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 zápise 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 $2 \uparrow k$ rekurzivním 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 nebude 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:

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

Jednoparametrová Ackermannova funkce $A(k)$ je pak rovna hodnotě $A_k(2)$, čili $A(2) = A_2(2) = 8$, $A(3) = A_3(2) = 2^{11}$, $A(4) = A_4(2) \approx 2 \uparrow 2048$ 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 $\alpha(N) \leq 4$.

Dnešní menu vám servírovali
Dan Král', Martin Mareš a Milan Straka

Milí řešitelé a řešitelky!

Zdravíme všechny, kteří to s námi vydrželi až do třetí série. Vánoce jsou za dveřmi, a tak jsme si pro vás připravili malou nadílku. Dejte si lžíci bramborového salátu, kousněte si do kapra nebo si alespoň uždibněte kousek vanilkového rohlíčku a s chutí se pusťte do čtení …

Vaše řešení třetí série budeme dychtivě očekávat do pondělí 19. ledna 2009. Řešení můžete odevzdávat elektronicky na *http://ksp.mff.cuni.cz/submit/*, nebo klasickou poštou na známou adresu:

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

Aktuální informace o KSP můžete nalézt na stránkách *http://ksp.mff.cuni.cz/*, diskutovat můžete na fóru *http://ksp.mff.cuni.cz/forum/* a záludné dotazy organizátorům lze zasílat e-mailem na adresu *ksp@mff.cuni.cz*.

Třetí série dvacátého prvního ročníku KSP

Dnes sáhneme do trochu jiného soudku. Po dvou vydařených reportážích jsme se rozhodli zařadit trochu oddychovější téma. O pár řádků níže se můžete zakousnout do krátké sci-fi povídky s nádechem cyberpunku, kterou pro vás připravil Martin „Bobřík“ Kruliš. Měli jste někdy sen, který vypadal jako skutečný? Co když to není jen zdání a sen začne žít vlastním životem …?

„Zase jsem měl ten sen.“

„Opravdu? A který?“ pozvedl doktor obočí a dál čmáral cosi do svého poznámkového bloku. Sám nevím, jestli je to vlastnost všech psychologů, nebo zda to dělá jen ten můj, ale občas mám pocit, že vám vůbec nevěnují pozornost a snaží se jen udržovat konverzaci hloupými dotazy.

„Ten o těch lidech,“ řekl jsem unaveně. „Vyspělá civilizace, utopická společnost a tak dále. Už ani nevím, kdy se mi naposledy zdálo něco jiného.“

„A zdá se vám pokaždé to samé, nebo pozorujete drobné rozdíly?“ zeptal se s dobře hraným výrazem neutuchajícího zájmu.

„Je to pokaždé … trochu jiné. Skoro jako seriál – každý sen je o něčem novém, ale zároveň stále o tom samém … “

⋆⋆⋆

Opět mi neřekl nic převratného. Stále ty samé řeči o přepracování a stresu. Nemyslím, že by to byl zrovna můj případ. Jmenuji se Harold a tohle je můj život. Pracuji na pozici řadového úředníka v účetní firmě. Procházím formuláře a počítám nejrůznější statistiky. Můj život je klidný a předvídatelný. Největší vzrušení jsem zažil, když kolega z kanceláře ztratil sešívačku a celá směna úředníků mu ji pomáhala hledat. Takže jak jsem říkal – s tím stresem se doktor trochu netrefil.

Po návštěvě ordinace mě čekala práce. Celý den proběhl poklidně – ostatně jako každý jiný den. Žádné vzrušení. Žádný stres. Kancelář jsem opustil přesně v 17 hodin a zamířil do podzemky. Byla ošklivě přeplněná, ale takový už je život. Stálo mě to zpoždění dlouhé přesně tři minuty a čtyřicet pět vteřin.

Doma následovala obvyklá večerní rutina. Nakrmit rybičky, večeře a televizní zprávy na kanálu 6. Dělám to tak každý den. Do postele jsem ulehl s rozečtenou knihou a pomalu se ukolébal čtením.

⋆⋆⋆

Pohled na město byl nádherný. Táhlo se od obzoru k obzoru a stříbrně lesklé výškové budovy šplhaly k nebi. Každý den vyrostla alespoň jedna nová. A mezi nimi na různých letových hladinách proplouvala vznášedla. Úchvatný pohled, a přitom nic neobvyklého. Běžná denní doprava. Lidé jedou do práce, děti do školy …

V jedné nevýznamné budově na okraji města uvnitř velikého sálu mezitím pobíhali lidé v bílých pláštích a křičeli na sebe nesrozumitelné věci. Podobali se mravencům, kterým právě někdo šlápl do mraveniště. Shlukovali se do skupinek a vášnivě debatovali. Co chvíli se zase rozprchli do všech směrů a vytvořili nové skupinky. Hluk sílil a s ním i zoufalství pobíhajících lidí.

Místnost prořízl ostrý pískavý zvuk. Všichni jako na povel utichli a obrátili se k řečnickému pultu, ke kterému právě z davu vystoupil postarší muž s plnovousem. Opatrně přejel ostatní nervózním pohledem a napjatou atmosféru ještě vylepšil váhavým odkašláním.

„Vážení kolegové,“ začal rozechvělým hlasem, „není o tom již nejmenších pochyb. Všechny naše hypotézy se potvrdily sérií nezávislých experimentů, a tak zbývá jediné rozumné vysvětlení … “ Na chvíli se odmlčel, aby si osušil pot z čela, a pak pokračoval: „Žijeme ve snu! Celá naše civilizace byla vytvořena snem někoho jiného! Navíc nejsme schopni určit, kdy tento sen –“

⋆⋆⋆

Probudil jsem se. Byly čtyři hodiny ráno a na okno tiše bubnoval déšť. Poslední útržky snu pomalu zahaloval mrak zapomnění. Oči se prodíraly tmou a postupně rozeznávaly obrysy nábytku mého pokoje. Posadil jsem se na posteli a snažil se uklidnit. Nešlo to. Uběhla skoro hodina a já stále nehybně seděl. Hlavou se mi honily nejrůznější myšlenky a paměť se snažila sen uspořádat. Únava mě přemáhala, ale strach mě držel vzhůru. Nakonec mě ukolébal déšť a já spal až do rána.

Budík zazvonil ve čtvrt na osm. Z rozespalosti mě probudila až studená sprcha společně s teplou kávou a dvěma croissanty. Začetl jsem se do ranních novin. Titulní stranu opět zdobil článek o nějakém vědeckém objevu. Oči byly zaměstnány čtením, ale mou mysl ovládaly vzpomínky na muže v bílých pláštích, na jejich experimenty a teorie … Z rozjímání mě probral pohled na hodiny. K čertu, prolétlo mi hlavou. Následovalo ještě mnoho slov. Nepěkných slov.

Do práce jsem dorazil s velkým zpožděním. Stálo mě to napomenutí nadřízeného a ostudu před všemi kolegy. První zameškání po tolika letech. Bohužel to nebyl jediný problém, který mě čekal. Má mysl se potulovala kdesi daleko. Soustředit se bylo extrémně těžké a práce mi nešla od ruky. V poledne mě můj nadřízený poslal domů, protože jsem za celé dopoledne nevyplnil jediný formulář správně.

Doma ale také nebylo k vydržení. Chodil jsem nervózně po svém bytě sem a tam. Únava mne pronásledovala a já se

jí snažil unikat. Nakonec mě ale přece jen přemohla. Krátký spánek během dne nemůže uškodit. Natáhl jsem se oblečený na postel a opatrně zavřel oči. Jen krátké zdřímnutí. Jen tak krátké, aby se mi nic nemohlo zdát ...

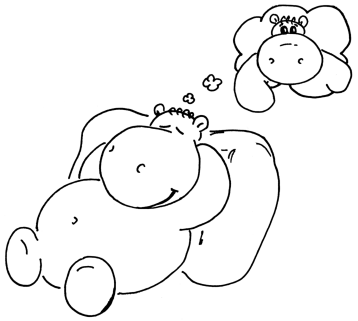
* * *

Pohled na město byl nádherný. Právě zapadalo slunce a jeho odraz se zrcadlil na lesklém povrchu budov. Vzduch se lehce tetelil a podtrhával tak atmosféru letního večera.

„Máme to! Máme to!“ ozvalo se odkudsi. V malé pracovně jedné bezvýznamné budovy na kraji města se rozlétly dveře. Do nich se vřítila postava mladého muže v brýlích a bílém plášti, který nad hlavou vítězoslavně mával štosem papírů.

„Máme co?“ podíval se na něj postarší muž, kterému očividně ona pracovna patřila.

„Přišli jsme na to, jak modifikovat memgramy! Tím můžeme posílat informace přímo do mozku našeho Spáče!“



21-3-1 Kódování memgramů — 9 bodů

Memgram, tedy záznam nesoucí jednu myšlenku, si můžeme představit jako mřížku obsahující 8×8 polí. Každé pole může nabývat dvou hodnot – 0 a 1. Vědci se snaží memgramy modifikovat, aby pomocí nich mohli přenášet své informace. Otázka je, kolik informace můžou zakódovat do jednoho memgramu.

Posílání informace probíhá tak, že vědci dostanou memgram, který nese (z jejich pohledu) náhodnou informaci – tj. nelze dělat žádné předpoklady o tom, kde jsou jedničky a kde nuly. Tomuto memgramu můžou – ba co víc, dokonce musí – změnit jeden bit (jednu jedničku překlopí na nulu nebo obráceně). Mřížka se nesmí otáčet, tzn. políčka jsou jasně a jednoznačně očíslována.

Příjemce (mozek Spáče) pak dostane upravený memgram. Přitom ale neví, jak vypadal původní memgram, tedy ani který bit byl změněn. Protože nás zajímá maximální velikost přenášené informace, předpokládejte, že mozek umí informaci rozkódovat, ať je jakkoli zakódovaná (zná komunikační protokol).

Pro lepší představu si uveďme triviální příklad, na kterém si ukážeme, jak přenést jeden bit. Řekněme, že hodnota tohoto bitu se bude přenášet na pozici $(1, 1)$ a všechna ostatní políčka budou pro příjemce nepodstatná. Když přijde memgram, podíváme se na naši pozici. Buď tam přímo dostaneme hodnotu, kterou chceme přenést, a pak změníme libovolný jiný bit (vždy musíme něco změnit, tak sáhneme jinam, abychom si políčko $(1, 1)$ nerozbili). Pokud tam je opačná hodnota, změníme políčko $(1, 1)$ a tím ho nastavíme správně. Příjemci pak stačí přečíst toto políčko, aby získal posílanou informaci.

Vašim úkolem je určit, kolik nejvíce bitů informace lze takto přenést v jednom memgramu, a popsat, jak bude tato informace kódována.

Pokud úlohu vyřešíte i obecně pro memgram obsahující libovolný počet (N) bitů, případně pokud dokážete, že vaše řešení je nejlepší možné, bonusové body vás jistě neminou.

„To vypadá zajímavě,“ pokýval hlavou starší muž, když pročítal papíry mladšího kolegy. „A za jak dlouho může být zařízení připravené?“

„Když budeme všichni pracovat jen na tomto projektu, mohli bychom to stihnout už za pár dní.“

„Hmm, to není špatné. Ale pomyslel jste, pane kolego, na možnost, že se mezitím Spáč probudí?“

* * *

Probudil jsem se a posadil na posteli. Bylo pozdní odpoledne a venku se začínalo stmívat. Tyhle sny začínají být čím dál realističtější. A také děsivější. Musím přijít na jiné myšlenky! Krátká procházka by mi mohla vyčistit hlavu. Oblékl jsem se a vyrazil.

V parku se pohybovali nejrůznější lidé. Bylo to ideální místo na odpočinek, přestože o sobě dával podzim vědět vtíravým chladem a všudypřítomným zažloutlým listím. Posadil jsem se na lavičku, pohodlně se opřel. Večerní vánek mi lehce foukal do tváře. Bylo to příjemně osvěžující.

Kolem prošel muž v bílém plášti. Snažil jsem se mu nevěnovat pozornost. Každý přece může nosit bílý plášť! V opačném směru prošla dvojice lidí. Také v bílých pláštích. Vášnivě diskutovali nad nějakými dokumenty. Nedaleko se zastavila žena ve středních letech. Byla zahalená do bílého pláště a venčila malého bílého psa. Vypadal roztomile, ale něco na něm nebylo v pořádku. Bližší pohled odhalil, že pes není bílý. Byl oblečený do bílého oblečku, který nápadně připomínal plášť. Kdo může obléknout psa do něčeho takového?!

Mám snad vidiny, nebo se ti lidé okolo zbláznili? Vydal jsem se směrem domů a snažil se příliš nerozhlížet po lidech. Přepracování, stres, to bude určitě ono! Nic jiného to nemůže být.

Teplá sprcha, několik prášků na spaní a rychle do postele. Prostě se z toho vyspím a bude to! Už žádné sny o bílých pláštích. Usilovně jsem se snažil myslet na jiné věci a prášky pomalu zabraly ...

* * *

Na rozlehlé, sytě zelené louce se pásly ovce. Uprostřed nich seděl baca a hrál na vlastnoručně vyřezanou píšťalku. Ovce klidně přežvykovaly trávu a sem tam se ozvalo zabečení. Baca odložil píšťalku a odkudsi vytáhl dřevěnou na mléko. Sedl si na stoličku vedle nejbližší ovce a začal ji dojit.

Něco nebylo v pořádku. Ovce znervózněly a začaly pobíhat sem a tam. Hustá bílá vlna na nich poskakovala a vlála. Jako by v ní byly jen oblečené a mohly ji kdykoli shodit. Tráva se podivně leskla a její zelená barva vybledla a změnila se na stříbrnou. Jednotlivá stébla se přestala kývat ve větru a stála vzpřímeně kolmo k nebi jako výškové budovy. Ovce si stouply na zadní. Stále pobíhaly kolem a vášnivě spolu diskutovaly bečivými hlasy. Už na sobě neměly vlnu – byly to bílé pláště! Baca zmizel neznámo kam a z louky se stala obrovská prosvětlená místnost ...

Jedna ovce – vědec se přitočil k jinému a do všeobecného hluku téměř zakřičel: „Už to můžeme spustit?!“

„Téměř! Právě provádíme kalibraci!“ odpověděl mu rovněž křikem druhý vědec.

povídat 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 stromě 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 stromě, 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.

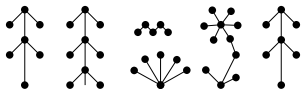
Než si obě metody blíže rozebereme, podívejme se, jak se změní implementace funkcí *root* a *union*:

```
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;

{zmena kvuli 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;

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

{zmena kvuli 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 $\mathcal{O}(\log N)$.

Grafům bez kružnic budeme obecně říkat *lesy,* jelikož každá komponenta souvislosti takového grafu je strom.



Les, jak ho vidí matematici

Někdy se hodí jeden z vrcholů stromu prohlásit za *kořen,* čímž jsme si v každém vrcholu určili směr nahoru (ke kořeni – je to zvláštní, ale informatici obvykle kreslí stromy kořenem vzhůru) a dolů (od kořene). Souseda vrcholu směrem nahoru pak nazýváme jeho *otcem,* sousedy směrem dolů jeho *syny.*

Ještě se nám bude hodit nahlédnout, že strom s n vrcholy má právě $n-1$ hran: Budeme postupovat matematickou indukcí podle počtu vrcholů stromu. Strom s jedním vrcholem neobsahuje žádnou hranu. Pokud máme strom s $n > 1$ vrcholy, vezměme libovolný jeho list a odeberme ho ze stromu. Tím získáme opět strom (souvislost jsme porušit nemohli a kružnici jsme také nevytvořili) a jeho počet vrcholů je o 1 menší. Podle indukčního předpokladu má o jednu hranu méně než vrcholů. Nyní list „přilepíme“ zpět, čímž zvýšíme počet vrcholů i hran o 1, a tvrzení stále platí.

A nyní k slibovaným kostrám. Mějme nějaký souvislý graf. Jeho *kostrou* nazveme libovolný podgraf, který obsahuje všechny vrcholy a nejmenší počet hran takový, aby každé dva vrcholy byly spojeny nějakou cestou. Všimněte si, že kostra musí být sama souvislá a navíc neobsahuje žádnou kružnici (jinak bychom mohli libovolnou hranu ležící na kružnici z kostry beze škody odebrat, čímž bychom získali menší kostru, a to nám definice zakazuje.) Čili každá kostra je strom. Na prvním obrázku je kostra levého grafu znázorněna silnými hranami.

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. Pokud si graf představíme jako města spojená silnicemi, problém nalezení minimální kostry můžeme vidět následovně: Chceme určit silnice, které se budou v zimě udržovat sjízdné tak, aby součet délek silnic, které je třeba udržovat, byl co nejmenší možný a zároveň se stále bylo možné přepravit mezi každými dvěma městy.

Algoritmus pro hledání minimální kostry

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 třídících 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 $\mathcal{O}(M \log M)$. 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 hladového algoritmu

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 třídícím 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.

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 jeda 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.

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 od-

Uprostřed místnosti stál přístroj, který vypadal jako jaderný reaktor zkřížený s obřím kávovarem. Okolo pobíhala hromada vědců v bílých pláštích, kteří usilovně pracovali na různých částech přístroje. Trvalo notnou chvíli, než dokončili všechny drobnosti.

„Jsme připraveni!“ zamával jeden vědec z vrcholu přístroje na kolegu u ovládacího pultu. Ten mu zamával nazpět a pomalu zatáhl za velkou páku. Navzdory všem předpokladům přístroj nezačal ani hučet, ani blikat, jen malá kontrolka signalizovala, že byl uveden do provozu. Vědci z celé místnosti se shromáždili u informační obrazovky, kde s napětím čekali na první výsledky. Číselné ukazatele se pohnuly a začaly stoupat. Ozvalo se hromadné oddechnutí a celý dav začal optimisticky švitořit.

„Výborně! A teď uložíme náš svět do dlouhodobé paměti Spáče,“ zavelel nejstarší vědec. Ostatní se rozprchli po nejbližším okolí a začali opět pilně pracovat.

21-3-2 Nadposloupnost — 10 bodů

Vědci se snaží uložit informace o jejich světě do paměti Spáče. Problém je, že v paměti už některé věci jsou a žádné nesmí zmizet – to by mohlo mít nedozírné následky.

Paměť si představte jako uspořádanou posloupnost vzpomínek. Jednu vzpomínku budeme pro jednoduchost brát jako řetězec. Zároveň je dána posloupnost vzpomínek, které by vědci rádi do paměti uložili. Některé vzpomínky se můžou překrývat s těmi, které už v paměti jsou.

Cílem je najít *nadposloupnost* takovou, aby původní paměť i nové vzpomínky představovaly podposloupnosti této *nadposloupnosti*. Vzhledem k tomu, že paměť není svou kapacitou neomezená, měla by být nadposloupnost nejkratší možná, aby se minimalizovalo riziko zapomínání.

Příklad: V paměti je „`snídaně`“, „`práce`“ a „`večeře`“. Vědci chtějí přidat „`večeře`“, „`sen`“ a „`snídaně`“. Jeden z možných výsledků je „`snídaně`“, „`práce`“, „`večeře`“, „`sen`“ a „`snídaně`“. Vyškrtnutím snu a druhé snídaně dostaneme původní paměť a vyškrtnutím první snídaně a práce dostaneme posloupnost, kterou by rádi vědci do paměti dostali.

Probudilo mě drnčení budíku. Musel jsem spát opravdu tvrdě, protože zvonil už několik minut v kuse. Hlava byla čistá a celým tělem mi pulzovala energie. Byl to nádherný pocit. Stačilo pár minut, abych se osprchoval, nasnídal a vyrazil do kanceláře.

Práce mi šla od ruky. Formuláře, které se mi nahromadily za včerejšek, zmizely ještě před dopolední poradou a chvíli po poledni jsem měl splněné všechny povinnosti na dnešní den.

„Dobrá práce, Harolde!“ pochválil mě nadřízený přede všemi kolegy. „Vidím, že jsi překonal tu včerejší krizi.“

„Potřeboval jsem se z toho jen pořádně vyspat,“ odpověděl jsem ledabyle a přibral si další formuláře navíc od svých kolegů.

Práce mě úplně pohltila. Čas ubíhal a kancelář se postupně vyprazdňovala. Vyrušil mě až vrátný, když kontroloval, zda v budově nikdo nezbyl. To byl ale produktivní den! Musel jsem udělat práci nejméně za pět lidí! Cesta domů byla klidnější než obvykle. Podzemka byla poloprázdná. Aby také ne, v tuhle dobu.

Čekal mne pravidelný večerní rituál. Nakrmit rybičky, večeře a večerní zprávy. Můj přesčas v práci ale způsobil, že zprávy na kanálu 6 už dávno skončily. Na obrazovce pobíhala nějaká zvířata, zatímco hlas na pozadí vyprávěl odborné pikantnosti z jejich života. Zajímavé, ale ne zas tolik. Vypnul jsem televizi a šel si číst do postele.

* * *

Pohled na město byl nádherný. Vycházelo slunce a společně s ním se probouzeli lidé. Proudy vznášedel houstly a město pomalu ožívalo. Do jedné už ne tak bezvýznamné budovy na kraji města se sbíhali vědci. Přednáška na téma Snových světů se bude konat od devíti hodin ve Velké aule, informovaly všudypřítomné plakáty.

Aula se plnila a s přibývajícím množstvím lidí se zvyšoval i hluk. Šepot se brzy změnil v křik a v místnosti nebylo slyšet vlastního slova. K řečnickému pultu vystoupil starší muž s plnovousem. Chvíli počkal, než hluk v sále utichl na přijatelnou úroveň, a začal přednášet.

Mluvil dlouze o nejrůznějších věcech. Jak je možné žít ve snu a jaké filozofické problémy jsou s tím spojené. Kolik takových snových světů může existovat, jak jsou propojené a zda se dá mezi nimi cestovat. Zda mohou existovat snové světy vytvořené snem osoby, která je rovněž uvězněna ve snu. Jak mohou snové světy ovlivňovat reálné světy a naopak. A na závěr upozornil posluchače na vážný problém.

„Jak jistě všichni víte, jsme uvězněni v mysli Spáče. Spuštěním přístroje na úpravu memgramů se nám podařilo zafixovat naši existenci přímo v jeho paměti a podvědomí, takže nehrozí, že by na nás v blízké době zapomněl. Ale stále tu existuje jeden problém . . . “ Přednášející se na chvíli odmlčel, prohrábl si plnovous a promítl další holografický obrázek. „Co když našeho spáče potká řekněme malá mozková příhoda? Co když ho zítra přejede cestou do práce auto? A i kdybychom měli štěstí a nic z toho se nestalo, spáč je jen obyčejný člověk. Za nějakých padesát, možná šedesát let stejně zemře přirozenou smrtí a naše civilizace zanikne s ním.“

Poslední věta visela ve vzduchu a v aule se rozhostilo úplné ticho. Po chvíli se mezi posluchači zvedla ruka těsně následovaná i jejím majitelem. Stoupl si, rozhlédl se po okolních posluchačích a pak se rozechvělým hlasem zeptal: „A myslíte, že s tím půjde něco udělat, pane profesore?“

„Upřímně řečeno, nevím. Stále nad tím bádáme. Pravděpodobně je naší jedinou možností vytvořit tunel mezi reálným a snovým světem. Problém je, že by si to vyžádalo obrovské množství energie a také úplnou znalost topologie všech snových světů Spáče.“

Aula začala tiše šumět vzrušenými debatami. O chvíli později se z davu zvedla jiná ruka. „Myslím, pane profesore, že bych pro vás mohl vyřešit to druhé . . . “

21-3-3 Topologie snů — 10 bodů

Topologie snových světů je reprezentována binárním stromem. Katedra snového inženýrství se pokusila zmapovat okolní světy, ale zatím mají k dispozici jen neúplná data.

Podařilo se jim získat tento strom vypsaný v prefixové a infixové notaci. Problém je, že pro další zpracování by velice potřebovali mít strom vypsaný také v postfixové notaci.

Napište algoritmus, který z prefixového a infixového výpisu sestaví výpis postfixový, případně oznámí, že ze zadaných dat nelze sestavit tento výpis jednoznačně.

Jednotlivé vrcholy jsou ve stromě označeny celými náhodnými čísly. Označení je navíc jednoznačné – tj. žádné dva vrcholy nemají stejné číslo.

Příklad: Pro strom s prefixovým výpisem 16 11 19 3 42 a infixovým 11 16 3 19 42 bude postfixový výpis 11 3 42 19 16.

Následujících několik dní bylo velmi zajímavých. Energie mne neopouštěla, a tak jsem strávil v práci i celý víkend. Můj plán byl našetřit nějaké přesčasy a pak si udělat dovolenou. Ale nešlo to. Mozek byl příliš aktivní a neustále potřeboval něco řešit. Každou noc se mi zdály tytéž sny o vědcích usilovně pracujících na zařízení, které by je mělo dostat ven ze snu. Začínalo mě to děsit. Opět jsem navštívil svého psychologa a převyprávěl mu své sny i to, co se se mnou v poslední době děje ...

„Takže říkáte, že oni – tedy jako v tom vašem snu – něco staví?“ zeptal se už asi po čtvrté.

„Ano. Jak jsem říkal, staví nějaké zařízení, které by je mělo dostat ven ze snu.“

„Hm. Velmi zajímavé,“ pokýval doktor hlavou a zamyslel se. „Myslím, že byste je měl nechat, aby to dostavěli.“

„A nemohlo by to být – já nevím – nebezpečné?“

„Snad nemyslíte, že by to mohlo být skutečné,“ usmál se. „Ten stroj symbolizuje nějakou věc ve vašem podvědomí, která čeká, až ji vyřešíte. Pomozte jim. Mějte dobrou vůli ten stroj dostavět. Jedině tak vaše podvědomí vyřeší problém, který zřejmě máte.“

Úžasné! Psychologové mají prostě na všechno vysvětlení. Cestou do práce se mi hlavou honily nejrůznější myšlenky. Na jednu stranu mohl mít pravdu. Na druhou stranu, co když je ten sen skutečný. Nebo je také možné, že mi šplouchá na maják! V duchu jsem se zasmál té hloupé myšlence, ale veselo mi nebylo.

Večer jsem dal na radu psychologa. Od teď bude mou jedinou myšlenkou před spaním dostavět ten zatracený přístroj. Ať to stojí co to stojí.

* * *

Pohled na město byl nádherný, ale zdaleka už ne tak úchvatný. Byl to stále ten stejný pohled, který se vracel noc co noc. Budova na okraji města, ve které vědci připravovali svůj přístroj, byla stále plná a žila čilou kreativní prací. Právě řešili další problém ...

„Už se nám podařilo rozmístit jednotlivé sny na různé frekvence, ale stále nemáme potřebný výkon, abychom je dokázali všechny pokrýt,“ říkal právě jeden vědec druhému.

„A co kdybychom optimalizovali hierarchii snů?“

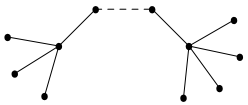
„To by mohlo jít, ale výsledná struktura by musela mít co nejmenší průměr!“ přikývl vědec a začal počítat potřebné úpravy.

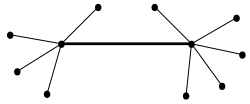
21-3-4 Optimalizace stromu — 11 bodů

Hierarchie, do které vědci přeuspořádali jednotlivé sny, je vlastně strom. Ale už nemusí být nutně binární a také není zakořeněný. Manipulace se sny je velmi složitá, takže můžete jen jednu hranu odebrat a jednu hranu přidat. Výsledek musí být opět souvislý strom a navíc musí mít nejmenší možný průměr.

Pro upřesnění: Termínem *průměr* zde myslíme počet hran na cestě mezi dvěma nejvzdálenějšími vrcholy grafu (v našem případě stromu).

Příklad: Na následujících obrázcích je uveden strom, který máme modifikovat. Na prvním obrázku je zvýrazněná hrana, kterou odebíráme, na druhém hrana, kterou jsme přidali. Původní strom má průměr 5, po úpravě se dostaneme na hodnotu 3.





„To by mělo stačit,“ přikývl postarší vědec, když prošel všechna čísla. „Zkontrolujte znovu všechny systémy! Zbývá necelá hodina do spuštění!“

Zařízení mělo tvar obrovského oválu. Po obvodu byly přidělány mohutné cívky, ke kterým se táhly tlusté kabely. Pobíhající vědci kontrolovali poslední detaily před prvním spuštěním. Čas pomalu ubíhal a jednotlivé týmy postupně potvrzovaly funkčnost jednotlivých systémů. Přípravy byly dokončeny a čekalo se pouze na pokyn z nejvyšších míst. Pan profesor, který celý vývoj řídil, se postavil před shromážděné vědce.

„Experiment, který nyní provedeme, je velice nebezpečný. Pokud jsme někde udělali chybu, ten, kdo projde bránou na druhou stranu, může skončit kdekoli. Třeba v nějaké noční můře, nebo ještě hůř ... “ Shromáždění vědci se podívali jeden na druhého.

„Nemohu po nikom z vás chtít, aby takto riskoval svůj život,“ navázal profesor. „Proto jsem se rozhodl, že bránou projdu sám.“ Z hloučku přítomných lidí se ozvalo překvapené zalapání po dechu.

„Spusťte to!“ zavelel profesor a otočil se k zařízení. Ozvalo se hlasité cvaknutí spínaných kontaktů a hluboké bručení transformátorů. Vzduch uvnitř oválu se začal vlnit a tmavnout. Ve vzduchu byl cítit silný elektrostatický náboj. Obraz uvnitř portálu se ustálil a skupinka vědců zírala do tmavé místnosti. Uprostřed ní byla postel, ve které kdosi spal.

Profesor pomalu vykročil k oválu. Naposledy se otočil a kývl svým kolegům na pozdrav. Zhluboka se nadechl a jedním krokem prošel skrz. Obraz ložnice se zavlnil. Pak začal rychle blednout, až zmizel docela ...

Probudil jsem se a posadil se. U postele stála postava. Oči pomalu přivykly šeru a rozeznaly dlouhý plášť a plnovous ...

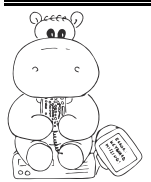
* * *

„Vzpomínáte na ty vědce z mého snu? Tak už dostavěli to zařízení.“

„Opravdu? A mělo to na vás nějaký účinek?“ zeptal se doktor.

„Myslím, že ano,“ přikývl jsem. „Každopádně se mnou dnes přišel někdo, kdo by se s vámi rád seznámil ... “

21-3-5 Praktická: Rozklad na součty — 10 bodů



V této sérii se nám praktická úloha nevešla do příběhu. Ale nebojte se, o to lépe se vám bude řešit ...

Tuto úlohu odevzdávejte výhradně prostřednictvím webové aplikace CodEx (*https://codex2.ms.mff.cuni.cz/ksp/*). Pokud jste s řešením začali teprve v této sérii a nevíte, co je CodEx, podívejte se třeba na úlohu 21-1-2 „Optimalizace kotlů“, ve které naleznete úvodní povídání o CodExu.

Zadání:

Na standardním vstupu je zadáno číslo N ($1 \leq N \leq 40$). Vypište na standardní výstup všechny možnosti, jak toto číslo rozložit na součet celých kladných čísel. Každý rozklad musí být uveden na samostatném řádku, sčítance vyjmenovány od nejmenšího k největšímu a odděleny znaménkem „+“. Na pořadí řádků nezáleží.

Například pro $N = 5$ je jeden ze správných výstupů následující:

```
1+1+1+1+1
1+1+1+2
1+1+3
1+2+2
1+4
2+3
5
```


21-3-6 Pan Cowmess — 12 bodů

Tuto úlohu musíte řešit v programovacím jazyce Rapl, jehož popis najdete v zadání úlohy 21-1-6 z první série.

Pan Cowmess [čtěte Koumes] si založil softwarovou firmu Zkrátil a Zrychlil, s.r.o., která se hodlá zabývat programováním nejkratších a nejrychlejších programů na světě. Pro svého prvního zákazníka napsal následující program:

```
        n = 0
        y = 1
cyklus: A[n] = x % 2
        n = n + 1
        x = x / 2
        if n < 32 => jump cyklus
        i = 0-1
hledej: i = i+1
        if i >= n => jump zkus
        if A[i] <> 0 => jump hledej
        j = 0-1
druhe:  j = j+1
        if j >= n => jump hledej
        if A[j] <> 1 => jump druhe
        A[i] = 2
        A[j] = 2
        jump hledej
zkus:   if A[k] = 2 => jump ok
        y = 0
ok:     k = k+1
        if k < n => jump zkus
```

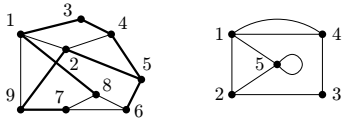
Vám se (zcela oprávněně) nezdá, že by tento program byl obzvlášť krátký nebo rychlý. Zkuste tedy přijít na to, co Cowmessův program dělá, a zdůvodnit, proč to dělá *(za 6 bodů)*. Pak napište daleko kratší program, který počítá totéž *(za dalších 6 bodů)*. Tím myslíme, že váš program má pro libovolnou počáteční hodnotu registru `x` odpovědět stejnou hodnotou `y` jako původní program. Vaše řešení může být i pomalejší, hlavní je, aby mělo co nejméně instrukcí.

Recepty z programátorské kuchařky

V dnešním dílu kuchařky si zavedeme základní pojmy z teorie grafů a ukážeme si, jak řešit problém nalezení minimální kostry grafu. Také si popíšeme datovou strukturu *Disjoint-Find-Union* (její název je často zkracován na DFU), kterou šikovně použijeme právě na řešení tohoto problému.

Grafy

Neorientovaný graf je určen množinou vrcholů V a množinou hran E, přičemž hrany jsou neuspořádané dvojice vrcholů. Hrana $e = \{x, y\}$ spojuje vrcholy x a y. Většinou požadujeme, aby hrany nespojovaly vrchol se sebou samým (takovým hranám říkáme *smyčky*) a aby mezi dvěma vrcholy nevedla více než jedna hrana (pokud toto neplatí, mluvíme o *multigrafech*). Neorientovaný graf většinou zobrazujeme jako body pospojované čarami.



Neorientovaný graf a jeho kostra; multigraf

Podgrafem grafu G rozumíme graf G', který vznikl z grafu G vynecháním některých (a nebo žádných) hran a vrcholů.

Často nás zajímá, zda se dá z vrcholu x dojít po hranách do vrcholu y. Ovšem slovo „dojít“ by mohlo být trochu zavádějící, proto si zavedeme pár pojmů:

- *sled* budeme říkat takové posloupnosti vrcholů a hran tvaru $v_1, e_1, v_2, e_2, \ldots, e_{n-1}, v_n$, že $e_i = \{v_i, v_{i+1}\}$ pro každé i. Sled je tedy nějaká procházka po grafu. Délku sledu měříme počtem hran v této posloupnosti.
- *tah* je sled, ve kterém se neopakují hrany, tedy $e_i \neq e_j$ pro $i \neq j$.
- *cesta* je sled, ve kterém se neopakují vrcholy, čili $v_i \neq v_j$ pro $i \neq j$. Všimněte si, že se nemohou opakovat ani hrany.

Lehce nahlédneme, že pokud existuje sled z vrcholu x do y ($v_1 = x$, $v_n = y$), pak také existuje cesta z vrcholu x do vrcholu y: Každý sled, který není cestou, obsahuje nějaký vrchol u dvakrát, nechť $u = v_i = v_j$, $i < j$. Z takového sledu ale můžeme vypustit posloupnost $e_i, v_{i+1}, \ldots, e_{j-1}, v_j$ a dostat také sled spojující v_1 a v_n, který je určitě kratší než původní sled. Tak můžeme po konečném počtu úprav dospět až ke sledu, který neobsahuje žádný vrchol dvakrát, tedy k cestě.

Kružnicí nazýváme cestu délky alespoň 3, ve které oproti definici platí $v_1 = v_n$. Někdy se na cesty, tahy a kružnice v grafu také díváme jako na podgrafy, které získáme tak, že z grafu vypustíme všechny ostatní vrcholy a hrany.

Ještě si ukážeme, že pokud existuje cesta z vrcholu a do vrcholu b a z vrcholu b do vrcholu c, pak také existuje cesta z vrcholu a do vrcholu c. To vyplývá z faktu, že existuje sled z vrcholu a do vrcholu c, který můžeme dostat například tak, že spojíme za sebe cesty z a do b a z b do c. A jak jsme si ukázali, když existuje sled z a do c, existuje i cesta z a do c.

V mnoha grafech (například v těch na předchozím obrázku) je každý vrchol dosažitelný cestou z každého. Takovým grafům budeme říkat *souvislé*. Pokud je graf nesouvislý, můžeme ho rozložit na části, které již souvislé jsou a mezi kterými nevedou žádné další hrany. Takové podgrafy nazýváme *komponentami souvislosti*.

Teď se podívejme na pár grafů z přírody: *Strom* je souvislý graf, který neobsahuje kružnici. *List* je vrchol, ze kterého vede pouze jedna hrana. Ukážeme, že každý strom s alespoň dvěma vrcholy má nejméně dva listy. Proč to? Stačí si najít nejdelší cestu (pokud je takových cest více, zvolíme libovolnou z nich). Oba koncové vrcholy této cesty musejí být nutně listy: kdyby z některého z nich vedla hrana, musela by vést do vrcholu, který na cestě ještě neleží (jinak by ve stromu byla kružnice), ale o takovou hranu bychom cestu mohli prodloužit, takže by původní cesta nebyla nejdelší.