nemôžeme použiť obyčajné prehľadávanie do šírky, keďže to nefunguje na grafe so zápornými hranami. Budeme musieť použiť napr. Bellman-Fordov algoritmus, ktorý si poradí aj so zápornými hranami. Opäť budeme potrebovať spraviť dve iterácie F-F algoritmu, ale s úpravou hľadania zlepšujúcej cesty. Časová zložitosť Bellman-Forda je lineárna od súčinu počtu hrán a vrcholov, čo je aj výsledná zložitosť celého algoritmu.

Nie je to ale optimálne riešenie. Brzdí nás práve hľadanie najkratšej cesty, kde ohodnotenie hrán môže byť záporne. Existuje ale spôsob ako sa tomu vyhnúť. Trik spočíva vo vhodnom ohodnotení hrán pri hľadaní zlepšujúcej cesty. Je to už ale nad rámec nášho riešenia.

Pavol „Pali“ Rohár

25-5-8 Boxy, z TEXu ven!

První úkol nebylo těžké vymyslet. Stačilo v upravené výstupní rutině vyprázdnit box 255 tak, aby se nikam nevypsal. Například jste mohli použít `\setbox0{\box255}`. Tedy, to bylo jádro úkolu, pak bylo potřeba zajistit, abyste tím nic nerozbili.

1. Vysypání dříve vypsaného materiálu. Bylo potřeba vložit na příslušné místo `\vfil\eject`. Dá se i napsat cyklus, kdyby po vysypání poslední strany ještě něco zbylo, ale nebylo to nutné. Makro `\stopoutput` se tedy smí volat jen na takovém místě, kde se smí objevit konec strany.
2. Uložení původní výstupní rutiny. Zde bylo potřeba například zakázat vnoření, nebo nějak chytře vynutit, aby se makra `\stopoutput` a `\startoutput` chovala jako třetí typ závorek.
3. Bylo třeba nulovat proměnnou `\deadcycles`, aby si TEX nemyslel, že se mu výstupní rutina zacyklila.
4. Vysypání vysázeného materiálu před `\startoutput`, tedy vložit i tam `\eject`.

Nebo jste mohli předefinovat `\shipout`, aby místo vypisování boxů svůj materiál zahazoval. Zde je jenom potřeba vědět, že některé zběsilejší pluginy také předefinovávají `\shipout` a dát si pozor na kolize. Taktéž je potřeba na správných místech vložit `\eject`.

```
\let\primitiveshipout\shipout
\newbox\stopoutputbox
\def\stopoutput{%
  \vfil\eject
  \def\shipout{%
    \deadcycles=0\setbox\stopoutputbox
  }%
}

\def\startoutput{%
  \vfil\eject
  \let\shipout\primitiveshipout
}
```

Ještě jednodušší verze, leč také funkční (v běžných případech), vypadala takto:

```
\newbox\stopoutputbox
\def\stopoutput{%
  \setbox\stopoutputbox\vbox\bgroup
}
\let\startoutput\egroup
```

Každý přístup má své ply a míny, záleží na tom, co si od maker slibujete. Přesná sémantika úmyslně nebyla zadána, abyste si mohli vybrat. Když jsem před časem psal podobné makro pro účely extrakce vzorců pro web, použil jsem první variantu s pár dalšími specifiky pro naše letáky, a každý extrahovaný vzorec pak je samostatnou stránkou, se kterou se pracuje dál.

Zlatý hřebíček posledního dílu seriálu v **úkolu 2** jste úspěšně zatloukli. Jak sázet do více sloupců?

0. Otevřeme skupinu (`\begingroup`), aby všechny změny, které napácháme, zůstaly lokální.
1. Předefinujeme výstupní rutinu tak, aby svůj výstup odložila do speciálního boxu. Pak ukončíme stránku, čímž si dosud nevysázený materiál uložíme stranou.
2. Předefinujeme rozměry stránky, aby odpovídaly situaci, kdy se pod sebe naskládají všechny sloupce, které se mají na straně objevit. Upravíme výstupní rutinu, viz následující body.
3. Když je ve výstupním seznamu dostatek materiálu, spustí se výstupní rutina, která sloupce nakrájí na správnou výšku a nasází vedle sebe. Zbytek se vrací do zpracování v dalším cyklu. Je potřeba nezapomenout na vrácení odloženého materiálu na správné místo.
4. Na konci víceloupcové sazby je nakonec potřeba ošetřit případy, kdy celá sekce vyjde na jednu stranu, nebo by se vešla, ale musí se rozdělit mezi dvě strany.
5. Drobná výhybka na začátku `\multicolumn` za pomoci podmínky `\ifinner` vybírá mezi vnitřní verzí (jako v minulé sérii) a vnější, která může být přes více stran.
6. Zavřeme skupinu, aby všechny napáchané změny zůstaly lokální. Tím jsme si zajistili, aby se nám to při vnořování nerozbilo úplně.

Co se týče vnořování, úplně stačilo, když jste vnitřní víceloupcovou část vysázeli bez dělení, jako podle minulé série. Hlavně šlo o to, aby se to tímto vnořením celé nerozbilo.

V řešení se nakonec proti původnímu plánu objevilo primitivum `\pagetotal`. To říká, kolik materiálu se už nashromáždilo k vysypání do tisku. Občas se to může hodit, například ve chvíli, kdy je potřeba na konci víceloupcové sekce zjistit, jestli se celý výsledek vejde na stránku, nebo je potřeba lámat.

Zdrojový text maker:
`http://ksp.mff.cuni.cz/viz/25-5-8.tex`

Řešení **úkolu 3** pak bylo přímočaré. Taková oddychovka.

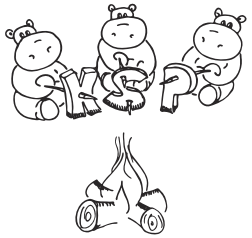
```
\def\defrgbcolor#1#2{%
  \def#1{\pdfliteral{#2 rg #2 RG}}%
}
\def\defcmykcolor#1#2{%
  \def#1{\pdfliteral{#2 k #2 K}}%
}
\def\defgrayscalecolor#1#2{%
  \def#1{\pdfliteral{#2 g #2 G}}%
}
```

A to je, milí přátelé, pro letošek všechno. Doufám, že jste se TEXu nezalekli a příští rok budou vaše řešení (nejen) KSP čistě a úhledně vysázena vaším vlastním makrobalíkem.

Přišel čas se s vámi, milí řešitelé, rozloučit. Napřesrok už organizovat nebudu, místo mě bude jiný, mladší, ale to bude sekáč . . . Děkuju vám za krásná řešení, rád jsem s vámi ušel kus vaší cesty ku vědění a k umění programování. Mnoho štěstí a hezké prázdniny vám přeje autor seriálu

Jan „Moskyto“ Matějka

Milí řešitelé, řešitelky a řešitelčata!



Hop hej! Vám skončila škola, matfyzákům zkouškové období a nám všem 25. ročník našeho semináře. Na závěr vám přinášíme vzorová řešení poslední série, celkovou výsledkovou listinu a těm nejlepším i pozvánku na podzimní soustředění.

Doufáme, že se vám uplynulý ročník líbil, a těšíme se, že se s vámi zase někdy potkáme. Třeba při řešení následujícího ročníku – úlohy první série se ostatně objeví na webu už v průběhu prázdnin. A pokud jste už odmaturovali, můžete se přidat k organizování.

Pěkné prázdniny přejí

Vaši organizátoři

Výsledková listina dvacátého pátého ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*2551*	*2552*	*2553*	*2554*	*2555*	*2556*	*2557*	*2558*	*série*	*celkem*
0.					11	9	11	11	9	11	13	15	61,0	298,0
1.	Rastislav Rabatin	GJHroncaBA	4	8	11	9	11	11		7	13		55,0	268,1
2.	Dominik Macháček	GLanškroun	4	10		4,5	11	7	1	7		14	44,7	232,6
3.	Martin Raszyk	G_Karvina	3	15		9		11	9	11	7		45,3	229,3
4.	Michal Punčochář	GJírovcČB	3	10	10,5				8				18,7	214,7
5.	Richard Hladík	GOAMarLaz	0	5		9	11	11		0		15	46,0	195,5
6.	Jakub Maroušek	G_Písek	3	5	6		11				4	6	34,4	194,0
7.	Štěpán Hojdar	GJírovcČB	3	5	0	2,5		1	4		4		18,2	188,5
8.	Dalimil Hájek	GKepleraPH	2	10		9	5	11	4	0	3,5		33,6	186,4
9.	Jakub Šafin	GHorMichal	4	5						11			11,0	180,4
10.	Petr Houška	GJírovcČB	3	6	1	8	11	11				3	37,0	175,0
11.	Jakub Svoboda	GKomHavíř	3	5		9	2	9	6	0	6		38,6	173,2
12.	Ondřej Mička	GJírovcČB	4	18		9	11	11			1,5	3	32,9	170,8
13.	Martin Španěl	ArcibisGPH	4	7									0,0	168,2
14.	Matej Lieskovský	GOmskPha	3	10	5		11	6	7		4,5		34,9	167,7
15.	Martin Černý	G_Sokolov	3	4		9	11	11					31,0	165,3
16.	Martin Šerý	GJírovcČB	3	6		3,5	11	6	1		1	2	28,9	162,2
17.	Marek Dobranský	GHorMichal	3	5		9	8		8	7	3,5		41,7	160,4
18.	Vojtěch Hlávka	GŠlapanice	4	20						7			4,0	153,1
19.	Jan Mikel	G_RožnovPR	4	4			10	11	7				29,8	135,0
20.	Mikuláš Hrdlička	MensaG	2	5		3,5	4		8	4			26,1	126,2
21.	Lukáš Ondráček	GVolgogrOS	4	9			11	8		2		12,5	35,1	109,5
22.	Jan-Sebastian Fabík	GJarošeBO	3	8	13	9	11	11		11			55,0	106,4
23.	Mark Karpilovskij	GJarošeBO	4	10						11			11,0	104,3
24.	Ondřej Hlavatý	GJirsíkaČB	4	2									0,0	102,5
25.	Petra Pelikánová	GJarošeBO	4	5									0,0	99,1
26.	Vojtěch Sejkora	SPSE_Pard	4	13									0,0	94,6
27.	Kateřina Zákravská	GJar	4	5									0,0	83,4
28.	Vladan Glončák	GĽŠtúraTN	4	4									0,0	82,1
29.	Vojtěch Vašek	GHli	4	8									0,0	75,9
30.	Sabína Fraňová	GDubNVahom	4	4									0,0	74,3
31.	Štěpán Trčka	GSlavičín	2	7	2		8		0,5	1	2		17,5	73,5
32.	Anna Zákravská	GJar	4	5									0,0	72,0
33.	Jan Knížek	G_Strakon	2	9									0,0	67,4
34.	Aneta Šťastná	GOmskPha	3	6									0,0	56,0
35.	Jonatan Matějka	SŠP_ČB	3	14		9				0			9,0	52,7
36.	Štěpán Šimsa	GJungmanLT	4	15		9	11	11	8	11			49,5	49,5
37.	Václav Rozhoň	GJirsíkaČB	2	1	6	9	11	2	8	7			47,6	47,6
38.	Jan Pokorný	G_Bučovice	1	1									0,0	46,7
39.	Alexander Mansurov	GNVPláníPH	4	11									0,0	44,4
40.	Jitka Fürbacherová	GKlatovy	4	10									0,0	40,9
41.	Jan Lejnar	GKlatovy	3	3									0,0	39,1
42.	Dominik Smrž	GOhradníPH	3	10		9	11	11					31,0	38,8
43.	Tomáš Velecký	GBezručeFM	2	5									0,0	34,8
44.	Tomáš Svítil	AES_NewDelhi	4	2									0,0	34,6
45.	Radovan Švarc	G_ČTřebová	2	2									0,0	34,4
46.	Milan Šorf	GNeumannŽR	3	3									0,0	33,5
47.	Ondřej Cífka	GNAlejíPH	4	9									0,0	32,0
48.	Václav Volhejn	GKepleraPH	0	5				7					8,8	29,5
49.	Ráchel Sgallová	GZborovPH	3	2		9		8			5,5		28,2	28,2
50.	Ondřej Theiner	GJírovcČB	3	1	1	2,5	6	7					26,5	26,5

51.	Ondřej Hübsch	GArabskáPH	3	18			0,0	26,1
52.	Jozef Kaščák	G_Svidník	4	1			0,0	23,3
53.	Tereza Hulcová	GKlatovy	4	8			0,0	23,2
54.	Michal Staruch	GOA_Vrchla	4	2			0,0	22,7
55.	Marek Dědič	GBNěmcovHK	3	1			0,0	19,3
56.	Veronika Klapová	GMHorPH	4	1			0,0	14,6
57.	Michal Kužela	GSlavičín	1	3		3,5	5,8	14,0
58.	Pavel Salva	VOŠSumperk	3	4			0,0	8,7
59.	Tadeas Friedrich	GOhradníPH	3	1			0,0	8,0
60.	Tomáš Zahradník	GOPavla PH	3	1			0,0	7,1
61.	Jan Horešovský	GMěl	3	1			0,0	6,4
62.	Dominik Roháček	SPŠLegioJI	3	1			0,0	6,0
63.	Vojta Staněk	PORGPha	-1	1		3	5,7	5,7
64.	Přemysl Šťastný	GZamberk	-1	1			0,0	4,7
65.	Dominika Macháčová	GSereď	3	1			0,0	4,4
66.	Martin Vzorek	SŠStarTura	2	1			0,0	1,4

Vzorová řešení páté série dvacátého pátého ročníku KSP

25-5-1 Cesta autobusem

Ukážeme si mírně inženýrské řešení. Vezmeme velmi zjednodušenou úlohu a budeme ji postupně opravovat, abychom se dostali k té složité (správný inženýr by začal od prázdného programu, ale nemusíme to přehánět).

Takže tedy od lesa. Co kdybychom nejezdili autobusem, ale chodili pěšky? To bychom nemuseli řešit, jakým směrem jsme otočení, takže by úloha byla jen obyčejný průchod do šířky, jak je popsaný například v grafové kuchařce.[1] Jak známo, to stihneme v $\mathcal{O}(n)$, kde n je počet políček plánu města.

Tak si úložku malinko ztížíme. Budeme jezdit mikrobusem – to bude autobus délky 1. Už musíme řešit otáčení, ale můžeme se otočit kdekoliv. A na vyřešení otáčení uděláme malý trik. Uděláme si dvě kopie plánu města a umístíme je na sebe – takže budeme mít jakýsi trojrozměrný prostor. V dolním patře budou sousedit políčka jen ve vodorovném směru, v horním patře jen ve svislém. A políčka nad sebou budou sousední vždy.

Všimněme si, že patra plánu odpovídají otočení autobusu. V dolním patře je autobus otočený vodorovně, v horním svisle. Přesun mezi patry odpovídá jeho otočení o 90 ° (nebo $\pi/2$, pokud máte tyto jednotky radši).

V tomto dvoupatrovém bludišti tedy opět nalezneme cestu ze startu do cíle (nebo cílů – je jedno, jak budeme natočení v cíli, proto jsou obě políčka nad sebou cílová). Protože jsme zvětšili plán jen dvakrát, časová i paměťová složitost je stále $\mathcal{O}(n)$.

A už se dostáváme k lehčí variantě úlohy ze zadání. Budeme jezdit klasickým autobusem délky k. Pozici autobusu si budeme reprezentovat pozicí jeho levého horního konce. Z toho, ve kterém se nacházíme patře, poznáme, jestli autobus vede dolů nebo doprava. Tak to by byla téměř stejná úloha jako minule. Až na to, že ne všechna políčka nad sebou sousedí (a občas sousedí některá políčka, která nejsou nad sebou – viz např. obrázek v zadání, kde se otočením změní levý horní roh autobusu). Kdybychom ale věděli, jestli stojíme v rohu volného čtverce velikosti alespoň $k \times k$, neměli bychom nejmenší problém určit, jestli se zde otočit umíme, či nikoliv. Hledání čtverců ale odložme až na konec řešení.

Tak tedy, zlatý hřeb úlohy. Potřebujeme si v průběhu prohledávání pamatovat ještě poslední navštívenou zastávku (protože do ní nesmíme hned vjet znovu) a aktuální délku autobusu. No, pro pamatování tohoto si vytvoříme další „patra“ bludiště. Naše dvě patra z lehčí varianty zkopírujeme tolikrát, kolik je zastávek, a v každé zastávce se teleportujeme do stejného políčka v kopii pro danou zastávku. V této kopii se do ní již nesmí vjet (ale vyjet ano – již máme orientovaný graf). Obdobný trik uděláme pro délky autobusů – celé skupinky pater z minulého kopírování nakopírujeme pro každou délku autobusu a budeme teleportovat mezi nimi při každé změně délky.

To je celé hrozně hezké, ale udělat všechny ty kopie by trvalo hrozně dlouho a zabralo zbytečně mnoho paměti – vždyť my z těch kopií většinu ani nikdy nepoužijeme. Proto si budeme jen „představovat“, že máme všechny tyhle kopie. Pozice autobusu v celé té naší struktuře bude reprezentovaná čtveřicí informací – pozicí na plánu, otočením, poslední navštívenou zastávkou a délkou. Ale plán budeme mít jen jeden. Jen to, která políčka v našem mnoharozměrném prostoru jsou sousední, budeme počítat podle celé čtveřice pokaždé, když budeme potřebovat sousední políčka některé pozice.

Tak a teď ty čtverce. Hodilo by se nám umět spočítat, jaký největší čtverec vede z každého políčka doleva nahoru (pro zbylé 3 směry to bude obdobné, prostě stejný algoritmus pustíme vícekrát v různých směrech). To je ale pouze drobné cvičení na dynamické programování.[2]

Chceme spočítat velikost čtverce pro pozici (x, y). Pokud máme spočítané velikosti maximálních čtverců pro pozice $(x-1, y)$, $(x, y-1)$ a $(x-1, y-1)$, pak z těchto hodnot jednoduše spočítáme velikost našeho čtverce (vezmeme minimum z těch tří a přičteme jedničku – kdo nevěří, ten si to nakreslí). A abychom tyto pozice měli již spočítané, tak půjdeme po řádcích shora zleva.

A jak je to se složitostmi? Určitě potřebujeme $\Omega(n)$ času i paměti na načtení a uložení mapy a spočítání velikostí čtverců. Horní odhad je ale trochu horší. Určitě ale nepotřebujeme navštívit žádný stav dvakrát (tedy, každá čtveřice se ve frontě vyskytne maximálně jednou). Políček je n, možných délek autobusů $\mathcal{O}(n)$ a zastávek nechť je z (kde

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

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

algoritmus si normálne s kapacitami vo vrcholoch neporadí, a preto je najprv potreba graf upraviť a previesť kapacity z vrcholov na hrany.

Ale poporiadku. Najprv sa pozrieme, ako sa dala vyriešiť ľahšia varianta. Úlohou bolo zistiť, či sa dá v neohodnotenom neorientovanom grafe zo štartu (vrchol A) dostať do cieľa (vrchol C) cez sklad (vrchol B) s tým, že niektoré vrcholy môžeme navštíviť iba raz (tie, na ktorých je polícia).

Hneď na začiatok si môžeme všimnúť, že nemá význam ľubovoľným vrcholom prechádzať viac ako dva krát (ak je to možné). Ak sme navštívili nejaký vrchol tretí krát, znamená to, že sled, ktorým prechádza ten vrchol, má dve slučky. Vrchol B potrebujeme navštíviť iba raz, a teda jedna z tých slučiek bude určite zbytočná. Môžeme ju zo sledu vyhodiť a vrchol navštívime už iba dva krát. Z toho vyplýva, že každý vrchol nám stačí navštíviť maximálne dva krát a neprídeme tým o žiadne riešenie.

Keďže naším cieľom je na túto úlohu použiť F-F algoritmus, musíme sa najprv vysporiadať s policajtmi vo vrcholoch. Vrcholy podrozdelíme. Každý takýto vrchol nahradíme dvojicou nových. Do prvého nového vrcholu budú viesť všetky hrany, čo viedli do pôvodného vrcholu, z druhého nového vrcholu budú viesť hrany, ktoré viedli z pôvodného vrcholu. Tieto dva nové vrcholy spojíme hranou.

Tým sa naša úloha nezmenila, iba sme obmedzenie na navštívenie vrcholov presunuli na hrany. Počet hrán sa zvýšil maximálne o počet vrcholov, čo nám nič nezhorší. Keďže graf v zadaní bol neorientovaný a F-F algoritmus pracuje s orientovanými hranami, každú (neorientovanú) hranu nahradíme dvojicou orientovaných.

Ďalej si vytvoríme nový vrchol (označme D) a spojme ho s A a C. Na obe hrany, ktoré z vrcholu D vedú, postavíme policajtov. Potom môžeme našu úlohu preformulovať na nájdenie dvoch ciest z vrcholu B do D s tým, že môžeme prejsť každou hranou, na ktorej je policajt, maximálne raz. Keďže na hranách C-D a A-D sú policajti, nie je možné, aby obe cesty viedli po spoločnej jednej hrane do D.

Teraz už môžeme použiť F-F algoritmus. Kapacita hrany nám bude udávať, koľko krát cez ňu môžeme prejsť. Hrany, na ktorých sú policajti, budú mať kapacitu 1. Keďže hľadáme práve dve cesty medzi A a D, kapacita väčšia ako 2 nemá význam (aj z toho dôvodu, že každý vrchol nám stačí navštíviť max. dva krát). Za zdroj zoberieme vrchol B, za stok vrchol D.

Spustíme F-F algoritmus a ten nám vráti maximálny tok v tom grafe. Ak bude tok nulový, znamená to, že neexistuje žiadna (zlepšujúca) cesta z B do D, a teda z B sa nevieme dostať ani do A a ani do C. V tom prípade úloha nemá riešenie. Ak tok bude veľkosti jedna, znamená to, že existuje nejaká cesta z B do D, ale neexistujú tie cesty dve, ktoré by spĺňali podmienku policajtov. Teda z B sme sa nevedeli dostať buď do A, alebo do C, keďže na oboch hranách pred vrcholom D sú policajti. Ani v tomto prípade riešenie zjavne neexistuje.

A posledný prípad, ktorý môže nastať, je tok veľkosti 2. Väčší už byť nemôže, lebo súčet kapacít hrán, ktoré vedú do stoku, je 2. Ak nám F-F našiel tok veľkosti 2, znamená to, že každou hranou, na ktorej je policajt, prechádzame maximálne raz a máme dva sledy z B do D, ktoré sú disjunktné na hranách s policajtmi. Teda nutne existuje sled z A do B a aj sled z B do C a pritom platí, že každá hrana, na ktorej sú policajti, sa vyskytne v sledoch A-B a B-C maximálne raz. Teda v tomto prípade, keď existuje tok o veľkosti 2, úloha má riešenie.

Ak tieto dva sledy chceme vypísať, môžeme použiť rovnaký algoritmus ako na vypísanie hranovo disjunktných ciest, ktorý je popísaný v kuchárke.

Úloha sa dá aj priamo previesť na hľadanie hranovo disjunktných ciest. Zdroj a stok zoberieme rovnaký ako vyššie a podrozdelíme tentokrát každý jeden vrchol, čím opäť presunieme políciu na hrany. Hrany, na ktorých polícia nestojí, zdvojíme a potom postavíme políciu na každú jednú hranu. Zdvojením sme zabezpečili, že sa medzi vrcholmi dá prejsť aj dva krát a teda pridaním polície na každú hranu nič nepokazíme. A teraz, keď už budú mať všetky hrany svojho policajta, môžeme im nastaviť kapacitu 1 a použiť kuchárkový algoritmus na hľadanie hranovo disjunktných ciest. V tomto prípade potrebujeme dve hranovo disjunktné cesty zo zdroju do stoku, čo bude ekvivalentné s existenciou riešenia ľahšej varianty úlohy.

Pozrime sa ešte na časovú zložitosť. Treba nám nájsť tok o veľkosti 2. To znamená, že potrebujeme spraviť max. dve iterácie F-F algoritmu. Teda nájsť max. dve zlepšujúce cesty. Nájsť zlepšujúcu cestu vieme prehľadaním do šírky. Teda časová zložitosť celého algoritmu bude rovnaká ako jedna iterácia F-F a bude to lineárne od počtu hrán a vrcholov. Pamäťová zložitosť taktiež.

Poďme sa teraz pozrieť na ťažšiu variantu tohto príkladu. V nej bolo potrebné zo všetkých ciest z A do C cez B nájsť tú najkratšiu, ktorá by spĺňovala podmienku policajtov. Ako prvé by nás mohlo napadnúť použiť rovnaký algoritmus ako pri ľahšej variante a upraviť hľadanie zlepšujúcich ciest tak, aby sme vyberali vždy tú najkratšiu.

Vyzerá, že by to mohlo fungovať, ale je v tom háčik. Niekedy pri hľadaní zlepšujúcej cesty sa nám oplatí tlačiť tok po nejakej hrane späť. A teda nemôžeme hľadať zlepšujúcu cestu, ktorá bude najkratšia v počte hrán. Môže existovať nejaká zlepšujúca cesta s väčším počtom hrán, ktorá bude na veľkom počte hranách tlačiť tok späť. Po pridaní takejto zlepšujúcej cesty už nebude po tých hranách nič tiecť, a teda tie hrany, po ktorých sme tlačili tok späť, nechceme započítať do dĺžky výslednej cesty medzi A-C (pretože po nich nakoniec nepôjdeme). Ba naopak potrebujeme ich odpočítať, lebo v nejakej predchádzajúcej iterácii hľadania zlepšujúcej ceste sme ich do dĺžky cesty pripočítali a teraz už po nich nič netečie. Teda v jednej F-F iterácii z pomedzi všetkých zlepšujúcich ciest potrebujeme vybrať takú, ktorá bude mať súčet ohodnotení hrán najnižší.

V prvej iterácii začíname s nulovým tokom, a teda tu nám stačí nájsť najkratšiu zlepšujúcu cestu (po žiadnej hrane nebudeme tlačiť tok späť, keďže je nulový). Pred začiatkom druhej iterácie, bude po všetkých hranách tiecť tok buď 0, alebo 1. V prípade, že budeme chcieť ísť v druhej iterácii po hrane, po ktorej zatiaľ nič netečie, ohodnotenie hrany bude naďalej 1. V prípade, že budeme chcieť ísť po hrane po smere toku (veľkosti 1), potom ohodnotenie hrany bude tiež 1. V prípade, že budeme chcieť ísť po hrane proti smeru toku (veľkosti 1), tak ohodnotenie takejto hrany bude -1. Tým docielime, že ak po hrane budeme tok tlačiť späť, tak sa nám naozaj zníži aj dĺžka cesty A-C presne o tok, čo pred tým po tej hrane tiekol. A teda ohodnotenie hrán nám bude udávať dĺžku cesty medzi A-C.

Nájdenie zlepšujúcej cesty bude spočívať v nájdení najkratšej cesty s naším novým ohodnotením hrán. Na to ale

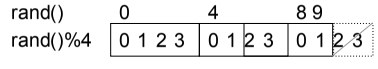
balíků s číslem větším než k – ty jsou vybírány později a nezávisle. Pozice prvních pár balíků se určitě změní, leč nás nezajímají přesně – zajímá nás jen to, které z nich zůstanou na prvních k místech (bez ohledu na to, kde konkrétně). I to se pochopitelně změní: např. pro $k = 2, N = 3$, pokud jsme původně po k-tém kroku měli permutaci $(2, 1)$ a poslední balík nahradíme třetím, dostaneme na konci k-tici $\{2, 3\}$, kdežto pokud ji nahradíme setříděným pořadím $(1, 2)$, skončíme na konci s $\{1, 3\}$. Podívejme se ale na to, jak postupně nahrazujeme balíky $\{1, \ldots, k\}$ v počátečním úseku jinými. Při každé úpravě permutace mají všechny „přeživší“ z prvních k balíků stejnou pravděpodobnost být odsunuty na konec – jinými slovy, permutační algoritmus balík k vyřazení vybírá náhodně. A pokud na začátku prvních k balíků nějak jednorázově přeuspořádáme, nebudou tyto výběry o nic méně náhodné (spíše k intuitivnímu nahlédnutí, poctivý důkaz by dal trochu práce, klíčovým je fakt, že naše přeuspořádání a tyto náhodné výběry jsou *nezávislé*).

Pokud z permutačního algoritmu odstraníme prvních k kroků a začneme $(k+1)$-ním s permutací $(1, 2, \ldots, k)$, ze všech prohození už se stanou přiřazení (vždy bude alespoň jeden prvek ležet mimo prvních k) a dostáváme algoritmus naprosto identický s původním k-ticovým. Jen jsme k němu přišli tak trochu z jiné strany.

Poznámky

1) Všichni řešitelé až na jednoho považovali za dostatečné dokázat, že každý *balík* má stejnou pravděpodobnost objevit se na výstupu. To ale ještě vůbec *neznamená*, že každá k-*tice* bude mít stejnou pravděpodobnost. Např. pro $k = 2$ a $N = 4$ a algoritmus, který vrací se stejnou pravděpodobností dvojice $\{1, 2\}$ a $\{3, 4\}$ se každý balík vyskytuje právě v jedné ze dvou možných dvojic, objeví se tedy na výstupu se stejnou pravděpodobností 1/2 (dokonce „tou správnou“, neb $k/N = 1/2$). Ovšem určitě není pravda, že by všechny dvojice ze čtyřech balíků měly stejnou pravděpodobnost být výstupem. Ba co víc, většinu (4) jich algoritmus vůbec vygenerovat neumí! Místo šesti dvojic se stejnou pravděpodobností 1/6 generuje dvě s pravděpodobností 1/2 a čtyři s nulovou! A jedno takové řešení nám opravdu přišlo. Samozřejmě chápeme, že to s teorií pravděpodobnosti na středních školách nebude nikterak slavné, pročež jsme řešením, která generovala k-tice rovnoměrně, ale pořádně to o sobě nedokázala (většina došlých), strhávali jen jeden bod.

2) Někteří řešitelé používali ke generování náhodných čísel z rozsahu $0 \ldots m-1$ ve svých zdrojácích konstrukci `rand() % m`, kde `rand()` je funkce vracející náhodná čísla z nějakého fixního velkého rozsahu $0 \ldots M-1$, typicky daného maximálním rozsahem celočíselného datového typu (M je 2^{31} či 2^{63} pro Céčkový `int`), a `%` operátor modula. Pro m nesoudělné s M nebude výsledek takovéhoto výrazu úplně rovnoměrně náhodný, jak naznačuje pro příklad $M = 10$, $m = 4$ obrázek níže:



Vidíme, že čísla 0 a 1 mají větší pravděpodobnost (3/10) než ostatní (2/10), neboť když rozdělíme interval $0 \ldots 9$ na úseky délky 4, objeví se i v posledním neúplném. Obecně mají čísla menší než $M \bmod n$ pravděpodobnost $\lceil M/m \rceil$ a ostatní $\lfloor M/m \rfloor$. Tyto pravděpodobnosti budou stejné právě tehdy, když M/m je celočíselné. Obecně to, jak moc velká nerovnoměrnost bude, záleží na poměru $(M \bmod m)/M$. Za tuto technickou drobnost jsme pochopitelně nestrhávali žádné body, ale pokud někdy budete programovat něco hodně závislého na rovnoměrně náhodných číslech, je dobré mít to na paměti.

Dalo by se to obejít tak, že pokud nám „padne“ 8 nebo 9, tento výsledek zahodíme a zkusíme to znovu (i opakovaně). Ale to není předmětem této úlohy. Bylo naprosto oprávněné předpokládat, že umíme generovat náhodná čísla z daného rozsahu v konstantním čase. Pokud na to váš oblíbený jazyk nemá žádnou funkci, můžete si nějakou vymyslet a v kódu tento fakt okomentovat (programy jen čteme, obvykle se je nesnažíme spouštět). V případě této úlohy by bylo asi nejjednodušší prostě místo zdrojáku poslat pseudokód.

Martin Mareš & Filip Štědronský

25-5-6 Dělení dortu

Přímočarým řešením je vyzkoušet všechny možné průběhy hry a z nich vybrat ten nejlepší. V prvním tahu je na výběr N způsobů jak táhnout, v následujících $N-2$ tazích jsou způsoby právě dva. Celkem tedy máme $N2^{N-2}$ možných scénářů hry. Získali bychom tedy řešení s exponenciální časovou složitostí.

Chceme-li se zbavit exponenciály, stojí za zamyšlení otázka, zda něco nepočítáme zbytečně. Opravdu nás pro získání výsledku zajímají všechny možné průběhy hry?

Povšimněme si následujícího – jedinou informací o pozici hry, která má vliv na volbu dalšího tahu je aktuální nesnědený úsek dortu. Pro každý úsek délky 1 až $N-1$ existuje N pozic, kde tento úsek začíná. Úsek reprezentující celý dort je pouze jeden, ale bude se nám pro naše řešení hodit uvažovat jej jako N různých úseků podle toho, kde pomyslně začíná. Uvažujeme tedy N^2 úseků.

Pro každý úsek zjistíme, kolik nejvýše může hráč získat, pokud na tomto dílku táhne jako první, a kolik, pokud táhne jako druhý.

Pro úseky délky 1 ukořistí první hráč dílek odpovídající tomuto úseku a druhý nic. Pro úsek délky ℓ začínající na dílku i a končící na dílku j má hráč na tahu 2 možnosti – sníst dílek na pozici i, nebo j. V prvním případě bude zisk s_i (velikost i-tého dílku) plus zisk druhého hráče na úseku délky $\ell-1$ končícím na pozici j, v druhém případě bude zisk s_j plus zisk prvního hráče na úseku délky $\ell-1$ začínajícím na pozici i.

Řešením úlohy pak bude největší hodnota z možných zisků na dílcích délky N. Hodnoty zisků pro jeden dílek naznačeným způsobem spočteme v konstantním čase, časová složitost postupu tak bude $\mathcal{O}(N^2)$. Všimněme si, že si stačí v průběhu řešení pamatovat pouze zisky pro úseky délky $\ell-1$ a ℓ, pak získáme paměťovou složitost $\mathcal{O}(N)$.

Lukáš Folwarczný

25-5-7 Policejní koridor

K zadaniu tejto úlohy bola priložená kuchárka o tokoch v sieťach. Malo vám to napomôcť, že úlohu treba riešiť pomocou tokov. Toho ste sa skoro všetci chytili a použili kuchárkový Ford-Fulkersonov (F-F) algoritmus na vyriešenie ľahšej varianty. Viacerí z Vás ste ale vo svojich riešeniach zabudli napísať alebo ste nedôsledne čítali, že F-F algoritmus dostane kapacity na hranách a nie vo vrcholoch. F-F

z je $\mathcal{O}(n)$). Orientace autobusu jsou jen dvě. Takže máme $\mathcal{O}(n^2 \cdot z)$ možných stavů.

Navíc, protože jsme si nevytvořili všechny kopie mapy, musíme si nějak pamatovat, které stavy jsme už navštívili. Pokud to uděláme např. ve stromu, zaplatíme za to ještě logaritmickým zpomalením, takže časová složitost by byla $\mathcal{O}(n^2 \cdot z \cdot \log n)$. Také bychom mohli místo stromu použít hešování a dostat složitost $\mathcal{O}(n^2 \cdot z)$ v průměru.

Lze očekávat, že na „slušně vychovaných“ mapách do takových extrémů nebudeme muset jít, ale jdou vytvořit i „neslušné“ mapy – například začneme s dlouhým autobusem a na jeho zkrácení o 1 je potřeba projet řádově n políček. A do cíle povede klikatá cesta, kterou projede jen kratičký autobus.

Taktéž, pokud bychom si vytvořili všechny ty kopie na začátku, zbavili bychom se onoho logaritmu. Ale za cenu toho, že by nám to vždy trvalo $\Omega(n^2 \cdot z)$. Tedy, musíme si rozmyslet, jestli čekáme spíše slušně vychované mapy, nebo ty neslušně vychované.

Program (Python):
`http://ksp.mff.cuni.cz/viz/25-5-1.py`

Michal „Vorner“ Vaner

25-5-2 Telefonní ústředna

Úloha byla poměrně jednoduchou aplikací principu dynamického programování.[3] Pokud tento pojem slyšíte poprvé, doporučuji přečíst si naši kuchařku.

Kromě polí pro záznam a hovor si pořídíme ještě pole `stavy` o délce s (tedy stejně dlouhé jako hovor). Do něj si na i-tou pozici budeme ukládat zatím objevený počet prefixů hovoru délky i.

Postupně tedy procházíme pole záznamu odzadu. V každém kroku tohoto průchodu začneme procházet od začátku celé pole hovoru a vždy porovnáme bity. Pokud se j-tý bit hovoru shoduje s příslušným bitem záznamu, znamená to, že můžeme prodloužit všechny dosud nalezené prefixy hovoru délky $j-1$. To odpovídá přičtení proměnné `stavy[j-1]` k `stavy[j]` v případě, že $j > 1$, a přičtení jedničky v případě, že $j = 1$.

Kýžený výsledek, tedy počet způsobů, jak z bitů záznamu vybrat podposloupnost odpovídající hovoru, se nachází na konci algoritmu ve `stavy[s]`.

Rozmysleme si ještě, že pole záznamu potřebujeme procházet odzadu. Pakliže tak neučiníme a j-tý a $(j-1)$-tý bit hovoru bude stejný, zvýšíme nejdříve `stavy[j-1]` o $k > 0$ a pak `stavy[j]` o k víc, než bychom měli.

Pokud bychom za každou cenu chtěli hovor procházet odpředu, museli bychom použít ještě jedno pomocné pole, kam bychom si změny ukládali a vždy až na konci zpracování indexu záznamu tyto změny k poli `stavy` přičetli.

Nakonec ke složitostem. Paměťová je očividně lineární k délce záznamu a hovoru, tedy $\mathcal{O}(n+s)$. Ohledně časové pak vidíme, že pro každý bit záznamu procházíme celý hovor, tedy výsledná časová složitost je $\mathcal{O}(ns)$.

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

Jan Bok

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

25-5-3 Špagety

Úloha se špagetami vás očividně zaujala, přišlo na ní skoro nejvíce řešení páté série (jen o jedno řešení méně, než kolik měla „nejoblíbenější“ úloha série 25-5-2).

Ve vašich řešeních se objevovaly dva přístupy, které nakonec vedly skoro k tomu samému. Jedním z nich bylo vytvořit si ze špaget graf. Z podlouhlých špaget se nám stanou vrcholy (z každé špagety jeden) a orientované hrany v tomto grafu natáhneme tam, kde jedna špageta leží přímo na druhé (tedy pokud ve směru gravitační osy jsou dílky špaget přímo nad sebou, nebo je mezi nimi jen prázdné místo).

Pokud do nějaké špagety ještě povedou vstupní hrany, znamená to, že nad ní ještě něco leží a nemůžeme ji tedy odebrat. Navíc si uvědomme, že nám stačí tyto hrany natáhnout vždy jen mezi přímo sousedícími špagetami. Pokud nad sebou totiž leží více špaget, tak nejspodnější špagetu zajímá jen špageta těsně nad ní a je jí jedno, jestli je to jediná špageta, která ji blokuje, nebo je jich nad ní více.

Pak již jen můžeme najít špagety, které nejsou pokryté žádnou hranou (mají vstupní stupeň nula) a postupně všechny odebereme. Během odebírání každé ze špaget zrušíme i příslušné hrany a současně se díváme, jestli jsme nezískali novou volnou špagetu. Pokud ano, vložíme ji do pomocného zásobníku. To je jeden krok.

V dalším kroku vezmeme pomocný zásobník a odebereme všechny špagety, které jsou v něm. Tím nám opět vzniknou nové volné špagety a takto budeme pokračovat, dokud nevyprázdníme talíř, nebo dokud to dál nepůjde.

Druhým přístupem, který v důsledku vede na stejnou strukturu, jen je v něm graf více schovaný, je procházet talíř špaget postupně po sloupcích a k oindexovaným špagetám si ukládat počet špaget, kterými je tato špageta přímo zakryta. Při odebírání se pak tento čítač snižuje a řešení tak funguje stejně, jako výše popsané.

K vypočítání složitosti si označme jako N počet špaget, jako M počet hran mezi nimi a jako V objem talíře. Je jasné, že $N, M \leq V$, protože maximálně můžeme mít jednotkovou špagetu na každém políčku talíře. Může nám také nastat situace, kdy bude hran řádově N^2 (představte si v jedné vrstvě $N/2$ rovnoběžných špaget a pod nimi stejnou vrstvu, jen zrotovanou). Paměťová složitost je tedy $\mathcal{O}(N+M)$, ale jelikož N a M nevíme na vstupu, je asi korektnější odhad udělat jako $\mathcal{O}(V)$.

Časová složitost je pak stejná, protože na každou špagetu se podíváme při výpočtu maximálně jednou a po každé hraně se vydáme také jen jednou, což je zase $\mathcal{O}(N+M)$ na výpočet. Na vytvoření grafu ale určitě potřebujeme projít celý talíř (navíc ho musíme i načíst), tedy časová složitost je také $\mathcal{O}(V)$.

Hodně štěstí i při dalších pokusech nepokecat se špagetami!

Program (C):
`http://ksp.mff.cuni.cz/viz/25-5-3.c`

Jirka Setnička

25-5-4 Výslechy

Niektorí z vás si správne všimli ako úlohu previesť na grafovú. Predstavme si, že na vstupe dostaneme výrok typu „A tvrdí, že B je mafián“. Čo všetko vieme povedať o A a B? Ak A je zamestnanec (budem značiť A_1), tak B bude

určite mafián. Ak je zas B mafián, tak A musí byť zamestnanec (mafián by klamal a netvrdil, že B je mafián). Je jednoduché podobne odvodiť, že ak A je mafián (značím A_0), tak B je zamestnanec a ak B je zamestnanec, tak A je mafián. Inými slovami, pri tomto výroku platí

$$A_1 \Leftrightarrow B_0, \quad A_0 \Leftrightarrow B_1.$$

V prípade výroku typu „A tvrdí, že B nie je mafián“ rovnakým postupom odvodíme, že platí

$$A_1 \Leftrightarrow B_1, \quad A_0 \Leftrightarrow B_0.$$

Zo vstupu teda môžeme vyrobiť graf. Za každú novú osobu X pridáme do grafu dva vrcholy, a to X_1 a X_0. Je asi zrejmé, ako budú vyzerať hrany v tomto grafe. V prípade výroku „A tvrdí, že B je mafián“ spojíme hranou vrcholy A_1, B_0 a vrcholy A_0 a B_1. Podobne pri výroku „A tvrdí, že B nie je mafián“ spojíme hranou vrcholy A_1, B_1 a vrcholy A_0, B_0.

K čomu nám pomôže takýto graf? Je vidieť, že ak nastane situácia, že existuje vrchol X_0, ktorý je spojený nejakou cestou s vrcholom X_1, tak práve vtedy si musel niekto protirečiť a počet možností je 0. Ak totiž vyjdeme z nejakého vrcholu a dostaneme sa do iného, tak to znamená, že z výpovedi človeka A dokážeme vyvodiť, že či X je mafián alebo nie podľa toho, či sme vo vrchole X_1 alebo X_0.

Ak situácia z predchádzajúceho odstavca nenastane, tak počet možností je $2^{k/2}$, kde k je počet komponent súvislosti grafu. Všimnime si, že pre každú komponentu je navyše v grafe ešte aj jej negácia. Komponenta súvislosti v našom grafe nám vlastne hovorí nasledovné. Predstavme si, že v nejakej komponente sa nachádza vrchol X_1. Ak poviem, že X je zamestnanec, tak som rozhodol o všetkých vrcholoch v komponente, že či sú zamestnanci alebo mafiáni. Negácia komponenty nám dáva druhú možnosť.

Ak by sme riešenie chceli naprogramovať, tak po skonštruovaní grafu nájdeme prehľadávaním do hĺbky (alebo do šírky) komponenty grafu a počas prehľadávania môžme kontrolovať, že či sa X_0 a X_1 nenachádza v tej istej komponente, pre každé X. Časová zložitosť je $\mathcal{O}(n+m)$, kde $2n$ je počet vrcholov grafu a m je počet hrán grafu. Pri tomto odhade predpokladám, že násobenie má zložitosť $\mathcal{O}(1)$. Všimnite si, že počet možností je v najhoršom prípade exponenciálny, napríklad keď nedostaneme na vstupe žiadne výroky.

Peter Zeman

25-5-5 Úklid trávníku

Označme si celkový počet balíků N (pro účely teoretického rozboru, toto číslo pochopitelně nesmíme v algoritmu použít), jednotlivé balíky $1, \ldots, N$ a k velikost vybíraného vzorku. Budeme předpokládat, že umíme generovat náhodná celá čísla z rozsahu $1, \ldots, m$ v konstantním čase pro libovolné $m \leq N$. Dále předpokládáme, že $N \geq k$ (jinak by úloha vůbec neměla smysl).

Řešení pro $k = 1$

Zařídíme si *odkladiště* (celočíselnou proměnnou), na kterém si budeme uchovávat jeden jakýsi „prozatímně vybraný“ balík z dosud zpracované části vstupu. Algoritmus potom bude pracovat takto: na začátku umístí do odkladiště první vstupní balík a poté postupně prochází všechny další v pořadí, v jakém přicházejí na vstupu. U každého se bude muset rozhodnout, zdali jej umístit do odkladiště (a tedy jím původní vybraný balík nenávratně nahradit), nebo zahodit. Balík, který v odkladišti zůstane po zpracování celého vstupu, označíme za hledaný vzorek.

Budeme postupovat induktivně: chceme, aby na konci každého kroku měly všechny zatím zpracované balíky stejnou pravděpodobnost nacházet se v odkladišti. Pokud toto zajistíme, pak už určitě na konci budou mít všechny balíky stejnou pravděpodobnost výběru.

Na začátku to určitě platí – máme jediný balík, který se v odkladišti nachází s pravděpodobností 1. Nyní předpokládejme, že už máme prvních $n-1$ balíků zpracovaných ($n \geq 2$) a dle indukčního předpokladu se každý z nich nachází v odkladišti s pravděpodobností $1/(n-1)$. Na konci kroku by se měl každý, tedy i nově přidaný, balík objevit na odkladišti s pravděpodobností $1/n$. Nezbývá nám tudíž nic jiného, než tam nový balík s pravděpodobností $1/n$ umístit. Zbývá ověřit, že dostaneme správnou pravděpodobnost i u ostatních balíků (1 až $n-1$). Každý z nich má pravděpodobnost $\frac{1}{n-1} \cdot \frac{n-1}{n} = 1/n$ obsazovat na konci n-tého kroku odkladiště (musel se tam nacházet v předchozím kroku a museli jsme se rozhodnout nenahradit jej n-tým).

Obecné k

Pro obecné k budeme postupovat obdobně, jen na odkladišti (nyní poli délky k) budeme skladovat prozatímně vybranou *k-tici* balíků. A pochopitelně budeme chtít, aby na konci n-tého kroku měly všechny možné k-tice ze zatím načtených balíků stejnou pravděpodobnost obsazovat odkladiště.

Indukci začneme až od k-tého balíku, kdy umístíme do odkladiště rovnou všechny balíky 1 až k – mezi nimi existuje jediná k-tice s pravděpodobností 1, takže naše podmínka je určitě splněna. Nyní předpokládejme, že už máme prvních $n-1$ balíků ($n \geq k+1$) zpracovaných a dle indukčního předpokladu se v odkladišti nachází náhodná k-tice z $\{1, \ldots, n-1\}$.

Nyní zpracováváme n-tý balík a chtěli bychom získat náhodnou k-tici z $\{1, \ldots, n\}$. Libovolná k-tice z $\{1, \ldots, n\}$:

a) Buď obsahuje n. Pak je tvořena $(k-1)$-ticí z $\{1, \ldots, n-1\}$ rozšířenou o n. Každé takové $(k-1)$-tici odpovídá právě jedna k-tice obsahující n a naopak, což má dva příjemné důsledky: (1) celkem jich je stejně, $\binom{n-1}{k-1}$, a (2) náhodnou k-tici obsahující n si můžeme pořídit tak, že vezmeme náhodnou $(k-1)$-tici z $\{1, \ldots, n-1\}$ a přidáme k ní balík n. Předkládáme k intuitivnímu uvěření, že náhodnou $(k-1)$-tici získáme z náhodné k-tice z $\{1, \ldots, n-1\}$ (kterou máme z indukčního předpokladu) zahozením náhodného prvku.

b) Anebo neobsahuje n. Pak ale není ničím jiným než k-ticí z $\{1, \ldots, n-1\}$. Tedy i náhodná k-tice neobsahující n je prostě jen náhodná k-tice z $\{1, \ldots, n-1\}$. A hle, jednu takovou máme z předchozího kroku!

Už umíme vygenerovat náhodnou k-tici obsahující a neobsahující n, teď bychom chtěli tyto výsledky dát dohromady. Pravděpodobnost, že náhodná (libovolná) k-tice obsahuje n, je rovna

$$\frac{\text{počet } k\text{-tic obsahujících } n}{\text{počet všech } k\text{-tic}} = \frac{\binom{n-1}{k-1}}{\binom{n}{k}} = \frac{k}{n}.$$

Tedy náš algoritmus by měl s pravděpodobností k/n vygenerovat náhodnou k-tici obsahující n a s pravděpodobností $(n-k)/n$ náhodnou k-tici neobsahující n.

Tím jsme tedy vlastně (přinejmenším neformálně) dokázali správnost následujícího postupu v n-tém kroku:

1. Vygenerujeme náhodné číslo $r \in \{1, \ldots, n\}$.
2. Pokud $r \leq k$ (nastane s pravděpodobností k/n):
3. Vygenerujeme náhodné číslo $a \in \{1, \ldots, k\}$
4. Nahradíme a-tý (tedy náhodný) balík na odkladišti balíkem n (vygeneruje náhodnou k-tici obsahující n).
5. Jinak (s pravděpodobností $(n-k)/n$):
6. Ponecháme odkladiště beze změny a n-tý balík zahodíme (a máme náhodnou k-tici neobsahující n).

Nyní si všimněme, že 3. bod vůbec není potřeba. V nahrazovací větvi máme zaručeno, že $r \in \{1, \ldots, k\}$ a všechny tyto hodnoty mají stejnou pravděpodobnost (r jsme generovali rovnoměrně náhodně). Tedy můžeme prostě vyhodit z odkladiště r-tý balík.

Tato na první pohled technická drobnost nám umožní se na řešení podívat ještě úplně jinak. Ale o tom až za chvíli, nejprve se podíváme na program a předvedeme formální důkaz správnosti našeho algoritmu.

```
from random import randint

def vyber(baliky, k):
    buf = []
    n = 0
    for balik in baliky:
        n += 1
        if n <= k:
            buf.append(balik)
        else:
            nahrad = randint(1, n)
            if nahrad <= k:
                buf[nahrad - 1] = balik
    return buf
```

Formální důkaz

Ukážeme si jen indukční krok, zbytek je stejný jako u neformálního důkazu. Uvažujme libovolnou k-tici P z $\{1, \ldots, n\}$. Chceme ukázat, že se na odkladišti bude po n-tém kroku nacházet s pravděpodobností $1/\binom{n}{k}$. Rozebereme dva případy:

a) $n \notin P$. Pak P pochází z předchozího kroku, kde se vyskytla s pravděpodobností $1/\binom{n-1}{k}$ (z indukčního předpokladu), a aktuální krok přežila s pravděpodobností $(n-k)/n$ (rozhodli jsme se n-tý balík zahodit). Tedy pravděpodobnost výskytu P po n-tém kroku je

$$\frac{1}{\binom{n-1}{k}} \cdot \frac{n-k}{n} = \frac{1}{\frac{n}{n-k} \cdot \frac{(n-1)\cdot\ldots\cdot(n-k)}{k!}} = \frac{1}{\binom{n}{k}},$$

což jsme přesně chtěli.

b) $n \in P$. Pak P vznikla z nějaké k-tice Q nahrazením nějakého balíku x za n, tedy $P = Q \setminus \{x\} \cup \{n\}$ pro libovolnou z $n-1-(k-1) = n-k$ možných voleb x. Pravděpodobnost, že z daného Q vznikne P, je $\frac{k}{n} \cdot \frac{1}{k}$ (musíme se rozhodnout nahrazovat a vybrat k nahrazení právě x). A pravděpodobnost, že jsme v minulém kroku skončili s jedním z $n-k$ možných Q, je $(n-k)/\binom{n-1}{k}$. Celková pravděpodobnost, že po tomto kroku dostaneme P, je tedy

$$\frac{n-k}{\binom{n-1}{k}} \cdot \frac{k}{n} \cdot \frac{1}{k} = \frac{1}{\frac{n}{n-k} \cdot \binom{n-1}{k}} = \frac{1}{\binom{n}{k}}.$$

A tím je důkaz správnosti hotov. Na konci algoritmu pak budou mít všechny k-tice stejnou pravděpodobnost $1/\binom{N}{k}$ a máme požadovaný výstup. Časová složitost algoritmu je $\mathcal{O}(N)$ a paměťová $\mathcal{O}(k)$.

Alternativní řešení

Náhodnou k-tici můžeme vygenerovat také tak, že vygenerujeme náhodnou permutaci balíků a z ní vezmeme prvních k prvků. To zní trochu zblátadoloužnicky – když nemůžeme použít ani samotné N, kde bychom přišli k takové permutaci? Inu, opět inkrementálně. Budeme chtít na konci n-tého kroku držet v ruce náhodnou permutaci prvků $\{1, \ldots, n\}$. Začneme s triviální permutací obsahující pouze první balík a v každém kroku ji budeme chtít rozšířit o jeden prvek. Nyní předpokládejme, že už máme náhodnou permutaci prvků $\{1, \ldots, n-1\}$.

Chceme vygenerovat náhodnou permutaci na $\{1, \ldots, n\}$. Všimneme si, že prvek n se může vyskytovat stejně pravděpodobně na všech pozicích a že pokud ho prohodíme s prvkem na n-té pozici, dostaneme na konci n a před ním náhodnou permutaci na $\{1, \ldots, n-1\}$. Lze to ale udělat i opačně: vzít náhodnou permutaci na $\{1, \ldots, n-1\}$, na konec přidat n a pak ho prohodit s náhodně vybraným prvkem.

Tedy náhodné permutace vyrábět umíme. Teď nám ještě život komplikuje fakt, že bychom na uložení takovéto permutace potřebovali $\mathcal{O}(N)$ paměti, což si určitě nemůžeme dovolit (kdybychom mohli, prostě si do ní načteme celý vstup, spočítáme délku a vzorky vybereme přímo). My si ovšem celou permutaci pamatovat nepotřebujeme – zajímá nás jen prvních k prvků a všimneme si, že při žádné z úprav nepřenášíme informaci z jednoho místa permutace na jiné. K tomu, abychom určili, jak se změní počáteční úsek permutace, nám stačí znát ten a nově přidávaný prvek. Všechny zásahy do prvků od $(k+1)$ dál můžeme prostě z programu vyházet. Zbude algoritmus nápadně podobný předchozímu:

```
from random import randint

def vyber(baliky, k):
    buf = [-1] * k
    n = 0
    for balik in baliky:
        n += 1
        # Zvolíme místo pro přidávaný prvek
        r = randint(1, n)
        if r <= k:
            # Prohození popisované v řešení:
            # provedeme jen ty části, které
            # zasáhnou prvních k prvků.
            if n <= k:
                buf[n] = buf[r]
            buf[r] = balik
```

Uložený začátek permutace odpovídá našemu odkladišti, a náhodná místa, na která prohazujeme nově přidané prvky, jsou přesně náhodná $r \in \{1, \ldots, n\}$, která generujeme v k-ticovém algoritmu. Tato verze se liší vlastně jen tím, že v k-tém kroku začíná s náhodnou permutací balíků $\{1, \ldots, k\}$ namísto uspořádaného pořadí. Ukážeme, že je to úplně jedno.

Představme si, že permutačnímu algoritmu v k-tém kroku prostě „pod rukama“ nahradíme náhodnou k-prvkovou permutaci za $(1, 2, \ldots, k)$ a zajímá nás, jaký to bude mít vliv na jeho další průběh. Určitě to nijak nezmění pozice