Milí řešitelé a řešitelky!

Úložky, úložky přicházejí, řešte je přátelé! Do domu úložky, úložky přicházejí, mastné a pečené! Je tu třetí série 25. ročníku a v ní tradiční nadílka úloh, pokračování seriálu o TEXu a zbrusu nová kuchařka o teorii čísel.

Za úspěšné řešení KSP je možno být přijat na MFF UK bez příjimacích zkoušek. Úspěšným řešitelem se stává ten, kdo získá za celý ročník alespoň 50 % bodů, přičemž půjde získat maximálně 300 bodů. Připomínáme, že z každé série se do celkového bodového hodnocení započítává 5 nejlépe vyřešených úloh.

Každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok a tužku. Navíc každému, kdo v této sérii vyřeší alespoň **tři libovolné úlohy na plný počet bodů, pošleme čokoládu**.

Termín odevzdání třetí série je stanoven na **pondělí 4. února v 8:00 SEČ**.

Řešení přijímáme elektronicky na stránce *https://ksp.mff.cuni.cz/submit/*. Chcete-li s námi komunikovat bezpečně, můžete si ověřit náš HTTPS certifikát – zde je jeho SHA1 hash: `7F:53:E7:00:60:F2:24:93:8F:52:51:EC:1E:A8:34:54:86:69:32:7D`.

Také nám řešení můžete poslat klasickou poštou. V tom případě byste jej měli podat do středy 30. ledna s naší adresou

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

Před tím ale vyplňte přihlášku (a to i tehdy, když jste se KSPčka účastnili loni) na *http://ksp.mff.cuni.cz/,* kde najdete i další informace o tom, jak KSP funguje. Na webu máme také fórum, kde se můžete na cokoli zeptat. Nebo nám můžete napsat na e-mail *ksp@mff.cuni.cz.*

Třetí série dvacátého pátého ročníku KSP

10. 12. 2042

Odkládat věci je snadné, proklatě snadné. Někdy kolem třicátých narozenin jsem si řekl, že jednou sepíšu některé ze svých zážitků a zanechám v nich otisk svých pocitů z tohoto světa. Každý rok jsem si říkal, že ještě není ta pravá chvíle, že to přece má svůj čas. A najednou... ani nevím jak, jsem starcem a tuším, že čas se krátí.

Když se tak zpětně ohlížím, asi jsem nikdy příliš nepřivykl tempu života. V mládí jsem usiloval o spoustu věcí, ale vždy se mi nakonec podařilo nechat si je proplout mezi prsty. Jednou jsem na přechodnou dobu přijal místo u policie. Z přechodné doby se stala záležitost na celý život. Asi jsem objevil klid, který jsem hledal. Práci jsem trávil pochůzkami, většinou jsem lidem pomáhal s různými výtržníky a chuligány, dělal jsem to velmi rád a po práci měl konečně klid na všechny ty věci, které mi dříve unikaly...

Chci vyprávět o mnohém, ale začnu historkou, která mi do dnešního dne občas nedá spát.

⋆⋆⋆

I když se odehrál před desítkami let, pamatuji si ten den velice dobře. Dopoledne zajímavé nebylo, začalo velkou poradou, před kterou si náš velitel, poručík Hamáček, neodpustil monolog o stavu disciplíny v policejním sboru, který korunoval okázalou kontrolou toho, zda se všichni dostavili.

25-3-1 Kontrola docházky — 12 bodů

Pro řádnou kontrolu docházky je nutno své podřízené přepočítat. Policejní poručík Hamáček na to má svůj systém osvědčený léty služby – ve svém notesu má N dvojic (M_i, K_i). Všechna M_i jsou po dvou nesoudělná a $0 \leq K_i < M_i$. Celkový počet policistů je menší než součin všech M_i.

Samotná kontrola probíhá v N krocích, v i-tém kroku se příslušníci srovnají do řad po M_i osobách a poručík Hamáček následně zkontroluje, zda odpovídá počet policistů, kteří už nemohli vytvořit celou řadu, hodnotě K_i.

Vrchní referent, strážmistr Borůvka, se stěhuje. Pomozte mu určit k takové, že vygumováním dvojice (M_k, K_k) z poručíkova notesu umožní co největšímu počtu kolegů mu místo porady pomoci se stěhováním.

Poručík nesmí nic poznat, tedy počty nezařazených policistů pro zbývajících $N - 1$ dvojic musí stále odpovídat.

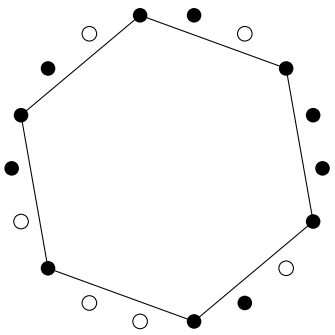
Příklad: Mějme 7157 policistů, kteří se postupně řadí do řad po 12, 13 a 49 osobách. Dvojice (M_i, K_i) tedy jsou $(12, 5)$, $(13, 7)$ a $(49, 3)$. Optimálním řešením je vygumovat dvojici $(13, 7)$. Na stanici pak musí zůstat 101 policistů.

Po úmorném přepočítávání, připomínajícím vojenské cvičení, jsme konečně mohli usednout k poradě.

25-3-2 Zasedání u kulatého stolu — 10 bodů

Na policejní schůzi se sešlo N příslušníků policejního sboru sedících u kulatého stolu, vzdálenost mezi každými dvěma sousedními policisty je shodná. Schůze je dlouhá, policisté v jejím průběhu všelijak odcházejí a přicházejí. Přítomnost policistů v pravé poledne je zadána jakožto posloupnost N nul a jedniček, kde i-tá jednička znamená, že policista na i-tém místě je na schůzi právě přítomen. Vaším úkolem je zjistit, zda existuje $K \geq 3$ takové, že lze vytvořit pravidelný K-úhelník, jehož vrcholy tvoří přítomní policisté.

Příklad: Pro 12 policistů a posloupnost 111010111011 je odpověď kladná, lze sestrojit trojúhelník nebo šestiúhelník. Pro 5 policistů a posloupnost 10111 pravidelný K-úhelník nesestrojíme. Další příklad je na obrázku (plné tečky představují přítomné policisty):



Po schůzi jsem se z naší tehdejší služebny ve Vlašské ulici vydal na pochůzku. Propletl jsem se spoustou aut před Schönbornským palácem, asi se na americké ambasádě konala důležitá recepce, a pak zahnul do spleti úzkých uliček.

V jedné velmi úzké uličce stála dost svérázně zaparkovaná černá dodávka s japonským osazenstvem. Po zdlouhavé komunikaci, která spíš než pomocí několika málo anglických slůvek probíhala především gestikulací, se mi podařilo domluvit, ať mi zavolají někoho, kdo se alespoň trochu dorozumí anglicky (možná německy, jistý jsem si naší domluvou příliš nebyl). Zatímco jsem čekal, začalo mě velmi silně zajímat, co se nachází v těch černých pytlích v dodávce.

V tom okamžiku se ale z mé vysílačky ozval rozkaz k okamžitému přesunu k japonské ambasádě. Má námitka, že zrovna něco zajímavého mám, byla smetena. Prý je nutné se okamžitě postarat o bezpečnost japonského konzula. Propletl jsem se tedy několika úzkými uličkami a za zvuku sekaček z nedalekého parku uháněl k ambasádě.

25-3-3 Do třetice sekání — 13 bodů

V této sérii pro změnu trávník vypadá jako jeden řádek čtvercové sítě a je již posekán, nyní je potřeba posvážet posekanou trávu. Vaším úkolem je zanalyzovat, kolik by různé možnosti svozu stály námahy.

Máte zadáno N přirozených čísel udávajících hmotnost trávy na jednotlivých polích a D intervalů $[a..b]$. Takový interval znamená, že se bude svážet z políček a až b. Námaha pro převoz L trávy z políčka k na políčko l je dána jako $L \cdot |k - l|$. Pro každý interval určete políčko, na které se dá všechna tráva posvážet s nejmenší celkovou námahou. Celková námaha je součet veškeré námahy, jež byla potřeba pro svezení trávy z celého intervalu na toto políčko.

Pokud existuje více správných políček, vypište libovolné z nich. Výpočty pro jednotlivé intervaly jsou nezávislé, tedy množství trávy na jednotlivých políčcích se mezi intervaly nemění. Složitost algoritmu by měla být optimalizována pro případy, kdy je D řádově stejně velké jako N.

Příklad: (První řádek obsahuje N a D, následují hodnoty L a pak intervaly.)

```
10 3
10 3 1 3 9 8 5 4 12 9

1 4
3 6
5 9
```

Odpovědi pro jednotlivé intervaly jsou `1 5 7`.

Před ambasádou postávalo několik lidí. Nejvíce pozornosti zde poutala žena hovořící s jedním Japoncem, pravděpodobně oním konzulem, v jeho rodné řeči. Jemu to zřejmě nebylo příliš příjemné.

Oslovil jsem je, abych zjistil, co se děje. Na to spustila přítomná žena dlouhý monolog o tom, že je novinářkou, že se zabývá důležitou mezinárodní zločinnou kauzou, že je ve veřejném zájmu, aby položila několik otázek konzulovi, že jí japonská ambasáda odpírá právo na informace a že v této zemi obecně není dostatečně ctěna svoboda tisku. Zatímco mi to vše tak emotivně sdělovala, jí ale pan konzul utekl.

V okamžiku, kdy si toho všimla, trochu znejistěla, řekla něco o tom, že mi vlastně vůbec nic není do toho, co dělá. Ona si prý jen tak postává před ambasádou a zrovna potřebuje něco spočítat na jakési speciální kalkulačce. A skutečně vytáhla z kabelky kalkulačku.

Ohlásil jsem stanici, že konzul je mimo nebezpečí. Trochu vzrušený hlas poručíka Hamáčka mi sdělil, že se mám okamžitě přesunout na místo na druhém konci Malé Strany. Prý naše jednotka bude provádět zásah.

Podařilo se mi tam dorazit v okamžiku, kdy probíhaly poslední přípravy. Zatímco chlapi z jednotky kontrolovali zbraně, vyprávěl poručík Hamáček svým skoro otcovským hlasem o tom, jak mu jeho pečlivá příprava jednou zachránila život při střetu se členy jednoho nebezpečného mezinárodního gangu.

25-3-4 Zločinná záležitost — 10 bodů

Ve špinavých vodách mezinárodního zločinu je obzvláště důležitá organizace, typickým příkladem zločinu je výroba něčeho ilegálního. Výroba probíhá ve fázích a obvykle na více než jednom místě. Převoz z jednoho místa na druhé je nejkritičtější částí výrobního procesu, proto je potřeba počet převozů mezi výrobními místy minimalizovat.

V této úloze budete mít na vstupu popsán výrobní proces ve dvou továrnách. První řádek obsahuje čísla N a M, kde N udává počet výrobních fází a M počet závislostí mezi nimi. Druhý řádek vstupu obsahuje N hodnot, kde i-tá hodnota je 1, pokud má i-tá fáze probíhat v první ilegální továrně, a 2 v případě, kdy má probíhat ve druhé továrně. Následuje M řádků obsahující dvojice a, b, které značí, že fáze b může proběhnout až tehdy, když byla provedena fáze a.

Určete pořadí fází výrobního procesu tak, aby každá fáze proběhla až poté, co jsou provedeny všechny fáze, na kterých je závislá, a aby počet převozů mezi výrobními místy byl minimální. Převoz je nutný vždy, když po fázi probíhající v továrně 1 bezprostředně následuje fáze v továrně 2, nebo naopak. (Můžete předpokládat, že řešení existuje.)

⇧ **Lehčí varianta (za 6 bodů):** Nabízíme k řešení i jednodušší variantu úlohy, kde všechny fáze probíhají v jediné továrně.

Příklad:

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

Potřebujeme alespoň 4 převozy. Výroba může proběhnout takto – začneme v první továrně, provedeme v ní fáze 3 a 4, pak se přesuneme do druhé továrny a provedeme fáze 2 a 1. Následují fáze 5 v první továrně a 6 v druhé továrně, končíme po čtvrtém převozu v první továrně fází 7.

Už se nepamatuji, jaké zločince tam naše jednotka očekávala. Každopádně, po pompézním vyražení dveří a sražení k zemi všech přítomných osob zjistili ozbrojení kolegové, že se jim podařilo zneškodnit všehovšudy tři zaměstnankyně jakéhosi asijského bufetu. Vydal jsem se tedy zase zpátky k dodávce, třeba tam ještě něco zajímavého bude.

Nešel jsem ani pět minut a opět mě volali vysílačkou. K dodávce se asi jen tak nedostanu. Měl jsem přivést japonského chlapce v černém tričku a modrých riflích s velkým fotoaparátem. Spatřen prý byl na nedalekém náměstí.

Na náměstí skutečně ještě byl. Působil nesmírně zmateně. V okamžiku, kdy mě zahlédl, ke mně natáhl ruku s peněženkou. Netušil jsem, co tím zamýšlí, zda to jsou jeho doklady, či nějaká lest k tomu, aby mohl následně utéci. Každopádně jsem k němu přistoupil, pokusil se na něj usmát, něco mu říct a raději jej chytil jemně za rameno a pro jistotu chytil i onen důležitý foťák, aby jej v té nervozitě ještě nerozbil.

V tom se zezadu přiřítila opět ona milá novinářka a vzala z jeho ruky peněženku. Povídala, že to je její peněženka a že si to klidně mohu zkontrolovat. Učinil jsem tak a dal jí podržet chlapcův fotoaparát. Dřív, než jsem stihl zareagovat, z něj vyndala paměťovou kartu. V duchu jsem si zanadával, věděl jsem, že tyhle novinářky jsou dost mazané a šikovné na to, abych tu kartu už nikdy neviděl a radši se tvářil, že jsem si toho nevšimnul. (Krátce před tím jsem udělal ještě jeden průšvih, a kdyby se k tomu přidalo, že jsem si nechal před nosem vzít paměťovou kartu, opravdu by mi to neprospělo.)

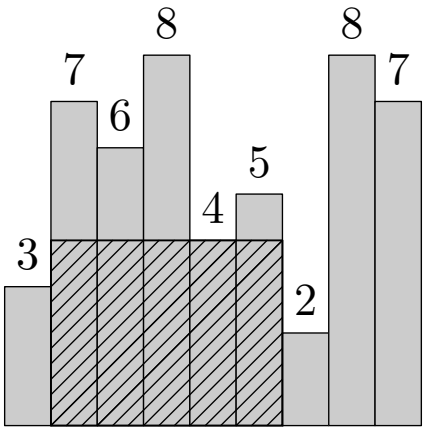
Pak jsem chlapce odvedl k nám na stanici. Na chodbě zrovna postával jeden z vyšších velitelů pražské policie, u nás na stanici jsem jej viděl asi podruhé. Vzal si ode mě chlapcův fotoaparát a řekl mi, ať se postarám o chlapce a najdu jeho rodiče.

Vzal jsem jej k nám do kanceláře, zdál se být hodně zaujat prací naší sekretářky. Zrovna přerovnávala přílohy ke spisům, především grafy.

25-3-5 Histogram — 9 bodů

Jedním z četně používaných typů grafů je histogram. Histogram je, jak praví Wikipedie, grafické znázornění distribuce dat pomocí sloupcového grafu se sloupci stejné šířky. Máte zadán histogram jakožto posloupnost N přirozených čísel udávajících výšky sloupců, šířka sloupců je jednotková. Určete obsah největšího obdélníka rovnoběžného s osami, který lze do grafu umístit tak, že celá jeho plocha leží na sloupcích histogramu.

Příklad: Pro 7 sloupců a výšky 3 6 7 4 2 3 1 je výsledný obsah 12. Existují hned 4 různé obdélníky s tímto optimálním obsahem. Další příklad s jednoznačným řešením je na obrázku:



Chlapec mi naštěstí dal kartičku pro podobné případy. Mimo jiné se na ní nacházelo číslo na japonskou ambasádu. Během telefonátu s ambasádou přišel velitel, jemuž jsem předával foťák. Byl dost naštván, že ve fotoaparátu nebyla paměťová karta a jestli o tom něco nevím, samozřejmě jsem odpověděl, že nikoliv. Sekretářka ambasády se při našem telefonátu musela dobře bavit.

Domluvili jsme se, že jim chlapce přivedu. Předal jsem jej vrátnému, ten člověk to asi s dětmi uměl lépe. Už když se pozdravili, objevil se na chlapcově tváři úsměv. Na jeho stole jsem dokonce zahlédl nějakou knihu s kresbou draka a princezny. Chlapec byl určitě v dobrých rukou.

25-3-6 Rytíř a princezny — 10 bodů

Rytíř v dalekém království se vydal na hrdinnou výpravu. Trasa jeho výpravy vypadá jako N polí uspořádaných do řádku. Na každém poli se nachází buď drak, který má u sebe H zlatých mincí, nebo princezna, která má koeficient krásy K. Rytíř se pohybuje při své výpravě z prvního políčka na poslední a je dostatečně silný na to, aby zabil kteréhokoliv draka po cestě. Je jeho volbou, zda draka zabije a získá mince, či jej obejde a ponechá drakovi život i mince.

V okamžiku, kdy přijde k princezně o krásе K, nastávají dvě možnosti. Pokud již rytíř zabil alespoň K draků, princezna se do něj zamiluje a chce si jej vzít. Rytíř není schopen odmítnout a jeho výprava končí. Pokud rytíř zabil menší počet draků, prohodí pár zdvořilých slov s princeznou a pokračuje ve výpravě.

Na posledním políčku sídlí princezna, kterou rytíř miluje. Pro zadanou trasu výpravy určete, zda je možné získat princeznu na posledním poli. Pokud ano, vypište seznam zabitých draků tak, aby rytíř získal nejen princeznu na posledním poli, ale i co nejvíce zlatých mincí.

Tato úloha je praktická a řeší se ve vyhodnocovacím systému CodEx.[1] Přesný formát vstupu a výstupu, povolené jazyky a další technické informace jsou uvedeny v CodExu přímo u úlohy.

Konečně jsem se mohl vrátit na místo dodávky, ta už tam přirozeně nebyla. Místo jsem prohledal. V nedalekém průchodu jsem objevil svého starého kolegu a přítele, říkejme mu Jan. Jan se tam bavil s dalším mužem, pravděpodobně Japoncem. K mé smůle si mne všimli a Japonec se dal na útěk. Rozběhl jsem se za ním, ale Jan se mi postavil do cesty.

Měl s sebou obří brašnu na spisy. Vypadal opravdu vyděšeně, říkal jen: „Prosím, nech mě odejít. Nikdy jsme se tady neviděli.“ Tak jsem to i udělal. Být to kdokoli jiný, okamžitě jej zatýkám a zabavuji tyto spisy. Toto ale byl Jan – můj nejlepší přítel, kterému jsem vděčil opravdu za mnoho. V dalších částech svého vypravování se k tomu snad ještě dostanu.

⋆⋆⋆

O kauze toho později proniklo mnoho ven. Došlo i k sérii vražd, asi 2 měsíce po dni, o němž jsem vyprávěl. O 20 let později se mi podařilo dokonce dostat k odtajněným spisům GIBS a ÚOOZ. Vyšetřovalo se velmi důkladně, ale nakonec byl případ uzavřen pro nedostatek důkazů.

25-3-7 Zkratky — 7 bodů

Svět je plný zkratek, nejen těch policejních. V této úloze máte zadáno nejvýše 10 zkratek o nejvýše pěti písmenech a slovo délky L. Vaším úkolem je rozhodnout, zda lze

[1] `http://ksp.mff.cuni.cz/viz/codex`

slovo sestavit ze zadaných zkratek. Zkratky je možno spojovat za sebe a jednu zkratku lze použít i opakovaně.

Příklad: Pokud bychom jako zkratky měli oblíbenou čtveřici *sus*, *usu*, *sss*, *Ssu* a ještě pro obohacení *su*. Pak slovo *Ssusss* postavit půjde, slovo *sususu* také a rovnou dvěma způsoby, ale slova *ssss* nebo *susSus* už nepostavíme.

S Janem jsme překvapivě zůstali v kontaktu. Říkal, že se svou nerozvážností dostal do obřího průšvihu a šlo mu o život. Domluvili jsme se, že další informace si raději nechá pro sebe. Mnoho času jsem přemítal nad tím, jaká je cena přátelství. Trápila mě otázka, zda právě tyto spisy nemohly případ rozuzlovat a třeba i zabránit dalšímu krveprolití...

Ke zpovědi policisty se dostal

Lukáš Folwarczný

25-3-8 Tabulatika — 13 bodů

Třetí díl seriálu o TEXu bude snad méně děsivý než druhý. Nebojte, budeme se věnovat „jenom“ tabulkám, pokročilé matematice a podrobnostem sazby odstavců.

Sazba na tabulátory

Nejjednodušší tabulky můžeme sázet jednoduchým způsobem. Následující konstrukcí rozdělíme stránku na 3 stejně široké sloupce a pak do nich sázíme řádky:

```
\settabs 3\columns
\+Sloupec 1&Sloupec 2&Sloupec 3\cr
\+&Jen druhý\cr
\+První&&a třetí\cr
\+Vykřičník je ve čtvrtém sloupci~-- mimo&&&!\cr
\+První sloupec je příliš dlouhý&
  a druhý se přesází přes něj.\cr
\+Mezery za \&&     se ignorují.\cr
\+\hfill Tento řádek&\hfill je zarovnán
  &\hfill na pravý okraj.&\cr

Ukončení tabulky se nijak zvlášť neřeší.
```

Sloupec 1	Sloupec 2	Sloupec 3
	Jen druhý	
První		a třetí
Vykřičník je ve čtvrtém sloupci – mimo		!
První sloupec je příliš dlouhý	a druhý se přesází přes něj.	
Mezery za &	se ignorují.	
Tento řádek	je zarovnán	na pravý okraj.

Ukončení tabulky se nijak zvlášť neřeší.

Poznámka: Důvod, proč zde má `\hfill` najednou dvě L, vysvětlíme v tomto díle seriálu v kapitole o různých typech lepidla.

Nelíbí se vám stejně široké sloupce? Vymyslete si vzorový řádek, podle kterého TEX nastaví šířky sloupců:

```
\settabs\+První sloupec &Druhý sloupec
        &Třetí sloupec &Zbytek&\cr
\+A&B&C&D&E\cr
\+První sloupec &Druhý sloupec
 &Třetí sloupec &Zbytek&!\cr
```

A	B	C	D	E
První sloupec	Druhý sloupec	Třetí sloupec	Zbytek!	

Vzorový řádek se nezobrazí, jen se podle něj nastaví šířka sloupců.

Úkol 1 [2b]: Vysázejte pomocí tabulátorů tuto jednoduchou tabulku z knihy jízd:

Odkud	*Kam*	*Kdy*	*Kolik km*
Praha	Olomouc	21. 12.	250
Olomouc	Uherské Hradiště	30. 12.	130
Uherské Hradiště	Vyšší Brod	5. 1.	350
Vyšší Brod	Jablonec nad Nisou	17. 2.	324

Správně odsazený zdrojový kód

Tabulátory vůbec nemusí být extra pevné. Můžeme zrušit všechny tabulátorové pozice vpravo od aktuální „buňky“ příkazem `\cleartabs`. A pokud použijeme mezi `\+` a `\cr` znak `&` na místě, kde ještě není definovaná tabulátorová pozice, tak se ta pozice jednoduše nadefinuje právě na ono místo.

Víc asi ukáže příklad:

```
\cleartabs
\+{\bf if} $x<0$: &{\bf if} $x<-1000$:
  &{\it print} \uv{$x$ je echt záporné}\cr
\+&{\bf else}:
  &{\it print} \uv{$x$ je trochu záporné}\cr
\+{\bf else}: &\cleartabs{\bf if} $x>0$:
  &{\it print} \uv{$x$ je kladné}\cr
\+&{\bf else}: &{\it print} \uv{$x$ je nula}\cr
```

if $x < 0$: **if** $x < -1000$: *print* „x je echt záporné“
else: *print* „x je trochu záporné“
else: **if** $x > 0$: *print* „x je kladné“
else: *print* „x je nula“

Tabulky

Při sazbě na tabulátory je potřeba odhadnout, který sloupec bude jak dlouhý, a podle toho nastavit šířku sloupců. Také pokud chcete tabulku s orámováním, nemáte moc rozumných možností. TEX však nabízí mocnější nástroj než sazbu na tabulátory – primitivum `\halign`.

Tabulku z **úkolu 1** vysázíme primitivem `\halign` takto:

```
\halign{
# \hfil&# \hfil&# \hfil& \hfil#\cr
\it Odkud&\it Kam&\it Kdy&\it Kolik km\cr
Praha&Olomouc&21. 12.&250\cr
Olomouc&Uherské Hradiště&30. 12.&130\cr
Uherské Hradiště&Vyšší Brod&5. 1.&350\cr
Vyšší Brod&Jablonec nad Nisou&17. 2.&324\cr
}
```

Tabulka se skládá z jednotlivých řádků oddělených od sebe značkou `\cr`. Buňky se od sebe oddělují znakem `&` (nebo jiným znakem kategorie 4 – alignment).

První řádek obsahuje vzor. V každé buňce musí být znak `#` (kategorie 6 – parameter) právě jednou (jinak vám TEX vynadá), jinak může být vzorem prakticky libovolný kus TEXu, který se dá použít uvnitř hboxu (třeba `\bye` není povolený příkaz). Buňky se pak sází tak, že se v příslušném vzoru nahradí výskyt znaku `#` za obsah buňky.

Dejte si pozor na mezery – mezery za `&` se ignorují, mezery před `&` se neignorují.

Tabulky a boxy

Při sazbě tabulky si TEX zjistí pro každý sloupec, jak bude široký, a podle toho nastaví šířku všem jeho buňkám. Každá buňka tabulky je separátní hbox široký právě tak jako celý sloupec. Pokud nechcete mít ošklivě roztažené mezery v textu, vložte na vhodné místo `\hfil`-y, vizte příklad výše.

Očárovaná tabulka

Potřebujete-li do tabulky vložit vodorovnou čáru (nebo libovolný jiný materiál), využijte prostředí `\noalign`. Tím začne vertikální box šířky přesně takové, jak je široká celá tabulka:

```
\halign{
\strut# \hfil&# \hfil&# \hfil& \hfil#\cr
\it Odkud&\it Kam&\it Kdy&\it Kolik km\cr
\noalign{\hrule\smallskip\line{\hfil
    2012 \hfil}\smallskip\hrule\smallskip}
Praha&Olomouc&21. 12.&250\cr
Olomouc&Uherské Hradiště&30. 12.&130\cr
\noalign{\hrule\smallskip\line{\hfil
    2013 \hfil}\smallskip\hrule\smallskip}
Uherské Hradiště&Vyšší Brod&5. 1.&350\cr
Vyšší Brod&Jablonec nad Nisou&17. 2.&324\cr
}
```

Odkud	*Kam*	*Kdy*	*Kolik km*
	2012		
Praha	Olomouc	21. 12.	250
Olomouc	Uherské Hradiště	30. 12.	130
	2013		
Uherské Hradiště	Vyšší Brod	5. 1.	350
Vyšší Brod	Jablonec nad Nisou	17. 2.	324

Potřebujeme-li i svislé čáry, je náš úkol o poznání složitější. TEX totiž vkládá každý řádek tabulky samostatně do stránkového vboxu jako jednotlivé hboxy. Takže mezi ně vloží `\lineskip` nebo `\baselineskip`. Proto je musíme vypnout a výšky řádků nastavit explicitně. Na vypnutí lineskipů „pořádně a důkladně“ použijte makro `\offinterlineskip`, které myslí i na zběsilé okrajové případy.

Aby byl každý řádek stejně vysoký, je potřeba do něj vložit vzpěru. Jinak by sazba vypadala ošklivě. K tomu se hodí makro `\strut`, které je v Plain TEXu definováno přibližně takto:

```
\def\strut{\vrule height 8.5pt
           depth 3.5pt width 0pt\relax}
```

Odkud	*Kam*	*Kdy*	*Kolik km*
	2012		
Praha	Olomouc	21. 12.	250
Olomouc	Uherské Hradiště	30. 12.	130
	2013		
Uherské Hradiště	Vyšší Brod	5. 1.	350
Vyšší Brod	Jablonec nad Nisou	17. 2.	324

```
{\offinterlineskip
\def\higher{\vrule height 11pt
            depth 3.5pt width 0pt\relax}
\halign{%
\strut# \hfil&# \hfil\vrule\ &# \hfil& \hfil#\cr
\it Odkud&\it Kam&\it Kdy&\it Kolik km\cr
\noalign{\hrule\smallskip\line{\hfil
    2012 \hfil}\smallskip\hrule}
\higher Praha&Olomouc&21. 12.&250\cr
Olomouc&Uherské Hradiště&30. 12.&130\cr
\noalign{\hrule\smallskip\line{\hfil
    2013 \hfil}\smallskip\hrule}
\higher Uherské Hradiště&Vyšší Brod&5. 1.&350\cr
Vyšší Brod&Jablonec nad Nisou&17. 2.&324\cr
}}
```

Ještě se vám můžou hodit dvě tabulkové operace: `\omit` na začátku buňky dočasně nahradí vzor pro tuto buňku za prosté `#`.

Místo `&` v běžném řádku můžete použít `\span`. V tu chvíli se příslušné dvě buňky spojí. To, co bylo před `\span`-em, se vloží na místo prvního `#`, a to, co je za `\span`-em, se vloží na místo druhého `#`.

```
{\offinterlineskip
\halign{
\strut1A#1B\hfil\vrule&\ 2A#2B\hfil\vrule
   &\ 3A#3B\hfill\cr
x&y&z\cr
\omit\strut x x&y\span z\cr
\omit\strut\hfil x\span\omit y&z\cr
xxx& yyy& zzz\cr
x\span y\span z\cr
\omit\span\omit\span\omit
   \strut\hfil abcde\hfil\cr
}}
```

1Ax1B | 2Ay2B | 3Az3B
x | x 2Ay2B | 3Az3B
xy 3Az3B
1Axxx1B | 2Ayyy2B | 3Azzz3B
1Ax1B | 2Ay2B | 3Az3B
abcde

Podrobnosti o tom, jak se vlastně TEX v tabulkách chová, si najděte v TEXbooku v kapitole 22 (nebo v TBN[2] v kapitole 4) – překračuje to rámec tohoto seriálu.

Úkol 2 [6b]: Definujte makra pro sazbu výsledkové listiny KSP. Zdroják výsledkovky pak může vypadat například takto:

```
\vysledkovka{
\radek 1. Petr Pilný (GABCD; 4; 7):
       12 7 - 4 9 5 - 7: 49,4 69,3
\radek 2. ...
}
```

Pokud se vám nelíbí současný desing naší výsledkovky, navrhněte lepší a přehlednější.

Poznámka: Pokud byste potřebovali tabulku ne po řádkách, ale po sloupcích, zkuste `\valign`. Funguje to stejně jako `\halign`, jenom otočeně.

Periodická hlavička

Zdvojíte-li v hlavičce na nějakém (nejvýše jednom) místě `&`, říkáte tím TEXu: „Zde začíná periodická část hlavičky.“ Tedy pokud by v nějakém řádku došly vzory pro buňky, tak začne recyklovat vzory od `&&` dál:

```
\halign{1# & 2# && 1P# & 2P# \cr
A&B&C&D&E&F&G&H&I&J&K&L\cr
A&B&C&D&E&F&G\cr
}
```

1A 2B 1PC 2PD 1PE 2PF 1PG 2PH 1PI 2PJ 1PK 2PL
1A 2B 1PC 2PD 1PE 2PF 1PG

Tolik k tabulkám.

Matematické závorky

Ve druhé části poněkud rozkouskovaného třetího dílu se budeme věnovat složitějšímu využití matematického módu.

[2] Petr Olšák: TEXbook naruby, Konvoj Brno 2001 (2. vydání), ISBN 80-7302-007-6

Složité vzorce jsou často různě zběsile vysoké a jedna velikost závorek nestačí. Prohlédněte si příklad:

$$\left(\sqrt{(x+\sqrt{x-1})\left(\sum_{k=1}^{\infty}\frac{1}{k^2}\right)}\right)$$

TEX si umí určit velikost závorek sám, jen je potřeba mu říct, která patří ke které. Používají se k tomu primitiva `\left` a `\right`, kterými se označí příslušné závorky:

```
$$\left(\sqrt{
    \left(x + \sqrt{x - 1}\right)
    \left(\sum_{k=1}^\infty {1 \over k^2}\right)
  }\right)$$
```

TEX sám zvolí velikost závorek takovou, aby byly vhodně vysoké. Pokud byste potřebovali, aby se jedna ze závorek nezobrazila, použijte místo ní tečku:

```
\left({x\over y}\right.
```

$$\left(\frac{x}{y}\right.$$

Typů závorek je hrozná spousta, Plain určitě zná tyto:

```
$$()[]\{\}\vert\Vert\lceil\rceil
  \lfloor\rfloor\langle\rangle
  \backslash/\uparrow\downarrow
  \Uparrow\Downarrow\updownarrow\Updownarrow$$
```

$$()[]\{\}\vert\Vert\lceil\rceil\lfloor\rfloor\langle\rangle\backslash/\uparrow\downarrow\Uparrow\Downarrow\updownarrow\Updownarrow$$

Pokud byste čirou náhodou potřebovali použít jiné závorky, konzultujte TEXbook, kapitolu 17 a následující (nebo TBN, kapitolu 5).

Může se vám však stát, že potřebujete jinak velkou závorku, než vám dá TEX. V takovém případě jsou pro vás připraveny varianty `\big`, `\bigg`, `\Big` a `\Bigg`:

```
$\Bigg(\bigg(\Big(\big(()\big)\Big)\bigg)\Bigg)$
```

$$\Bigg(\bigg(\Big(\big(()\big)\Big)\bigg)\Bigg)$$

Zákulisí matematické sazby

Ani celý rozsah seriálu by zdaleka nevystačil na vyložení sazby matematiky v celé její kráse a síle. Donald Knuth na to v TEXbooku vynaložil přes 60 stran. Naznačíme si však některé principy, které by vám při pokročilé sazbě neměly zůstat utajeny.

TEX umí velmi sofistikovaně rozložit mezery do matematického vzorce, pokud ví, jakého typu je která část vzorce. Rozlišuje těchto 13 typů:

Ord	Standardní „atom“	x či 1
Op	Velký operátor	$\sum$ či $\int$
Bin	Binární operátor	$+$ či $-$
Rel	Relace	$=$ nebo $<$
Open/Close	Začátek/konec závorky	(/), [/], …
Punct	Interpunkce	čárka, středník
Inner	Složitější konstrukce	zlomek
Over/Under	Atom s čárou nad/pod	$\overline{x}$, $\underline{x}$
Acc	Atom s „diakritikou“	$\hat{x}$, $\vec{x}$
Rad	Cosi pod odmocninou	$\sqrt{2}$
Vcent	Výstup z `\vcenter`	

Poznámka: PlainTEX má v matematice definována tato diakritická znaménka: `\acute` ($\acute{a}$), `\bar` ($\bar{a}$), `\breve` ($\breve{a}$), `\check` ($\check{a}$), `\ddot` ($\ddot{a}$), `\dot` ($\dot{a}$), `\grave` ($\grave{a}$), `\hat` ($\hat{a}$), `\vec` ($\vec{a}$), `\tilde` ($\tilde{a}$), `\widetilde` ($\widetilde{abc}$), `\widehat` ($\widehat{abc}$). Používají se stejným způsobem jako nematematická.

Poznámka: `\vcenter` je explicitní vbox v matematickém módu, který je stejně vysoký jako hluboký, takže se vysází centrovaný na osu. Vůbec, matematika se sází ne na účaří, ale na pomyslnou vodorovnou osu, neboť vzorce jsou často takto symetrické. Ale toho už jste si jistě všimli.

Mezi jednotlivými kategoriemi je pak definována tabulka, která určuje, kam patří jak velká mezera. Tato tabulka je pro drtivou většinu použití dostačující.

Pokud vám připadá, že TEX nějakou mezeru určil chybně, zkuste označit atomy v jejím okolí jejich typem. K tomu použijte `\mathord`, `\mathop`, `\mathbin`, `\mathrel`, `\mathopen`, `\mathclose`, `\mathpunct` a `\mathinner`. Zbytek typů se značí automaticky a nelze je označit ručně.

Pokud selže i označení atomu vhodným typem, můžete použít mezery různých velikostí – `\,`, `\;` nebo `\!`:

```
$$x\!y\quad xy\quad x\,y\quad x\;y$$
```

$$xy\quad xy\quad x\,y\quad x\;y$$

Matematický mód se dělí na několik dílčích módů. Všimněte si, jak se vysází různé výrazy na různých místech – `$$\sum$$`, `$\sum$`, `$x^{\sum}$` či `$x^{x^{\sum}}$`. Nejde jen o velikost, ale také o umístění indexů a rozložení mezer.

Ony dílčí módy jsou D, T, S a SS: „display“, „text“, „script“ a „scriptscript“ uvedené ve stejném pořadí jako v předchozím odstavci.

Chcete-li si vyzkoušet víc magie okolo módů, poradím primitivum `\mathchoice`:

```
\def\te{{\mathchoice{D}{T}{S}{SS}}}
$$\te {\te^{\te^\te} \over \te^{\te^\te}}$$
```

$$D\frac{T^{S^{SS}}}{T^{S^{SS}}}$$

Nebo můžete vynutit konkrétní mód použitím `\displaystyle`, `\textstyle`, `\scriptstyle` a `\scriptscriptstyle`.

Tabulkové konstrukce v matematice

Plain TEX definuje některé maticové konstrukce:

```
$$\pmatrix{a_{11}&a_{12}&\ldots&a_{1n}\cr
           a_{21}&a_{22}&\ldots&a_{2n}\cr
           \vdots&\vdots&\ddots&\vdots\cr
           a_{m1}&a_{m2}&\ldots&a_{mn}\cr}$$
```

$$\pmatrix{a_{11}&a_{12}&\ldots&a_{1n}\cr a_{21}&a_{22}&\ldots&a_{2n}\cr \vdots&\vdots&\ddots&\vdots\cr a_{m1}&a_{m2}&\ldots&a_{mn}\cr}$$

Použití je obdobné jako u tabulek, jen nemusíte definovat vzorový řádek. Matici bez závorek můžete získat použitím `\matrix`. A na matici s okraji je definováno makro `\bordermatrix` (vyzkoušejte sami).

Úkol 3 [3b]: Vysázejte tento vzorec:

$$Z = \begin{cases} N > 0: & \displaystyle\sum_{i=1}^{N} \left(2 \sum_{i<j} \log |\lambda_i - \lambda_j| - \sum_{i=1}^{N} V(\lambda_i) \right) \\ N < 0: & -\frac{1}{N^2} \\ N = 0: & 0 \end{cases}$$

Sázíme odstavec

Další kousek třetího dílu věnujeme podrobnějšímu náhledu na algoritmus lámání odstavce.

Když se na vstupu objeví něco, co by mělo začít odstavec (znak, `\indent`, ...),[3] zkontroluje TEX několik věcí. Na začátek horizontálního boxu, který bude později zalámán na jednotlivé řádky v odstavci, se vloží prázdné místo velikosti `\parindent` (pokud nebyl odstavec zahájen pomocí příkazu `\noindent`).

Pak se expanduje `\everypar`, což je seznam tokenů (pseudomakro), které se má vložit na začátek každého odstavce. Přiřazuje se do něj zavoláním `\everypar={něco}`. Standardně je prázdný. Pak se do horizontálního boxu postupně vkládají znaky, dokud se neobjeví `\par`.

Nyní se na úplný konec boxu vloží prázdné místo velikosti `\parfillskip` (standardně `\hfil`). Pak se TEX pokusí zalámat odstavec s přihlédnutím k nastavenému tvaru odstavce. Každý řádek musí obsahovat nejprve prázdné místo velikosti `\leftskip`, pak příslušný kus odstavce a pak prázdné místo velikosti `\rightskip`. Dohromady je vždy široký přesně `\hsize`.

Poznámka: Pro zjednodušení nyní vynecháváme možnost úpravy tvaru odstavce primitivem `\parshape` a nastavením `\hangindent` a `\hangafter`, které nás čekají v další sérii.

TEX se pokouší zalámat odstavec celkem třikrát. Nejprve v mezerách mezi slovy. Pokud mu to nejde, zkusí rozdělit slova podle pravidel pro dělení slov. Pokud se mu ani toto nepovede, zkusí nepatrně roztáhnout mezery a rozdělit slova. Pokud selže i poslední pokus, vyhlásí chybu, nechá nějaký řádek přetéct a vykreslí slimáka.

Lámání odstavce probíhá najednou, TEX se snaží, aby nebyl jeden řádek ošklivě stažený a hned ten další ošklivě roztažený – má nějaký základní smysl pro estetiku. Nicméně když při druhém a třetím pokusu dělí slova, musí vědět, kde všude může dělit, jinak může odstavec vyjít zbytečně ošklivě. To je ten zásadní důvod, proč jsem v první sérii všem řešitelům bez `\language\czech` strhával body.

Přesný algoritmus včetně všech podrobností, které jste nikdy nechtěli znát, najdete v TEXbooku v kapitole 14 (nebo v TBN v kapitole 6).

Lepidlo

Prázdné místo je v sazbě důležitá věc. Někdy se s nadsázkou dokonce říká, že celá typografie je věda o prázdném místě. V poslední části tohoto dílu seriálu se naučíme pracovat s jeho roztažností.

TEX nahlíží na prázdné místo jako na kusy „lepidla“, které se může různě roztahovat a smršťovat. Český odborný název je „výplněk“. Je několik druhů roztažnosti:

Pevný výplněk svoji velikost nemění. Je vždy stejně velký:

```
\hbox{A\hskip 2cm\relax B}
```

A B

Omezeně roztažitelný výplněk má základní velikost a meze, kam až se může roztáhnout.

```
\hbox to 15mm{A\hskip 2cm plus 1cm minus 1cmB}
\hbox to 25mm{A\hskip 2cm plus 1cm minus 1cmB}
```

A B
A B

Roztažnost nemusí být na obě strany stejná: `10mm plus 5mm minus 3mm`

Nekonečně roztažitelný výplněk má základní velikost a jinak se může roztáhnout neomezeně podle potřeby.

```
\hbox to 5cm{A\hskip 2cm plus 1fil\relax B}
\hbox to 1cm{A\hskip 2cm minus 1fil\relax B}
```

A B
A B

Nekonečných roztažností jsou tři druhy: `fil`, `fill` a `filll`. Čím víc L, tím agresivněji se roztahuje.

Jeden výplněk je nuda. Co kdyby se někde potkalo více výplňků? Když TEX skládá box, u kterého dopředu neví, jak bude velký, tak se vůbec roztažností neřídí. Ideální je vždycky, když se nic natahovat nemusí, takže použije vždy pouze základní velikost. V opačném případě má nařízeno, jak musí být výsledný box velký, a musí se do něj chtě nechtě vejít.

Pokud je nutné roztahovat a stahovat bílé místo, TEX určí správné mezery přibližně takovýmto algoritmem:

1. Zjistí, jakou velikost by měl box, kdyby se použily základní velikosti. Určí rozdíl mezi požadovanou a základní velikostí. Je-li rozdíl kladný, bude nás dále zajímat pouze kladná roztažnost; je-li rozdíl záporný, budeme řešit pouze zápornou roztažnost. Tento rozdíl si označme jako w.
2. Sečte povolenou roztažnost přes celý box – zvlášť omezenou a zvlášť každý druh nekonečné roztažnosti.
3. Vybere nejagresivnější roztažnost, která je nenulová, a tou se bude zabývat.
4. Rozpočítá w mezi všechny výplňky, které přispěly do roztažnosti, podle poměru, ve kterém přispěly.

Vizte příklad:

```
\hbox{\vrule
\hbox to 6cm{\strut%
  \hskip 5mm plus 3cm
  \vrule width 1cm
  \hskip 1cm plus 1cm minus 1cm
  \vrule width 1cm
  \hskip 5mm minus 2cm
}\vrule}
```

Vnitřní hbox má být široký 6 cm. Základní šířka boxu vyjde $0{,}5 + 1 + 1 + 1 + 0{,}5 = 4$ cm, $w = 6 - 4 = 2$ cm. Zajímá nás tedy kladná roztažnost. Čáry se neroztahují, řešíme tedy jen skipy: $3 + 1 + 0 = 4$ cm, jiná roztažnost není.

[3] Primitivum `\indent` explicitně započne odstavec; `\indent\indent` si vyžádá dvojité odsazení (možno opakovat vícekrát pro vícenásobné odsazení). A konečně explicitní začátek odstavce bez odsazení vynutíte primitivem `\noindent`.

Potřebujeme tedy rozdělit 2 cm mezi první a druhý skip v poměru 3 : 1, tedy prvnímu skipu se přidělí 15 mm a druhému 5 mm. Box se tedy vysází, jako by byl zadán takto:

```
\hbox to 5cm{\hskip 20mm\vrule width 1cm
  \hskip 15mm\vrule width 1cm\hskip 5mm}
```

Druhý příklad bude složitější:

```
\hbox{\vrule
\hbox to 5cm{%
  \hskip 5mm plus 1fil
  \vrule width 1cm
  \hskip 1cm plus 2cm minus 1cm
  \vrule width 1cm
  \hskip 5mm minus 1fil
}\vrule}
```

Základní šířka boxu je 5 cm, obsah má základní šířku 4 cm, $w = 1$ cm. Celková kladná roztažnost je 2 cm + 1 fil, takže nás zajímá jen 1 fil. Prvnímu skipu se tedy přidělí celý 1 cm.

Rozmyslete si, co se stane, když:

- poslední hskip nebude mít `minus 1fil`, ale `plus -1fil`;
- první hskip bude mít `plus 1fill`;
- bude hbox `to 4cm` nebo `to 3cm`.

Nyní je čas na důležitou poznámku. Roztažnost mají pouze skipy (výplňky), mezi které se počítají i běžné mezery mezi slovy. Všechno ostatní (boxy, čáry) má pevnou velikost, jakmile je to vytvořeno. Není možné vytvořit box, jehož rozměry by byly pružné podle jeho okolí. Najděte si třeba ve své sazbě řádek s hodně roztaženými mezerami a část toho řádku uzavřete do hboxu. Všimněte si, co se stane s mezerami, a zkuste si rozmyslet, proč to tak může být.

Připomínám, že je nanejvýš vhodné si vyzkoušet práci s tabulkami i pokročilou matematikou na uvedených příkladech. Zkuste je dál modifikovat a hrát si s nimi, ať si to všechno důkladně zažijete. Příště budeme konečně programovat.

> **Úkol 4** [2b]: Vymyslete nastavení parametrů odstavce tak, aby se zarovnal na střed, přibližně tak, jako je zarovnáno zadání tohoto úkolu.
>
> Řešení nemusí být univerzální, stačí, aby se zadaný odstavec povedlo vysázet v běžném případě. Nemusíte řešit okrajové případy, kdy je odstavec extrémně krátký apod.

A to bude ze třetí série vše. Těším se na vaše řešení.

Jan „Moskyto“ Matějka

Recepty z programátorské kuchařky: Teorie čísel

Dnes si budeme povídat o různých užitečných vlastnostech celých čísel, především o dělitelnosti a kongruencích. Mohlo by se zdát, že to nemá s informatikou nic společného, ale překvapivě v informatice zakopáváme o teorii čísel takřka na každém kroku. Někdy se jedná o hledání velkých prvočísel, jindy o rychlé násobení čísel s miliony cifer nebo všudypřítomnou asymetrickou šifru RSA.

Začneme vyjasněním základních pojmů, postupně se prokoušeme kongruencemi k hledání největšího společného dělitele a Bézoutových koeficientů, chvilku se zamyslíme nad prvočísly a nakonec si také ukážeme, jak to všechno souvisí s čínskou armádou.

Definice na úvod

Množinu celých čísel si označíme $\mathbb{Z}$ a každé její podmnožině $\{0, 1, \ldots, n-1\}$ budeme říkat $\mathbb{Z}_n$.

Často nás bude zajímat *dělitelnost:* $a \setminus b$ (nebo $a \mid b$) budeme značit, že číslo a je dělitelem čísla b (nebude-li hrozit mýlka, čteme prostě „a dělí b“).

Pro *největšího společného dělitele* dvou čísel zavedeme symbol $\mathrm{nsd}(a, b)$. Pokud $\mathrm{nsd}(a, b) = 1$, říkáme, že čísla a a b jsou *nesoudělná,* zkráceně $a \perp b$. Když budou naopak a a b soudělná, napíšeme $a \parallel b$. Podobně nejmenší společný násobek dvou čísel označíme $\mathrm{nsn}(a, b)$ a všimneme si, že je roven $a \cdot b \,/\, \mathrm{nsd}(a, b)$.

Není-li jedno číslo dělitelné druhým, znamená to, že při celočíselném dělení vznikne zbytek: například pokud vydělíme 23/8, dostaneme zbytek 7, protože $23 = 8 \cdot 2 + 7$. Obecně *zbytkem po dělení* a/b nazveme hodnotu z v rovnici $a = b \cdot x + z$, kde x je celé číslo a z je nezáporné celé číslo menší než b. Obvyklé programovací jazyky mívají takovouto operaci zabudovanou a říkají jí *modulo.* Programátoři počítají zbytky po dělení často nějakou konstrukcí podobnou `z = a % b`, zatímco matematici spíše píší $z = a \bmod b$.

Dodejme, že pro záporná čísla už není definice zbytku po dělení tak jednoznačná: v některých programovacích jazycích je `(-7)%3 = -1`, jiné se shodnou s naší definicí na tom, že vyjde `2`. Přitom oba výsledky vycházejí z jedné rovnice $a = b \cdot x + z$. Aby měla jednoznačné řešení, požadovali jsme $0 \leq z < b$. Lze na to ovšem jít i jinak: řekneme, že x má být celočíselný podíl a/b. Ten jde ale definovat dvěma způsoby: buď se zaokrouhlením dolů (což se shodne s naší definicí), nebo se zaokrouhlením k nule (to dělá většina procesorů), což dá pro záporné a záporný zbytek. Zkuste zjistit (a vysvětlit), jak je to ve vašem oblíbeném jazyce a co se stane, když je záporné i číslo, kterým modulíme.

Kongruence

Když čísla p a q dávají stejný zbytek po dělení číslem m, píšeme

$$p \equiv q \pmod{m}$$

a čteme „p je *kongruentní* s q modulo m“. To platí právě tehdy, je-li rozdíl $p - q$ dělitelný m.

Zápis kongruence tak trochu připomíná rovnici. To není náhoda – kongruence totiž můžeme upravovat podobně jako rovnice.

Součet dvou kongruencí: Pokud $a \equiv A$ a $b \equiv B$, pak také platí $a + b \equiv A + B$ (to vše modulo totéž m). Že je to pravda, nahlédneme snadno. Napišme si a jako $n_a \cdot m + z_a$ a čísla A, b, B obdobně. Pak dostaneme:

$$\begin{aligned} a + b &= (n_a \cdot m + z_a) + (n_b \cdot m + z_b) = \\ &= (n_a + n_b) \cdot m + (z_a + z_b), \\ A + B &= (n_A \cdot m + z_A) + (n_B \cdot m + z_B) = \\ &= (n_A + n_B) \cdot m + (z_A + z_B). \end{aligned}$$

Protože $a \equiv A$ a $b \equiv B$, musí být $z_a = z_A$ a $z_b = z_B$. Můžeme si tedy všimnout, že rozdíl mezi $a + b$ a $A + B$ je $((n_a + n_b) - (n_A + n_B)) \cdot m$. To je násobek m, takže $a + b \equiv A + B$.

Rozdíl dvou kongruencí: Nahlédneme obdobně.

Přičtení téhož čísla k oběma stranám: Pokud $a \equiv A$, pak platí $a + k \equiv A + k$ pro libovolné k. Stačí totiž přičíst evidentně platnou kongruenci $k \equiv k$.

Přičtení násobku m k jedné straně: Z $a \equiv A$ plyne $a+k \equiv A$ pro libovolné k, které je násobkem modulu m. Přičítáme totiž kongruenci $k \equiv 0$.

Vynásobení dvou kongruencí: Z $a \equiv A$ a $b \equiv B$ plyne $ab \equiv AB$. Stejně jako u součtů a rozdílů, i zde stačí čísla rozepsat na součty násobků m a zbytků po dělení m:

$$\begin{aligned} a \cdot b &= (n_a \cdot m + z_a) \cdot (n_b \cdot m + z_b) = \\ &= (n_a \cdot n_b \cdot m + n_a \cdot z_b + n_b \cdot z_a) \cdot m + (z_a \cdot z_b) \equiv \\ &\equiv z_a \cdot z_b, \\ A \cdot B &= (n_A \cdot m + z_A) \cdot (n_B \cdot m + z_B) = \\ &= (n_A \cdot n_B \cdot m + n_A \cdot z_B + n_B \cdot z_A) \cdot m + (z_A \cdot z_B) \equiv \\ &\equiv z_A \cdot z_B. \end{aligned}$$

Přitom opět víme, že $z_a = z_A$ a $z_b = z_B$.

Vynásobení obou stran kongruence tímtéž číslem: Pokud $a \equiv A$, platí také $ax \equiv Ax$ pro libovolné x. To plyne z násobení kongruencí $x \equiv x$.

Ekvivalentnost úprav: Běžné úpravy rovnic jsou takzvaně ekvivalentní – to znamená, že fungují oběma směry, takže řešení rovnic ani neubírají, ani nepřidávají. Jak je to s kongruencemi? Sčítání kongruencí ekvivalentní musí být, protože opačný směr odpovídá odečtení kongruencí, což víme, že je také korektní úprava.

U násobení kongruencí to už tak jasné není. Zkusme zjistit, jestli je pravda, že z kongruence $ax \equiv Ax$ plyne $a \equiv A$. Pro $x = 0$ to jistě neplatí, ale co když zvolíme jiné x?

Vyzkoušíme to třeba na následujícím příkladě:

$$a \equiv 5 \pmod{14}$$

Takováto kongruence má jednoduché řešení: a je každé celé číslo, které dostanu sečtením 5 a nějakého násobku 14. To se dá zapsat třeba takhle:

$$a \in \{5 + k \cdot 14 \mid k \in \mathbb{Z}\},$$

tudíž a může tedy například 5, 19 nebo 33.

Vyzkoušíme nyní obě strany vynásobit… třeba trojkou:

$$3 \cdot a \equiv 15 \equiv 1 \pmod{14}.$$

Tato kongruence platí pro všechna a, která po vynásobení 3 dávají modulo 14 zbytek 1. Když máme nějaké a z první kongruence, je ve tvaru $5+k \cdot 14$. Vynásobíme-li ho 3, dostaneme $15 + 3 \cdot k \cdot 14$. To je určitě kongruentní s 1 modulo 14, takže žádné řešení jsme neztratili a po chvíli uvažování zjistíme, že jsme ani žádné nepřidali.

Další pokus: původní kongruenci vynásobíme místo trojky dvojkou. Dostaneme:

$$2 \cdot a \equiv 10 \pmod{14}.$$

Řešení původní kongruence pořád sedí, ale nová kongruence platí například i pro $a = 12$. Posuďte sami:

$$2 \cdot 12 = 24 = 10 + 14 \equiv 10 \pmod{14}.$$

Ouha, najednou vynásobení obou stran konstantou není ekvivalentní úprava!

Proč nám násobení trojkou fungovalo, ale násobení dvojkou si vymýšlí kořeny navíc? Postupně se ukáže, že násobení k je ekvivalentní úprava právě tehdy, když $k \perp 14$ (či obecněji $k \perp m$, počítáme-li modulo m).

Než k tomu dojdeme, nejdřív na chvíli odbočíme k největším společným dělitelům.

Euklidův algoritmus

Největšího společného dělitele dvou čísel můžeme vypočítat pomocí prvočíselného rozkladu, ale to je pro velká čísla velmi pomalé. Daleko lepší je použít prastarý Euklidův algoritmus. (Jmenuje se podle starověkého matematika Euklida, v jehož díle Základy se nachází první dochovaná verze. Jedná se zřejmě o nejstarší netriviální algoritmus, jaký se s drobnými úpravami používá dodnes. Dokonce je pravděpodobné, že Euklidés pouze sepsal dávno známý trik.)

Pojďme si odvodit, jak Euklidův algoritmus funguje. Nahlédneme, že pro libovolná čísla a, b $(a > b)$ platí:

$$\mathrm{nsd}(a, b) = \mathrm{nsd}(a - b, b).$$

Proč je to pravda? Dokážeme, že dvojice (a, b) a $(a - b, b)$ sdílejí dokonce všechny společné dělitele, takže i toho největšího:

- Nechť d je společným dělitelem a a b. Platí tedy $a = a' \cdot d$, $b = b' \cdot d$ pro nějaká celá čísla a' a b'. Pak ovšem můžeme zapsat $a - b$ jako $(a' - b') \cdot d$, což je zase dělitelné číslem d.
- Nechť naopak d je společným dělitelem $a - b$ a b. Opět zapíšeme $a - b = c' \cdot d$, $b = b' \cdot d$ a získáme $a = (a - b) + b = (c' + b') \cdot d$.

Euklidův algoritmus dostane na vstupu nějaká dvě čísla a a b a opakovaně odčítá menší z nich od většího. Jak už víme, tato operace zachovává největšího společného dělitele. Pokaždé se přitom součet $a + b$ zmenší, takže po konečně mnoha krocích musíme jedno z čísel vynulovat. Pak víme, že největším společným dělitelem je druhé z nich (platí přeci $\mathrm{nsd}(0, x) = x$).

Pojďme si takhle nějakého největšího společného dělitele spočítat. A abychom netroškařili, zkusme rovnou čísla 1518 a 945.

a	b
1518	945
573	945
573	372
201	372
201	171
30	171

Zastavme se na chvíli. Teď bychom mohli pracně odečítat 30 od b, dokud bychom nenašli v b něco menšího než 30. Buďme trochu líní: když to budeme dělat dost dlouho, zbude nám v b zkrátka zbytek po dělení b číslem 30. Můžeme tedy místo odečítání modulit. Pokračujeme:

a	b
30	171
30	$21 = 171 \bmod 30$
$30 \bmod 21 = 9$	21
9	$3 = 21 \bmod 9$
$9 \bmod 3 = 0$	3

V a nám zbyla 0, takže $\mathrm{nsd}(1518, 945) = 3$.

Pojďme si tento postup převést do návodu pro počítač. Při implementaci Euklidova algoritmu se hodí držet si v jedné proměnné pořád to větší z čísel a a b. Navíc můžeme využít toho, že každým krokem algoritmu se z většího čísla stane

menší, takže je stačí prohodit a není potřeba znovu porovnávat. (Dokonce i porovnání před cyklem bychom si mohli ušetřit, kdyby nám nevadilo, že první průchod cyklem může projít „naprázdno“.)

```
def Euclid(a, b):
    # Prohodíme, je-li třeba
    if b > a:
        a, b = b, a

    # Zde je vždy a >= b
    while b > 0:
        # nsd(a % b, b) = nsd(a, b).
        a = a % b
        a, b = b, a

    # nsd(0, a) = a.
    return a
```

Jak rychle náš algoritmus běží? Podívejme se, co se stane, když pustíme dva kroky algoritmu na čísla a_1 a b_1 ($a_1 \geq b_1$):

$a_2 = a_1 \bmod b_1$
$b_2 = b_1$ (nyní $a_2 < b_2$)
$a_3 = a_2 = a_1 \bmod b_1$
$b_3 = b_2 \bmod a_2 = b_1 \bmod (a_1 \bmod b_1)$ (nyní $a_3 > b_3$)

Dokážeme, že $a_3 < a_1/2$. Rozebereme přitom dva případy podle toho, jestli bylo b_1 menší, nebo větší než $a_1/2$:

- Pokud $b_1 \leq a_1/2$, pak určitě platí $a_3 < b_1$, a tedy i $a_3 < a_1/2$. (Zde využíváme toho, že zbytek po dělení čehokoliv číslem b_1 musí být menší než b_1.)
- V opačném případě leží b_1 mezi $a_1/2$ a a_1, takže $a_3 = a_1 \bmod b_1 = a_1 - b_1 < a_1/2$. (Poslední rovnost platí, protože $\lfloor a_1/b_1 \rfloor = 1$.)

Dokázali jsme tedy, že po dvou krocích algoritmu se větší z obou proměnných zmenší přinejmenším na polovinu a opět bude větší. Po $\mathcal{O}(\log n)$ krocích tedy musí větší proměnná klesnout pod 1, čímž se algoritmus zastaví. Euklidův algoritmus proto provede $\mathcal{O}(\log n)$ elementárních operací.

Jak dlouho ale trvá jedna elementární operace? Pokud počítáme s malými čísly, která se našemu počítači vejdou do celočíselné proměnné, zvládneme ji v konstantním čase. Jsouli ovšem čísla větší, musíme ještě zohlednit složitost aritmetických operací: porovnání čísel a operace modulo. Když použijeme modulení pomocí školního dělení, které je kvadratické v počtu cifer, strávíme v každém z $\mathcal{O}(\log n)$ kroků Euklidova algoritmu čas $\mathcal{O}(\log^2 n)$. Celková složitost algoritmu tedy vzroste na $\mathcal{O}(\log^3 n)$.

Rozšířený Euklidův algoritmus

Právě jsme našli největšího společného dělitele d nějakých dvou obrovských čísel a a b. Jak ale přesvědčíme svého pochybovačného kolegu, že je náš výsledek správně? Snadno ověříme, že d dělí obě čísla. Ale jak ukážeme, že žádné větší číslo už a ani b nedělí? Překvapivě to jde jednoduše dosvědčit: pokud pro nějaká u a v platí

$$a \cdot u + b \cdot v = d,$$

musí d být dělitelné každým společným dělitelem a a b, takže i číslem $\mathrm{nsd}(a, b)$. Nemůže tedy být menší než $\mathrm{nsd}(a, b)$.

Dobrá – kde taková u a v vzít? Kupodivu snadno: trochu upravíme Euklidův algoritmus. Nejprve ale prozradíme, že rovnici

$$a \cdot u + b \cdot v = \mathrm{nsd}(a, b)$$

se říká *Bézoutova identita* a číslům u a v *Bézoutovy koeficienty*.

Dokážeme, že spustíme-li Euklidův algoritmus na čísla a a b, v každém okamžiku se v proměnných `a` a `b` budou nacházet čísla tvaru $\alpha \cdot a + \beta \cdot b$ (kde α a β jsou nějaká celá čísla). Na začátku to triviálně platí, neboť $\mathtt{a} = a$ a $\mathtt{b} = b$, a pokaždé, když se proměnné mění, buď se prohazují, nebo se jedna odčítá od druhé. Obě tyto operace z výrazů uvedeného tvaru dělají opět výrazy uvedeného tvaru. Takže i konečný výsledek algoritmu, tedy $\mathrm{nsd}(a, b)$, musí jít zapsat v takovém tvaru.

Algoritmus proto upravíme tak, aby si stále udržoval proměnné α_a, α_b, β_a a β_b a vždy platilo

$$\mathtt{a} = \alpha_{\mathtt{a}} \cdot a + \beta_{\mathtt{a}} \cdot b,$$
$$\mathtt{b} = \alpha_{\mathtt{b}} \cdot a + \beta_{\mathtt{b}} \cdot b.$$

Ke konci algoritmu je, jak víme, $\mathtt{b} = \mathrm{nsd}(a, b)$, takže $\alpha_{\mathtt{b}}$ a $\beta_{\mathtt{b}}$ jsou hledané Bézoutovy koeficienty. Opět si to vyzkoušejme na výpočtu $\mathrm{nsd}(1518, 945)$:

a	$\alpha_{\mathtt{a}}$	$\beta_{\mathtt{a}}$	b	$\alpha_{\mathtt{b}}$	$\beta_{\mathtt{b}}$
1518	1	0	945	0	1
573	1	−1	945	0	1
573	1	−1	372	−1	2
201	2	−3	372	−1	2
201	2	−3	171	−3	5
30	5	−8	171	−3	5
30	5	−8	21	−28	45
9	33	−53	21	−28	45
9	33	−53	3	−94	151
0	315	−506	3	−94	151

Algoritmus tedy tvrdí, že hledaný nsd splňuje rovnost

$$1518 \cdot (-94) + 945 \cdot 151 = \mathrm{nsd}(1518, 945) = 3.$$

Snadným výpočtem ověříme, že je to pravda. (Poznamenejme, že Bézoutova identita má nekonečně mnoho řešení. Jak byste našli ta další?)

Převeďme své myšlenky do zdrojového kódu. Do `Aa`, `Ba`, `Ab`, `Bb` budeme ukládat koeficienty $\alpha_{\mathtt{a}}$, $\beta_{\mathtt{a}}$, $\alpha_{\mathtt{b}}$ a $\beta_{\mathtt{b}}$.

```
def ExtEuclid(a, b):
    Aa, Ba = 1, 0 # a = 1 * a + 0 * b
    Ab, Bb = 0, 1 # b = 0 * a + 1 * b
    # Prohodíme, je-li třeba
    if b > a:
        a, b = b, a
        Aa, Ab = Ab, Aa
        Ba, Bb = Bb, Ba

    # Zde je vždy a >= b
    while b > 0:
        # Odečteme od proměnné a proměnnou b
        # tolikrát, kolikrát se tam vejde.
        # ("/" značí celočíselné dělení)
        Aa = Aa - (a / b) * Ab;
        Ba = Ba - (a / b) * Bb;

        # nsd(a % b, b) = nsd(a, b).
        a = a % b

        # Prohodíme
        a, b = b, a
        Aa, Ab = Ab, Aa
        Ba, Bb = Bb, Ba
    # nsd(0, a) = a.
    # Vrátíme také Bézoutovy koeficienty.
    return [a, Aa, Ba]
```

Žádná operace, kterou děláme s koeficienty pro proměnné `a` a `b`, netrvá asymptoticky déle než operace modulo. Přidáním počítání Bézoutových koeficientů si tedy časovou složitost Euklidova algoritmu nezhoršíme.

Řešení lineárních kongruencí

Bézoutovy koeficienty jsou užitečné také k řešení kongruencí. Pojďme si to na jedné kongruenci vyzkoušet.

Máme nakoupena 4 vajíčka. V obchodě se vajíčka prodávají pouze v balíčcích po 6 kusech, zatímco my je skladujeme v platech po 20 kusech. Kolik si musíme koupit balíčků, abychom neměli v žádném platu volno?

Přepišme si tento příklad do formy kongruence:

$$4 + 6 \cdot x \equiv 0 \pmod{20},$$

čili

$$6 \cdot x \equiv 16 \pmod{20}.$$

To je totéž, jako že pro x a nějaké další celé číslo y platí

$$6 \cdot x + 20 \cdot y = 16.$$

Ejhle, to je rovnice podobná Bézoutově identitě. Kdyby na její pravé straně byl $\mathrm{nsd}(6, 20) = 2$, byla by to přesně Bézoutova identita a rozšířený Euklidův algoritmus by nám prozradil, že platí

$$6 \cdot (-3) + 20 \cdot 1 = 2.$$

Tím bychom měli vyřešeno.

Jenže v našem případě je na pravé straně 8krát víc, než bychom potřebovali. Tak obě strany Bézoutovy identity vynásobíme 8:

$$6 \cdot (-3) \cdot 8 + 20 \cdot 1 \cdot 8 = 2 \cdot 8.$$

Řešením naší rovnice tedy je $x = -24$, $y = 8$.

Tak hurá do obchodu nakoupit -24 balíčků vajec. Cože? Že záporné nemají? Nevadí – stačí si vzpomenout, že jsme původně počítali modulo 20, takže k x můžeme přičíst libovolný násobek 20 a dostaneme další řešení. Můžeme tedy jít třeba pro 16 balíčků.[4]

Teď už můžeme zformulovat obecný *návod na řešení kongruence*

$$ax \equiv b \pmod{n}$$

s neznámou x. Kongruenci přepíšeme do tvaru

$$ax - ny = b$$

a označíme $d = \mathrm{nsd}(a, n)$. Rozlišíme 3 případy:

- $d = b \ldots$ tehdy jsou hledaná x a y rovná Bézoutovým koeficientům a najdeme je rozšířeným Euklidovým algoritmem.
- $d \setminus b \ldots$ pak najdeme řešení x' a y' rovnice s d na pravé straně a položíme $x = x' \cdot b/d$ a $y = y' \cdot b/d$.
- b není násobkem $d \ldots$ v tomto případě kongruence nemůže mít žádné řešení, neboť levá strana rovnice je pro každé x a y dělitelná d, zatímco pravá strana dělitelná d nikdy není.

Inverzní prvky modulo m

Vraťme se teď zpátky z dlouhé odbočky a zkusme se znovu zamyslet nad tím, kdy je vynásobení obou stran kongruence ve tvaru

$$x \equiv z \pmod{m}$$

konstantou k ekvivalentní úprava. Tak říkáme úpravě, která neubírá ani nepřidává řešení. Už máme dokázáno, že když $x \equiv z$, tak pro každé k platí i $k \cdot x \equiv k \cdot z$. Takže zbývá zajistit, aby každé řešení kongruence $k \cdot x \equiv k \cdot z$ bylo i řešením $x \equiv z$.

Nejprve ukážeme, že pokud k je soudělné s m, je naše snaha předem ztracená. Označme $d = \mathrm{nsd}(k, m) > 1$. Vezměme libovolnou dvojici x a z splňující kongruenci $x \equiv z$, což je totéž jako $x - z \equiv 0$. Nyní vytvořme novou dvojici $x' = x$ a $z' = z + m/d$. Pro tu dostaneme

$$x' - z' \equiv x - (z + m/d) \equiv x - z - m/d \equiv -m/d \not\equiv 0.$$

Ovšem kongruenci vynásobenou k tato nová dvojice stále splňuje:

$$kx' - kz' \equiv kx - k(z + m/d) \equiv kx - kz - km/d \equiv$$
$$\equiv -km/d \equiv m \cdot (-k/d) \equiv 0.$$

Dokázali jsme tedy, že pokud číslo k, kterým násobíme obě strany kongruence, je soudělné s modulem m, nejedná se o ekvivalentní úpravu. Teď naopak ukážeme, že jsou-li k a m nesoudělná, ekvivalentní to je.

Nahlédneme, že kdykoliv $k \perp m$, existuje nějaké číslo $k^{-1} \in \mathbb{Z}_m$ takové, že $k \cdot k^{-1} \equiv 1$. Tomuto číslu se říká *inverzní prvek ke k* (nebo také *multiplikativní invers čísla k*) a pokud jím kongruenci $kx \equiv kz$ vynásobíme, získáme

$$k \cdot k^{-1} \cdot x \equiv k \cdot k^{-1} \cdot z \pmod{m},$$

což je kýžená kongruence $x \equiv z$.

Kongruenci $k \cdot k^{-1} \equiv 1$ přitom už umíme vyřešit – předchozí kapitola nám říká, že takové k^{-1} existuje právě tehdy, je-li $k \perp m$, a že se dá najít Euklidovým algoritmem. Dodejme ještě, že prvkům, které mají multiplikativní invers, se říká *invertibilní prvky modulo m*.

Konečná tělesa

Když speciálně zvolíme za m nějaké prvočíslo, budou všechny prvky $\mathbb{Z}_m$ kromě nuly invertibilní. Tím pádem se $\mathbb{Z}_m$ bude chovat dost podobně racionálním nebo reálným číslům. Má s nimi například tyto společné vlastnosti (sčítáním a násobením v případě $\mathbb{Z}_m$ myslíme operace modulo m):

- *Sčítání* je asociativní a komutativní.
- Pro každé a platí $a + 0 = a$.
- Pro každé a existuje $(-a)$ takové, že $a + (-a) = 0$.
- *Násobení* je asociativní a komutativní.
- Pro každé a je $a \cdot 1 = a$.
- Pro každé nenulové a existuje a^{-1} takové, že $a \cdot a^{-1} = 1$.
- Násobení a sčítání jsou distributivní: $a \cdot (b - c) = a \cdot b - a \cdot c$.

Obecněji, máme-li libovolnou množinu, můžeme v ní označit „jedničku“ a „nulu“ a „přibalit“ operace sčítání, násobení, „dej mi $(-a)$“ a „dej mi a^{-1}“. Operací přitom myslíme libovolnou funkci, která prvkům množiny nebo jejich dvojicím přiřazuje prvky. Pokud navíc pro naši množinu s operacemi platí všechny vyjmenované vlastnosti, říká se jí *komutativní těleso*.

Racionální, reálná i komplexní čísla jsou příklady takových těles a my jsme k nim přidali *konečná tělesa* velikosti prvočísla. (Na okraj poznamenejme, že je známo, že všechna konečná tělesa mají velikost mocniny prvočísla, což ovšem

[4] Mimochodem, není to nejmenší počet balíčků, který vyhovuje úloze: 6 balíčků by také fungovalo. Rozmyslete si, jak najít *nejmenší* řešení kongruence.

neznamená, že se vždy chovají jako celá čísla modulo nějakým m.)

Malá Fermatova věta

S prvočísly úzce souvisí takzvaná *Malá Fermatova věta.* Říká, že pokud je p prvočíslo a a libovolné číslo od 1 do $p-1$, tak

$$a^{p-1} \equiv 1 \pmod{p}.$$

Tato věta má mnoho různých použití (třeba ve známém šifrovacím algoritmu RSA nebo níže v algoritmu na testování prvočíselnosti), nám se bude především hodit jako další způsob invertování čísel modulo prvočíslo:

$$a^{p-2} \cdot a \equiv a^{p-1} \equiv 1 \pmod{p},$$

takže a^{p-2} je inverzní prvek k a.

Pojďme nyní Malou Fermatovu větu dokázat. Indukcí podle a budeme dokazovat ekvivalentní tvrzení

$$a^p \equiv a \pmod{p}.$$

(Jelikož a je určitě nesoudělné s modulem p, tak už víme, že násobení obou stran kongruence je ekvivalentní úprava.)

Pro $a = 1$ je snadné vidět, že věta platí:

$$a^p = 1^p = 1 \equiv 1 \pmod{p}.$$

Teď uděláme indukční krok. Řekněme, že máme dokázáno, že naše věta platí pro nějaké a, a chceme ji dokázat i pro $a+1$. K tomu se bude hodit známá Binomická věta, která říká, že pro každé reálné x a y a přirozené n platí:

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i},$$

přičemž $\binom{n}{i}$ je takzvané *kombinační číslo* tvaru

$$\binom{n}{i} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-i+1)}{i \cdot (i-1) \cdot \ldots \cdot 1},$$

mající v čitateli i jmenovali zlomku právě i členů.

Indukce po nás chce, abychom dokázali, že $(a+1)^p \equiv a$. Rozepíšeme tedy levou stranu kongruence pomocí Binomické věty:

$$(a+1)^p = \binom{p}{0} a^0 + \binom{p}{1} a^1 + \ldots + \binom{p}{p} a^p.$$

Jelikož $\binom{p}{0}$ i $\binom{p}{p}$ jsou rovny 1, tvoří první a poslední člen součtu dohromady $a^p + 1$. To je podle indukčního předpokladu kongruentní s a.

Zbývá tedy dokázat, že všechny ostatní členy jsou dělitelné p, takže se v kongruenci modulo p neprojeví. Vskutku: pro $0 < i < p$ se ve zlomku definujícím $\binom{p}{i}$ objeví p v prvočíselném rozkladu čitatele, ale ne v rozkladu jmenovatele, takže se nemá s čím zkrátit.

Tím je indukce hotova.

Fermatův test prvočíselnosti

Jako malou odměnu za dlouhý důkaz předvedeme, jak Malou Fermatovu větu využívat ke zjištění, zda je nějaké obrovské číslo n prvočíslem. Jistě bychom mohli zkoušet všechny kandidáty na dělitele od 2 do $n-1$ (nebo chytřeji do $\lfloor \sqrt{n} \rfloor$), ale to by trvalo příliš dlouho.

Raději zkusíme vybrat nějaké náhodné $a \in \{1, \ldots, n-1\}$ a spočítat, kolik je $a^{n-1} \bmod n$. Pro prvočíselné n musí vyjít jednička, takže pokud vyjde něco jiného, usvědčili jsme n z toho, že není prvočíslem (aniž jsme našli jediného dělitele – zvláštní, že?).

Pokud pro toto konkrétní a jednička vyjde, samozřejmě to neznamená, že n je určitě prvočíslo. Vyzkoušíme proto několik různých a a pokud test pro žádné z nich neselže, drze prohlásíme, že n je pravděpodobně prvočíslo.

Jak moc velká drzost to je? Překvapivě ne moc velká. Pro skoro každé složené číslo n platí, že alespoň polovina a-ček dosvědčí, že se nejedná o prvočíslo. Takže jeden pokus selže s pravděpodobností nejvýše $1/2$ a pokud uděláme t pokusů, pravděpodobnost chybného výsledku je nanejvýš $1/2^t$.

Jedinou výjimku z našeho pravidla tvoří takzvaná *Carmichaelova čísla* (nejmenší z nich je číslo 561). To jsou čísla, jejichž složenost prokážeme jen tehdy, když se strefíme do a soudělného s n, a takových a je velmi málo. Naštěstí není Carmichaelových čísel moc (relativně k prvočíslům), takže Fermatův test funguje docela spolehlivě.

Existují i důmyslnější testy, které se Carmichaelovými čísly obalamutit nenechají. Jejich popis, jakož i důkaz našeho tvrzení o spolehlivosti Fermatova testu, najdete v literatuře zmíněné na konci kuchařky.

Rychlé mocnění

Ve Fermatově testu nebo při počítání inverzí pomocí Malé Fermatovy věty potřebujeme spočítat $a^k \bmod m$ pro velké k. Pokud budeme mocninu a^k počítat přímo podle definice, tedy jako $a \cdot a \cdot \ldots \cdot a$, budeme potřebovat $\mathcal{O}(k)$ násobení, což je příliš.

Jednoduchou fintou lze počet operací snížit na $\mathcal{O}(\log k)$. Například a^{16} můžeme spočítat jako:

$$a^{16} = (a^8)^2 = ((a^4)^2)^2 = (((a^2)^2)^2)^2.$$

Pro obecný exponent bude rychlejší umocňování vypadat takto:

```
def FastExp(a, k):
    # Nejdříve ošetříme triviální případy.
    if k == 0: return 1
    if k == 1: return a

    # Když je x sudé, vrátíme a^(k/2) * a^(k/2).
    # Když je x liché, vrátíme a * a^(k - 1).

    if k % 2 == 0:
        i = FastExp(a, k / 2)
        return i * i
    else:
        return a * FastExp(a, k - 1)
```

Každé volání `FastExp` pro sudé k jednou zavolá `FastExp` s poloviční k a jednou vynásobí dvě čísla. Když je k liché, převede se na sudé a provede se jedno vynásobení. `FastExp` tedy provede $\mathcal{O}(\log k)$ násobení.

Je ale důležité uvědomit si, že kdybychom si neuložili výsledek $a^{k/2}$ do pomocné proměnné `i`, ale rovnou vraceli `FastExp(a, k / 2) * FastExp(a, k / 2)`, byl by náš kód stejně pomalý, jako kdybychom počítali mocninu podle definice!

Pro použití ve Fermatově testu (nebo obecně na spočítání mocniny $a^k \bmod m$) stačí po každém násobení výsledek vymodulit m.

Síto na prvočísla

Už jsme zjistili, že se nám hodí umět najít prvočísla. Kde je ale vezmeme? Můžeme určitě zkoušet jedno číslo po druhém a pokaždé otestovat, jestli držíme prvočíslo (třeba Fermatovým testem nebo zkoušením všech dělitelů). Už staří Řekové ale znali algoritmus, který najde všechna prvočísla menší než n efektivněji. Říká se mu *Eratosthenovo síto.*

Síto funguje na docela jednoduchém principu. Budeme si uchovávat v paměti pro každé číslo od 2 do n příznak, jestli je prvočíslo, nebo složené. Začneme u dvojky a označíme všechny násobky 2 ležící mezi 4 a n jako složená čísla. Další prvočíslo je 3. Označíme všechny násobky 3 ležící od 6 do n jako složená čísla. Další číslo na řadě je 4. Když jsme ale vyškrtávali násobky 2, vyškrtli jsme i 4. Nebudeme tedy provádět nic a rovnou přejdeme na 5. Takto najdeme všechna prvočísla od 2 do n a stihneme to rychle.

Ukažme si ještě zdrojový kód v Pythonu:

```
def Eratosthenes(n):
    prvocislo = [ True ] * n

    for i in range(2, n):
        if prvocislo[i]:
            print("%d je prvocislo." % i)

            # Násobky prvočísla jsou složené
            j = i * 2
            while j < n:
                prvocislo[j] = False
            j += i
```

Jak dlouho síto poběží? Dá se dokázat, že jeho časová složitost činí $\mathcal{O}(n \log\log n)$, ale není to snadné. My si zde předvedeme jenom slabší odhad $\mathcal{O}(n \log n)$. Zájemce o těžší důkaz menší složitosti odkazujeme na vzorové řešení úlohy 24-3-5.[5]

Síto tráví čas $\mathcal{O}(n)$ hledáním prvočísel a mezitím škrtá jejich násobky. Když škrtáme násobky dvojky, vyškrtneme nejvýše $n/2$ čísel, když škrtáme násobky trojky, vyškrtneme jich nejvýše $n/3$, atd. Složitost Eratosthenova síta tedy bude shora omezena součtem

$$n + \frac{n}{2} + \frac{n}{3} + \ldots + \frac{n}{n} = n \cdot \sum_{i=1}^{n} \frac{1}{i}.$$

Sumě

$$H_n = \sum_{i=1}^{n} \frac{1}{i}$$

se říká n-té harmonické číslo a dokážeme o něm, že leží v $\mathcal{O}(\log n)$.

Uvažujme, o co se zvětší H_{2n} oproti H_n:

$$H_{2n} - H_n = \frac{1}{n+1} + \frac{1}{n+2} + \ldots + \frac{1}{2n}.$$

To je součet n členů, z nichž každý je menší než $1/n$. Celý součet je tedy menší než 1.

Zjistili jsme tedy, že $H_{2n} < H_n + 1$. Funkce, které rostou takhle pomalu, jdou shora omezit nějakým logaritmem n, takže $H_n = \mathcal{O}(\log n)$.

Proto složitost celého Eratosthenova síta činí $\mathcal{O}(n \log n)$.

Čínská zbytková věta

Následující věta dostala své jméno po staročínském způsobu počítání vojáků. Čínská armáda je velká, a kdybychom chtěli počítat vojáky jednoho po druhém, trvalo by to dlouho. Pomáhalo prý armádu rozřadit do řad o velikostech m_1, m_2, …, m_n (součin všech m_i si označíme jako M bez indexu). Někdy zbyli nezařazení dva, někdy třicet, někdy se seřadili všichni. Tyto zbytky si označíme z_1, z_2, …, z_n.

A co Čínská zbytková věta říká? Tvrdí, že když jsou všechna m_i navzájem nesoudělná a počet vojáků je menší než M, lze ho ze zbytků z_i jednoznačně určit. Když například rozdělujeme vojáky do řad velikostí 2, 3, 5, 7, 11 a 13, můžeme zbytky z řad jednoznačně vyjádřit každý počet vojáků menší než $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 = 30\,030$.

Formálněji řečeno: Jsou-li dána navzájem nesoudělná přirozená čísla $m_1, \ldots, m_n$ (jejichž součin označíme M) a zbytky $z_1, \ldots, z_n$, pak existuje právě jedno číslo $x \in \mathbb{Z}_M$ takové, že pro všechna i je

$$x \equiv z_i \pmod{m_i}.$$

A jak dokážeme, že něco takového platí? Mějme 2 čísla $a, b \in \mathbb{Z}_M$ taková, že mají stejné zbytky po dělení všemi m_i. Ukážeme, že musí nutně být stejná.

Víme, že pro všechna i platí

$$a \equiv b \pmod{m_i}.$$

To podle definice kongruence znamená, že rozdíl $a - b$ je dělitelný všemi m_i. Proto je dělitelný i nejmenším společným násobkem všech m_i, což ovšem díky nesoudělnosti musí být jejich součin M.

Máme tedy dvě čísla ze $\mathbb{Z}_M$, jejichž rozdíl je dělitelný M. To nutně znamená, že jsou stejná.

Dokázali jsme tedy, že jedna sada zbytků $z_1, \ldots, z_n$ odpovídá jednoznačně určenému číslu $x \in \mathbb{Z}_M$, ale ještě nevíme, jak bez zkoušení všech možností toto x najít. Půjdeme na to od lesa.

Nejprve se hodí všimnout si toho, že když sečteme dvě čísla, sečtou se i jejich zbytky modulo všemi m_i.

Co kdybychom nyní dokázali sehnat čísla $Q_1, \ldots, Q_n$ taková, že Q_j je dělitelné všemi m_i kromě m_j a že $Q_j \equiv 1 \pmod{m_j}$?

To by potom stačilo položit

$$x = (z_1 \cdot Q_1 + z_2 \cdot Q_2 + \ldots + z_n \cdot Q_n) \bmod M.$$

Vskutku: počítáme-li $x \bmod m_i$, všechny členy $z_j \cdot Q_j$ pro $j \neq i$ vyjdou nulové a člen $z_i \cdot Q_i$ bude roven z_i. To, že celý výsledek nakonec vymodulíme M, na věci nic nemění, protože přičtení či odečtení libovolného násobku M zbytek po dělení žádným m_i neovlivní.

Jak se ale k číslům Q_i dostaneme? Číslo Q_i má být dělitelné všemi m_j kromě m_i. Uvažujme tedy součin

$$S_i = m_1 \cdot \ldots \cdot m_{i-1} \cdot m_{i+1} \cdot \ldots \cdot m_n.$$

Ten modulo každé m_j ($j \neq i$) dá nulu, zatímco modulo m_i nějaké číslo r_i nesoudělné s m_i (nesoudělné musí být, protože jinak by m_i bylo soudělné s některým m_j). Speciálně to znamená, že r_i není 0.

Potřebujeme tedy z tohoto nenulového zbytku udělat jedničku. To zařídíme snadno: pořídíme si r_i^{-1}, což bude inverzní prvek k r_i modulo m_i, a tímto prvkem celé S_i vynásobíme:

$$Q_i = S_i \cdot r_i^{-1}.$$

[5] `http://ksp.mff.cuni.cz/viz/24-3-5/reseni`

Toto Q_i už má požadované vlastnosti: $Q_i \bmod m_j$ pro $j \neq i$ vyjde nulové, protože Q_i je násobkem S_i, které bylo dělitelné m_j. A modulo m_i získáme

$$Q_i \equiv S_i \cdot r_i^{-1} \equiv r_i \cdot r_i^{-1} \equiv 1.$$

„Kouzelná“ čísla Q_i tedy dokážeme sestrojit a jejich zkombinováním i hledané x.

Pojďme si to teď zkusit v praxi. Chceme najít nejmenší x takové, že platí následující kongruence:

$$\begin{aligned} x &\equiv 3 \pmod 5 \\ x &\equiv 1 \pmod 9 \\ x &\equiv 14 \pmod{16} \end{aligned}$$

Spočítáme si nejdříve M a všechna S_i:

$$\begin{aligned} M &= 5 \cdot 9 \cdot 16 = 720, \\ S_1 &= 9 \cdot 16 = 144, \\ S_2 &= 5 \cdot 16 = 80, \\ S_3 &= 5 \cdot 9 = 45. \end{aligned}$$

Teď zjistíme, kolik vychází každé S_i modulo m_i a určíme příslušné multiplikativní inverze (například pomocí rozšířeného Euklidova algoritmu):

$$\begin{aligned} r_1 &= 144 \bmod 5 = 4, \\ r_2 &= 80 \bmod 9 = 8, \\ r_3 &= 45 \bmod 16 = 13, \\ r_1^{-1} &= 4 \quad (4 \cdot 4 \bmod 5 = 1), \\ r_2^{-1} &= 8 \quad (8 \cdot 8 \bmod 9 = 1), \\ r_3^{-1} &= 5 \quad (13 \cdot 5 \bmod 16 = 1). \end{aligned}$$

Z toho vypočteme Q_i jako $S_i \cdot r_i^{-1}$:

$$\begin{aligned} Q_1 &= S_1 \cdot r_1^{-1} = 144 \cdot 4 = 576, \\ Q_2 &= S_2 \cdot r_2^{-1} = 80 \cdot 8 = 640, \\ Q_3 &= S_3 \cdot r_3^{-1} = 45 \cdot 5 = 225. \end{aligned}$$

Nakonec sečteme příslušné násobky Q_i a zjistíme x:

$$x \equiv 3 \cdot 576 + 1 \cdot 640 + 14 \cdot 225 = 5518 \equiv 478 \pmod{720}.$$

Výsledek opravdu vypadá správně:

$$x = 478 = 3 + (5 \cdot 95) = 1 + (9 \cdot 53) = 14 + (16 \cdot 29).$$

Pár slov na závěr

Doufáme, že se vám naše povídání o teorii čísel líbilo a že jste poznali, že i tak základní objekty, jako jsou celá čísla, mají spousty zajímavých vlastností.

Přejete-li si dozvědět se více o prvočíselných testech nebo o RSA, můžeme navrhnout ke studiu textík *Algoritmy okolo teorie čísel*[6] od jednoho z autorů kuchařky. Důkladný rozbor Eratosthenova síta a jiné zajímavosti o prvočíslech najdete v článku *Tři věty o prvočíslech*[7] od téhož autora.

S teorií čísel také souvisí algebra, která zobecňuje různé poznatky na libovolné množiny opatřené nějakými operacemi (například tělesa). Máte-li o ni zájem, mohla by vám pomoci například skripta *Základy algebry* od Davida Stanovského.

Kuchařku pro vás namíchali

Michal Pokorný a Martin Mareš

[6] `http://mj.ucw.cz/papers/numth.pdf`
[7] `http://mj.ucw.cz/papers/bert.pdf`

Čokolády

Oproti první sérii, ve které získali čokoládu jen 4 řešitelé, se v této sérii roztrhl s čokoládami pytel. Pro tuto sérii jsme čokoládu rozdávali za vyřešení některé ze tří nejvíce bodovaných úloh na plný počet bodů. Tedy buď za *25-2-1 Vytíženost dopravy*, *25-2-4 Organizace vykládky* (CodExová úloha), nebo za *25-2-7 Zaléváme dokument* (seriál).

Jen za CodEx získalo čokoládu 10 lidí. Celkem pak čokoládu dostává 11 z vás. Gratulujeme a užijte si sladkou odměnu.

25-2-1 Vytíženost dopravy

Na tuto úlohu přišla spousta vašich řešení. Jedním z největších problémů některých z vás se ale ukázalo být to, jak správně rozdělit čas mezi předvýpočet a mezi odpovědi na dotazy. Zaveďme značení N pro počet zastávek (vrcholů našeho stromu) a K pro počet dotazů.

Správné rozdělení času

Zamysleme se nejdříve nad dvěma extrémy: Pokud bychom očekávali malý (konstantní) počet dotazů, bylo by asi nejlepší odpověď pro každý dotaz vyhledat samostatně (třeba jednoduchým procházením do šířky – BFS) a zabralo by nám to čas $\mathcal{O}(N)$ na dotaz a $\mathcal{O}(N)$ celkem. Druhým extrémem by bylo $K \geq N^2$. V takovém případě si můžeme předvypočítat pomocí BFS cesty z každé zastávky na každou v $\mathcal{O}(N^2)$ a odpovídat pak už jen v konstantním čase na dotaz. Tím se dostaneme na celkovou složitost $\mathcal{O}(N^2 + K)$.

My jsme se ale zabývali nejzajímavějším případem, a to když K je řádově stejně velké jako N. Pak je druhý postup příliš pomalý. Ukážeme si, jak udělat předvýpočet v čase $\mathcal{O}(N \log N)$ a odpověď na dotaz v čase $\mathcal{O}(\log N)$.

Lehčí varianta

Nejdříve se zamyslíme nad lehčí variantou s pouhou cestou. To je jako situace přímo stavěná pro maximové intervalové stromy. Intervalový strom je stromová struktura postavená nad nějakou posloupností, která je schopná vracet hodnotu (součet, maximum a podobně) v nějakém intervalu. Dělá to tak, že kořen drží hodnotu celé posloupnosti, jeho synové hodnotu levé a pravé poloviny a tak dále, až na úroveň jednotlivých prvků posloupnosti.

Každý interval jsme pak schopni poskládat z maximálně $\log N$ menších intervalů zastoupených vrcholy a dotaz na intervalový strom tedy trvá $\mathcal{O}(\log N)$, strom se dá vystavět v lineárním čase. Toť řešení jednodušší varianty. Pokud si chcete přečíst něco více, podívejte se do naší kuchařky o intervalových stromech.[8]

Složitější varianta

Nyní se pokusíme některé myšlenky intervalových stromů zobecnit, aby fungovaly nejenom na graf tvaru cesty. Strukturu ale nebudeme potřebovat aktualizovat, stačí nám ji pouze jednou vybudovat a pak nad ní pokládat dotazy. Tím se bude lišit od intervalových stromů, které umožňují rychle provádět i aktualizace.

Zakořeníme si naši grafovou síť zastávek v libovolném vrcholu a postavíme strom. Ve chvíli, kdy dostaneme dotaz na úsek A-B, můžeme ho složit (vzít maximum) z dotazů na úseky A-P a L-P, kde P je společný stromový předchůdce obou vrcholů (tedy nejvýše umístěný vrchol, přes který cesta z A do B musí jít). K rychlému hledání P se vrátíme později.

Tím jsme si problém zredukovali na nalezení maxima nějaké vertikální cesty ve stromě. To bychom mohli udělat tak, že bychom jí celou prošli, ale to může trvat až lineárně dlouho vzhledem k N (například v grafu tvaru cesty). My bychom ale chtěli dosáhnout času $\mathcal{O}(\log N)$. Zaveďme si tedy v každém vrcholu *zpětné odkazy* různých úrovní. Zpětný odkaz úrovně 0 bude odkaz na otce, zpětný odkaz úrovně 1 bude odkaz na otce otce (tedy o 2 výš) a obecně zpětný odkaz úrovně m povede o 2^m vrcholů výše. Takových zpětných odkazů bude v každém vrcholu maximálně $\log n$ a každý si navíc bude pamatovat maximum na úseku, který pokrývá.

Když budeme chtít vystoupit od A k P, dokážeme tento úsek pokrýt jen pomocí těchto zpětných odkazů a použijeme jich jen $\mathcal{O}(\log N)$ (jednoduchým argumentem: když budeme skákat po největším možném zpětném odkazu, tak každým skokem zmenšíme vzdálenost alespoň o polovinu). Obdobně úsek od B k P. Pokud tedy dostaneme takovouto strukturu, dokážeme odpovědět na libovolný dotaz v $\mathcal{O}(\log N)$.

Teď se vrátíme ke slibovanému hledání P. V každém vrcholu si budeme pamatovat, v jaké je hloubce, a budeme stoupat od A a B zároveň. Nejdříve vystoupáme po zpětných odkazech z toho hlubšího do stejné hloubky (v $\mathcal{O}(\log n)$ krocích) a pak zkoušíme stoupat z obou vrcholů naráz. Vezmeme postupně všechny délky zpětných odkazů (od největších po nejmenší) a když se přes takto dlouhé zpětné odkazy ještě nedostaneme do stejného vrcholu (mohl by to totiž být až nějaký předchůdce P), vystoupáme po nich. Tímto nalezneme snadno P a současně si nerozbijeme logaritmickou délku cest od A a od B k P.

Jak si takovou strukturu rychle pořídit? To už je jednoduché. Zakořenění stromu můžeme udělat pomocí prohledávání do hloubky od libovolného vrcholu, to nám zabere lineárně kroků. V každém vrcholu pak zkonstruujeme maximálně $\log N$ zpětných odkazů a s využitím předchozích odkazů nám každý nový odkaz zabere jen konstantní čas. Vzdálenost od A k B překlenutá zpětným odkazem délky 2^m je totiž překlenutá dvěma zpětnými odkazy: od A k C délky 2^{m-1} a od C k B délky také 2^{m-1}. Když vezmeme z těchto dvou odkazů maximum, máme maximální počet cestujících i na zpětném odkazu délky 2^m.

V každém vrcholu tedy strávíme $\mathcal{O}(\log N)$ a celou strukturu zvládneme vystavět v $\mathcal{O}(N \log N)$. Celkově předvýpočet i odpověď na K dotazů trvá $\mathcal{O}((N+K) \log N)$, což je ideální.

Program (C++):
`http://ksp.mff.cuni.cz/viz/25-2-1.cpp`

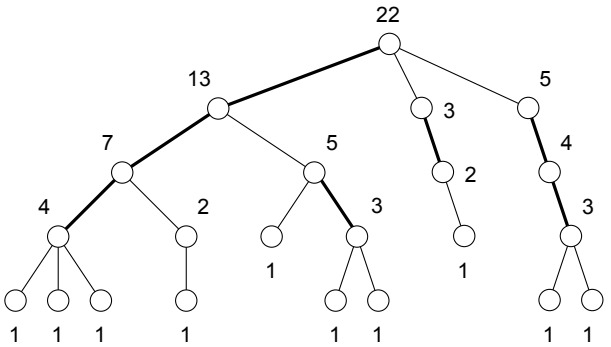
Jirka Setnička

[8] `http://ksp.mff.cuni.cz/viz/kucharky/intervalove-stromy`

Heavy-light dekompozice

Datová struktura z našeho vzorového řešení má jednu nevýhodu: pro strom na N vrcholech zabere řádově $N \log N$ buněk paměti. Načrtneme ještě jeden způsob, jak úlohu vyřešit, který si vystačí s lineární pamětí. Použijeme k tomu takzvanou *heavy-light dekompozici* stromu neboli rozklad stromu na lehké a těžké hrany.

Strom zakořeníme a pro každý vrchol v spočítáme $T(v)$, což bude počet vrcholů v podstromu, jehož kořenem je v. Za *těžké* prohlásíme ty hrany, které vedou z nějakého vrcholu v do jeho syna w, přičemž $T(w) > T(v)/2$. Všechny ostatní hrany budou *lehké*. Dobře je to vidět na následujícím obrázku (čísla udávají velikosti podstromů, těžké hrany jsou nakresleny tučně):



Povšimneme si, že z každého vrcholu může dolů vést nejvýše jedna těžká hrana. Těžké hrany proto tvoří cesty, kterým budeme říkat *těžké cesty* a jejich nejvyšším vrcholům *stopky cest*. Pokud stopka cesty není kořen, vede z ní nahoru lehká hrana, která ji napojí na nadřazenou těžkou cestu (ta ovšem může být triviální – jednovrcholová).

O lehkých hranách platí jiná zajímavá věc: kdykoliv se vydáme z kořene do listu, projdeme po nejvýše $\log_2 N$ lehkých hranách. To proto, že kdykoliv projdeme po lehké hraně, velikost podstromu, v němž se nacházíme, klesne alespoň na polovinu.

Teď popíšeme, jak pomocí naší dekompozice hledat nejbližšího společného předchůdce dvou vrcholů. Stačí si pro každou těžkou cestu zapamatovat pole všech vrcholů, které na ní leží, a naopak si pro každý vrchol zapamatovat, na jaké těžké cestě leží a kolikátý v pořadí je.

Když nám nyní někdo zadá vrcholy x a y, půjdeme z nich do kořene a podíváme se, kde se obě cesty poprvé potkaly. Do kořene přitom vyskáčeme tak, že pokaždé určíme stopku těžké cesty, na níž se nacházíme, a z té vystoupíme po jedné lehké hraně. To může nastat nejvýše $\mathcal{O}(\log N)$-krát a pokaždé nás to stojí konstantní čas.

Podobně zvládneme hledání maxim na cestách. Pro každou těžkou cestu postavíme intervalový strom, pomocí kterého budeme umět rychle nalézt maximum v libovolném úseku cesty.

Kdykoli nám pak někdo zadá cestu z x do y, rozdělíme ji na úsek z x nahoru do nejbližšího společného předchůdce a úsek z něj dolů do y. Stačí tedy umět počítat maxima pro „svislé“ cesty ve stromu. Každá svislá cesta ovšem obsahuje $\mathcal{O}(\log N)$ lehkých hran; těmi jsou spojeny úseky těžkých cest, takže úseků musí být také $\mathcal{O}(\log N)$.

Pro každou těžkou cestu se zeptáme příslušného intervalového stromu, jaké je maximum z příslušného úseku, a vypočteme maximum z těchto maxim a z ohodnocení lehkých hran spojujících těžké úseky. To dává celkem $\mathcal{O}(\log N)$ dotazů na intervalové stromy, z nichž každý trvá $\mathcal{O}(\log N)$. Dohromady $\mathcal{O}(\log^2 N)$, což je příliš.

Učiníme tedy ještě jedno pozorování: skoro všechny dotazy, které intervalovým stromům klademe, se týkají intervalů od nějakého vrcholu těžké cesty k její stopce. Jedinou výjimku tvoří nejvyšší cesta, na kterou se zeptáme. Můžeme si tedy navíc pro každou cestu předpočítat maxima úseků od stopky do ostatních vrcholů. Pak položíme $\mathcal{O}(\log N)$ dotazů trvajících $\mathcal{O}(1)$ a jeden trvající $\mathcal{O}(\log N)$. To je dohromady $\mathcal{O}(\log N)$ na celé nalezení maxima cesty z x do y.

Celá struktura nám přitom zabere $\mathcal{O}(N)$ buněk paměti a jsme ji schopni vystavět v lineárním čase.

Dodejme ještě, že trochu složitější struktury tohoto druhu jde i aktualizovat. To si ale necháme na jindy; pokud jste zvědavi, zkuste si najít něco o Sleatorových-Tarjanových stromech, známých také pod názvem Link-Cut Trees.

Martin „Medvěd“ Mareš

25-2-2 Sekání trávy podruhé

Řešení této úlohy je krátké, jednoduché, ale je třeba uznat, že i nemálo trikové. Pro úplnost zadání budeme předpokládat, že startovní políčko již je posekané. Pak má vyhrávající strategii první hráč. A jaká tedy bude jeho strategie?

Hrací plocha se skládá ze sjednocení obdélníků se sudým obsahem. Tedy alespoň jeden z rozměrů každého obdélníku je sudý. Tedy je možné celý herní plán vyskládat dominovými kostkami. Jak se to přesně udělá, si každý jednoduše rozmyslí.

Nyní k samotné strategii. Hráč jedna začíná na nějaké poloposekané dominové kostce. Tak jediné, co udělá, je, že přejde do druhé části této kostky. Nyní druhý hráč buď už nemůže nikam táhnout, nebo přejde do jiné dominové kostky. Tato kostka zatím nebyla použita, a tedy má volnou druhou půlku. Takže první hráč zas jen přejde do druhé poloviny. Tuto strategii bude opakovat až do doby, kdy druhý hráč nebude mít kam táhnout.

Na závěr ještě poznamenejme, že obecně se této taktice říká *Párovací strategie* a funguje ve všech grafech, které mají perfektní párování (tj. vrcholy grafu lze rozdělit do dvojic, kde každá dvojice je spojena hranou).

Karel Tesař

25-2-3 Doplňování operátorů

Túto úlohu sme pôvodne zamýšľali ako jednoduchú, teda takú, že jej riešenie je priamočiare a jasné. Nevšimli sme si však značné množstvo slepých uličiek, ktoré vás zmiatli. Úlohu jsme zadávali s tým, že pôjde jednoducho riešiť v lineárnom čase, čo sa nakoniec ukázalo ako zlý predpoklad. Za to sa vám ospravedlňujeme a i kvôli tomu sme bodovali zlé riešenia miernejšie.

Drvivá väčšina riešiteľov sa úlohu pokúšala riešiť hladovo, čo ale nefungovalo. Ďalšia vec je, že skoro všetky riešenia, ktoré prišli, boli zle popísané a často sa stávalo, že si opravujúci musel toho dosť veľa domýšľať. Navyše väčšina z vás zabúdala uvádzať časovú zložitosť.

Najčastejšími protipríkladmi na hladové riešenie boli postupnosti, v ktorých sa vyskytovalo viac jednotiek vedľa seba – napríklad v postupnosti, ktorá je tvorená dvojkou, desiatimi jednotkami a opäť dvojkou, je lepšie sčítať. Na podobných protipríkladoch zlyhali všetky pokusy o lineárne hladové riešenie.

Ukážeme si ako riešiť úlohu v kvadratickom čase pomocou dynamického programovania.[9] Dynamické programovanie je veľmi užitočná programovacia technika, ktorá spočíva v rozdelení problému na nejaké menšie podproblémy, ktoré vieme vyriešiť. Z riešení pre menšie podproblémy potom poskladáme finálne riešenie. Ukážeme si to na našej úlohe.

Predstavme si, že na vstupe dostaneme n čísel, označme ich $a_1, \ldots, a_n$. Uvážme teraz každé $i \in \{2, \ldots, n-1\}$. Predpokladajme, že vieme doplniť operátory medzi $a_1, \ldots, a_i$, tak že po vyhodnotení $a_1\ a_2\ \ldots\ a_i$ dostaneme najlepší výsledok. Ako zistiť, aký najlepší výsledok môžeme dosiahnúť, keď uvažujeme $a_1\ a_2\ \ldots\ a_n$?

Zvoľme nejaké $j \in \{2, \ldots, n-1\}$. Z predchádzajúceho odstavca vieme, že sme schopní optimálne doplniť operátory medzi $a_1\ a_2\ \ldots\ a_j$, označme výsledok v_j. Jeden z možných výsledkov pre $a_1\ a_2\ \ldots\ a_n$ je $v_j + a_{j+1} \cdot \ldots \cdot a_n$, označme ho V_j. Najlepší výsledok dostaneme tak, že vezmeme maximum zo všetkých V_j, pre $j \in \{2, \ldots, n-1\}$.

Čo teda vlastne robíme? Všimnime si, že sme predpokladali, že problém vieme vyriešiť pre všetky i také, že $i < n$ a z toho sme zistili riešenie pre n. Pre úplnosť dodajme ešte, že pre a_1 sme schopní úlohu vyriešiť triviálne, tam žiadne operátory dopĺňať netreba. To znamená, že so znalosťou optimálneho riešenia pre a_1 sme schopní aplikovaním vyššie uvedeného postupu získať optimálne riešenie pre $a_1\ a_2$, ďalej so znalosťou optimálneho riešenia pre a_1 a $a_1\ a_2$ sme schopní aplikovaním rovnakého postupu získať optimálne riešenie pre $a_1\ a_2\ a_3$ atď.

Predchádzajúci odstavec celkom jasne popisuje, ako bude algoritmus fungovať. Jeho časová zložitosť bude $\Theta(n^2)$. Je tomu tak preto, lebo potrebujeme spočítať najlepšie výsledky pre $a_1\ a_2\ \ldots\ a_i$, pre každé $i \in \{1, \ldots, n\}$. Pri počítaní každého takéhoto výsledku spravíme $\Theta(i)$ krokov. Celkový počet krokov je teda $\Theta(1) + \cdots + \Theta(n) = \Theta(n^2)$. Pamäťová zložitosť je $\Theta(n)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/25-2-3.py`

Peter Zeman

25-2-4 Organizace vykládky

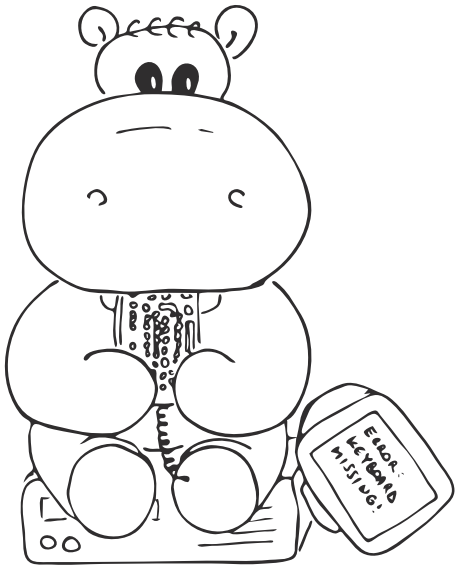
V této praktické úloze byly vstupy rozděleny do čtyř sad podle očekávané efektivity řešení.

Hledaný součet vzdáleností od daného bodu ke všem ostatním v maximové metrice nazveme cenou vykládky pro daný bod. Nejjednodušším řešením úlohy je přímý výpočet ceny vykládky pro každý bod tak, že projdeme všechny ostatní body a sečteme vzdálenosti podle vzorečku. Takové řešení má časovou složitost $\mathcal{O}(N^2)$ a stačilo pro vyřešení prvních dvou sad.

Ve druhé sadě se hodnoty nevlezly do 32-bitových proměnných, museli jste tedy použít 64-bitové celočíselné typy. Co jsem do úlohy nevkládal, ale doporučuji k promyšlení (a nejlépe i naprogramování), je případ, kdy by se čísla nevlezla ani do 64-bitových proměnných a váš jazyk by nepodporoval celočíselné typy s dynamickou délkou. A jak byste postupovali, pokud by čísla nebylo potřeba pouze sčítat a porovnávat, ale provádět i další operace jako například násobení a dělení?

Ve třetí sadě již byl počet zadaných bodů vysoký ($N \leq 100000$), ale počet navzájem různých bodů byl stále nízký ($K \leq 1000$). Taková situace se dala vyřešit seskupením shodných bodů do jediného bodu, u kterého byla udána jeho násobnost. Lze to provést třeba setříděním bodů (nejpřirozenější způsob je asi lexikografický – nejprve podle souřadnice x, v případě shody podle souřadnice y).

Setřídění vede k tomu, že se shodné body umístí v poli na souvislém úseku. Pak lze pole projít a pro každý bod zkontrolovat, zda je shodný s předchozím prvkem pole. V případě shody je zvýšena násobnost naposledy přidaného bodu, v opačném případě je přidán nový bod. Protože má (rozumné) třídění časovou složitost $\mathcal{O}(N \log N)$, je výsledná časová složitost $\mathcal{O}(N \log N + K^2)$. (Použitím vyhledávacích stromů lze docílit $\mathcal{O}(N \log K + K^2)$, což ale není příliš atraktivní vylepšení.)



Efektivní řešení

Problém dosavadního přístupu tkví v tom, že kvůli nepříjemné formuli pro vzdálenost mezi dvěma body nedokážeme cenu vykládky počítat nějak hromadně. Možností, se kterou jsem původně počítal, je zametat body podél y-ové souřadnice a vhodně si v intervalových stromech udržovat body

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

tak, aby se daly efektivně pro zpracovávaný bod spočítat 4 součty – součty souřadnic všech bodů, pro které se bude: přičítat a odečítat x-ová souřadnice a přičítat a odečítat y-ová.

Asymptoticky stejně efektivní, ale jednodušší je použít trik převzatý z řešení Ondry Hübsche. Trik spočívá ve využití vztahu

$$2\max(|a|,|b|) = |a-b| + |a+b|,$$

ověřte si jej například rozborem případů. Vzdálenost dvou bodů $d = \max(|x_1-x_2|, |y_1-y_2|)$ pak můžeme zapsat jako

$$2d = |(x_1-x_2)-(y_1-y_2)| + |(x_1-x_2)+(y_1-y_2)|.$$

Budeme pracovat s novými souřadnicemi u a v, které vzniknou jako:

$$u = x - y \qquad v = x + y$$

Vzdálenost dvou bodů nám pak bude vycházet jako

$$2d = |u_1-u_2| + |v_1-v_2|.$$

Celou dobu budeme počítat dvojnásobné vzdálenosti a pouze na konci výsledek vydělíme dvěma.

Jaká je výhoda tohoto převodu? Zbavili jsme se nepříjemné operace maxima. Součet už dokážeme počítat efektivně, provedeme to ve dvou krocích – oddělíme $|u_1-u_2|$ a $|v_1-v_2|$. V prvním kroku setřídíme body vzestupně podle souřadnice u a spočítáme si prefixové součty pro toto setříděné pole. Pro aktuálně zpracovávaný prvek p pole je pak $|u_p-u_i| = u_p-u_i$ pro všechny prvky i před ním a $|u_p-u_i| = u_i-u_p$ pro prvky za ním.

S prefixovými součty dokážeme započítat všechny body do ceny vykládky pro bod p v konstantním čase. (Pole P prefixových součtů je takové pole, které na i-té pozici obsahuje součet prvních i prvků. Dokážeme jej předpočítat v lineárním čase pomocí vztahu $P[i] = P[i-1] + Q[i]$, kde Q je původní pole. Součet prvků na pozicích k až l je pak $P[l] - P[k-1]$.)

Analogicky postupujeme pro souřadnici v. Nyní máme pro každý bod spočtenou cenu vykládky a stačí vybrat nejmenší z nich. Jedinou složitější operací je třídění, ostatní operace jsou pouhé lineární průchody. Časová složitost řešení je tak $\mathcal{O}(N \log N)$, paměťová $\mathcal{O}(N)$.

Program (C++) – N kvadratický:
`http://ksp.mff.cuni.cz/viz/25-2-4-Nkvadr.cpp`

Program (C++) – K kvadratický:
`http://ksp.mff.cuni.cz/viz/25-2-4-Kkvadr.cpp`

Program (C++) – $N \log N$:
`http://ksp.mff.cuni.cz/viz/25-2-4-NlogN.cpp`

Lukáš Folwarczný

25-2-5 Sbírání papírů

Než se vrhneme na řešení úlohy, učiňme několik stěžejních pozorování. Ta nám už řešení úlohy dají takřka zadarmo.

Novinářka se nesmí vracet dolů. Tím pádem před přechodem na další řádek (nahoru) musí vybrat všechny papíry na řádku.

Zároveň nemá smysl uvažovat jiné papíry než ten nejvíce vlevo a vpravo. Všechny ostatní papíry totiž novinářka sebere automaticky při pohybu mezi nimi.

Dále je důležité uvědomit si, že po sebrání posledního papíru nemá smysl se na tomto řádku dále pohybovat. Stejné kroky totiž lze provést o řádek výše s výsledkem přinejhorším stejným (na aktuálním řádku už žádný další papír nesebereme).

Musíme ještě vyřešit, jak se na řádku pohybovat. Označme si index papíru položeného nejvíce nalevo l a index toho nejvíce napravo r. Index políčka, na které vstupujeme při přechodu na řádek, označme c. Nyní nastávají tři možnosti.

$l \geq c$ Nemá smysl se pohybovat jinak než pořád doleva.
$r \leq c$ Nemá smysl se pohybovat jinak než pořád doprava.
$l < c < r$ Můžeme jít nejdříve k l a následně k r. Také je možné jít nejdříve k r a poté k l. Obě varianty mohou dát jinou délku cesty.

V tuto chvíli mnoho z řešitelů sáhlo po hladovém algoritmu. Tedy vždy se podívat, zda je výhodnější z c jít nejdříve do l a až poté do r, nebo opačně. Následně lepší z výsledků vzít jako součást řešení a pokračovat stejným způsobem na řádku výše. Tento postup je však chybný. Zkuste si ho například odsimulovat na následujícím vstupu:

```
3 8
Řádek 3: 0 1 0 0 0 0 0 0
Řádek 2: 0 1 0 0 0 0 0 1
Řádek 1: 0 0 1 0 0 0 0 0
```

Z toho plyne, že je třeba počítat s oběma variantami průchodu řádkem a obě si uložit. Všech možných cest je tak sice exponenciálně mnoho, lze ale využít přístupu z dynamického programování a ukládat si mezivýsledky.

Buďme v nějakém řádku i. V řádku $i-1$ jsme dostali dvě optimální řešení. Jedno pro l_{i-1} a jedno pro r_{i-1}. Pro spočítání řešení v l_i tedy vyzkoušíme oba z výsledků pro předchozí řádek a lepší z nich si uložíme. Totéž provedeme pro r_i, čímž dostaneme opět dvě možné varianty. V posledním řádku pak z těchto dvou variant vybereme tu s kratší délkou cesty.

Speciálním případem je první řádek, a to v tom, že musíme vždy dojít do r_1 a optimální řešení je zde tak pouze jedno (o délce r_1). V programu toto snadno ošetříme.

Že je řešení správné si lze snadno rozmyslet přes matematickou indukci. Řešení pro první řádek je indukčním předpokladem. Řešení pro i-tý řádek pak indukčním krokem.

Ještě však nekončíme. Jelikož chceme také zpětně rekonstruovat cestu, musíme si navíc ukládat pole předchůdců a směry pohybu. Pak lze již cestu rekonstruovat. Směr pohybu se navíc bude měnit pro řádek maximálně dvakrát, tedy paměťová složitost uložení cesty je $\mathcal{O}(N)$. Taková je i paměťová složitost celého algoritmu, jelikož není třeba ukládat si celou matici. Pozorní čtenáři si jistě dokážou rozmyslet proč.

Ke zjištění indexů l a r pro každý řádek stačí jednou matici v $\mathcal{O}(NM)$ projít. Následná dynamika nám zabere $\mathcal{O}(N)$ kroků, tedy celková časová složitost je lineární – $\mathcal{O}(NM)$.

Zdrojový kód je z větší části přepisem programu Rastislava Rabatina. Děkujeme.

Program (C++):
`http://ksp.mff.cuni.cz/viz/25-2-5.cpp`

Jan Bok

25-2-6 Optimalizace v redakci

Naším úkolem je pro každou hranu zadaného ohodnoceného grafu určit, zda se vyskytuje ve všech, v pouze některých, nebo v žádných minimálních kostrách. K řešení využijeme upravený Kruskalův algoritmus. Z kuchařky[10] budeme potřebovat především poznatek o jeho správnosti a také poznatek o tom, že v případě více hran stejné váhy může být pořadí zpracování hran libovolné (algoritmus může vydat různé kostry, ale všechny budou minimální).

Popis algoritmu

Algoritmus si bude, stejně jako ten původní, budovat les T – část výsledné kostry. Hrany stejné váhy w_k budeme zpracovávat najednou. Hranu, jejíž oba vrcholy leží v jedné komponentě, rovnou označíme za hranu, která se v žádné minimální kostře nevyskytuje, a dále s ní nepracujeme.

Pro ostatní hrany rozhodneme, zda se vyskytují ve všech, nebo jen některých minimálních kostrách. Ve skutečnosti zjistíme, zda Kruskalův algoritmus hranu (v závislosti na pořadí zpracování) přidá vždy, nebo jen někdy. Později dokážeme, že je obojí ekvivalentní.

Uvažme pro tento účel graf G, který vznikne tak, že do T vložíme všechny hrany váhy w_k. Pokud se odebráním hrany e (váhy w_k) z grafu G sníží počet komponent, přidal by algoritmus hranu ve všech případech, neb nemá jinou hranu téže váhy, jíž by spojil příslušné komponenty. Pokud se naopak počet komponent nezmění, existuje další hrana, kterou lze použít místo e.

Hrana, po jejímž odebrání vzroste počet komponent grafu, se nazývá *most*. Algoritmus na hledání mostů je detailně popsán v kuchařce o grafech,[11] zde si jej popíšeme pouze stručně. Upravíme algoritmus průchodu do hloubky (DFS) tak, aby u vrcholů počítal tzv. hladinu. Graf si zakoříníme, pro lepší představu umístíme kořen „dolů“ a přidělíme mu hladinu 0. Hladina pak ukazuje to, jak vysoko při průchodu vystoupáme, tedy počet hran na cestě od kořene při průchodu do hloubky.

Pro každý vrchol se určí nejnižší hladina, do které se lze z něj dostat při průchodu do hloubky (hrana, po které jsme do něj přišli je přirozeně zapovězena). Každý vrchol si tuto hodnotu spočte rekurzivně jako minimum z hladin vrcholů, do kterých se z něj lze dostat.

Nyní, když se v DFS stromu vracíme z vrcholu w zpátky do jeho otce v, nastávají dvě možnosti: z vrcholu w se lze dostat pouze do hladin vyšších než v, pak je hrana vw most, neb po odebrání se už z w do nižších hladin nijak nedostaneme. Pokud se lze dostat alespoň do hladiny vrcholu v, pak se o most nejedná, protože i po odebrání hrany vw bude existovat cesta z v do w a graf tak bude souvislý.

Časová složitost průchodu do hloubky je $\mathcal{O}(N + M)$, kde N je počet vrcholů a M počet hran. Počet skupin hran různé váhy označme jako ℓ a počty hran v těchto skupinách jako m_1 až m_ℓ. V jednom průchodu procházíme pro každou skupinu graf, ve kterém je nejvýše $N - 1 + m_i$ hran (kostra grafu na N vrcholech má $N - 1$ hran). Časová složitost je tak $\mathcal{O}(\ell \cdot N + \sum_i m_i)$, což je v nejhorším případě $\mathcal{O}(MN)$.

Pro vylepšení si uvědomíme, že vrcholy a hrany uvnitř už existujících komponent nás ve skutečnosti vůbec nezajímají. Zajímá nás pouze to, jak propojují aktuálně zpracovávané hrany. Graf proto budeme konstruovat tak, že jeho vrcholy budou tyto komponenty. Navíc komponenty, do kterých nevede žádná z aktuálně zpracovávaných hran, do grafu nezahrneme.

Takový graf už nemusí být klasickým grafem, ale mohou v něm existovat i násobné hrany (více hran mezi dvěma vrcholy, tzv. multihrany). To nevadí, uvědomíme si, že multihrany nemohou být mosty a nic jiného není ovlivněno. Takový graf bude mít m_i hran a nejvýše $2m_i$ vrcholů, kde m_i je počet hran i-té váhy.

Celková složitost hledání mostů bude $\mathcal{O}(m_1 + m_2 + \ldots + m_\ell) = \mathcal{O}(M)$. V Kruskalově algoritmu hrany na začátku setřídíme nějakým efektivním algoritmem, např. QuickSortem, a pak používáme strukturu Disjoint-Find-Union. To nám dává časovou složitost $\mathcal{O}(M \log M)$, což je i složitost celého algoritmu. Paměťová složitost je $\mathcal{O}(M)$.

Důkaz správnosti

Zatím jsme pro hrany nějaké váhy w_i rozhodli pouze to, zda se objeví v nějaké kostře za předpokladu, že už je určena částečná kostra T, která vznikla během Kruskalova algoritmu na všech hranách menší váhy. Dokážeme, že hrana, která spojuje 2 vrcholy stejné komponenty v průběhu našeho algoritmu, se neobjeví v žádné minimální kostře.

Předpokládejme pro spor, že existuje minimální kostra, ve které se vyskytuje tato hrana e. Odeberme tuto hranu – vzniknou dvě komponenty. Uvažme všechny hrany původního grafu, které vedou mezi komponentami (z vrcholu jedné komponenty do vrcholu druhé komponenty). V průběhu našeho algoritmu byly tyto komponenty spojeny lehčí hranou než e, tedy mezi vybranými hranami je tato hrana. Když ji přidáme místo e do kostry, vznikne nová kostra s menší váhou – to je spor s tím, že se jednalo o minimální kostru.

O ostatních hranách víme, že se vyskytují v alespoň jedné minimální kostře. Předpokládejme, že existuje minimální kostra, ve které se nevyskytuje hrana $e = uv$, o které jsme prohlásili, že se vyskytuje v každé kostře. Uvažme cyklus, který vznikne přidáním hrany e do této kostry.

Pokud mají všechny hrany na tomto cyklu menší váhu, pak bychom hranu e do kostry vůbec nepřidávali, místo toho bychom vrcholy u a v spojili těmito levnějšími hranami. Pokud se na cyklu vyskytuje ostře větší hrana, je možno ji za e vyměnit, tím ale vznikne lehčí kostra, což je spor.

Zbývá případ, kdy se na cyklu vyskytuje alespoň jedna hrana $f = wx$ stejné váhy. Pak se ale dvojice (u, w), (v, x) nebo (u, x), (v, w) dají spojit hranou váhy nejvýše stejné jako e a v algoritmu bychom vyhodnotili, že ani e ani f nejsou mosty, to je opět spor.

Pro zbývající hrany jsme přímo ukázali, jak sestrojit minimální kostru, kde se vyskytují, i minimální kostru, kde se nevyskytují. Důkaz je tímto hotov.

[10] `http://ksp.mff.cuni.cz/viz/kucharky/minimalni-kostra`

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

Závěrem si dovolím malou poznámku. Jak asi vidíte, tento důkaz je dosti nepěkný a nepřehledný. Kolem koster existuje zajímavá teorie, pomocí které lze tvrzení podobná tomu našemu dokazovat daleko příjemněji. Doporučuji nahlédnout do učebního textu Martina Mareše Krajinou grafových algoritmů.[12] Lze tam nalézt například i to, že pro zavedení minimální kostry vlastně vůbec nemusíme umět váhy hran sčítat – stačí je umět porovnávat. (Což lze možná i odtušit z toho, že Kruskalův algoritmus sčítání nevyužívá.)

Program (C++):
`http://ksp.mff.cuni.cz/viz/25-2-6.cpp`

Lukáš Folwarczný

Teorie minimálních koster

Z té zmíněné teorie minimálních koster si ostatně můžeme malý kousek ukázat. Uvažme nějakou minimální kostru T a hranu $e = uv$, která v této kostře neleží. Víme, že vrcholy u a v jsou v kostře T spojené nějakou cestou $T[u, v]$. Označme f nejtěžší hranu této cesty.

Rozlišíme tři případy:

- f je těžší než e – tento případ nemůže nastat. Tehdy bychom totiž mohli z kostry T odebrat hranu f a vzniklé dvě komponenty spojit přidáním hrany e. Tím bychom získali lehčí kostru, než byla minimální kostra T.
- f je stejně těžká jako e – pak můžeme použít tentýž trik a získat jinou kostru stejně těžkou jako T (tedy též minimální), která obsahuje hranu e. Tím pádem hrana e v některých minimálních kostrách leží a v jiných ne.
- f je lehčí než e – dokážeme, že tehdy se e nemůže vyskytovat v žádné minimální kostře. K tomu se nám bude hodit následující lemma (použijeme ho na cyklus tvořený cestou $T[u, v]$ a hranou e).

Cyklové lemma: Nechť C je cyklus v grafu a $e = uv$ jeho hrana, která je těžší než všechny ostatní hrany cyklu C. Potom se e nevyskytuje v žádné minimální kostře.

Důkaz: Pro spor předpokládejme, že existuje nějaká minimální kostra K, která obsahuje hranu e. Odebereme-li z K tuto hranu, kostra se rozpadne na nějaké dva stromy K_u a K_v (označíme je tak, aby $u \in K_u$ a $v \in K_v$). Budeme obcházet cyklus C: začneme ve vrcholu u, půjdeme vzdálenější cestou k vrcholu v (tedy ne po hraně e) a budeme sledovat, ve kterém stromu se zrovna nacházíme. Na začátku to je strom K_u, na konci K_v, takže někde cestou musíme potkat nějakou hranu h, jejíž jeden konec leží v K_u a druhý v K_v. Tato hrana se ovšem nevyskytuje v kostře K a je lehčí než hrana e (protože hrana e byla nejtěžší na cyklu). Proto nahrazením e za h získáme kostru lehčí než K, což je spor s minimalitou K.

Naše tři pravidla nám tedy říkají, jak pro každou hranu, která neleží ve zvolené minimální kostře T, rozhodnout, zda leží v nějaké / všech / žádné minimální kostře. Stačí umět hledat nejtěžší hrany na cestách v T, na což se dá elegantně použít datová struktura z druhé úlohy této série.

Zbývá dořešit, jak je to s hranami, které leží v T, ale to už si zkuste rozmyslet sami. Opět pomůže Cyklové lemma.

Martin „Medvěd“ Mareš

[12] `http://mj.ucw.cz/vyuka/ga/`

25-2-7 Zaléváme dokument

Nepřišlo sice tolik vašich řešení jako v první sérii, ale stále se jednalo o úlohu s největším počtem odevzdaných řešení. Vypadá to, že vás TEX zaujal a to je dobře.

Úkol 1

Řešení tohoto úkolu jste se zhostili velmi úspěšně a vynalézavě. Podívejme se, jakým způsobem se dal řešit. Nejprve bylo potřeba nadefinovat bílá a černá pole.

```
\def\bile#1{\hskip #1}
\def\cerne#1{\vrule height #1 width #1}
```

Objevily se i jiné, stejně dobré definice bílého pole:

```
\def\bile#1{\hbox to #1{\hfil}}
\def\bile#1{\vrule height 0cm width #1}
```

Pak je potřeba políčka nějak poskládat do šachovnice. Ondra Hlavatý přišel s nápaditým a čistým řešením – definovat oblasti 2×2.

```
\def\ctyrka#1{\vbox{%
  \hbox{\bile{#1}\cerne{#1}}%
  \hbox{\cerne{#1}\bile{#1}}%
}}
```

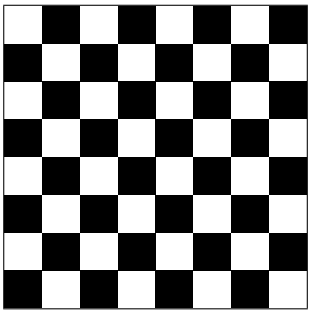
Tyto oblasti pak naskládáme do šachovnice a orámujeme.

```
\def\sachovnice#1{\vbox{\offinterlineskip%
  \hrule\hbox{%
    \vrule\vbox{%
      \hbox{\ctyrka{#1}\ctyrka{#1}%
            \ctyrka{#1}\ctyrka{#1}}%
      \hbox{\ctyrka{#1}\ctyrka{#1}%
            \ctyrka{#1}\ctyrka{#1}}%
      \hbox{\ctyrka{#1}\ctyrka{#1}%
            \ctyrka{#1}\ctyrka{#1}}%
      \hbox{\ctyrka{#1}\ctyrka{#1}%
            \ctyrka{#1}\ctyrka{#1}}%
    }\vrule
  }\hrule
}}
```

Šachovnici vykreslíme zavoláním `\sachovnice{20mm}`.

Vypnutí meziřádkové mezery (`\offinterlineskip`) jste si mohli najít na fóru, případně jste mohli nastavit rozměry `\baselineskip` a `\lineskip` na nulu.

Typickou chybou bylo vynechání vypnutí meziřádkových mezer, což vytvořilo ošklivé bílé pruhy mezi řádky. To jsem penalizoval obvykle jedním bodem. Drobné penalizace jste se také mohli dočkat za nečitelný kód.



Úkol 2

Druhý úkol bylo prosté cvičení na vyloženou látku. Někteří jej hrubě odflákli, objevilo se nezarovnání počtu bodů na pravou stranu nebo ošklivě malá mezera pod textem.

Takto vypadá vzorová implementace používaná v KSP (jen máme okolo přidané ještě nějaké mezery, aby nám nadpisy úloh lépe sedly do sazby):

```
\def\thrule{\hrule height 0.8pt depth 0pt}
\def\dblrule{\thrule\nobreak\vskip 1pt\thrule}
\def\taskheading#1#2#3{\vtop{%
  \dblrule\line{%
    \strut\bf #1 \enspace #2 \hfil #3}
  \dblrule}%
}
```

Makro `\thrule` definuje čáru, `\dblrule` definuje dvojčáru. Makro `\enspace` je definováno v Plainu jako mezera velká přesně `0.5em`.

Asi dva z vás použili i `\strut`, což je zkratka definovaná v Plainu za podlý trik (leč naprosto běžný a standardní): `\vrule width 0pt height 8.5pt depth 3.5pt\relax`

Účel použití této konstrukce naleznete ve třetí sérii. Ti, kdo na to přišli sami, u mě mají malé bezvýznamné plus.

Úkol 3

Toto byl nejtěžší úkol. Definice vlastního prostředí typu verbatim dala mnohým zabrat. Objevilo se několik variant, obvykle jste si zvolili nějaký znak nebo fixní sekvenci, která prostředí verbatim ukončí. Zde uvádíme vzorovou implementaci, ve které si můžete zvolit ukončovací řetězec sami.

Příkaz `\verbatim` je vstupní rozhraní celého prostředí. Otevře se skupina, přestaví se kategorie speciálních znaků, změní se práce s mezerami a předá se řízení do dalšího makra.

```
\def\verbatim{%
  \begingroup
  \verbcats
  \verbspaces
  \verbwork
}
```

Přestavení speciálních tisknutelných znaků se provede s výjimkou složených závorek, které ještě budeme potřebovat, aby nám orámovaly ukončovací řetězec.

Všimněte si speciální sekvence `^^I` a `^^M`. Když TEX potká dva stejné znaky kategorie 7 za sebou, spolkne je a následujícímu znaku přehodí šestý bit. Operace se provádí *před tokenizací* (takže `\^^I` je token (TAB, 0), tedy řídící sekvence se jménem TAB).

Takže `^^I` je znak s kódem 9, tedy tabulátor; `^^@` je znak s kódem 0; `^^.` je znak `n`; `^^?` je znak s kódem 127, zvaný též DEL, jediný znak, který je běžně kategorie 15. Tento zápis se hodí pro přehlednost, aby se v kódu jen tak nepoflakovaly netisknutelné znaky.

Ještě pro úplnost, pokud se za `^^` objeví dvě malé hexadecimální číslice (0 až 9, a až f), nahradí se celá čtveřice za znak s uvedeným hexadecimálním kódem.

```
\def\verbcats{%
  \catcode`\\=12
  \catcode`\$=12
  \catcode`\&=12
  \catcode`\#=12
  \catcode`\^=12
  \catcode`\_=12
  \catcode`\%=12
  \catcode`\~=12
  \catcode`\ =13
  \catcode`\^^I=13
  \catcode`\^^M=13
}
```

Nyní nastavíme chování verbatimu na bílých znacích.

Mezera, tabulátor a konec řádku se stanou aktivními znaky s významem `\verbspc`, `\verbtab` a `\verbcr`. Konec řádku zajistí, že se vstoupí do odstavcového módu a vynutí se vysázení odstavce. Odstavec musí obsahovat nějaký materiál, jinak se nevysází, proto ten `\hskip`.

Mezera se vysází jako explicitní mezera. Tabulátor vysázíme jako čtyři mezery. Pokud odkomentujete šestý řádek, budou mezery vysázeny viditelně. Mezery na koncích řádků se ořezávají ještě před tokenizací, takže je vysázet neumíme.

```
\catcode`\ =13\catcode`\^^I=13\catcode`\^^M=13
\def\verbspaces{\let \verbspc\let
\verbcr\let^^I\verbtab}
\catcode`\ =10\catcode`\^^I=10\catcode`\^^M=5
\def\verbspc{\ }
%\chardef\verbspc 32
\def\verbtab{\verbspc\verbspc\verbspc\verbspc}
\def\verbcr{\noindent\hskip 0pt\par}
```

Nakonec se přečte ukončovací řetězec (který může obsahovat libovolné znaky, jen musí být dobře uzávorkovaný, co se týče složených závorek), nastaví se i kategorie složených závorek na 12, zruší odsazení prvního řádku odstavce a provede finální magie.

```
\def\verbwork#1{%
  \catcode`\{=12%
  \catcode`\}=12%
  \parindent 0pt%
  \def\verbdo^^M##1#1{\tt##1\endgroup}%
  \verbdo
}
```

Makro `\verbatim` se volá takto:

```
\verbatim{EOV}
Nějaký zdrojový kód
   odsazený mezerami
        nebo tabulátorem

s prázdnými řádky
a obsahující různé speciální znaky
jako jsou { a } nebo %
EOV
```

Nyní vysvětlíme magii okolo `\verbdo`. Parametr `{EOV}` se přečte až při volání makra `\verbwork`. To pak definuje `\verbdo` vlastně takto:

```
\def\verbdo^^M#1EOV{\tt#1\endgroup}
```

Pak se makro zavolá, spolkne konec řádku za `{EOV}` a zbytek až do `EOV` vysází. Pak uzavře skupinu, čímž všechny

zběsilé změny kategorií zruší a můžeme zase sázet klasicky dál. Místo `EOV` můžeme použít libovolný jiný řetězec, který bude verbatim ukončovat.

Za rozumně funkční řešení jsem dával plný počet bodů. Za vážnější prohřešky jsem pak něco strhával.

Balíky maker

Čím více budete používat TEX, tím více budete mít pocit, že si na začátek souboru kopírujete děsnou spoustu věcí. TEX umí vkládat externí soubory primitivem `\input`, za které uvedete jméno souboru. Můžete si tedy například vytvořit svůj soubor se spoustou maker, který si pak vložíte do každého sázeného textu. Například všechny letáky KSP začínají příkazem `\input kspmac3.2.tex`, tedy vložením souboru s makry KSP, verze 3.2.

Na spoustu různých úkolů pak existují specializované balíky, které si uživatelé můžou vyměňovat přes CTAN.[13] Na adrese `http://www.ctan.org/` tedy najdete několik tisíc různých balíků všeho druhu.

A to je pro dnešek vše. Děkuji vám všem za hezká řešení a těším se na příští sérii.

Jan „Moskyto“ Matějka

[13] Comprehensive TEX Archive Network

Výsledková listina druhé série dvacátého pátého ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*2521*	*2522*	*2523*	*2524*	*2525*	*2526*	*2527*	*série*	*celkem*
0.					13	11	6	13	8	9	13	59,0	118,0
1.	Rastislav Rabatin	GJHroncaBA	4	5	13	7,5		13	8	9	8,5	54,8	109,5
2.	Ondřej Hlavatý	GJirsíkaČB	4	2	4	5		13	8	4	13	49,6	102,5
3.	Michal Punčochář	GJírovcČB	3	7	6,5	11		13	8		13	53,2	98,9
4.	Martin Raszyk	G_Karvina	3	12	13	11		13			13	50,0	97,0
5.	Dominik Macháček	GLanškroun	4	7	8			13	5	9	13	50,6	94,6
6.	Martin Španěl	ArcibisGPH	4	5	9	11			8		11,5	42,2	93,1
7.	Martin Černý	G_Sokolov	3	2	5	4,5		4	8	4	5,5	41,0	89,1
8.	Štěpán Hojdar	GJírovcČB	3	2	6			6	3	6	12,8	45,4	87,0
9.	Vojtěch Hlávka	GŠlapanice	4	17	10			13	8	9	2,5	39,2	83,3
10.	Dalimil Hájek	GKepleraPH	2	7	7	7,5	4	4		6	6	37,8	81,6
11.	Jakub Maroušek	G_Písek	3	2	5,5	4		4		0	10,7	36,1	73,5
12.–13.	Mikuláš Hrdlička	MensaG	2	2	4		3,5	6	2,5			26,8	72,9
	Matej Lieskovský	GOmskPha	3	7	2,5		4		3	7	9	30,7	72,9
14.	Jakub Svoboda	GKomHavíř	3	2		7	4	4	1,5	2		29,6	71,2
15.	Jakub Šafin	GHorMichal	4	2	13	11		13	8	9	0	54,0	65,2
16.	Ondřej Mička	GJírovcČB	4	15	12				8		5	23,0	63,6
17.	Petra Pelikánová	GJarošeBO	4	3			4	4	3			17,4	62,6
18.	Lukáš Ondráček	GVolgogrOS	4	7							11,5	12,1	62,1
19.	Petr Houška	GJírovcČB	3	3	3,5	4			1,5		8,5	27,4	58,5
20.	Martin Šerý	GJírovcČB	3	3		1			0,5	5	10	22,3	58,2
21.	Richard Hladík	GOAMarLaz	0	2			3		3		5	18,6	53,6
22.	Marek Dobranský	GHorMichal	3	2	0		3,5		1	0,5	5	17,3	52,6
23.	Vojtěch Vašek	GHli	4	6	8			6		6	4	31,3	51,1
24.	Kateřina Zákravská	GJar	4	3							11,5	12,5	50,4
25.	Sabína Fraňová	GDubNVahom	4	3	1,5	3	4	6	0	3		28,5	48,2
26.	Vladan Glončák	GĽŠtúraTN	4	3			4	4			4,5	20,2	47,2
27.	Jan Pokorný	G_Bučovice	1	1								0,0	46,7
28.	Alexander Mansurov	GNVPláníPH	4	11				6				6,0	44,4
29.	Jan Mikel	G_RožnovPR	4	1								0,0	43,5
30.	Vojtěch Sejkora	SPSE_Pard	4	11					1,5		13	14,5	41,8
31.	Aneta Šťastná	GOmskPha	3	5								0,0	39,7
32.	Jonatan Matějka	SŠP_ČB	3	12	7			13	1			20,5	37,5
33.	Mark Karpilovskij	GJarošeBO	4	7				13				13,0	37,0
34.	Štěpán Trčka	GSlavičín	2	5								0,0	35,2
35.	Tomáš Velecký	GBezručeFM	2	5							5	7,5	34,8
36.	Tomáš Svítil	AES_NewDelhi	4	2			3,5		2,5			9,7	34,6
37.	Radovan Švarc	G_ČTřebová	2	2								0,0	34,4
38.	Ondřej Cífka	GNAlejíPH	4	9								0,0	32,0
39.	Milan Šorf	GNeumannŽR	3	2		1	3,5		0	0	4	14,9	31,3
40.	Jitka Fürbacherová	GKlatovy	4	8			5	6	4			17,4	29,3
41.	Jan Knížek	G_Strakon	2	7	5							6,7	28,3
42.	Anna Zákravská	GJar	4	3							6,5	9,7	27,6
43.	Jozef Kaščák	G_Svidník	4	1								0,0	23,3
44.	Tereza Hulcová	GKlatovy	4	8								0,0	23,2
45.	Ondřej Hübsch	GArabskáPH	3	17				13				13,0	21,8
46.	Jan Lejnar	GKlatovy	3	2	3		4	6				21,0	21,0
47.	Marek Dědič	GBNěmcovHK	3	1								0,0	19,3
48.	Jan-Sebastian Fabík	GJarošeBO	3	6								0,0	12,0
49.	Michal Staruch	GOA_Vrchla	4	1			4,5		1,5			9,0	9,0
50.	Pavel Salva	VOŠŠumperk	3	4								0,0	8,7
51.	Jan Horešovský	GMěl	3	1								0,0	6,4
52.	Dominik Roháček	SPŠLegioJI	3	1								0,0	6,0
53.	Václav Volhejn	GKepleraPH	0	3			4					5,2	5,2
54.	Přemysl Šťastný	GZamberk	-1	1								0,0	4,7