Milé řešitelky a řešitelé!

Dostává se vám do rukou poslední série jubilejního 25. ročníku KSP. Jsme rádi, že jste s námi tento ročník prožili a zde vám přinášíme posledních několik úloh, včetně posledního dílu seriálu o TEXu, než si začátkem léta budete moci dát zasloužený oddych (a orgové taky).

Také se zde konečně rozuzlí zamotaný příběh, který jsme vám po kouscích celý rok odhalovali. Vzpomenete si ještě na příběhy japonského kluka, investigativní novinářky, policisty a konzula? Ne? Tak si je připomeňte přečtením minulých sérií, nebo se naopak odvážně ponořte do příběhu aktuálního, během něhož vám minulé příběhy snad připomeneme.

Každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok a tužku. Navíc **pošleme čokoládu každému, kdo v této sérii z libovolných pěti úloh dostane alespoň polovinu možných bodů, které lze za tyto úlohy získat**.

Připomínáme, že z každé série se do celkového bodového hodnocení započítává 5 nejlépe vyřešených úloh.

Termín odevzdání páté série je stanoven na **pondělí 27. května v 8:00 SELČ**. CodExová úloha má termín o den posunutý, protože nám ji opravuje automat – odevzdejte ji do 28. května, 8:00 SELČ.

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 22. května 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.*

Pátá série dvacátého pátého ročníku KSP

Při tlačenici v metru se muž už po několikáté podíval na hodinky. Byl to pobočník vysoce postaveného důstojníka policie a spěchal do práce. Tedy ne že by byl pobočníkem již nějak dlouho, na tohle místo byl přidělený asi před měsícem, ale už stačil zjistit, že šéf nemá rád nedochvilnost.

Konečně dojel na Malostranskou a rychle vyběhl po eskalátorech. Dělníci stále kopali tramvajové koleje, takže i dnes se musel svézt náhradním autobusem. „Tak co, jednou přijdu pozdě. Snad jen, kdyby ten řidič byl dneska obzvláště rychlý. . . “ pomyslel si při nastupování do nezvykle vypadajícího vozidla.

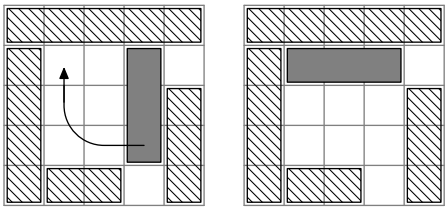
25-5-1 Cesta autobusem — 11 bodů

Dopravní podnik testuje nový druh autobusu – autobus s roztažitelnou karosérií. Bohužel vytočit se s ním v úzkých uličkách není vždy snadné a vyžaduje to velké řidičské umění.

Představte si plán města jako klasickou čtvercovou síť, volná místa představují ulice a náměstí, na zaplněných políčkách jsou domy, parky, fontány a jiné věci, přes které by autobus projíždět neměl. Autobus obsazuje několik políček za sebou (tedy je to jakýsi obdélník o šířce jedna a délce k) a může jet buď vodorovně, nebo svisle, a to oběma směry (buď jede dopředu, nebo couvá).

Na plánu města je start, cíl a navíc jsou zde nástupní a výstupní zastávky. Pokud autobus projede přes nástupní zastávku, tak se o jedno políčko do délky natáhne (proti směru, ze kterého na políčko přijel), a pokud naopak přes výstupní, tak se zkrátí. Nemůže se však zkrátit na nulovou délku. Zastávkou nelze projet dvakrát těsně za sebou, je nutné mezitím navštívit alespoň jednu jinou.

Aby se autobus mohl otočit, potřebuje dostatek místa. Otáčí se kolem některého ze svých konců, a to tak, že pokud má délku k, musí stát tímto koncem v rohu volného prostoru rozměru $k \times k$. Pak se otočí jako na obrázku a stojí ho to právě jeden krok. Přirozeně, druhý konec autobusu se také musí nacházet v onom volném prostoru.



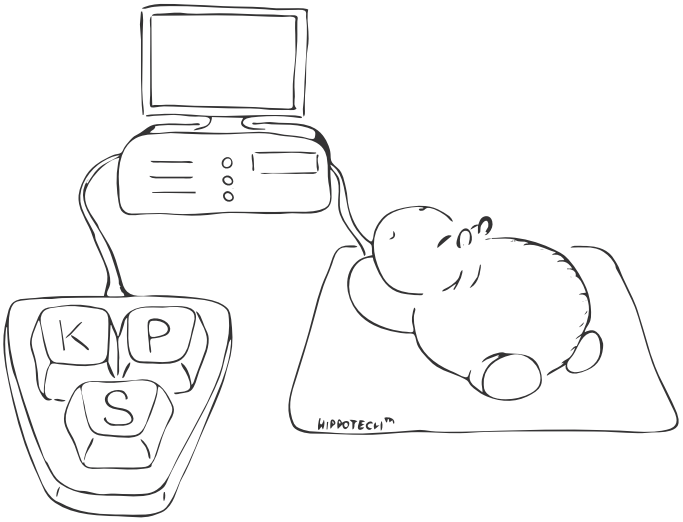
Vaším úkolem je nalézt nejkratší cestu ze startu do cíle (nemusíte projet všemi zastávkami) takovou, aby autobus projel a měl dostatek místa na otáčení.

Lehčí varianta (za 6 bodů): Vyřešte to samé, ale bez zastávek (tedy autobus má jen pevnou délku k a během cesty se jeho délka už nemění).

Po klikaté jízdě skrz uličky vyběhl pobočník z autobusu a rychle běžel na stanici, kde se měl se šéfem setkat. Cestou minul na dvoře nějaký policejní nástup, místní poručík si asi přepočítával přítomné policisty. To už ale pobočník vešel do budovy a u dveří kanceláře zaslechl hlas šéfa. Zrovna domlouval nějaké odvolání nepohodlného pochůzkáře na jiné místo, jak pobočník pochopil.

„Tak na to se podívejme!“ zamumlal si potichu, aby to nikdo neslyšel. To by zapadalo do obrázku, který si o svém šéfovi udělal během toho měsíce, který u něj zatím strávil. Podivná setkání, divné zprávy, rozkazy a náhody. Začínal mít vážné podezření, že na něj šéf nehraje čistou hru. Zaslechl od kolegů nějaké zvěsti o rozrůstající se japonské mafii.

Chvíli váhal, ale pak se rozhodl. Tuhle nahrávku musel mít celou. Šéf udělal chybu a tenhle rozhovor vedl přes telefon na stanici, který byl samozřejmě nahráván. Opatrně tedy vešel do vedlejší místnosti, zavřel za sebou dveře a posadil se k počítači.



25-5-2 Telefonní ústředna — 9 bodů

Program telefonní ústředny na policejní stanici zaznamenává všechny provedené hovory. Bohužel věcí, co zaznamenává, je spousta, a proto se musí ukládat komprimovaným způsobem.

Všechny záznamy tvoří dohromady posloupnost 0 a 1 dlouhou n. Hovor je identifikovaný konkrétním vzorcem 0 a 1 o délce s, který se v posloupnosti může vyskytovat jako vybraná podposloupnost.[1]

Máme zadaný vzorec námi hledaného hovoru a ptáme se, kolik hovorů si budeme muset přinejhorším poslechnout, abychom nalezli ten pravý. Neboli kolik je možností, jak vybrat z celé posloupnosti daný vzorec.

Příklad: Pro posloupnost 010111 existuje 9 způsobů, jak v ní nalézt vzorec 011.

„Konečně, už to mám!“ zaradoval se v duchu pobočník. V tom ale málem dostal infarkt. Těžká ruka mu dopadla na rameno. Za ním se ozval šéfův hlas.

„Jane, co to tu děláte?“

„Já... pane... já...“ rychle se pokoušel schovat okno se záznamy hovorů.

Šéf si výpisu záznamů však všiml. Chvíli stál mlčky s rukou na Janově rameni. Jan skoro cítil, jak šéf v hlavě prochází spoustu možností – nedopadne Jan stejně jako předchozí pobočník, o kterém se povídá, že už ho nikdo nikdy neviděl? Pak si to šéf zjevně rozmyslel a jen suše řekl: „Pojďte Jane, půjdeme se někam najíst.“

Jan, nevěda co očekávat, ho následoval. Skoro mlčky došli do blízké restaurace. Jan přemýšlel, jestli nemá utéct, ale nějaký tajemný pocit mu říkal, že teď šéfovi může věřit. Usadili se v osamělém rohu a objednali si jídlo. Mezitím, co Jan opatrně uždiboval špagety, začal mu šéf líčit spoustu věcí.

25-5-3 Špagety — 11 bodů

Představte si špagety v typické italské restauraci v centru Prahy. Je to jakási směs zamotaných těstovin a člověk musí hodně dávat pozor, jak je nabírat, aby si je na sebe při jídle neplácl. Proto je nejlepší odebírat špagety jen z vrchu talíře a netahat je zespoda.

Talíř špaget si můžeme představit jako trojrozměrnou mřížku. Špageta je vždy nějaký souvislý „had“ složený z trojrozměrných jednotkových krychliček, který se může libovolně kroutit. Jednotlivé špagety se v trojrozměrné mřížce vzájemně neprotínají.

Navíc máme daný směr gravitace, tedy osu, ve které budeme špagety postupně jíst. V každém kroku můžeme vzít právě ty špagety, na kterých ve směru této osy neleží žádná jiná špageta (sama na sobě však ležet může, to nám nevadí). Tím se nám uvolní některé další špagety, které zase můžeme odebrat při dalším kroku, a tak dále. Skončíme ve chvíli, kdy buď sníme celý talíř, nebo už nebudeme mít žádnou špagetu volnou.

Vaším úkolem pro zadaný talíř špaget je tedy spočítat minimální počet kroků pro snězení celého talíře a vypsat špagety odebírané v každém kroku, nebo určit, že špagety sníst nelze. Jako vstup můžete předpokládat popis celého talíře po jednotlivých souřadnicích (tedy buď se na souřadnici nachází kus nějaké určité špagety, nebo je zde volné místo).

⇧ **Lehčí varianta (za 5 bodů):** Uvažujte jen rovinnou situaci, tedy když se špagety budou proplétat jen ve dvou rozměrech a odebírat je budeme ve směru jedné z os.

„To snad nemyslíte vážně, pane!“ řekl Jan, když konečně dojedl špagety. Během uplynulých dvaceti minut se mu úplně převrátil pohled na šéfa. Žádný mafián, agent speciálního útvaru policie to byl!

„Dobře jste všechny okolo vodil za nos. A proč jste si vlastně vybral mě?“

„Inu Jane, to byla součást plánu. Mafiány nejlépe dostanete zevnitř. A proč jsem si vybral vás? Nejste z Prahy, takže vás nemůžou znát, vaše hodnocení ze služby v Brně je přímo ukázkové a můj kamarád, šéf vašeho okrsku, mi vás doporučil. A navíc jste byl rok v Japonsku a prý umíte trochu japonsky, což se možná bude hodit. Ale teď –“ náhle ho přerušil telefon.

Šéf rychle prohodil několik slov do telefonu a pak se na Jana podíval. „Ale teď provedeme pár výslechů, poručíkovi zdejšího oddělení Hamáčkovi se právě povedlo udělat razii v nějakém čínském bistru,“ zlověstně se usmál.

25-5-4 Výslechy — 11 bodů

Policii se povedlo při razii v čínském bistru zadržet několik osob. Bohužel nevíme, kdo z nich je spořádaný zaměstnanec bistra a kdo z nich je mafián. Jediné, co víme, je, že mafiáni vždy lžou a zaměstnanci bistra vždy mluví pravdu.

Máme množinu výroků dvou typů: „A tvrdí, že B je mafián“ a „A tvrdí, že B není mafián“. Protože policisté chtějí mít při rozklíčování této situace alespoň nějaká vodítka, chtějí po vás zjistit počet možných řešení, neboli počet různých způsobů, kterými lze podezřelé označit za mafiány nebo zaměstnance bistra. Počet chceme spočítat modulo nějakou konstantou K (tedy tato úloha není myšlená jako úloha na velká čísla).

Navíc byste měli poznat, pokud si výpovědi nějakým způsobem protiřečí. Přesně řečeno že neexistuje žádné možné rozdělení na mafiány a zaměstnance, které by při daných

[1] Vybraná podposloupnost vznikne z původní posloupnosti čísel tak, že vynecháme některé její prvky. Pořadí zbylých prvků zůstane zachováno.

výrocích dávalo smysl (v tom případě se pak už policisté nějak zařídí).

Příklad: Pro množinu tří osob A, B a C a pro výroky: „A tvrdí, že B je mafián“ a „A tvrdí, že C není mafián“ máme jen dvě možnosti: A a C jsou mafiáni a B zaměstnanec bistra, nebo přesně naopak. Kdybychom k nim však přidali ještě osobu D, tak se nám počet možností zdvojnásobí (protože D může být v obou případech jak mafián, tak zaměstnanec bistra).

Výslechy byly zdlouhavé a táhly se až do večera. Nakonec ale Jan se šéfem zjistili něco, co se jim vůbec nelíbilo. Vypadá to, že právě teď se chystá velká dodávka nelegálního zboží.

Aby toho nebylo málo, tak hned venku přinesl nějaký rychlý posel šéfovi zprávu. Šéf si ji přečetl a pak zaklel. To bylo poprvé, co ho Jan slyšel mluvit sprostě.

„Právě dostali mého člověka. Nevím, jak se o něm doslechli, ale leží ve vážném stavu v nemocnici. Dnes večer měl domluvenou tajnou schůzku s konzulem, měl hrát prostředníka jistému bohatému podnikateli.“

Šéf chvíli přemýšlel. Pak ho něco napadlo. Vysvětlil Janovi svůj plán.

Jan chvíli přemýšlel. To, co po něm šéf chtěl, nebylo lehké. Jestli tohle vyjde, můžou nachytat celou japonskou mafii i s konzulem. Ale pokud ne… Ale co na tom, rodinu nemá, o rybičky se mu doma už někdo postará – „Jdu do toho, pane.“

⋆⋆⋆

V drahém obleku se Jan cítil trochu nesvůj, ale už si na něj zvykal. Jen se bez zbraně na boku a odznaku v kapse cítil jako nahý. Před chvílí navíc minul svého starého známého, jednoho z pochůzkářů. Jen tak tak, že ho nepředvedl na služebnu, když přebíral aktovku s dokumenty od jednoho kontaktu.

Teď šel rozvážným krokem k japonské ambasádě. Když už byl skoro u ní, všiml si znaveně vypadajících zahradníků. „Zajímavé, jako by celý den vozili sem a tam trávu,“ podivil se. Teď už to vypadalo, že zahradu u ambasády konečně uklízí.

25-5-5 Úklid trávníku — 9 bodů

Zahradníci starající se o trávník u japonské ambasády jsou po celém náročném dni už silně unavení, ale ještě na ně čeká poslední úkol. Musí z posekané trávy vybrat reprezentativní vzorek, který pošlou do laboratoře na rozbor, jestli je trávník zdravý.

Na sběr trávy používají zmenšenou verzi balíkovacího stroje, kterým se tráva svazuje do malých krychlových úhledných balíků. Do laboratoře chtějí zaslat právě k náhodně vybraných balíků z celého trávníku, ale neví, na kolik balíků sběr vší posekané trávy vyjde.

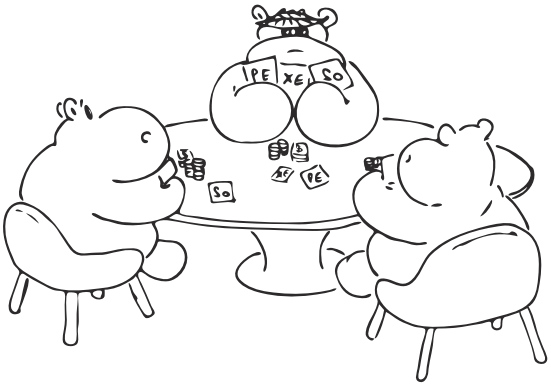
Chtějí tedy od vás nějaký postup, jak z posloupnosti balíků neznámé délky vybrat právě k balíků. Každá k-tice balíků musí mít stejnou pravděpodobnost, že bude vybrána. Již prošlé balíky nelze vracet (nakládají se na valník a ten je odváží na kompost), tedy nelze si počet balíků nejdříve spočítat a pak teprve vybírat. Vše je nutné udělat během jednoho průchodu.

Lehčí varianta (za 4 body): Řešte úlohu pro $k = 1$, tedy pokud chceme vybrat jen jeden náhodný balík.

To už ale Jan došel na ambasádu a nechal se uvést ke konzulovi. Ještě než mohl konzul cokoliv říct, tak Jan spustil.

„Předně bych vám chtěl poděkovat, pane Yamado. Nevím, jak ten špeh ke mně proklouzl, ale příště budu své lidi mnohem více prověřovat. Jsem vám zavázán. A teď bychom se mohli věnovat započatému obchodu,“ uctivě se uklonil. Pokoušel se držet si sebejistou tvář, ale srdce měl strachem až v krku.

„Takže pan Kebner osobně, jsem rád, že konečně vidím vaši tvář.“ Luskl prsty a rázem přiskočili dva bodyguardi, kteří ho během pár sekund prohledali a pak rychle kývli na konzula. „V pořádku, chtěl jsem si být jistý. Posadíte se a dáte si se mnou partičku Triád? Můžeme nad nimi prodiskutovat tu množstevní slevu, kterou jste navrhoval.“



Jan, potěšen, že mu konzul zatím věří, se s ním posadil nad herní stolek. Bohužel konzul byl příliš dobrý a Jan stále vystrašený, takže konzul snadno vyhrál. Během toho mluvili o obchodu a konzulovi nedalo příliš práce požadovanou slevu srazit na minimum. Však Janovi o peníze vlastně ani nešlo. Navzájem si také potvrdili vše, co už dříve domlouval nyní raněný agent, jen změnili data a časy. Mělo to proběhnout již dnes v noci.

Ještě než se však dostali k samotnému naplánování, přerušila je konzulova žena. „Drahý pane, můj milý muži…“ pokoušela se mluvit česky, ale bylo na ní znát, že se musí hodně soustředit. „Dnes já upekla tento… dort se tomu říká u vás?“ Jan přikývl. „Prosím, dejte si.“

Pak potichu odešla. Jan se na dort podíval. Byl celkem malý a již nakrájený, ale docela nepravidelně. Etiketa sice vyžadovala, aby ho celý nesnědl sám, ale měl už děsný hlad, a tak se ho chtěl najíst co nejvíc.

25-5-6 Dělení dortu — 11 bodů

Kulatý dort je nakrájený na jednotlivé kousky různé velikosti, kousky mají tvar kruhové výseče. První strávník si vybere jakýkoliv kousek dortu a ten sní. Pak se postupně střídají s druhým strávníkem, dokud nesnědí celý dort. Poté, co už je odebrán první dílek, je možné odebírat pouze z okraje odebrané výseče (tedy vždy jsou na výběr maximálně dva dílky).

Uvažujte, že oba strávníci chtějí snést co největší množství dortu a že druhý strávník vždy odebírá optimálně. Jaké největší množství dortu může první strávník snést při použití optimální strategie?

Tato úloha je praktická a řeší se ve vyhodnocovacím systému CodEx.[2] Přesný formát vstupu a výstupu, povolené jazyky a další technické informace jsou uvedeny v CodExu přímo u úlohy.

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

Poté, co dojedli dort – Janovi se povedlo sníst více, což ho trochu zasytilo a dodalo mu sebejistoty – sáhl konzul někam za sebe a spustil umně ukrytý projektor. Na zdi se za chvíli objevila celkem podrobná mapa Prahy.

„Tohle už asi znáte, pane Kebnere?“ zeptal se. Jan opatrně přikývl, i když mapu v životě neviděl. Na okrajích bylo pár poznámek v japonštině, které s vypětím sil přelouskal. Popisovaly něco o rozmístění policejních hlídek.

„Teď se ale trochu mění situace. Nevím, proč to plutonium a další věci chcete, ale už si po dnešku nemůžu dovolit nechat si to ve svých skladech. Musíme to provést ještě dnes.“

„Povedlo se mi z jistého zdroje získat aktuální rozestavění policie.“ Po konzulových slovech se Jan opět podíval na mapu. V duchu si oddechl, to důležité tam scházelo. Musel naplánovat trasu předání nelegálního zboží tak, aby to konzulovi nebylo podezřelé, ale tak, aby ho dostal tam, kam potřebuje...

25-5-7 Policejní koridor — 13 bodů

Máme zadanou mapu města jako neohodnocený neorientovaný graf (křižovatky pospojované ulicemi), na některých křižovatkách stojí policejní kontroly. Dále máme zadaný start, sklad a cíl jako nějaké křižovatky a chceme vyjet ze startu, naložit věci ve skladu a dojet do cíle.

Okolo každé policejní kontroly můžeme projet za celou cestu maximálně jednou. Kdybychom okolo ní projeli vícekrát, tak už jí to přijde podezřelé, zastaví nás a podrobí nás prohlídce – a to přesně nechceme. Současně chceme cestu absolvovat co nejrychleji.

Najděte tedy pro zadanou mapu s policejními hlídkami co nejkratší cestu mezi startem, skladem a cílem tak, aby každou křižovatkou s kontrolou procházela nejvýše jednou.

Lehčí varianta (za 7 bodů): Zjistěte jen, jestli taková cesta existuje.

Janovi se konečně povedlo vymyslet trasu, která vypadala alespoň trochu rozumně. Ukázal ji konzulovi. Ten nad ní chvíli taky váhal, ale pak přikývl. „Tohle vypadá dobře. Ano. Ale pojedete s námi, ať máme jistotu.“ S tímhle Jan nepočítal, chtěl odejít. Ale teď tu operaci přece nemůže pokazit, teď už je to nutné dohrát až do konce, ať bude jakýkoliv. „Dobře, kdy vyrážíme?“

Auto se pomalu blížilo k místu setkání. Celý nákladový prostor byl zaskládán nelegálním zbožím a uprostřed něj trůnila zlověstná bedna se znaky radioaktivity na boku. Jelo pomalu, bez světel.

Na smluveném místě z něj vystoupil jeden muž. Došel doprostřed plácku ohraničeného starými průmyslovými budovami. Tam čekalo druhé auto s jedním osamoceným mužem. První muž se rozhlédl, něco mu tady nehrálo. Najednou se ozvalo několik kovových cinknutí.

Jan věděl, co čekat. Vyskočil z auta, pevně zavřel oči a přitiskl si ruce na uši. Okolí najednou zaplavilo nesnesitelné světlo a zvuk o omračující síle. Šokové granáty. Světlo zmizelo stejně rychle, jako se objevilo. Jan otevřel oči, kopem skolil jednoho z japonských bodyguardů a vrhl se do bezpečí mezi průmyslové budovy.

Plácek mezitím zaplavilo světlo z mnoha reflektorů a výkřiky policie. Mafiáni byli natolik zaskočeni, že se nikdo nezmohl na žádný odpor, nikomu nebylo ublíženo. Během několika sekund složili zbraně a policisté je odvedli.

„Dobrá práce Jane,“ ozvalo se nad ním. Byl to šéf a natahoval ruku, aby mu pomohl se zvednout. „Nechcete pro mě pracovat i dál?“

Závěr příběhu vyprávěného z různých pohledů sepsal

Jirka Setnička

25-5-8 Boxy, z TEXu ven! — 15 bodů

Poslední díl seriálu věnujeme převážně výstupním rutinám a stránkovému zlomu. Vysvětlíme si, jak fungují penalty a špatnost sazby, a stručně si ukážeme okolí TEXu – formáty a nadstavby. Nakonec vložíme obrázek a vysázíme barevný dokument.

Stránkový zlom

TEX při sazbě stránky skládá boxy pod sebe do speciálního vertikálního boxu. Ve chvíli, kdy zjistí, že se už nevejde s výškou sazby do `\vsize`, najde správné místo, na kterém je nejlepší stránkový zlom, a tam uřízne box. Co se nevešlo, to si schová pro příští stranu.

Nejvýhodnější stránkový zlom se počítá tak, že se mezi každými dvěma položkami v boxu spočítá hodnota

$$c = \begin{cases} \infty, & \text{pokud } b = \infty \text{ nebo } q \geq 10\,000; \\ p, & \text{pokud } p \leq -10\,000; \\ b + p + q, & \text{pokud } b < 10\,000; \\ 100\,000, & \text{pokud } b = 10\,000, \end{cases}$$

přičemž b je „badness“, hodnota určující ošklivost roztažení nebo stažení stránky při zlomu na tomto místě;[3] p je penalta, hodnota určující nevhodnost zlomu na tomto místě (například mezi prvním a druhým řádkem odstavce); q je hodnota `\insertpenalties`, což je součet penalt pro speciální objekty jako poznámky pod čarou odpovídající zlomu.

Jediné, co můžete ovlivnit přímo, je penalta. Uvedete-li `\penalty 15`, vloží se na to místo penalta s hodnotou 15. Čím nižší penalta, tím spíš se na daném místě zlomí. Penalta $-10\,000$ a nižší vyvolá zlom vždy; penalta $10\,000$ a vyšší zlom zakáže. Pokud se někde vyskytnou dvě penalty za sebou, jejich hodnoty se sčítají.

Navíc je povoleno lámat jen na některých místech. TEX rozlišuje „zahoditelné“ a „nezahoditelné“ objekty. První z nich se za zlomem zahazují. Jedná se hlavně o penalty a výplňky.

[3] Badness spočítáme podle vzorce $b = \min(10\,000, 100 \cdot (g/g_0)^3)$, kde g je součet roztažení nebo stažení mezer oproti normálu a g_0 je celkové maximální povolené roztažení nebo stažení.

Lámat se pak smí jen před výplňkem, před kterým je něco nezahoditelného, nebo na penaltě. V TEXbooku nebo TBN si to můžete přečíst precizně.

Zde se můžou hodit vysvětlit některé zkratky, které jsme dříve definovali bez vysvětlení:

```
\def~{\penalty 10000 \ } % nedělitelná mezera
  % Mezera se považuje za výplněk a penalta
  % je zahoditelná ...

\def\break{\penalty-10000 } % zlom vždy
\def\nobreak{\penalty 10000 } % nelámej nikdy
\def\allowbreak{\penalty 0 } % povol zlom
% Na některých místech se nesmí lámat,
% například mezi dvěma čarami.
% Na penaltě se smí lámat vždy.

\def\filbreak{\par\vfil\penalty-200\vfilneg}
% \filbreak využívá skutečnosti, že na začátku
% každé stránky se zahodí všechny skipy.
% Přejde na novou stránku a zbytek vyplní
% prázdným místem, tedy pokud je záporná
% penalta dostatečná.
% Jinak se výplně vyruší:
% \def\vfil{\vskip 0pt plus 1fil}
% \def\vfilneg{\vskip 0pt plus -1fil}
\def\goodbreak{\par\penalty-500 }
\def\eject{\par\break}
\def\supereject{\par\penalty-20000 }
% Penalta -20000 se využívá pro požádání
% výstupní rutiny, aby vysázela všechny
% poznámky pod čarou a podobné elementy.
```

TEX si pak vybere takové místo, pro které je c nejmenší, a tam uřízne box. Co je před řezem, to vloží do vboxu číslo 255 a spustí výstupní rutinu.

Výstupní rutina

Na místo, kde došlo ke stránkovému zlomu, se vloží `{`, obsah seznamu tokenů `\output` a `}`. Cokoli, co vysázíte během výstupní rutiny, se přilepí před to, co zůstalo za stránkovým zlomem, a pokračuje se dál. Takto se tedy může výstupní rutina rozhodnout, že kus materiálu nevysází, a přesunout jej na další stranu. Na konci výstupní rutiny musí zůstat vbox 255 prázdný.

Dejte si pozor na to, že výstupní rutina se může aktivovat pokaždé, kdy vložíte nějaký materiál do hlavního vboxu, mimo jiné tam, kde se objeví `\par`, vložení boxu, čára, ... Pokud tedy v nějakém makru používáte stejné proměnné jako ve výstupní rutině (například `\count0` až `\count9`), pohlídejte si, aby se nespustila výstupní rutina zrovna v tu chvíli, kdy je máte předefinované.

Ve výstupní rutině se provedou všechny takové věci jako zvýšení čísla stránky, připojení hlaviček, patiček a poznámek pod čarou. Ve chvíli, kdy je poskládaná celá stránka, zavolá se `\shipout` a za toto primitivum se vloží box, který tvoří stránku. Tento box se ukotví svým levým horním rohem do bodu vzdáleného 1 in od levého i horního okraje. Tyto hodnoty se dají nastavit jako `\pdfhorigin` a `\pdfvorigin`.

Vzniklá stránka má rozměry `\pdfpagewidth` × `\pdfpageheight`, leda by nějaký z těch rozměrů byl nastaven na nulu. V takovém případě se příslušný rozměr vypočítá jako $x = x_0 + 2(f + r)$, kde x_0 je rozměr boxu předhozeného primitivu `\shipout`, f je `\hoffset` resp. `\voffset` a r je `\pdfhorigin` resp. `\pdfvorigin`.

Veškeré odložené operace (`\write` apod.) se provádějí ve chvíli, kdy příslušné místo projde `\shipout`em. Je tedy potřeba zajistit, aby všechna použitá makra byla definována v místě výstupní rutiny. Dokonce když zadáváte odložený `\write`, tak nemusíte mít použitá makra definována, stačí uvnitř výstupní rutiny.

Když se objeví `\end`, zavolá se výstupní rutina. Pokud po ní něco zbylo, vloží se do výstupu `\line{}\vfill\penalty-'10000000000` a znova se zpracovává token `\end`. Zkuste si předefinovat `\line`, vysázet extrémně dlouhý odstavec, a uvidíte, co se stane. Ve chvíli, kdy už není co zpracovat, TEX skončí.

Známé makro `\bye` je definováno takto:

```
\outer\def\bye{\par\vfill\supereject\end}
```

Výstupní rutina plainTEXu

```
\output{\plainoutput}
\def\plainoutput{%
  \shipout\vbox{%
    \makeheadline\box255\makefootline}%
  \advance\pageno by 1 }
\def\makeheadline{\vbox to 0pt{\vskip-22.5pt
  \line{\vbox to8.5pt{}\the\headline}\vss}
  \nointerlineskip}
\def\makefootline{%
  \baselineskip24pt\lineskiplimit0pt
  \line{\the\footline}}
```

Toto je zjednodušená verze výstupní rutiny plainTEXu. Jejím centrem je makro `\plainoutput`, které pošle stránku do výstupu a zvýší číslo stránky. Stránku poskládá tak, že nahoru vloží `\headline` (vhodně vysázenou), pak přidá samotnou stránku `\box255` a nakonec připojí `\footline`.

Ve skutečnosti se ve výstupní rutině plainu dělá trochu víc věcí, například se vkládají poznámky pod čarou.

Může se vám hodit umět nahradit kus výstupní rutiny plainu nějakým jiným kódem. V reálné výstupní rutině je například použito makro `\pagebody` místo `\box255`, které si můžete předefinovat.

Stejně tak můžete potřebovat například jinak pozicovanou hlavičku nebo patičku stránky. Stačí předefinovat příslušné makro.

Úkol 1 [3b]: Definujte makro `\stopoutput`, které vložením do zdrojáku způsobí, že od toho místa dál se na výstup nic nepošle. Definujte také makro `\startoutput` s opačným efektem, které na výstup data pošle. Vaše makro musí fungovat s libovolnou výstupní rutinou – o jejích vlastnostech nesmíte předpokládat prakticky nic.

Při definici neřešte patologické a okrajové případy, stačí, když bude makro fungovat při obvyklém použití (a dokumentujte, co se v tomto případě myslí obvyklým použitím). Například můžete vyžadovat, aby makro nebylo použito uvnitř explicitního hboxu nebo vboxu, nebo zakázat vnoření.

Může se vám hodit vědět, že TEX inkrementuje čítač `\deadcycles` pokaždé, když vstupuje do výstupní rutiny. Pokud jeho hodnota přeteče 25, skončí s chybou, neboť se domnívá, že máte ve výstupní rutině chybu a jste zacyklení. Čítač se nuluje při použití `\shipout`, nebo ho musíte snižovat ručně.

Úkol 2 [9b]: Upravte (vaši nebo vzorovou) implementaci `\multicolumn` z minulé série tak, že bude možno sázet text a další materiál do více sloupců přes více stran, podobně jako sázíme leták KSP.

Neuvažujte poznámky pod čarou, zkuste však implementovat makro tak, abyste umožnili vnoření. `\multicolumn` uvnitř jiného `\multicolumn` prostě vysází vícesloupcovou sazbu uvnitř vícesloupcové sazby.

Stejně tak se pokuste o to, aby se makro chovalo stejně jako v minulé sérii v případě, že jej použijete uvnitř jiného boxu.

Nezapomeňte na dokumentaci.

Formát

Samotný TEX je poměrně holá a osekaná kostra. Umí jen to nejnutnější, zbytek se definuje ve formátu, což je soubor v běžné syntaxi TEXu, který končí příkazem `\dump`. Tím se vygeneruje komprimovaný vnitřní stav TEXu na konci zpracovávání formátu. Během generování formátu platí omezení, že se nesmí vůbec nic vysázet.

TEX tedy umí pracovat ve dvou módech. První z nich jsme používali celou dobu v seriálu. Vezme uložený formát (v našem případě csplain), načte uložené hodnoty do paměti a zpracovává a sází vstup. Ve druhém módu vezme vstup pro formát a vygeneruje jej. Tomu se také říká iniTEX.

Chcete-li TEXu nařídit, jaký formát použít, použijte na příkazové řádce parametr `-fmt` a za něj připojte název formátu. Chcete-li TEX spustit jako iniTEX, použijte parametr `-ini`.

Vzpomenete-li si na první díl a instalaci TeXworks, pak stejně jako pdfcsplain si můžete nastavit TEX s libovolným jiným formátem, když do pole Arguments napíšete správné argumenty.

Například známý LATEX, ConTEXt a další jsou jen různé formáty pro TEX, stejně jako plain.

Nadstavby

Původní TEX má mnohá omezení. Generuje výstup ve formátu DVI („device independent“), což bývalo užitečné v dobách, kdy ještě tiskárny neuměly žádný jednotný jazyk a příkazy v DVI se překládaly přímo do jazyka konkrétní tiskárny jejím ovladačem. Navíc se pracovalo na řádkových terminálech, kde nebylo možné si požadovaný výstup zobrazit.

Současné tiskárny umí prakticky všechny PostScript a před tiskem si prohlížíte PDF. Vytvářet DVI je tedy prakticky zbytečné. Proto vzniknul pdfTEX,[4] který generuje přímo výstup v PDF. Nad rámec toho, co umí TEX, implementuje další užitečné vlastnosti a funkce, například přímé vkládání obrázků, základní práci s barvami apod. Některá z těchto rozšíření jste už v seriálu potkali, konkrétně všechno, co začíná `\pdf...`

V dnešním multilingválním a internacionalizovaném světě je TEX se svým 8bitovým chápáním vstupu silně zastaralý. Světem hýbe UTF-8. Situaci se snaží zachránit encTEX,[5] rozšíření, díky kterému je možno mapovat sekvence 8bitových znaků (například znaky z UTF-8) na sekvence tokenů.

Všechny funkce pdfTEXu a encTEXu by vydaly na samostatnou sérii, tak jen poznamenejme, že běžně dodávaný formát `plain-utf8-cs` se zapnutým encTEXem (argument `-enc` pro iniTEX) je csplain v UTF-8:

```
% vygenerování formátu
pdftex -enc -ini plain-utf8-cs
% použití formátu
pdftex -fmt plain-utf8-cs vstup.tex
```

Jako slibný projekt se pak jeví luaTEX,[6] což je implementace TEXu s možností vkládat do vstupního souboru kusy kódu v jazyce Lua. Ten již pracuje v Unicode a otevírá velmi zajímavé možnosti při psaní maker – některé konstrukce jsou v klasickém TEXu dosti nepraktické, až nemožné (složitější cyklus, opakovaná tokenizace, zavěšená interpunkce apod.). Některé z těchto nedostatků se snaží napravit rozšíření eTEX. Ještě jste se v těch TEXech neztratili?

Obrázky

Obrázky se vkládají primitivem `\pdfximage` (v pdfTEXu). Je možno nadiktovat si rozměry vkládaného obrázku i další parametry vytvářeného objektu ve výsledném PDF. Kompletní syntaxi a možnosti tohoto primitiva najdete v dokumentaci na webu pdfTEXu.

Primitivum `\pdfximage` pouze vloží obrázek jako objekt do PDF. Pokud jej chcete vložit do stránky, potřebujete primitivum `\pdfrefximage`, za které patří číslo objektu. To získáte primitivem `\pdflastximage` pro poslední obrázek vložený do PDF. (Pokud chcete vkládat jeden obrázek do stránky vícekrát, vložte jej do PDF jen jednou a pak se na něj vícekrát odkažte.)

```
\pdfximage width 2cm height 2cm depth 1cm {o.jpg}
\pdfrefximage\pdflastximage
```

Podporované formáty jsou JPEG pro fotografie, PNG pro bitmapovou grafiku, JBIG2 pro dvoubarevné bitmapy a PDF pro vektorovou grafiku.

Obrázek vložený ve stránce se chová jako vrule, resp. hrule. Pokud s ním potřebujete dělat nějaké speciality, zavřete jej do boxu.

Barvy

Každý objekt vykreslený TEXem má nějakou barvu, základní je černá. Její nastavení není v původním TEXu podporováno. V pdfTEXu je nutno vložit přímo kus kódu z formátu PDF.

Nejjednodušší způsob, jak změnit barvu, je přímé nastavení:

```
\def\red{\pdfliteral{1 0 0 rg}}
\def\black{\pdfliteral{0 0 0 rg}}
\def\green{\pdfliteral{0 0.5 0 rg}}
Černý text, \red červený text, \green
zelený text, \black černý text.
```

Černý text, červený text, zelený text, černý text.

Příkaz `rg` nastavuje barvu v prostoru RGB. Tři parametry se uvádí před ním, oddělené mezerou. Jsou to reálná čísla v rozsahu 0 až 1. První je červená, druhé je zelená a třetí modrá složka.

Dejte si pozor na to, že přímý zápis do PDF naprosto ignoruje nějaké uzavření do skupin, které vidí TEX, naopak je třeba uvažovat uzávorkování uvnitř PDF. Barva je nastavena obvykle do konce strany.

[4] `http://www.tug.org/applications/pdftex/`
[5] `http://petr.olsak.net/enctex.html`
[6] `http://www.luatex.org/`

Chcete-li si uložit na zásobník stav grafiky v PDF, můžete použít příkazy `q` a `Q`:

```
% Ulož stav grafiky
\def\beginpdfgroup{\pdfliteral{q}}
% Vrať stav grafiky
\def\endpdfgroup{\pdfliteral{Q}}
```

Analogicky k příkazu `rg` funguje příkaz `k` se čtyřmi parametry, který pracuje v prostoru CMYK, a příkaz `g` s jedním parametrem, jenž nastavuje barvu ve stupních šedé. Vyrábíte-li tedy PDF pro tisk, použijte CMYK, pokud se má výstup zobrazovat na obrazovce, použijte RGB.

Celé je to ještě trochu ztížené tím, že uvedené PDF příkazy platí jen pro čáry. Některé objekty se vykreslují jako výplň. Pokud se ve výstupu objevují objekty, které nerespektují nastavení barev, přidejte k nastavení barvy ještě jednou totéž, ale velkými písmeny. Všimněte si zlomkových čar:

```
\def\red{\pdfliteral{0 1 1 0 k}}
\def\green{\pdfliteral{1 0 1 0 k}}
\def\black{\pdfliteral{0 g}}
\def\Red{\pdfliteral{0 1 1 0 K}}
\def\Green{\pdfliteral{1 0 1 0 K}}
\def\Black{\pdfliteral{0 G}}
\def\fr{{a+b\over c}\quad}
$\displaystyle\fr\red\fr\green\fr\black
\fr\Red\fr\Green\fr\Black\fr$
```

$$\frac{a+b}{c} \quad \frac{a+b}{c} \quad \frac{a+b}{c} \quad \frac{a+b}{c} \quad \frac{a+b}{c} \quad \frac{a+b}{c} \quad \frac{a+b}{c}$$

Formát PDF je daleko mocnější, co se týče barev, ale to už výrazně přesahuje možnosti našeho seriálu. Máte-li zájem o přímé barvy Pantone, ICC profily a další, zeptejte se na fóru.

Úkol 3 [3b]: Implementujte makra pro pohodlnější práci s barvami. Váš balík musí umět definovat barvu v systémech RGB, CMYK a stupních šedé a pohodlně pak definovanou barvu nastavit. Použití může vypadat například takto:

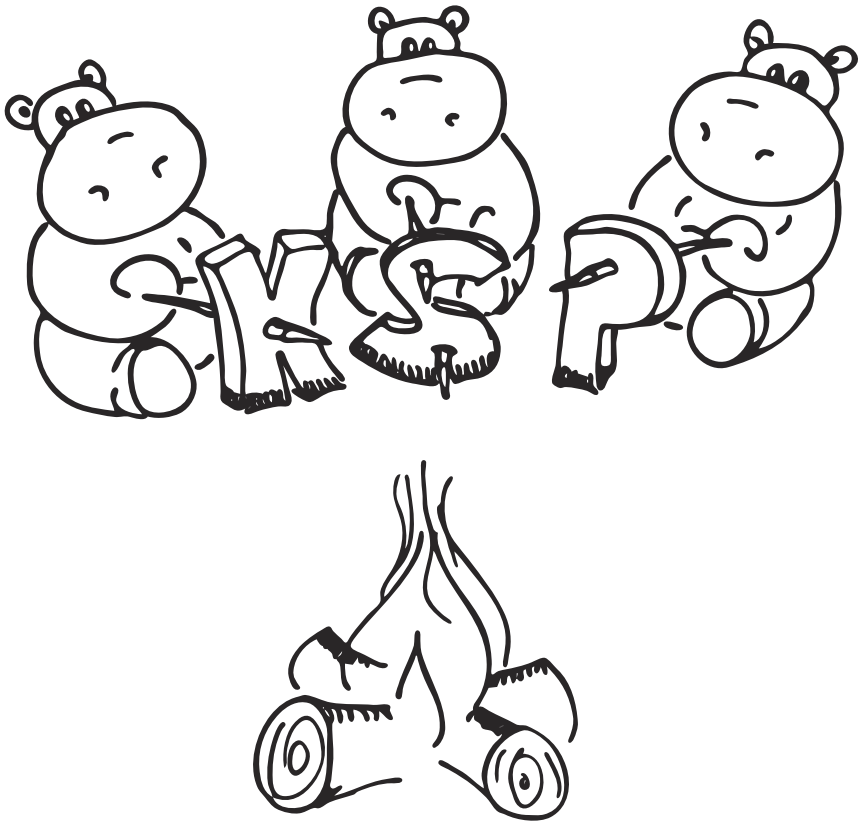
```
\defrgbcolor\red{1 0 0}
\defcmykcolor\green{1 0 1 0}
\defgrayscalecolor\halfgray{0.5}
\defgrayscalecolor\black{0}

Černý, \red červený, \green zelený,
\halfgray šedý, \black černý text.
```

Při řešení úkolů se vám možná budou hodit nějaké triky, které se objeví v řešení čtvrté série. Nezapomeňte tam nahlédnout.

A to je vše, přátelé. Doufám, že TEXu zůstanete věrni i nadále.

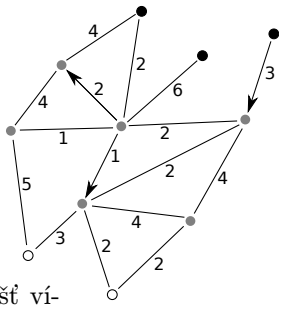
Jan „Moskyto“ Matějka



Ukážeme si uměle znějící úlohu, kterou posléze zmatematizujeme, vyřešíme a dokážeme vlastnosti řešení. Nakonec přijdou četná užití, která ozřejmí, proč jsme se snažili.

Látka je lehce pokročilá, takže vězte, že budete potřebovat znát grafy.

Uměle znějící úloha



Ruský petrobaron vlastní ropná naleziště na Sibiři a trubky vedoucí do Evropy. Trubky vedou mezi nalezišti, uzlovými body a koncovými body, kde ropu přebírají odběratelé.

Každá trubka může a nemusí mít definováno, kterým směrem jí má téci ropa. Pro každou trubku zvlášť víme, kolik nejvýše jí za hodinu protlačíme.

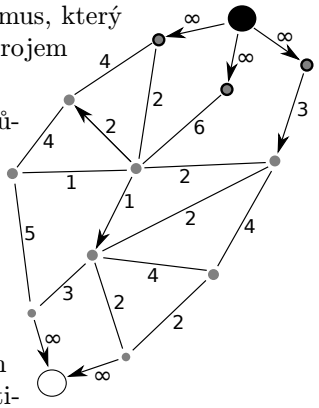
Naleziště jsou bezedná a mohou posílat neomezená množství ropy. Odběratelé také dokáží neomezená množství ropy z koncových bodů odebírat. Petrobaron čelí problému, jak protlačit danou distribuční sítí co nejvíce ropy za hodinu ze zdrojů k odběratelům.

Zapeklité je to zejména kvůli tomu, že v uzlových bodech nelze ropu hromadit, ani pálit – rozhodně tedy nejde bez rozmyslu přikázat, ať každou trubkou teče maximum, protože bychom poškodili cenná zařízení a v uniklé ropě utopili vše živé.

Zmatematizování

V zadání vidíme graf, který obsahuje orientované i neorientované hrany, kde je nějaká podmnožina vrcholů označená jako zdroje a jiná jako… říkejme tomu třeba stoky.

Abychom měli situaci jednodušší, zbavíme se hned na úvod mnohočetnosti zdrojů a stoků. Přikreslíme si dva nové vrcholy – z nadzdroje budeme posílat ropu do všech zdrojů, do nadstoku budeme posílat ropu ze všech stoků. Kapacitu přikreslených hran pak nastavíme na nekonečno.

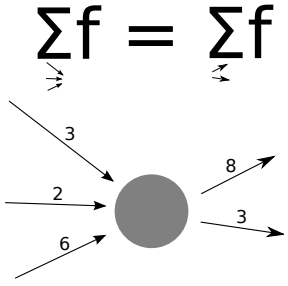


Teď nám stačí vymyslet algoritmus, který řeší problém s právě jedním zdrojem a právě jedním stokem.

Každý vstup totiž popsaným způsobem převedeme, pošleme ho algoritmu a z výstupu prostě jen odstraníme dva přidané vrcholy a připojené hrany.

Podobně se zbavíme neorientovaných hran.

Každou takovou hranu v každém zadání změníme na dvojici protisměrných orientovaných hran se stejnou kapacitou. V algoritmu pak už můžeme počítat jen s hranami orientovanými.



Dostáváme se nyní k nejdůležitějšímu – podmínkám na hledaný tok.

Na vstupu dostáváme ohodnocení hran nezápornými čísly a naším úkolem je sestavit jiné ohodnocení těch samých (všech) hran.

Je důležité, aby se nám to nepletlo – ohodnocení ze vstupu se říká kapacita a značí se $c(e)$, konstruované ohodnocení se jmenuje tok a říkáme mu $f(e)$.

Konstruované ohodnocení se snažíme maximalizovat, ale omezuje nás kapacita a Kirchhoffův zákon.

Tak budeme říkat podmínce na to, že součet toku na hranách, které do vrcholu vstupují, musí být stejný jako součet toku na hranách, které z vrcholu vystupují. Máte-li rádi fyziku nebo berete-li školu vážně, důvod k takovému pojmenování jistě chápete.

Formálně ony dvě podmínky vypadají takto:

$$\forall e \in E : f(e) \leq c(e)$$

$$\forall v \in V \setminus \{z, s\} : \sum_{\overrightarrow{uv} \in E} f(\overrightarrow{uv}) = \sum_{\overrightarrow{vu} \in E} f(\overrightarrow{vu})$$

Kirchhoffova podmínka se samozřejmě netýká ani zdroje, ani stoku – tam nám naopak jde o to ji co nejvíce porušit. Velikost toku je nejsnazší měřit na nich. Budeme ji definovat jako rozdíl mezi součtem odtoků a součtem přítoků ve zdroji.

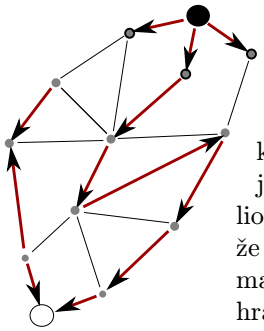
K zamyšlení

- Nastavit ohodnocení hrany (kapacitu) na skutečné nekonečno v našem programovacím jazyce nemusí jít. Pak se to řeší tím, že se zvolí dostatečně velké číslo. Jak co nejmenší, ale stále bezpečné, rychle ze zadání určit? Stejný problém se řeší třeba v Dijkstrově algoritmu, ale i ve spoustě dalších.
- Neorientované hrany, neboli obousměrné trubky, si zaslouží podrobnější rozbor, než jaký jsme jim věnovali v textu. Jak spolehlivě převedeme řešení algoritmu do původní sítě?
- Vymysleli jsme, jak vyřešit více zdrojů a stoků a jak ošetřit obousměrné trubky. Co kdyby bylo v zadání omezení na průtok vrcholy?
- Umíte dokázat, že je absolutní hodnota rozdílu přítoků a odtoků stejná na zdroji i na stoku? Tedy že bychom mohli velikost toku stejně tak dobře měřit i na stoku?

Řešení

Problém je velmi studovaný a k jeho řešení existují dva velké přístupy, které jsou humorně protikladné. Ten první vezme nulový tok a opatrně ho zlepšuje. Druhý si napíská veliké ohodnocení hran, které ani tokem není, a pak ho opravuje.

Předvedeme si onen první způsob a algoritmus, který se podle svých autorů jmenuje Fordův-Fulkersonův. Bude se nám odteď hodit tvářit se, jako že mezi každými dvěma vrcholy vede oběma směry hrana. Tam, kde ze vstupu nepřišla, si domyslíme jednu s nulovou kapacitou.



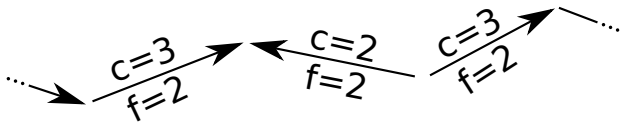
Představme si graf, na kterém počítáme tok a dejme tomu, že už nějaký tok máme – třeba prázdný. Představme si, že jsme ropný magnát a každý rozdíl mezi kapacitou potrubí a jejím využitím (tokem) nás stojí miliony dolarů. Už jsme se smířili s tím, že každá trubka nemůže být využita na maximum, ale zkusme si vyznačit ty hrany, kde $c(e) \neq f(e)$.

Co když existuje cesta z nadzdroje do nadstoku, která vede pouze po takových hranách?

Můžeme vzít minimum z rozdílů na každé hraně a o toto číslo navýšit tok na každé z nich!

Ani kapacitní, ani Kirchhoffovu podmínku to jistě nepoškodí.

Pokud žádnou takovou cestu nevidíme, znamená to, že tok vylepšit nejde? Ne úplně. Představte si následující situaci:



Copak nejde zlepšit? Jde! Není na to první pohled úplně jasné, ale můžeme zlepšovat výsledný tok i tím, že ho na protisměrné části cesty snížíme. Samozřejmě však nesmíme nastavovat tok záporný.

(Je smutné, že si teď trochu kazíme grafovou terminologii – co je to za cestu v orientovaném grafu, která nemusí respektovat orientaci hran?)

Takže jaká je přesně podmínka pro „vyznačení“ hrany $\vec{uv}$? Nastává $f(\vec{uv}) < c(\vec{uv})$ nebo $f(\vec{vu}) > 0$. Potom ji lze zlepšit o $c(\vec{uv}) - f(\vec{uv}) + f(\vec{vu})$.

Hledání všech vhodných („zlepšujících“) cest tedy můžeme dělat prostým prohledáváním do šířky přes vyznačené hrany. Budeme to dělat opakovaně znovu a znovu, až žádnou takovou nenajdeme, a pak vrátíme získaný tok jako výsledek.

Analýza algoritmu

Správnost

Zavolali jsme algoritmus na prázdný tok, ten ho zlepšil do situace, ve které neexistuje zlepšující cesta.

Znamená tato neexistence, že je výsledný tok maximální? Opačná implikace je jasná – maximální tok zlepšit žádným způsobem nepůjde, takže ani přes zlepšující cestičky.

Když zkusíme algoritmus pustit na graf, kde už žádná taková cesta není, můžeme si poznamenat všechny vrcholy, kam jsme se pomocí prohledávání zlepšitelných hran ještě dostali. Tato množina bude jistě obsahovat zdroj (tam jsme začali) a jistě nebude obsahovat stok (to by existovala zlepšující cesta).

Na hranách mezi touto množinou a jejím doplňkem nemůžeme zlepšovat, jinak by se po nich náš program pustil dál a množinu vrcholů, kam se dostal, by rozšířil. Všechny hrany směřující ven tedy mají $f(e) = c(e)$, pro všechny hrany směřující dovnitř platí $f(e) = 0$.

Tyto hrany tvoří *řez* naším grafem. Odvolám se v tuto chvíli na vaši intuici – tok nemůže být větší než libovolný řez. Z toho už dostáváme, že náš algoritmus našel tok maximální, protože našel také řez, který zaručuje, že nemůže existovat tok větší.

Formálnější předvedení najdete ve skriptíčkách z kombinatoriky.[7]

Časová složitost

Je možné dobu běhu omezit počtem vrcholů a hran? Výše uvedeným postupem na grafu s celočíselnými kapacitami každou nalezenou cestou zvýšíme tok alespoň o jednotku, takže program nebude běžet déle, než je součet všech kapacit. Ale to není moc uspokojivý odhad, protože záleží na ohodnocení.

Pokud budeme hledat cesty skutečně prohledáváním do šířky, bude počet kroků v $\mathcal{O}(nm^2)$, protože se dá ukázat, že se hrany, které při zlepšování cesty tvoří minimum, postupně vzdalují od zdroje. Pak máme $\mathcal{O}(m)$ času k nalezení cesty a m hran, které se nejvýše n-krát mohou vzdálit. Že to tak skutečně je, je lehce zdlouhavé intelektuální cvičení. Nechat si prozradit postup můžete třeba v druhém vydání Introduction to Algorithms na straně 662.

O vylepšení daného postupu si můžete přečíst v záznamu[8] z jedné Medvědovy přednášky předmětu ADS2, ukázka druhého přístupu k řešení hledání maximálního toku je na záznamu[9] jejího pokračování.

K zamyšlení

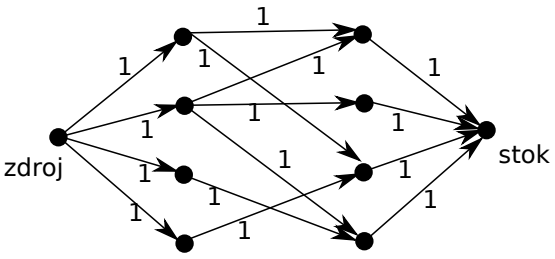
- Důležitou vlastností algoritmu je, že když dostane celočíselné kapacity, vrátí celočíselný tok. Bude se nám to hodit v aplikacích. Dokážete to?
- Rozdíl mezi Fordem-Fulkersonem, který hledá cesty obecným způsobem, a takovým, který to dělá prohledáváním do šířky, je ze složitostního hlediska docela velký, a proto se tomu druhému občas říká Edmondsův-Karpův. Najděte malý graf a nevhodnou posloupnost cest, která způsobí, že F-F poběží skutečně v závislosti na velikosti kapacit.
- Můžete dokonce zkusit využít zlatého řezu k nalezení grafu s reálnými kapacitami, na kterém F-F pro danou (nešikovnou) posloupnost cest nikdy neskončí.
- Skončí algoritmus v konečném čase, jsou-li kapacity čísla racionální?

Užití

Párování v bipartitních grafech

Máme-li za úkol najít na plese co nejvíce tanečnicím tanečníka, kterého znají, stojíme před zásadním a nelehkým úkolem.

Co třeba postavit na základě známosti bipartitní graf mezi partitou tanečníků a partitou tanečnic, přidat zdroj za kluky a stok za holky, tyto k nim připojit hranami s jednotkovou kapacitou, hranám v bipartitním grafu také nastavit jednotkové kapacity a nakonec všechno zorientovat směrem do stoku?



Maximální celočíselný tok, který na tomto grafu získáme, nám hrany bipartitního grafu rozdělí na nevybrané s tokem 0 a vybrané s tokem 1. Můžou vybrané hrany sdílet tanečníka? Těžko, když do něj teče nejvýše jednotkový tok a musí platit Kirchhoffův zákon. A podobně s tanečnicemi.

Vybrané hrany nám proto vytvoří párování. A protože jsme našli maximální tok, jde o párování největší. Kdyby existovalo párování větší, dokázali bychom z něj zvětšit tok.

[7] http://kam.mff.cuni.cz/~valla/kg.html

[8] http://mj.ucw.cz/vyuka/1112/ads2/3-dinic.pdf

[9] http://mj.ucw.cz/vyuka/1112/ads2/4-goldberg.pdf

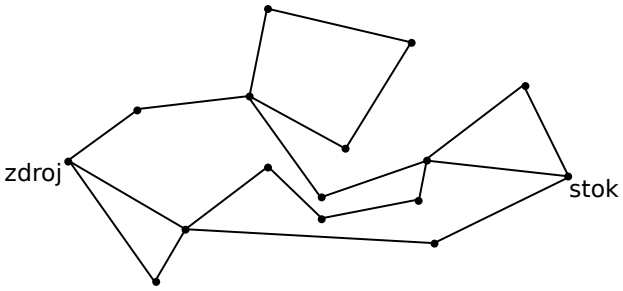
Hledání hranově a vrcholově disjunktních cest

Chceme-li se v grafu G dostat z vrcholu u do vrcholu v, může nás zajímat (třeba kvůli spolehlivosti, s jakou se umíme dostat do cíle), kolik mezi nimi existuje cest, které:

- nesdílí hrany, nebo
- nesdílí vrcholy. (Tato podmínka je silnější. Když dvě cesty nesdílí vrcholy, nesdílí hrany.)

Oba tyto problémy lze převést na hledání maximálního toku. V obou případech nastavíme u jako zdroj a v jako stok. V prvním případě nastavíme jednotkové kapacity všem hranám, v druhém navíc všem vrcholům.

Ford-Fulkerson nastavil některým hranám jednotkový tok, některým nulový. Nulové nyní z grafu vyhodíme. Pokud jsme hledali hranově disjunktní cesty, můžeme nyní získat třeba takovýto graf:



Jak z něj vykřesat kýžený výsledek? Začneme procházet ze zdroje zbylé hrany. Vždy, když se dostaneme do vrcholu, ve kterém už jsme v tom samém průchodu byli, vyhodíme z grafu všechny hrany cyklu, který jsme tímto objevili. (Hodnota toku se tím nezmění.)

Průchodem grafu se vždy můžeme dostat až do stoku (všude jinde budeme moci podle Kirchhoffova zákona jít dál – dost to připomíná úvahu o eulerovských tazích)[10] a protože jsme mezitím agilně odstraňovali cykly, dostali jsme cestu. Vrátíme ji jako jeden výsledek, smažeme její hrany a pokud ještě tok není nulový, pokračujeme dál.

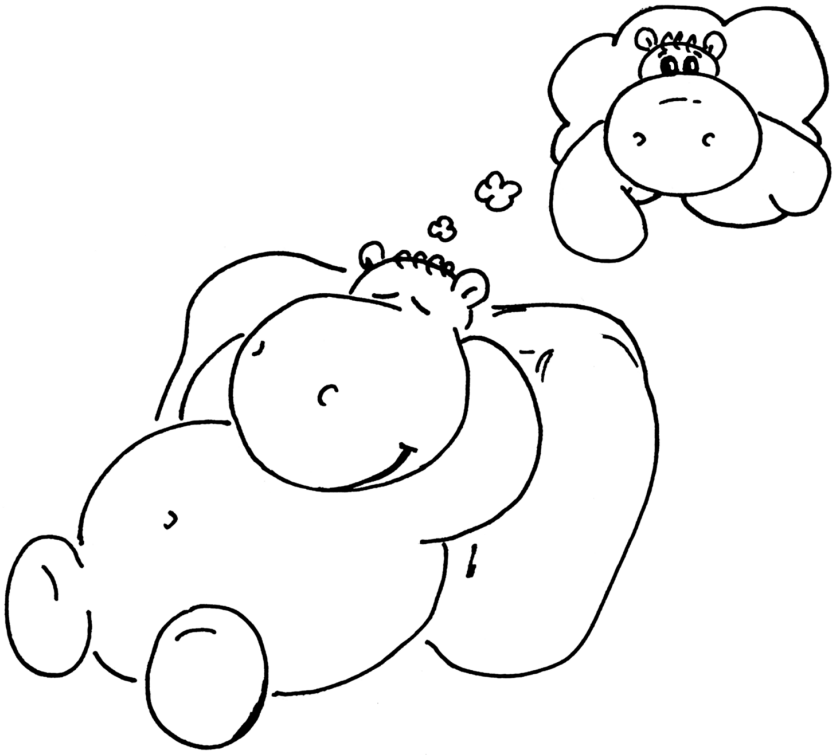
Počet cest je tedy velikost toku. Podle Mengerovy věty je navíc počet hranově/vrcholově disjunktních cest roven stupni hranové/vrcholové souvislosti grafu – máme tedy nyní algoritmus, který ji najde.

K zamyšlení

- Úvaha nebyla naprosto přímočará kvůli cyklům v nalezeném toku. Říká se jim cirkulace. Je jasné, že v případě hledání hranově disjunktních cest vzniknout mohou. Co v případě vrcholově disjunktních, tedy v situaci, kdy jsme omezili tok vrcholy?
- Nepracuje náhodou neupravený Edmondsův-Karpův algoritmus rychleji, pokud je graf, jak jsme teď opakovaně viděli, ohodnocený toliko nulami a jedničkami?

Dnešní menu servíroval

Lukáš Lánský



[10] `http://ksp.mff.cuni.cz/viz/kucharky/eulerovske-tahy`

25-4-1 Přesmyčky

Při řešení této úlohy budeme pro jednoduchost předpokládat, že K se nám vejde do nějaké normální proměnné, a tedy že s ním ještě dokážeme provádět aritmetické operace v konstantním čase (v opačném případě bychom pak jen časovou složitost museli vynásobit $\log K$). Druhou věcí, kterou jsme v zadání asi ne úplně přesně uvedli, je to, že operujeme s konstantně velkou abecedou (26 písmen). Pokud by však abeceda byla větší, tak bychom její velikostí museli časovou i paměťovou složitost vynásobit. Při hodnocení vašich řešení však ani jedna z možností neměla na bodový zisk vliv, protože jsme zadání zformulovali volně.

Lehčí varianta

Nejdříve provedeme několik pozorování. Pro jednodušší případ a slovo délky N máme přesně $N!$ možností, jak můžeme toto slovo uspořádat. Když však první písmeno zvolíme pevně, tak máme již jen $(N-1)!$ možností uspořádání zbylých písmen.

Přesmyčka začínající na lexikograficky nejmenší písmeno tak může mít pořadové číslo v rozsahu $1, \ldots, (N-1)!$, přesmyčka začínající na v pořadí druhé písmeno může mít pořadové číslo mezi $(N-1)!+1, \ldots, 2\cdot(N-1)!$ atd. Pokud tedy má hledaná přesmyčka pořadové číslo K, tak jako první znak zvolíme písmeno s pořadovým číslem k (indexujeme od nuly):

$$k = \left\lfloor \frac{K}{(N-1)!} \right\rfloor$$

Tím jsme vyřešili první znak, jak s ostatními? Stačí si uvědomit, že vlastně hledáme nějakou přesmyčku s pořadovým číslem K' na $N-1$ zbylých znacích. Stačí nám od původního K odečíst tolik přesmyček, kolik jsme jich volbou k-tého písmene přeskočili. Tedy zvolíme $K' = K - k\cdot(N-1)!$ a rekurzivně postupujeme pro celé slovo (jen v každém kroku nesmíme zapomenout brát k-té písmeno jen ze zatím nepoužitých písmen).

Implementace je v tomto případě jednoduchá, jen si přepočítáváme průběžně K a N. Pro nalezení a průběžné odmazávání k-tého písmena v pořadí můžeme použít pole nebo nějaký vyhledávací strom.

Těžší varianta

V případě opakování písmen se nám úloha mírně komplikuje. Po zvolení prvního znaku již nemáme právě $(N-1)!$ možností poskládání zbytku slova, ale pokud si jako m označíme počet různých znaků a jako p_i pro i od 1 do m jejich četnosti, tak je to:

$$\frac{(N-1)!}{p_1!\cdot p_2!\cdot \ldots \cdot p_m!}$$

(můžeme si všimnout, že to přesně odpovídá jednoduššímu případu pro všechny četnosti rovny jedné).

Postup je pak už stejný jako v jednodušším případě, jen musíme vymyslet, jak budeme rychle upravovat tento vzorec. Při snížení faktoriálu v čitateli o jedna ho jen vydělíme odpovídajícím N, při snížení četnosti některého z písmen z hodnoty p_i na $p_i - 1$ ho vynásobíme p_i. Obě tyto operace zvládneme stejně rychle jako jiné aritmetické operace.

Paměťová složitost je $\mathcal{O}(N)$, protože si musíme všechny znaky přečíst do paměti a ke každému si pamatovat konstantně mnoho údajů, jako je četnost (můžeme dokonce odhadnout paměťovou složitost jako $\mathcal{O}(m)$, ale m může být až N a tedy se složitost asymptoticky nezmění).

Časová složitost je také lineární k délce vstupu, tedy $\mathcal{O}(N)$. Pokud bychom však pracovali s velkým vstupem a velkou abecedou (viz poznámka v úvodu řešení), tak by se nám změnila až na $\mathcal{O}(N \cdot L \log K)$. Vzorový program implementuje těžší variantu.

Program (C):
`http://ksp.mff.cuni.cz/viz/25-4-1.c`

Jirka Setnička

25-4-2 Plánování trasy

Úloha vypadala na první pohled velmi jednoduše. Proto se do ní pustila téměř polovina z vás, kteří jste poslali řešení alespoň jedné z úloh čtvrté série. Úloha však skrývala několik záludností. Podívejme se na řešení, které se jim vyhýbá.

Nejprve si pro každé políčko předpočítáme vzdálenosti od překážek ve všech čtyřech směrech. Díky tomu pak v programu dokážeme okamžitě určit, na kterém místě budeme příště zatáčet, pokud se vydáme daným směrem.

Vzdálenosti od levé překážky určíme tak, že projdeme postupně celou mapu po řádcích zleva doprava. Pokud je první políčko na řádku volné, přiřadíme mu vzdálenost rovnou nule. Pokud volné není, přiřadíme mu číslo -1. Každému dalšímu políčku, které je volné, přiřadíme vždy hodnotu o jedna větší. Políčkům, která volná nejsou, přiřadíme opět hodnotu -1.

Podobně vypočítáme vzdálenosti od překážek v ostatních směrech: od pravé překážky postupujeme po řádcích zprava doleva, od horní překážky po sloupcích shora dolů a od dolní překážky po sloupcích zdola nahoru.

K čemu nám tato čísla pomohou? Když se z libovolného políčka vydáme některým směrem, budeme vědět, že na překážku narazíme až v políčku, které má příslušnou souřadnici větší nebo menší o takto vypočtenou vzdálenost. Pouze v těchto bodech budeme měnit směr. Libovolnou trasu pak popíšeme jako posloupnost políček, na nichž jsme směr měnili.

Zbývá zajistit, abychom nepřejeli přes cílové políčko. K tomu nám může pomoci malý trik. Pokud se při úvodním výpočtu vzdáleností dostaneme do políčka s cílem, hodnotu vzdálenosti vynulujeme. Tím zabezpečíme, že se zastavíme v cílovém políčku a nepřejedeme je až k následující překážce.

Celý předvýpočet dokážeme provést v čase $\mathcal{O}(MN)$, kde M a N jsou rozměry mapy. Mapu totiž projdeme čtyřikrát, počet průchodů je tedy konstantní.

Teď již můžeme hledat trasu od startu do cíle, která bude obsahovat co nejméně zatáček, druhotně co nejméně políček.

Při hledání optimálních cest se často vyplatí použít nějakou úpravu algortimu prohledávání do šířky. Prohledávání do šířky je grafový algoritmus. Přečtěte si o něm v grafové kuchařce.[11]

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

Nyní si místo mapy představme graf, v němž vrcholy odpovídají políčkům změn směrů a hrany odpovídají rovným trasám mezi nimi. Samotný algoritmus prohledávání do šířky nám zajistí minimalizaci počtu obratů.

Potřebujeme ještě mezi trasami se stejným počtem obratů vybrat tu nejkratší. K vrcholům, k nimž při prohledávání do šířky dorazíme, si poznamenáme počet políček, která jsme museli na celé trase od startu k nim překonat. Tuto hodnotu nebudeme nikdy zvyšovat a přepíšeme ji jenom v případě, že tím nezvýšíme počet zatáček na cestě do daného vrcholu.

Na závěr si jenom musíme dát pozor: nemůžeme se zastavit okamžitě, když dojdeme do cíle, ale až tehdy, kdy cílové políčko vyndáváme z fronty.

Složitost celého algoritmu je $\mathcal{O}(MN)$, tedy lineární s počtem políček. Je tomu tak proto, že vrcholů není více než políček mapy a z každého vrcholu vedou maximálně čtyři[12] hrany.

Program (C):
`http://ksp.mff.cuni.cz/viz/25-4-2.c`

Jirka Setnička a Jenda Hadrava

25-4-3 Rozpis svozu

Podobně jako u úlohy 25-3-3[13] i tentokrát přímočaré řešení spočívalo ve vyzkoušení všech políček, spočítání příslušné námahy a průběžném přepisování minima.

I tentokrát by takové řešení bylo dost pomalé, přesněji by mělo časovou složitost $\mathcal{O}((NM)^2)$ na každý dotaz, pro K dotazů tedy celkem $\mathcal{O}(KN^2M^2)$. Pokud by M i K řádově odpovídaly N, máme $\mathcal{O}(N^5)$.

Pojďme se tedy zase podívat, jestli to umíme lépe. A začněme bližším prozkoumáním toho, jak se počítá námaha a co z toho plyne.

Nechť t_p je množství trávy na políčku p a p_x, resp. p_y jsou souřadnice políčka p. Námaha na svoz trávy z každého políčka p v nějaké oblasti na políčko se souřadnicemi $[x,\ y]$ pak odpovídá výrazu:

$$\sum_p (|p_x - x| + |p_y - y|) \cdot t_p$$

Tenhle vzoreček můžeme ale roznásobením a rozepsáním upravit na tvar:

$$\left(\sum_p |p_x - x| \cdot t_p\right) + \left(\sum_p |p_y - y| \cdot t_p\right)$$

Právě jsme ukázali, že souřadnice jsou nezávislé, takže můžeme nezávisle na sobě hledat nejvýhodnější sloupec a nejvýhodnější řádek.

Samo zadání upozorňovalo na podobnost s úlohou minulé série 25-3-3, pojďme tedy prozkoumat, jestli úlohu neumíme převést na jednorozměrnou variantu. Ta pracovala s prefixovými součty trávy a prefixovými součty těchto prefixových součtů na jediném řádku.

Uvažujme bez újmy na obecnosti, že hledáme nejvýhodnější sloupec. Při dotazu na oblast bychom tak potřebovali mít k dispozici nikoli prefixové součty pro řádek, ale pro oblast, resp. součty prefixových součtů přes všechny řádky oblasti.

Představme si, že pro každé políčko víme, kolik námahy stojí svézt do něj trávu z oblasti vymezené levým horním rohem a naším políčkem, to celé za předpokladu, že přesuny po y-ové ose máme zadarmo. Námahu tedy počítáme pouze za přesuny doprava a doleva.

Řekněme, že tuto námahu máme v poli $S\ell$, podobně v Sr budeme mít námahu pro svoz z oblasti vymezené pravým horním rohem a naším políčkem. Ještě se nám budou hodit pole $P\ell$, resp. Pr udávající, kolik je v těchto oblastech celkem trávy.

Nechť máme oblast vymezenou souřadnicemi $[x,\ y]$ a $[X,\ Y]$ a chceme spočítat námahu za svoz trávy na políčko $[a,\ b]$. Stejně jako v 1D variantě si námahu rozdělíme na námahu za svoz zleva a námahu za svoz zprava.

Námaha zleva bude $S\ell_{a,Y} - S\ell_{a,y-1} - (S\ell_{x-1,Y} - S\ell_{x-1,y}) - (P\ell_{x-1,y} - P\ell_{x-1,y-1} \cdot (a - (x-1)))$. Základem je $S\ell_{a,Y}$. $S\ell_{a,b}$ totiž bere v úvahu pouze řádky $0 \ldots b$, zatímco $S\ell_{a,Y}$ pokrývá celou zadanou oblast. Připomeňme ještě, že pro hodnoty $S\ell$ počítáme s tím, že přesuny nahoru a dolů máme zadarmo.

Rozdílem $S\ell_{a,Y} - S\ell_{a,y-1}$ jsme tedy získali námahu za přesun veškeré trávy z oblasti $[1,\ y]$, $[a,\ Y]$ na políčko $[a,\ Y]$ (nebo kterékoli jiné v sloupci a).

Dál jsme podobně jako v 1D variantě odečetli námahu za svoz trávy z oblasti $[1,\ y]$, $[x-1,\ Y]$ na políčko $[x-1,\ Y]$ a nakonec námahu na přesun trávy ze stejné oblasti mezi políčky $[x-1,\ Y]$ a $[a,\ Y]$.

Stejným způsobem můžeme spočítat námahu za svoz trávy zleva. Ideální sloupec tedy můžeme najít stejně jako v jednorozměrné variantě úlohy upraveným binárním vyhledáváním tak, že vždy porovnáme námahu pro dvě sousední políčka.

Podobně dokážeme najít ideální řádek. Místo $S\ell$, resp. Sr budeme mít Rh, resp. Rd (shora, zdola).

Zatím jsme předpokládali, že všechna pomocná pole máme k dispozici, ale neukázali jsme, že si je opravdu umíme opatřit. Pojďme to teď napravit.

Pole $P\ell$ vyrobíme iterováním přes řádky. Na začátku máme $P\ell_{x,0} = 0$. Pro každý řádek si pamatujeme dosavadní součet trávy na tomto řádku, řekněme s, pak platí $P\ell_{x,y} = P\ell_{x,y-1} + s$.

Pro pole $S\ell$ platí $S\ell_{0,y} = 0$ a $S\ell_{x,y} = S\ell_{x-1,y} + P\ell_{x-1,y}$ (potřebujeme vynaložit námahu na svoz trávy do vedlejšího sloupce a pak všechnu dosud potkanou trávu převézt ještě o jeden sloupec dál). Podobně $Rh_{x,0} = 0$, $Rh_{x,y} = Rh_{x,y-1} + P\ell_{x,y-1}$.

Pravostranné varianty, resp. varianta zdola, fungují stejným způsobem.

Předpočítat pomocná pole tedy dokážeme v lineárním čase. Výpočet námahy umíme konstantně, vyhledání optimálního sloupce tak umíme v $\mathcal{O}(\log N)$, optimálního řádku v $\mathcal{O}(\log M)$. Celková složitost tedy je $\mathcal{O}(MN + K(\log N + \log M))$. Paměťová složitost je $\mathcal{O}(NM)$.

Program (C):
`http://ksp.mff.cuni.cz/viz/25-4-3.c`

Karolína „Karryanna“ Burešová

[12] Stačí dvě hrany. Snadno nahlédneme, že návrat se nikdy nevyplatí a mimo cílové a startovní pole nelze pokračovat rovně.

[13] `http://ksp.mff.cuni.cz/viz/25-3-3`

25-4-4 Podplácení

Úloha byla velmi snadná a v drtivé většině jste si s ní hravě poradili. Pojďme si pro ty, co ji neřešili, řešení ukázat.

Dokážeme, že první hráč má vyhrávací strategii, a to pro libovolné N.

První případ nastává pro N lichá. V takovém případě první hráč podplatí policistu ve prostředku poslední řady. Tím vzniknou dvě stejné pyramidy o délce základny $(N-1)/2$. Jakkoli teď zahraje druhý hráč, zahraje v dalším tahu první hráč úplně stejně na druhé pyramidě. Taková strategie se nazývá zrcadlová. Je snadno vidět, že poslední bude táhnout právě první hráč.

Pro sudá N je situace obdobná. První hráč podplatí prostředního policistu v předposlední řadě, čímž vzniknou opět dvě stejné pyramidy. Stačí hrát opět zrcadlově a vítězství je v kapse.

Jan Bok

25-4-5 Účetnictví

Jedno řešení, které můžeme rychle zamítnout, je zkoušet všechny možnosti. Počet způsobů roste plus minus exponenciálně rychle.

Něco nad polovinu bodů dostali ti, které osvítilo dynamické programování. Řekněme, že víme, kolika způsoby je možné se po pěti dnech dostat na všechny částky, které můžeme mít na účtu: 0 Kč tam můžeme dostat třeba pěti způsoby, 1 Kč dvěma, atd.

Kolika způsoby se můžeme do nějaké částky X dostat za šest dní? V šestém dni jsme mohli buď přidat 6 Kč, nebo je odebrat. Stačí tedy sečíst, kolika způsoby jsme se zvládli za pět dní dostat do $(X-6) \bmod N$ a $(X+6) \bmod N$. (Připomeňme si, že N značí číslo, kterým úřad modulí, a K je počet dní naší defraudace.)

Můžeme si takhle postupně stavět počty způsobů, a jakmile projdeme všechny dny, vypíšeme, kolika způsoby se můžeme vrátit na nulu. Jak dlouho tohle bude trvat? $\mathcal{O}(NK)$: pro každý den musíme přepočítat počet způsobů jak se dostat do všech N možných částek. Mohlo by se zdát, že budeme potřebovat i $\mathcal{O}(NK)$ paměti, protože pro každý den počítáme počty způsobů, ale dokážeme to i s $\mathcal{O}(N)$. Stačí si totiž ukládat vždy jenom počty způsobů v předchozím dni a do dočasného pole postupně přičítat způsoby v dalším dni.

Program (C):
`http://ksp.mff.cuni.cz/viz/25-4-5.c`

Na plný počet bodů dosáhli ti, které napadl krok stranou – vyjádření přes matice a jejich rychlé násobení.

Učiníme drobné pozorování: každých N dní algoritmus dělá v podstatě to samé! Když třeba přidáváme $N+10$ Kč, je to stejná operace, jako kdybychom přidávali jenom 10 Kč.

Použijeme trik a uložíme si do matice (třeba jménem M) popis toho, co se s počty způsobů jak dosáhnout jednotlivé částky stane, když přidáme nebo odebereme nejdřív 1 Kč, pak 2 Kč, pak 3 Kč, a tak dále až do N. A takovouhle matici si můžeme vystavět například tak, že si vytvoříme matice „přesuň 1 Kč“, „přesuň 2 Kč“, ..., a vynásobíme je.

Když M umocníme na $\lfloor K/N \rfloor$, dostaneme tím matici, která spočítá počty způsobů po $K-(K \bmod N)$ dnech. Násobit matice velikosti $N \times N$ umíme za čas $\mathcal{O}(N^3)$.[14] Takových násobení provedeme $\mathcal{O}(K/N)+\mathcal{O}(N)$ – první člen je za „skok“ na den $K-(K \bmod N)$, druhý za dopočítání do K. Celkem by to tedy trvalo $\mathcal{O}(KN^2)+\mathcal{O}(N^3)$, ale protože v naší úloze je K podstatně větší než N, zpomaluje nás nejvíc $\mathcal{O}(KN^2)$.

S tímhle členem ale ještě umíme zamávat. Mocnění matice M na K/N přece umíme rychleji než za $\mathcal{O}(K/N)$ násobení! Můžeme použít trik popsaný v kuchařce o teorii čísel,[15] kterými $\mathcal{O}(K/N)$ umoříme na $\mathcal{O}(\log(K/N))$.

Když použijeme rychlé mocnění matic, najednou vypadá složitost už o něco lépe: $(\mathcal{O}(\log(K/N))+\mathcal{O}(N)) \cdot \mathcal{O}(N^3) = \mathcal{O}(N^2 \log K)+\mathcal{O}(N^4)$. Teď nás zase ale straší $\mathcal{O}(N^4)$. Toho se ale dokážeme zbavit. Pochází totiž z násobení matic, které posouvají o 1 Kč, 2 Kč, ... Takovými maticemi jde ale násobit rychleji než v $\mathcal{O}(N^3)$, protože každý řádek obsahuje právě 2 nenulové prvky – nemusíme počítat celý skalární součin řádku a sloupce, stačí ze sloupce sečíst ty dva prvky, které chceme.

Tímhle krokem stranou jsme umlátili časovou složitost do $\mathcal{O}(N^2 \log K + N^3)$. Na první pohled vypadá zlověstněji než $\mathcal{O}(NK)$ (už jenom kvůli mocninám, v jakých se v ní vyskytuje N), ale pro $N = 250$, $K = 10^9$ vyjde podstatně lépe.

Pro úplnost ještě uveďme paměťovou složitost, i když na ní příliš nesejde. Sice počítáme $\log K + N$ matic velikosti $N \times N$, ale většinu z nich stejně zahodíme: budeme potřebovat jenom matici posouvající o $1, \ldots, K \bmod N$ a matici posouvající o $1, \ldots, N$. Vejdeme se tedy do $\mathcal{O}(N^2)$.

Program (C) – maticová varianta:
`http://ksp.mff.cuni.cz/viz/25-4-5-matice.c`

Michal Pokorný

25-4-6 Triády

Kdo se do úlohy pustil, triády by hledal spolehlivě, pokud by však měl dost výpočetního času. Jen nemnozí řešitelé zvládli přijít na relativně rychlé řešení.

Jednoduché řešení za pár bodů se prostě podívá na každou trojici karet a ověří, jestli netvoří triádu. Takto dostaneme časovou složitost $\mathcal{O}(n^3k)$ a paměťovou $\mathcal{O}(nk)$. Faktor k ve složitosti je důležitý, neboť potřebujeme čas $\mathcal{O}(k)$ na ověření, jestli trojice tvoří triádu.

Základní myšlenka asymptoticky rychlejšího řešení nebyla těžká: podíváme se na každou dvojici karet, dopočítáme k nim, jak by měla vypadat třetí karta, a zkusíme ji vyhledat.

Základním pozorováním je, že pro danou dvojici karet máme jednoznačně určenu kartu, která s nimi může tvořit triádu. Pokud se totiž na jedné vlastnosti dané dvě karty shodují, musí mít stejnou hodnotu na této vlastnosti i třetí karta. Jestliže jsou na nějaké vlastnosti dvě karty různé, třetí karta musí mít tu jedinou hodnotu, kterou nemají dané dvě karty.

Nyní už zbývá jenom umět najít třetí kartu. Jedním z řešení je na začátku setřídit karty (stačí i kvadraticky). Pak pro každou dvojici binárně vyhledáme, kde by se třetí karta měla nacházet, a ověříme, jestli tam skutečně je. Ještě je potřeba doplnit ověření, že jsme našli skutečně novou

[14] Kdybychom chtěli, můžeme rychlost násobení matic vylepšit, ale v téhle úloze to není potřeba.

[15] `http://ksp.mff.cuni.cz/viz/kucharky/teorie-cisel`

kartu, pokud jsme dostali dvojici identických karet. Takto dosáhneme složitosti $\mathcal{O}(n^2 \log n \cdot k)$.

Ještě rychlejšího řešení dosáhneme pomocí písmenkového stromu neboli trie. (Všimněte si skryté a neplánované nápovědy, totiž podobnosti slov triáda a trie.) Nyní si trii stručně popíšeme, jejich podrobnější vysvětlení najdete v kuchařce o hledání v textu.[16]

Trie je zakořeněný strom, který se staví pro nějakou množinu slov v dané abecedě. Kořen odpovídá prázdnému slovu, synové kořene znakům, kterým začíná nějaké slovo, čili jednoznakovým prefixům. Pokud více slov začíná jedním znakem, syn s tímto znakem je jen jeden. V další úrovni stromu budou dvouznakové prefixy slov (prefix je souvislá část slova, která obsahuje začátek), ve třetí úrovni stromu budou tříznakové prefixy a tak dále.

Stavba trie probíhá tak, že se začne s kořenem a postupně se přidávají slova. Slovo přidáme jednoduše tak, že jdeme do vrcholů odpovídajícím aktuálnímu znaku slova. Pokud vrchol chybí, doplníme ho a přejdeme na další znak.

My použijeme trii na karty, které si můžeme představit jako slova o délce k v abecedě 1, 2, 3. Na začátku algoritmu tedy všechny karty naskládáme do trie. U každého listu v trii si navíc budeme pamatovat, kolik karet k němu náleží, abychom poznali, že tři karty jsou stejné. Pokud se v nějakém listu počet dostane na 3, hned ohlásíme triádu a můžeme skončit.

Pak pro každou dvojici karet dopočteme třetí a zkusíme ji vyhledat v trii. Uspějeme-li, máme triádu. Pokud se třetí karta neliší od karet z dané dvojice, nemusíme ji hledat, neboť identické karty jsme ošetřovali při stavbě trie. Díky tomu také nemusíme ověřovat, jestli jsme v trii našli skutečně novou kartu, tedy že jsme nenalezli jednu z karet z dané dvojice.

Hledání v trii zabere čas $\mathcal{O}(k)$, takže celková časová složitost je $\mathcal{O}(n^2 k)$. V paměti se trie vejde do prostoru velikosti $\mathcal{O}(nk)$, neboť každá vlastnost každé karty vytvoří maximálně jeden nový vrchol. Paměťová složitost tedy je $\mathcal{O}(nk)$.

Umíte řešit úlohu asymptoticky rychleji, když k může být velké? Pak budeme rádi, když se s námi o řešení podělíte.

Mimochodem, pokud by k bylo zhruba logaritmicky velké oproti n (což dle zadání nebylo povoleno), vyplatilo by se karty skládat do k-dimenzionální krychle o hraně 3 a procházet všechny úsečky krychle, jež tvoří triádu. To už však přesahuje rámec tohoto řešení.

Pavel „Paulie“ Veselý

25-4-7 Šifrovací knoflíky

Úloha, v té verzi jak jsme ji zadali, se nakonec ukázala být o něco lehčí než jsme původně zamýšleli. Nejdříve ukážeme postup, jakým budeme knoflíky otáčet a pak ukážeme, že tento postup opravdu projde všechny možnosti a skončí opět v počáteční pozici.

Knoflíky si očíslujeme čísly $0, 1 \ldots, n-1$ a kroky otáčení si očíslujeme $1, 2, \ldots, n^k$.

Nejprve tedy postup otáčení. Celkový počet možností, které musíme navštívit, je n^k, a takový je i celkový počet otočení. Stačí tedy jen určit, kdy otáčíme kterým knoflíkem. V kroku i otočíme knoflíkem j takovým, že j je největší číslo, které splňuje $n^j \mid i$ (n^j beze zbytku dělí i).

Nyní nahlédneme, že platí následující dvě tvrzení:

1. Mezi dvěma otočeními knoflíku s číslem větším nebo rovným j se na knoflících $\{1, \ldots, j-1\}$ vystřídají všechny možné kombinace.
2. Po provedení n^k kroků budou všechny knoflíky v počátečních pozicích.

Tvrzení 1 dokážeme matematickou indukcí podle j. Pro $j = 0$ je to jasné, pro $j = 1$ si všimneme, že knoflík s číslem alespoň 1 se otočí každý n-tý krok a zbylých n kroků se otočí knoflík číslo 0. Tedy se na něm opravdu vystřídají všechny možnosti.

Nyní budeme předpokládat, že tvrzení platí pro $j-1$ a dokážeme, že platí pro j. Z podmínek pro otáčení vidíme, že mezi tím, co dvakrát otočíme knoflík s číslem alespoň j, otočíme $(n-1)$-krát knoflíkem $j-1$ a jelikož mezi každými těmito dvěma otočeními se nám na knoflících $0, \ldots, j-2$ vystřídají všechny možnosti, tak po $n-1$ opakování se nám vystřídají všechny možnosti na knoflících $0, \ldots, j-1$. A to jsme přesně chtěli.

Teď nám jen zbývá dokázat Tvrzení 2. Chceme ukázat, že počet otočení každého knoflíku je dělitelný číslem n. To dokážeme také indukcí, ale tentokrát budeme postupovat z druhé strany, od knoflíku s největším číslem. Ten se otočí pokaždé, když $n^{k-1} \mid i$, což se stane právě n-krát.

Nyní provedeme indukční krok. Předpokládáme, že knoflíky s čísly $k-1, k-2, \ldots, j+1$ skončí v počáteční pozici a ukážeme, že pak i knoflík s číslem j skončí v počáteční pozici. Knoflík j se otočí právě $(n^{k-j} - l)$-krát, kde l je počet otočení větších knoflíků. A jelikož víme, že počet otočení všech větších knoflíků je dělitelný číslem n, tak i počet otočení knoflíku j je dělitelný n. A máme vyhráno.

Na závěr se ještě podívejme na časovou složitost algoritmu. Otočení knoflíku provádíme celkem n^k-krát. Podmínky na dělitelnost budeme zkoušet postupně od nejnižšího j. Spočítáme, kolikrát kterou podmínku testujeme. První podmínku testujeme pokaždé, druhou podmínku jen pokud je splněna první, tedy n^{k-1}-krát. Všechny podmínky dohromady testujeme v čase $\sum_{i=0}^{k-1} n^{k-i} = \mathcal{O}(n^k)$.

V každém kroce vypíšeme jen číslo knoflíku, s kterým otáčíme. Časová složitost je tedy $\mathcal{O}(n^k)$. Lepší ani být nemůže, protože algoritmus vydává takto velký výstup.

Karel Tesař

Alternativní řešení

Pro každé n a k chceme najít $R_{n,k}$, posloupnost otáčení k knoflíků s n pozicemi takovou, že každou možnou konfiguraci projde právě jednou a z koncové konfigurace se lze jedním otočením dostat zpět do počáteční. To je jen drobná přeformulace zadání, kde poslední „návratový“ krok za součást řešení nepočítáme (ale víme, že jej lze udělat), což se nám bude za chvíli hodit, abychom mohli tato řešení skládat za sebe. Bez většího rozmýšlení je jasné, že pokud má $R_{n,k}$ projít všech n^k konfigurací, musí ji tvořit $n^k - 1$ otočení.

Zvolíme si pevné n a budeme postupně (induktivně) konstruovat řešení $R_{n,k}$ pro jednotlivá k. Tedy nejdříve vytvoříme $R_{n,1}$ a potom ukážeme, jak z libovolného $R_{n,k}$ vyrobit $R_{n,k+1}$.

Pro situaci s jedním knoflíkem je řešení $(R_{n,1})$ zřejmé: prostě jím $(n-1)$-krát otočíme doprava. Takto určitě projde-

[16] `http://ksp.mff.cuni.cz/viz/kucharky/hledani-v-textu`

me postupně všechny pozice a jedním (n-tým) otočením se můžeme vrátit zpět na začátek. Například pro $n = 3$ dostaneme postupně pozice $0, 1, 2(, 0)$.

Nyní chceme z $R_{n,k}$ vyrobit $R_{n,k+1}$. Rozdělíme si knoflíky na dvě skupiny: první (*hlavu*) a všechny ostatní (2 až $k+1$, *ocas*). Je asi jasné proč – ocas je dlouhý k, tedy na něm můžeme nějakým způsobem použít $R_{n,k}$ zděděné z indukce. Zkusíme začít tak, že budeme na ocas postupně aplikovat jednotlivé kroky $R_{n,k}$. Ukážeme si to na příkladu $k = 1$. Pro něj dostáváme postupně konfigurace (BÚNO začínáme v $(0, 0)$): $(0, 0), (0, 1), \ldots, (0, n-1)$. V tuto chvíli jsme prošli všechny konfigurace začínající nulou. Dále už nemůžeme pokračovat s ocasem, neb bychom se vrátili do již navštíveného stavu.

Budeme tedy postupovat podobně, jako bychom přičítali jedničku: provedeme jakýsi „přenos do vyššího řádu“ – tedy otočíme hlavovým knoflíkem. Při normálním sčítání bychom zároveň i vynulovali všechny řády ocasu (a dostali bychom v tomto případě konfiguraci $(1, 0)$) a pokračovali přičítáním opět od nejnižšího řádu, dostávajíce tentokrát všechny konfigurace začínající jedničkou. Kdybychom tohle zopakovali celkem n-krát, dostaneme všechny konfigurace začínající postupně 0 až $n-1$, tedy úplně všechny.

Ale to nemůžeme, neb smíme otočit jen jedním knoflíkem, dostáváme tedy konfiguraci $(1, n-1)$. To ovšem vůbec nevadí! Díky tomu, že vše je cyklické, je úplně jedno, kde opětovné přičítání na nejnižším řádu začneme: pokud ho provedeme $(n-1)$-krát, vystřídá se na daném knoflíku $n-1$ různých hodnot, tedy opět projdeme každou konfiguraci, začínající tentokrát jedničkou, právě jednou. Následuje další přenos, dalších $n-1$ otočení, etc. Od sčítání se to liší jen tím, že prvky naší posloupnosti nebudou seřazeny vzestupně. Nejlépe to bude vidět na příkladu: $P_{3,2}$ vypadá takto (čteno po sloupcích):

```
00    12    21
01    10    22
02    11    20
```

Zkusme to nyní zapsat obecně. Označíme-li si jako H operaci „otoč hlavovým knoflíkem o jedna doprava“ a jako O operaci „proveď postupně všechny kroky $R_{n,k}$ na ocas“, pak bude $R_{n,k+1}$ vypadat takto:

$$\underbrace{O, H, O, H, \ldots, H, O}_{n\text{-krát } O,\ (n-1)\text{-krát } H}$$

Pro příklad $n = 3$ a $k = 2$ bude výsledná posloupnost otáčení

$$\underbrace{B, B,}_{O}\ \underbrace{A,}_{H}\ B, B, A, B, B.$$

Snadno ověříte, že opravdu vygeneruje posloupnost konfigurací v příkladu výše.

Takováto posloupnost splňuje všechny požadavky na $R_{n,k}$. Ukážeme, že to platí obecně. Operace O díky vlastnostem $R_{n,k}$, které máme zaručené z indukčního předpokladu, projde v $n^k - 1$ krocích všech n^k možných konfigurací ocasu (počítáme i počáteční a koncovou), bez ohledu na to, kterou začala. A to zopakujeme postupně pro všechny možné hodnoty hlavy, dostáváme tedy nejdřív všechny konfigurace začínající nulou, pak všechny začínající jedničkou, atd., dohromady tedy úplně všechny.

Co už je méně jasné je, že se z koncového stavu půjde dostat jedním otočením do počátečního. To nahlédneme takto: nejdříve ukážeme, že konfigurace ocasu bude na konci stejná jako na začátku. S ním hýbou jen operace O, kterých provedeme celkem n, přičemž všechny jsou stejné. Tedy pokud O otočí nějakým i-tým knoflíkem p_i-krát, celkem jím bude otočeno $n \cdot p_i$-krát, vrátí se tedy do původní pozice. A hlavovým knoflíkem otočíme celkem $(n-1)$-krát. Tedy pokud s ním otočíme ještě jednou, dostaneme se opravdu zpět do výchozí konfigurace, což jsme přesně chtěli.

Filip Štědronský

25-4-8 TEXgramy

Řešitelů utěšeně ubývá, ale stále je vás dost. Je radost číst řešení, která jdou k věci a dávají smysl. Nikdo není mimo, občas se objeví ukrutně komplikované řešení, ale nic moc hrozného. Až je to občas líto mému zlomyslnému já.

Tentokrát bylo správných přístupů habakuk a vzorové řešení je dlouhé, ukážeme si tedy pouze základní princip. Implementační detaily si prohlédnete ve vzorovém kódu.

Řešení **úkolu 1** bylo poměrně jednoduché. Bylo potřeba zavést si tři číselné registry, ve kterých jste si udržovali aktuální číslo nadpisu. Při vytváření nadpisu jste inkrementovali příslušný registr a případně vynulovali čítače nadpisů nižších úrovní.

Z estetického pohledu bylo potřeba vhodně nastavit mezery pod a nad nadpisem, včetně problémů typu: „Pokud se hned pod sebou sejdou dva nadpisy různých úrovní, tak mezi nimi nesmí být moc velká mezera.“

Taktéž se ve vzorovém řešení ošetřuje případ, kdy se pod sebou sejdou dva nadpisy stejné úrovně s jinak širokými čísly. Na začátku se změří šířka čísla 00, 00.00, resp. 00.00.00 a pak se číslo sází do hboxu fixní šířky, který je zprava doplněn pružným výplňkem.

Sazba obsahu v **úkolu 2** byl o něco větší oříšek. Použití `\immediate\write` nepřicházelo v úvahu, neboť TEX se může pokusit vložit příslušný nadpis ještě do předchozí strany, než přijde na to, že by bylo lepší dopustit se stránkového zlomu někde jinde. Pak by neseděla čísla stran v obsahu.

Naopak vůbec nebylo třeba sypat si do pomocného souboru čísla jednotlivých nadpisů – ta se přece dala vypočítat znovu při načítání obsahu stejným algoritmem.

Při vypisování obsahu se objevil jiný problém – před vložením obsahu bylo třeba přejít na novou stránku, jinak se do něj nezapsaly nadpisy z poslední strany. Bylo třeba také zavřít soubor s obsahem (`\closeout`), jinak se mohlo stát, že jste jej nevložili celý, ale jenom část, nebo dokonce prázdný (zbytek zůstal v zápisovém bufferu).

Sázení do více sloupců v **úkolu 3** nakonec nebylo tak zlé, jak se na první pohled zdálo. V makru `\multicolumn` se spočítá šířka sloupce, nastaví se podle toho `\hsize` a otevře vbox (`\setbox0\vbox\bgroup`). Primitivum `\bgroup` je definované jako `\let\bgroup{`.

Makro `\endmulticolumn` zavře box (`\let\egroup}`), rozseká box 0 na správně vysoké části (správná výška se určí vydělením celkové výšky počtem sloupců) a naskládá je vedle sebe do hboxu oddělené správně širokou mezerou.

A to je protentokrát vše. Těším se na vaše řešení páté série a přeju vám všem hezké jaro ... konečně přišlo.

Program (TEX):
`http://ksp.mff.cuni.cz/viz/25-4-8.tex`

Jan „Moskyto“ Matějka

Výsledková listina čtvrté série dvacátého pátého ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*2541*	*2542*	*2543*	*2544*	*2545*	*2546*	*2547*	*2548*	*série*	*celkem*
0.					10	9	13	8	12	12	10	14	61,0	237,0
1.	Rastislav Rabatin	GJHroncaBA	4	7	10	9	11		7	12	10		52,8	213,1
2.	Michal Punčochář	GJírovcČB	3	9	10	9		8	7	9	7		44,4	196,0
3.	Dominik Macháček	GLanškroun	4	9	9	8	9			2,5		14	44,2	187,9
4.	Martin Raszyk	G_Karvina	3	14			10	8	12				29,0	184,0
5.	Štěpán Hojdar	GJírovcČB	3	4	8,5	4,5		8			5	8	42,0	170,3
6.	Jakub Šafin	GHorMichal	4	4		8		8	12	12	10		50,6	169,4
7.	Martin Španěl	ArcibisGPH	4	7	10	7	4	8			10		41,4	168,2
8.	Jakub Maroušek	G_Písek	3	4	6,5	8,5				3	7	5	39,2	159,6
9.	Dalimil Hájek	GKepleraPH	2	9	6,5	8,5	7	4	2		10		38,2	152,8
10.	Richard Hladík	GOAMarLaz	0	4	6,5		7				10	14	42,1	149,5
11.	Vojtěch Hlávka	GŠlapanice	4	19	6,5	7	8		7	8,5			23,7	149,1
12.	Petr Houška	GJírovcČB	3	5	7	8,5					5	14	38,2	138,0
13.	Ondřej Mička	GJírovcČB	4	17		8,5		8			6,5	4	22,7	137,9
14.	Jakub Svoboda	GKomHavíř	3	4	8,5	8		8		2,5	6,5		38,9	134,6
15.	Martin Černý	G_Sokolov	3	3									0,0	134,3
16.	Martin Šerý	GJírovcČB	3	5	10			8			5	13	38,5	133,3
17.	Matej Lieskovský	GOmskPha	3	9	7			8			5		21,3	132,8
18.	Marek Dobranský	GHorMichal	3	4	7		4			2,5	7		28,8	118,7
19.	Jan Mikel	G_RožnovPR	4	3	7			8			10	7	37,3	105,2
20.	Ondřej Hlavatý	GJirsíkaČB	4	2									0,0	102,5
21.	Mikuláš Hrdlička	MensaG	2	4	2								3,7	100,1
22.	Petra Pelikánová	GJarošeBO	4	5		4,5							6,2	99,1
23.	Vojtěch Sejkora	SPSE_Pard	4	13		8,5		8				14	30,4	94,6
24.	Mark Karpilovskij	GJarošeBO	4	9					12				12,0	93,3
25.	Kateřina Zákravská	GJar	4	5								8	10,6	83,4
26.	Vladan Glončák	GĽŠtúraTN	4	4									0,0	82,1
27.	Vojtěch Vašek	GHli	4	8		3							3,9	75,9
28.	Lukáš Ondráček	GVolgogrOS	4	8									0,0	74,4
29.	Sabína Fraňová	GDubNVahom	4	4									0,0	74,3
30.	Anna Zákravská	GJar	4	5								8	10,6	72,0
31.	Jan Knížek	G_Strakon	2	9	5	4,5						6	17,8	67,4
32.–33.	Štěpán Trčka	GSlavičín	2	6	7			7,5		3			20,8	56,0
	Aneta Šťastná	GOmskPha	3	6	7			8					16,3	56,0
34.	Jan-Sebastian Fabík	GJarošeBO	3	7									0,0	51,4
35.	Jan Pokorný	G_Bučovice	1	1									0,0	46,7
36.	Alexander Mansurov	GNVPláníPH	4	11									0,0	44,4
37.	Jonatan Matějka	SŠP_ČB	3	13					1		6		6,2	43,7
38.	Jitka Fürbacherová	GKlatovy	4	10							3,5		3,8	40,9
39.	Jan Lejnar	GKlatovy	3	3									0,0	39,1
40.	Tomáš Velecký	GBezručeFM	2	5									0,0	34,8
41.	Tomáš Svítil	AES_NewDelhi	4	2									0,0	34,6
42.	Radovan Švarc	G_ČTřebová	2	2									0,0	34,4
43.	Milan Šorf	GNeumannŽR	3	3									0,0	33,5
44.	Ondřej Cífka	GNAlejíPH	4	9									0,0	32,0
45.	Ondřej Hübsch	GArabskáPH	3	18					7				4,3	26,1
46.	Jozef Kaščák	G_Svidník	4	1									0,0	23,3
47.	Tereza Hulcová	GKlatovy	4	8									0,0	23,2
48.	Michal Staruch	GOA_Vrchla	4	2									0,0	22,7
49.	Václav Volhejn	GKepleraPH	0	4									0,0	20,7
50.	Marek Dědič	GBNěmcovHK	3	1									0,0	19,3
51.	Veronika Klapová	GMHorPH	4	1				8			3,5		14,6	14,6
52.	Pavel Salva	VOŠŠumperk	3	4									0,0	8,7
53.	Michal Kužela	GSlavičín	1	2	2								4,2	8,2
54.	Tadeas Friedrich	GOhradníPH	3	1				8					8,0	8,0
55.	Dominik Smrž	GOhradníPH	3	9					7				7,8	7,8
56.	Tomáš Zahradník	GOPavla PH	3	1									0,0	7,1
57.	Jan Horešovský	GMěl	3	1									0,0	6,4
58.	Dominik Roháček	SPŠLegioJI	3	1									0,0	6,0
59.	Přemysl Šťastný	GZamberk	-1	1									0,0	4,7
60.	Dominika Macháčová	GSereď	3	1		2							4,4	4,4
61.	Martin Vzorek	SŠStarTura	2	1									0,0	1,4