmi – vnitřní prostory této základny nebyly příliš rozlehlé. Největší utěsněnou místností byl hangár spojený s dílnou, kde zrovna pracovali na úpravě jedné z družic, a pak sekce s několika laboratořemi, ve kterých to sršelo blesky. S utěsněným hangárem ještě sousedil větší neutěsněný hangár, ve kterém objevili malou transportní loď, kterou pravděpodobně vynášeli družice na oběžnou dráhu.

Na druhé straně základny pak byla obytná část a řídící centrum. Netrvalo dlouho a mariňákům se povedlo najít všech šest zbývajících obyvatel základny. Nikdo z nich očividně nebyl voják a podle některých nalezených papírů to celé byla akce velké korporace, která částečně financovala i výstavbu celé kolonie. Kapitánka Laren si v duchu oddechla – aspoň že to není nějaká tajná akce koloniální armády.

Poté, co do základny přivedli z havarovaného Hefaista všechny kolonisty, se ředitel kolonie přišel podívat na vězně.

„Taylore, jak jen jsi mohl? Co jste tu probůh dělali a proč?“ nemohl uvěřit tomu, že člověk, jehož ještě před měsícem považoval pouze za jednoho z geologů kolonie, kteří rádi vyráželi na dlouhé výpravy do pustiny, šéfuje nějaké skupině vědců zahrávajících si s iontovými bouřemi.

Než však mohl pokračovat, přerušilo všechny hlasité zakvílení alarmu od hlavní přehledové mapy. Po chvíli zkoumání a vyzvídání od zajatých vědců zjistili, že na celé planetě se v několika uzavřených stanicích „pěstují“ iontové bouře držené na místě a posilované magnetickými silovými paprsky. A stanice nejblíže této základně vlivem poškození energetické sítě zkolabovala a je potřeba ji opět nahodit.

32-4-5 Svázání bouře — 13 bodů

Iontová bouře byla držena ve své „líhni“ pomocí magnetických silových paprsků generovaných na přímých spojnicích mezi emitory. Každý emitor zvládne generovat i více magnetických silových paprsků, ale nikdy se žádné dva magnetické silové paprsky nesmí křížit (podle staré zásady „Nikdy nekřižte paprsky!“ by mohlo dojít až ke konci světa, nebo minimálně ke konci daných emitorů).

Známe souřadnice jednotlivých emitorů v rovině. Chceme naplánovat, mezi kterými dvojicemi emitorů budeme generovat paprsky tak, aby platilo:

- Žádné dva paprsky se nekříží.
- Neexistuje žádná dvojice emitorů, mezi něž bychom ještě mohli přidat paprsek tak, aniž by se porušila první podmínka.

Vymyslete algoritmus, který pro zadané souřadnice vydá dvojice emitorů, mezi kterými generovat paprsky, aby byly podmínky výše splněné. Vaše řešení nemusí obsahovat co nejvíce paprsků, ani paprsky nemusí tvořit žádný speciální tvar. Jde jen o to, že tam již nepůjde žádný další paprsek přidat.

Iontovou bouři se povedlo opět svázat do její „líhně“. Podle všeho byly tyto líhně na planetě celkem čtyři a dvě z nich teď držely připravené iontové bouře (jedna iontová bouře byla použita na prvotní poničení kolonie a druhá pak na druhý útok na kolonii, vyrobit nové prý trvá několik měsíců).

Byli teď ale v nezáviděníhodné situaci. Na planetě byly dvě subprostorové vysílací stanice – jedna z nich na vážně poškozeném Hefaistu bez fůzního reaktoru zabořeném do země a druhá na hlavní základně kolonie, která byla prakticky smetena z povrchu planety. Navíc v této malé základně nebylo dostatek prostoru ani jídla, aby zvládla všechny kolonisty i posádku. Před posádkou Hefaista a před kolonisty tak stálo (již poslední) náročné rozhodnutí. Co dělat teď? Opět zahlasujte v anketě.[1]

⋆⋆⋆

Možná vám vrtá hlavou, v jaké smyčce že se to zasekl hlavní počítač Hefaista po poškození lodě při sestupu. K vysvětlení této části příběhu se musíme vrátit zhruba dvacet minut před zahájením útoku na cizí základnu, dokud byl Hefaistos ještě na oběžné dráze.

V tu chvíli se dva kolonisti, shodou náhod nejmladší a nejstarší ze všech kolonistů, rozhodli ukrátit si čekání na další vývoj situace nějakou hrou. Hra s výbušným koťátkem je přestala bavit a tak si mladý Tom spustil na jednom z panelů na stěně své oblíbené sudoku. A starý Azachiáš si zase chtěl zahrát partii šachů proti lodnímu počítači.

V tu chvíli se ale objevila v sekci pro posádku kapitánka a začala vysvětlovat detaily nadcházející akce. Tom i Azachiáš nechali hry na panelech spuštěné a úplně na ně zapomněli.

Jenže hlavní počítač ne. V momentě jeho kritického poškození došlo k pomíchání běžících programových bloků, přepsání paměti procesů a ke spojení programů pro řešení sudoku a hraní šachů do jednoho. A v této smyčce hlavní počítač stále běží, napájen z nouzových článků na Hefaistovi...

32-4-6 Sudoku s koněm — 15 bodů

Toto je optimalizační kompetitivní úloha – na problém, který vám zde ukážeme, pravděpodobně neobjevíte optimální algoritmus, který by vám našel nejlepší možný výsledek. Ale můžete se pokusit najít dostatečně dobrý výsledek – nejlépe takový, který bude lepší, než výsledky všech ostatních řešitelů.

Bodovat tuto úlohu budeme tak, že nejlepší řešení dostane plný počet bodů a ostatní dostanou body odstupňovaně podle toho, jak dobrého skóre dosáhnou. Vše vyhodnotíme po konci série. A přidáme i nějaká referenční organizátorská řešení :)

Úloha

Pro Sudoku, neboli mřížku 9×9, existuje posloupnost tahů šachovým jezdcem, která navštíví každé políčko právě jednou (bude tak mít právě 81 kroků), nemusí se však vrátit na startovní pozici. Takové posloupnosti často říkáme *otevřená jezdcova procházka* a najít jí není tak těžké. Také je to pouze jedna z částí naší úlohy.

Pro nějaké vyřešené Sudoku a pro konkrétní procházku jezdcem definujme *skóre* jako posloupnost 81 cifer v pořadí, v jakém na ně jezdec na vyřešeném Sudoku vstoupí.

Příkladem může být třeba následující Sudoku s následujícím pořadím návštěv jezdcem (začínáme na nule) a následujícím skóre:

```
1 2 3 8 7 4 5 6 9
8 4 5 1 9 6 7 2 3
6 7 9 2 3 5 1 4 8
2 1 4 7 5 9 8 3 6
3 9 6 4 8 1 2 7 5
7 5 8 6 2 3 9 1 4
9 6 2 3 1 8 4 5 7
4 8 1 5 6 7 3 9 2
5 3 7 9 4 2 6 8 1
```

[1] `https://poll.ly/#/GxbD1xrP`

```
80 77 50 41 52 73 70 43 12
49 40 53 78  3 42 11 72 69
76 79  2 51 74 71 44 13 10
39 48 75 54 57  4  9 68 65
30  1 58 47  8 55 66 45 14
23 38 29 56 59 46  5 64 67
 0 31 24 35 28  7 60 15 18
37 22 33 26 61 20 17  6 63
32 25 36 21 34 27 62 19 16
```

```
999999988887945513787987235218365143745248661734
183275716562466216243355243462171
```

Vaším úkolem je najít takové vyřešené Sudoku, na kterém půjde najít procházku takovou, která dá co nejvyšší skóre (když budeme skóre chápat jako číslo o 81 cifrách).

Pro řešení Sudoku klidně můžete použít externí knihovny či nějaké weby, není naším cílem zkoušet vás z napsání solveru na Sudoku. Ale napište nám prosím odkaz na použité nástroje do řešení.

Co odevzdávat

Budeme po vás chtít jednak co nejvyšší nalezené skóre (a Sudoku s procházkou, které k němu vedou), ale také slovní popis řešení říkající, jakou cestou jste k výsledku došli. Můžete přibalit i zdroják programu či sbírku skriptů, které jste použili.

Upozorňujeme, že bez slovního popisu a vysvětlení, jak jste k výsledku došli, nebudeme výsledky uznávat. Pamatujte na to.

Své řešení odevzdávejte jako ZIP obsahující:

- Soubor `sudoku.txt` se zápisem Sudoku ve formátu jako výše (9 řádků, čísla oddělená mezerami)
- Soubor `prochazka.txt` s procházkou ve formátu jako výše (9 řádků, čísla oddělená mezerami)
- Soubor `skore.txt` s 81-ciferným skóre získaným z procházky
- Slovní popis řešení (třeba v PDF) a případně jakékoliv další použité programy.

U prvních tří souborů se prosím pokoušejte držet zadaného formátu, velmi nám to usnadní kontrolu. Děkujeme.

Čtvrtý díl příběhu pro vás sepsal

Jirka Setnička

Recepty z programátorské kuchařky: Minimální kostra

Představme si následující problém: Chceme určit silnice, které se budou v zimě udržovat sjízdné, a to tak, abychom celkově udržovali co nejméně kilometrů silnic, a přesto žádné město od ostatních neodřízli.

Města a silnice si můžeme představit jako graf, o kterém nyní budeme předpokládat, že je souvislý. Kdyby nebyl, náš problém nijak vyřešit nelze. Výsledný podgraf/seznam silnic, který řeší náš problém se sněhem, nazývají matematici *minimální kostra grafu*.

Pokud vůbec netušíte, co je to graf, přečtěte si úvodní grafovou kuchařku na našem webu.[2]

Co se v souvislém grafu přesně myslí pod pojmem *kostra*? Nazveme jí libovolný podgraf, který obsahuje všechny vrcholy a zároveň je stromem. *Strom* jsme si definovali v kuchařce o grafech; jsou to přesně ty grafy, které jsou souvislé (z každého vrcholu „dojedeme“ do každého jiného) a bez kružnice (takže nemáme v silniční síti žádné přebytečné cesty).

Pokud každou hranu grafu ohodnotíme nějakou *vahou,* což v našem případě bude vždy kladné číslo, dostaneme *ohodnocený graf.* V takových grafech pak obvykle hledáme mezi všemi kostrami *kostru minimální,* což je taková, pro kterou je součet vah jejích hran nejmenší možný.

Graf může mít více minimálních koster – například jestliže jsou všechny váhy hran jedničky, všechny kostry mají stejnou váhu $n-1$ (kde n je počet vrcholů grafu), a tedy jsou všechny minimální.

Pro vyřešení problému hledání minimální kostry se nám bude hodit datová struktura *Disjoint-Find-Union* (DFU). Ta umí pro dané disjunktní množiny (disjunktní znamená, že každé 2 množiny mají prázdný průnik neboli žádné společné prvky) rychle rozhodnout, jestli dva prvky patří do stejné množiny, a provádět operaci sjednocení dvou množin.

Algoritmus

Algoritmus na hledání minimální kostry, který si předvedeme, je typickou ukázkou tzv. hladového algoritmu. Nejprve setřídíme hrany vzestupně podle jejich váhy. Kostru budeme postupně vytvářet přidáváním hran od té s nejmenší vahou tak, že hranu do kostry přidáme právě tehdy, pokud spojuje dvě (prozatím) různé komponenty souvislosti vytvořeného podgrafu. Jinak řečeno, hranu do vytvářené kostry přidáme, pokud v ní zatím neexistuje cesta mezi vrcholy, které zkoumaná hrana spojuje.

Je zřejmé, že tímto postupem získáme kostru, tj. acyklický podgraf grafu, který je souvislý (pokud vstupní graf je souvislý, což mlčky předpokládáme). Než si ukážeme, že nalezená kostra je opravdu minimální, podívejme se na časovou složitost našeho algoritmu: Pokud vstupní graf má N vrcholů a M hran, tak úvodní setřídění hran vyžaduje čas $\mathcal{O}(M \log M)$ (použijeme některý z rychlých třídicích algoritmů popsaných v jednom z minulých dílů kuchařky) a poté se pokusíme přidat každou z M hran.

V druhé části kuchařky si ukážeme datovou strukturu, s jejíž pomocí bude M testů toho, zda mezi dvěma vrcholy vede hrana, trvat nejvýše $\mathcal{O}(M \log N)$. Celková časová složitost našeho algoritmu je tedy $\mathcal{O}(M \log N)$ (všimněte si, že $\log M \leq \log N^2 = 2 \log N$). Paměťová složitost je lineární vzhledem k počtu hran, tj. $\mathcal{O}(M)$.

Důkaz správnosti

Zbývá dokázat, že nalezená kostra vstupního grafu je minimální. Bez újmy na obecnosti můžeme předpokládat, že váhy všech hran grafu jsou navzájem různé: Pokud tomu tak není již na začátku, přičteme k některým z hran, jejichž váhy jsou duplicitní, velmi malá kladná celá čísla tak, aby pořadí hran nalezené naším třídicím algoritmem zůstalo zachováno. Tím se kostra nalezená hladovým algoritmem nezmění, a pokud bude tato kostra minimální s modifikovanými váhami, bude minimální i pro původní zadání.

[2] `http://ksp.mff.cuni.cz/viz/kucharky/grafy`

Označme si nyní T_{alg} kostru nalezenou hladovým algoritmem a $T_{\min}$ nějakou minimální kostru. Co by se stalo, kdyby byly různé? Víme, že všechny kostry mají stejný počet hran, takže musí existovat alespoň jedna hrana e, která je v T_{alg}, ale není v $T_{\min}$. Ze všech takových hran si vyberme tu, která má nejmenší váhu, tedy kterou algoritmus přidal jako první. Když se podíváme na stav algoritmu těsně před přidáním e, vidíme, že sestrojil nějakou částečnou kostru F, která je ještě součástí jak $T_{\min}$, tak T_{alg}.

Přidejme nyní hranu e ke kostře $T_{\min}$. Tím vznikl podgraf vstupního grafu, který zjevně obsahuje nějakou kružnici C – už před přidáním hrany e totiž $T_{\min}$ byla souvislá. Protože kostra T_{alg} neobsahuje žádnou kružnici, na kružnici C musí být alespoň jedna hrana e', která není v T_{alg}.

Všimněme si, že hranu e' nemohl algoritmus zpracovat před hranou e: hrana e' neleží v $T_{\min}$ na žádném cyklu, takže tím spíš netvoří cyklus v F, a kdyby ji algoritmus zpracoval, musel by ji přidat do F, což, jak víme, neučinil. Z toho plyne, že váha hrany e' je větší než váha hrany e. Když nyní z kostry $T_{\min}$ odebereme hranu e' a přidáme místo ní hranu e, musíme opět dostat souvislý podgraf (e a e' přeci ležely na společné kružnici), tudíž kostru vstupního grafu. Jenže tato kostra má celkově menší váhu než minimální kostra $T_{\min}$, což není možné. Tím jsme došli ke sporu, a proto $T_{\min}$ a T_{alg} nemohou být různé.

Cvičení

- V důkazu jsme předpokládali, že váhy hran jsou různé (resp. jsme je různými udělali). Není potřeba i v samotném algoritmu přičítat velmi malá čísla k hranám se stejnou vahou?

Disjoint-Find-Union

Datová struktura *DFU* slouží k udržování rozkladu množiny na několik disjunktních podmnožin (čili takových, že žádné dvě nemají společný prvek). To znamená, že pomocí této struktury můžeme pro každé dva z uložených prvků říci, zda patří či nepatří do stejné podmnožiny rozkladu.

V algoritmu hledání minimální kostry budou prvky v DFU vrcholy zadaného grafu a budou náležet do stejné podmnožiny rozkladu, pokud mezi nimi v již vytvořené části kostry existuje cesta. Jinými slovy podmnožiny v DFU budou odpovídat komponentám souvislosti vytvářené kostry.

S reprezentovaným rozkladem umožňuje datová struktura DFU provádět následující dvě operace:

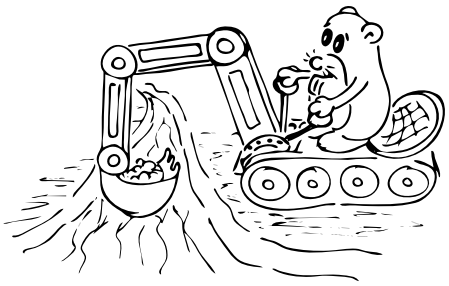
- **find:** Test, zda dva prvky leží ve stejné podmnožině rozkladu. Tato operace bude v případě našeho algoritmu odpovídat testu, zda dva vrcholy leží ve stejné komponentě souvislosti.
- **union:** Sloučení dvou podmnožin do jedné. Tuto operaci v našem algoritmu na hledání kostry provedeme vždy, když do vytvářené kostry přidáme hranu (tehdy spojíme dvě různé komponenty souvislosti dohromady).

Povězme si nejprve, jak budeme jednotlivé podmnožiny reprezentovat. Prvky obsažené v jedné podmnožině budou tvořit zakořeněný strom. V tomto stromu však povedou ukazatele (trochu nezvykle) od listů ke kořeni. Operaci *find* lze pak jednoduše implementovat tak, že pro oba zadané prvky nejprve nalezneme kořeny jejich stromů. Jsou-li tyto kořeny stejné, jsou prvky ve stejném stromu, a tedy i ve stejné podmnožině rozkladu. Naopak, jsou-li různé, jsou zadané prvky v různých stromech, a tedy jsou i v různých podmnožinách reprezentovaného rozkladu. Operaci *union* provedeme tak, že mezi kořeny stromů reprezentujících slučované podmnožiny přidáme ukazatel a tím tyto dva stromy spojíme dohromady.

Implementace dvou výše popsaných operací, jak jsme se ji právě popsali, následuje. Pro jednoduchost množina, jejíž rozklad reprezentujeme, bude množina čísel od 1 do N. Rodiče jednotlivých vrcholů stromu si pak pamatujeme v poli *parent*, kde 0 znamená, že prvek rodiče nemá, tj. že je kořenem svého stromu. Funkce *root*(v) vrátí kořen stromu, který obsahuje prvek v.

```
var parent: array[1..N] of integer;

procedure init;
var i: integer;
begin
  for i:=1 to N do parent[i]:=0;
end;

function root(v: integer):integer;
begin
  if parent[v]=0 then root:=v
  else root:=root(parent[v]);
end;

function find(v, w: integer):boolean;
begin
  find:=(root(v)=root(w));
end;

procedure union(v, w: integer);
begin
  v:=root(v); w:=root(w);
  if v<>w then parent[v]:=w;
end;
```

S právě předvedenou implementací operací *find* a *union* by se ale mohlo stát, že stromy odpovídající podmnožinám budou vypadat jako „hadi“, a pokud budou obsahovat N prvků, na nalezení kořene bude potřeba čas $\mathcal{O}(N)$.



Ke zrychlení práce DFU se používají dvě jednoduchá vylepšení:

- **union by rank:** Každý prvek má přiřazen *rank*. Na začátku jsou ranky všech prvků rovny nule. Při provádění operace *union* připojíme strom s kořenem menšího ranku ke kořeni stromu s větším rankem. Ranky kořenů stromů se v tomto případě nemění. Pokud kořeny obou stromů mají stejný rank, připojíme je libovolně, ale rank kořenu výsledného stromu zvětšíme o jedna.
- **path compression:** Ve funkci *root*(v) přepojíme všechny prvky na cestě od prvku v ke kořeni rovnou na kořen, tj. změníme jejich rodiče na kořen daného stromu.

Než si obě metody blíže rozebereme, podívejme se, jak se změní implementace funkcí *root* a *union*:

```
var parent: array[1..N] of integer;
    rank: array[1..N] of integer;
```

```
procedure init;
var i: integer;
begin
  for i:=1 to N do
    begin
      parent[i]:=0;
      rank[i]:=0;
    end;
end;

{změna path compression}
function root(v: integer): integer;
begin
  if parent[v]=0 then root:=v
  else begin
    parent[v]:=root(parent[v]);
    root:=parent[v];
  end;
end;

{stejná jako minule}
function find(v, w: integer):boolean;
begin
  find:=(root(v)=root(w));
end;

{změna kvůli union by rank}
procedure union(v, w: integer);
begin
  v:=root(v);
  w:=root(w);
  if v=w then exit;
  if rank[v]=rank[w] then
    begin
      parent[v]:=w;
      rank[w]:=rank[w]+1;
    end
  else if rank[v]<rank[w] then
    parent[v]:=w
  else
    parent[w]:=v;
end;
```

Zaměřme se nyní blíže na metodu *union by rank.* Nejprve učiníme následující pozorování: Pokud je prvek v s rankem r kořenem stromu v datové struktuře DFU, pak tento strom obsahuje alespoň 2^r prvků.

Naše pozorování dokážeme indukcí podle r. Pro $r = 0$ tvrzení zřejmě platí. Nechť tedy $r > 0$. V okamžiku, kdy se rank prvku v mění z $r-1$ na r, slučujeme dva stromy, jejichž kořeny mají rank $r-1$. Každý z těchto dvou stromů má dle indukčního předpokladu alespoň 2^{r-1} prvků, a tedy výsledný strom má alespoň 2^r prvků, jak jsme požadovali.

Z našeho pozorování ihned plyne, že rank každého prvku je nejvýše $\log_2 N$ a prvků s rankem r je nejvýše $N/2^r$ (všimněme si, že rank prvku v DFU se nemění po okamžiku, kdy daný prvek přestane být kořen nějakého stromu).

Když tedy provádíme jen *union by rank,* je hloubka každého stromu v DFU rovna ranku jeho kořene, protože rank kořene se mění právě tehdy, když zvětšujeme hloubku stromu o jedna. A protože rank každého prvku je nanejvýš $\log_2 N$, hloubka každého stromu v DFU je také nanejvýš $\log_2 N$. Potom ale procedura *root* spotřebuje čas nejvýše $\mathcal{O}(\log N)$, a tedy operace *find* a *union* stihneme v čase $\mathcal{O}(\log N)$.

Amortizovaná časová složitost

Abychom mohli pokračovat dále, musíme si vysvětlit, co je *amortizovaná* časová složitost. Řekneme, že nějaká operace pracuje v amortizovaném čase $\mathcal{O}(t)$, pakliže provedení libovolných k takových operací trvá nejvýše $\mathcal{O}(kt)$. Přitom provedení kterékoliv konkrétní z nich může vyžadovat čas větší. Tento větší čas je pak v součtu kompenzován kratším časem, který spotřebovaly některé předchozí operace.

Nejdříve si předveďme tento pojem na jednoduchém příkladě. Řekněme, že máme číslo zapsané ve dvojkové soustavě. Přičíst k tomuto číslu jedničku jistě netrvá konstantní čas, neboť záleží na tom, kolik jedniček se vyskytuje na konci zadaného čísla. Pokud se nám ale povede ukázat, že N přičtení jedničky k číslu, které je na počátku nula, zabere čas $\mathcal{O}(N)$, pak můžeme říci, že každé takové přičtení trvalo amortizovaně $\mathcal{O}(1)$.

Jak tedy ukážeme, že N přičtení jedničky k číslu zabere čas $\mathcal{O}(N)$? Použijeme k tomu „penízkovou metodu“. Každá operace nás bude stát jeden penízek, a pokud jich na N operací použijeme jen $\mathcal{O}(N)$, bude tvrzení dokázáno.

Každé jedničce, kterou chceme přičíst, dáme dva penízky. V průběhu celého přičítání bude platit, že každá jednička ve dvojkovém zápisu čísla má jeden penízek (když začneme jedničky přičítat k nule, tuto podmínku splníme). Přičítání bude probíhat tak, že přičítaná jednička se „podívá“ na nejnižší bit (tj. ve dvojkovém zápisu na poslední cifru) zadaného čísla (to ji stojí jeden penízek). Pokud je to nula, změní ji na jedničku a dá jí svůj zbylý penízek. Pokud to je jednička, vezme si přičítaná jednička její penízek (čili už má zase dva), změní zkoumanou jedničku na nulu a pokračuje u dalšího bitu, atd.

Takto splníme podmínku, že každá jednička v dvojkovém zápisu čísla má jeden penízek. Tedy N přičítání nás stojí $2N$ penízků. Protože počet penízků utracených během jedné operace je úměrný spotřebovanému času; vidíme, že všech N přičtení proběhne v čase $\mathcal{O}(N)$. Není těžké si uvědomit, že přičtení některých jedniček může trvat až $\mathcal{O}(\log N)$, ale amortizovaná časová složitost přičtení jedné jedničky je konstantní.

Dokončení analýzy DFU

Pokud bychom prováděli pouze *path compression* a nikoliv *union by rank*, dalo by se dokázat, že každá z operací *find* a *union* vyžaduje amortizovaně čas $\mathcal{O}(\log N)$, kde N je počet prvků. Toto tvrzení nebudeme dokazovat, protože tím bychom si nijak oproti samotnému *union by rank* nepomohli. Proč tedy vlastně hovoříme o obou vylepšeních? Inu proto, že při použití obou metod současně dosáhneme mnohem lepšího amortizovaného času $\mathcal{O}(\alpha(N))$ na jednu operaci *find* nebo *union*, kde $\alpha(N)$ je inverzní Ackermannova funkce. Její definici můžete nalézt na konci kuchařky, zde jen poznamenejme, že hodnota inverzní Ackermannovy funkce $\alpha(N)$ je pro všechny praktické hodnoty N nejvýše čtyři. Čili dosáhneme v podstatě amortizovaně konstantní časovou složitost na jednu (libovolnou) operaci DFU.

Dokázat výše zmíněný odhad časové složitosti funkcí $\alpha(N)$ je docela těžké, my si zde předvedeme poněkud horší, ale technicky výrazně jednodušší časový odhad $\mathcal{O}((N+L)\log^* N)$, kde L je počet provedených operací *find* nebo *union* a $\log^* N$ je tzv. iterovaný logaritmus, jehož definice následuje. Nejprve si definujeme funkci $2 \uparrow k$ rekur-

zivním předpisem:

$$2 \uparrow 0 = 1, \quad 2 \uparrow k = 2^{2 \uparrow (k-1)}.$$

Máme tedy $2 \uparrow 1 = 2$, $2 \uparrow 2 = 2^2 = 4$, $2 \uparrow 3 = 2^4 = 16$, $2 \uparrow 4 = 2^{16} = 65536$, $2 \uparrow 5 = 2^{65536}$, atd. A konečně, iterovaný logaritmus $\log^* N$ čísla N je nejmenší přirozené číslo k takové, že $N \leq 2 \uparrow k$. Jiná (ale ekvivalentní) definice iterovaného logaritmu je ta, že $\log^* N$ je nejmenší počet, kolikrát musíme číslo N opakovaně zlogaritmovat, než dostaneme hodnotu menší nebo rovnu jedné.

Zbývá provést slíbenou analýzu struktury DFU při současném použití obou metod *union by rank* a *path compression*. Prvky si rozdělíme do skupin podle jejich ranku: k-tá skupina prvků bude tvořena těmi prvky, jejichž rank je mezi $(2 \uparrow (k-1)) + 1$ a $2 \uparrow k$. Např. třetí skupina obsahuje ty prvky, jejichž rank je mezi 5 a 16. Prvky jsou tedy rozděleny do $1 + \log^* \log N = \mathcal{O}(\log^* N)$ skupin. Odhadněme shora počet prvků v k-té skupině:

$$\frac{N}{2^{(2\uparrow(k-1))+1}} + \cdots + \frac{N}{2^{2\uparrow k}} = \frac{N}{2^{2\uparrow(k-1)}} \cdot \left(\sum_{i=1}^{2\uparrow k - 2\uparrow(k-1)} \frac{1}{2^i} \right) \leq$$

$$\leq \frac{N}{2^{2\uparrow(k-1)}} \cdot 1 = \frac{N}{2 \uparrow k}.$$

Teď můžeme provést časovou analýzu funkce *root*(v). Čas, který spotřebuje funkce *root*(v), je přímo úměrný délce cesty od prvku v ke kořeni stromu. Tato cesta je pak následně rozpojena a všechny prvky na ní jsou přepojeny přímo na kořen stromu. Rozdělíme rozpojené hrany této cesty na ty, které „naúčtujeme“ tomuto volání funkce *root*(v), a ty, které zahrneme do faktoru $\mathcal{O}(N \log^* N)$ v dokazovaném časovém odhadu. Do volání funkce *root*(v) započítáme ty hrany cesty, které spojují dva prvky, které jsou v různých skupinách. Takových hran je zřejmě nejvýše $\mathcal{O}(\log^* N)$ (všimněte si, že ranky prvků na cestě z listu do kořene tvoří rostoucí posloupnost).

Uvažme prvek v v k-té skupině, který již není kořenem stromu. Při každém přepojení rank rodiče prvku v vzroste. Tedy po $2 \uparrow k$ přepojeních je rodič prvku v v $(k+1)$-ní nebo vyšší skupině. Pokud v je prvek v k-té skupině, pak hrana z něj na cestě do kořene bude účtována volání funkce *root*(v) nejvýše $(2 \uparrow k)$-krát. Protože k-tá skupina obsahuje nejvýše $N/(2 \uparrow k)$ prvků, je počet takových hran pro všechny prvky této skupiny nejvýše N. A protože počet skupin je nejvýše $\mathcal{O}(\log^* N)$, je celkový počet hran, které nejsou započítány voláním funkce *root*(v), nejvýše $\mathcal{O}(N \log^* N)$. Protože funkce *root*(v) je volána $2L$-krát, plyne časový odhad $\mathcal{O}((N+L) \log^* N)$ z právě dokázaných tvrzení.

Inverzní Ackermannova funkce $\alpha(N)$

Ackermannovu funkci lze definovat následující konstrukcí:

$$A_0(i) = i + 1, \quad A_{k+1}(i) = A_k^i(i) \text{ pro } k \geq 0,$$

kde výraz A_k^i zastupuje složení i funkcí A_k, např. $A_1(3) = A_0(A_0(A_0(3)))$. Platí tedy následující rovnosti:

$$A_0(i) = i + 1, \quad A_1(i) = 2i, \quad A_2(i) = 2^i \cdot i.$$

Ackermannova funkce s jedním parametrem $A(k)$ je pak rovna hodnotě $A_k(2)$, takže $A(2) = A_2(2) = 8$, $A(3) = A_3(2) = 2^{11}$, $A(4) = A_4(2) \approx 2 \uparrow 2048$ atd... Hodnota inverzní Ackermannovy funkce $\alpha(N)$ je tedy nejmenší přirozené číslo k takové, že $N \leq A(k) = A_k(2)$. Jak je vidět, ve všech reálných aplikacích platí, že $\alpha(N) \leq 4$.

Dan Kráľ, Martin Mareš a Milan Straka