Korespondenční Seminář z Programování

28. ročník **KSP** **Listopad 2015**

Milí řešitelé a řešitelky!

Počasí letos dělalo psí kusy, ale zdá se, že podzim nám připravilo takový, jaký má být. A jak už to k podzimu patří, přinášíme vám druhou sérii KSP. Najdete v ní nejrůznější úložky, které můžete řešit nejen u šálku horkého čaje, stejně jako další díl seriálu o evolučních algoritmech.

Za úspěšné řešení KSP je možno být přijat na MFF UK bez přijímacích zkoušek. Úspěšným řešitelem se stává ten, kdo získá za celý ročník alespoň 50 % bodů. Za letošní rok půjde získat maximálně 300 bodů, takže hranice pro úspěšné řešitele je 150. Maturanti pozor, pokud chcete prominutí využít letos, musíte to stihnout do konce čtvrté série, pátá už bude moc pozdě.

Připomínáme, že každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme propisku, blok, tužku, a možná i něco navíc.

V závěru úvodu chceme všechny řešitele, stejně jako kohokoli dalšího se zájmem o studium na MFF UK, pozvat na **Den otevřených dveří MFF UK**, který proběhne ve **čtvrtek 26. listopadu**. Více informací naleznete na adrese *http://www.mff.cuni.cz/verejnost/dod/*

Termín série: **Pondělí 14. prosince 2015 v 8:00 SEČ** (CodEx má termín stejný)

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

Značky úloh: Lehčí úloha vhodná pro začátečníky; Těžká úloha pro zkušené; Praktická úloha do systému CodEx; Praktická open-data úloha; Úloha řešitelná algoritmem z kuchařky; Seriálová úloha

Odměna série: Každému, kdo vyřeší **tři libovolné úlohy na plný počet bodů**, pošleme **sladkou odměnu**.

Druhá série dvacátého osmého ročníku KSP

Opojná vůně bankovek

„Dobrý oběd v restauraci.“ Co si pod touto frází představíte vy? Jistě se ve vašich myšlenkách objeví chutné jídlo, čisté stoly, cinkot příborů nebo přijatelná cena. Já, provozovatel restaurace v centru města, přitom cítím ještě něco navíc – pocit z dobře odvedené práce. Je jistě podobný tomu, co zažívá například programátor při doladění své nové aplikace, nebo dirigent, jenž posledním rozmáchlým gestem skončil symfonii a sklízí ovace diváků. A právě tímto pocitem začaly zvláštní události, o kterých vám chci vyprávět.

Mohly být tak dvě hodiny odpoledne, když jsem ve svém podniku od výčepu spokojeně sledoval hosty dojídající svůj oběd. Číšník odnášel prázdné talíře a obratem nosil na stoly dezerty. Většina lidí přišla ve skupinkách, jen blízko výčepu seděl osamoceně člověk s otevřenými novinami. Na zadní straně jsem si všiml reklamy na jakýsi katastrofický film.

28-2-1 Potopa ve městě — 10 bodů

Film se odehrává ve městě, jemuž hrozí katastrofa – mají přijít výjimečně silné deště. Město stojí na kopci a voda, která přeteče přes jeho okraj, bez problémů zmizí. Intenzivní srážky by však mohly poškodit budovy. Vědci naštěstí vymysleli způsob, jak zakrýt zdi domů, aby jimi voda neprošla. Potřebují však zjistit, jak bude celé zaplavení probíhat.

Dostanete popis města jako čtvercovou síť $N \times M$ a výšku budovy na každém poli (může být i nulová, třeba pokud se na něm nachází silnice). Voda, která nepřeteče přes okraj, je zadržena mezi budovami.

Ptáme se na celkový objem zadržené vody. Předpokládejte, že jí napršelo alespoň do výšky největší budovy.

Ukázkový vstup:	*Ukázkový výstup:*
0430 0304 3223 0450	5

Pro ukázkový příklad zůstane na „prostředních“ třech políčkách (dvě dvojky a pole nulové výšky nad pravou z nich) voda do výšky 3, vše ostatní odteče.

Dotyčný odložil noviny a já uviděl malého, shrbeného a celkem stydlivě vypadajícího muže. Mohlo mu být tak čtyřicet a byl pravidelným návštěvníkem. Všiml si, že ho sleduji, a plaše se ke mně otočil.

„Dobrý den... Jak se vám daří?“ zeptal se nejistým hlasem. „Jde to. Chutnalo vám?“ zajímalo mě.

„Jistě že ano,“ odpověděl rychle, jako by nad odpovědí nepřemýšlel a instinktivně ji vypustil z úst. Asi si to uvědomil a doplnil: „Á-ano, dnes mi chutnalo moc. Jistě, určitě.“

Byl to podivný člověk. Poprvé se tu objevil asi před dvěma měsíci a dlouhou dobu si sedal ke stolku blízko vchodu. Tvářil se, jako by chtěl mít jistotu, že může v případě potřeby rychle utéct. Až poslední dva týdny se rozhodl vyzkoušet pohodlnější místa a nakonec obsadil stůl blízko baru. Nic z toho mi nevadilo, ale jedna věc mi přišla zvláštní a nepříjemná: vypadalo to, že mě při jídle pokradmu sleduje a prohlíží. A dnes vypadal ještě nervózněji než kdy předtím.

„Nemám vám ještě něco přinést?“ s úsměvem jsem se zeptal. „Ne, já... stejně musím domů, mám nějakou práci s přerovnáváním své knihovny,“ řekl a nasadil křečovitý úsměv.

28-2-2 Řazení knih — 8 bodů

Mějme knihovnu. Je tvořena jednou dlouhou policí se spoustou knih, narovnaných těsně vedle sebe. Abychom v nich mohli vyhledávat, máme záznamy o knihách (reprezentovaných celými čísly) načtené v paměti počítače. Jsou uložené v poli a seřazené stejně jako odpovídající knihy v poličce.

Kvůli přerovnání knihovny jsme K knih z pravého konce vzali a přesunuli na levý konec. Vy máte za úkol provést změnu v záznamech v poli, aby odpovídala novému pořadí. Ale pozor, nemáte dostatek prostředků na vytvoření nového pole, změnu je třeba provést přímo v původní struktuře a smíte alokovat jen konstantně mnoho paměti navíc.

Příklad: Pro pole s hodnotami $(2, 3, 8, 7, 5, 1)$ a $K = 2$ je správným výsledkem $(5, 1, 2, 3, 8, 7)$.

„Vidíte, tohle mám na práci. Mimochodem, no, hm, jestli vás to zajímá, jmenuji se Konrád," představil se neznámý.

Konrád. To je ale jméno...

Usmál jsem se, také se představil a chystal se odejít do kuchyně, když vtom se ten člověk znovu ozval: „Já, vlastně, totiž, chtěl jsem se, víte, no, zeptat se... "

„Tak se vymáčkněte. Na co se chcete zeptat?" pohlédnu na něj a v duchu si povzdechnu. Chce si na něco postěžovat? Tak ať to vybalí, já kvůli tomu nikoho nevyháním!

Chvilku se na mě dívá. Pak kývne hlavou. A pak tu otázku položí.

„Ronny. Ronny Tlouštík. Neznáte to jméno?" vydechne.

Ale to ne! To jméno, zvuk toho jména proběhne celým mým tělem. To přeci ne! V mé mysli se rozvíří zapadlé vzpomínky a pocity překvapení. Vůbec si neuvědomuji, že na Konráda teď zírám s dokořán otevřenými ústy a zvednutým obočím. Najednou nejsem majitelem restaurace, vracím se do zašedlé minulosti a znovu slyším ten hutný hlas —

⋆⋆⋆

Asi bych vám měl povědět o tom, čím jsem se kdysi živil.

Po letech strávených v rodném městečku jsem odjel studovat na vysokou školu. Ale do jejího výběru jsem až příliš nechal mluvit své rodiče. Sice jsem jakž takž plnil své povinnosti, ale obor mě vůbec nebavil. Většinu času jsem trávil osaměle, mrzutým procházením se po městě – až jsem se jednoho temného večera dostal do městské čtvrti, která neměla mezi obyvateli dobrou pověst.

Po hodině bloumání tmavými ulicemi jsem spatřil bar a řekl jsem si, že zakončím den skleničkou něčeho ostřejšího. Podnik byl umístěný ve sklepě omšelé budovy s popadanou omítkou, a proto mě překvapil čistý a kupodivu poměrně luxusně zařízený interiér. Jedinými hosty byla halasně oslavující skupinka mužů, sedící v rohu místnosti. Krátký pohled na nabídku nápojů prozradil, že se svým stavem financí si nemohu dovolit snad ani lahev vody.

„Bez peněz? Zvu vás!" ozval se jasný, veselý hlas. Přede mnou najednou stál muž vysoké postavy, očividně silný, ale také tlustý. Hlavu měl plešatou. Byl oblečený ve stylovém bílém obleku a smál se od ucha k uchu.

„Oslavujeme!" zvolal nadšeně a přivedl mě ke stolu. Jeho spolusedící, evidentně opilí, mě vzali mezi sebe. „Ronnymu se očividně líbíš," podotkl jeden z nich. „Možná by tě mohl pozvat na nějakou naši recesi," zasmál se druhý.

„Recesi?" zeptal jsem se. Ale to už na stole přistál kalíšek whisky určený pro mě.

Oslava pokračovala, já zůstal a k jedné skleničce se přidala druhá, třetí. Ronny (o jeho nelichotivém přízvisku Tloušťík jsem se dozvěděl později) se mě vyptával na jméno, život, bydliště a neustále se smál. Když čas hodně pokročil, dal Ronny pokyn barmanovi a ten zamkl dveře.

„Ne abys mluvil o tom, co teď uvidíš," řekl. Neznělo to jako výhrůžka, ale jako nepsaná dohoda dvou kamarádů. Jeden z lidí na stůl přinesl dvě černé tašky, dosud nevinně stojící v koutě, a na stůl vytáhl jejich obsah.

Z jedné vypadla pistole s náboji. A ta druhá byla plná peněz, skutečných bankovek.

Tak tohle byla ta jejich recese. Sedím tady s kriminálníky, o kterých jsem předtím jen četl v novinách, a oslavuji s nimi jejich povedené vyloupení banky!

Ronny vzal několik bankovek vysoké hodnoty, srovnal je do úhledného balíčku a zeptal se: „Co bys řekl na to přidat se k nám? Co může člověk jako ty ztratit?"

Vrátil jsem se domů celý rozechvělý. Do rána jsem vystřízlivěl, ale opojení z peněz, jež ke mně takhle jednoduše doputovaly, zůstalo.

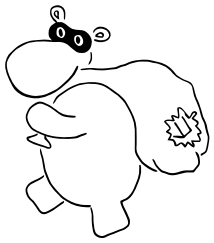
A tak jsem se brzy ke zločinecké partě připojil. Ronny, ač vypadal tak nevinně, byl jejich kápo a měl všude své kontakty. Setkání, kterému jsem byl přítomen, nedělali kvůli bezpečnosti příliš často. Zasílání zpráv všem členům obvykle probíhalo přes editora článků zaměstnaného v městských novinách.

28-2-3 Zprávy pro lupiče — 10 bodů

V novinovém článku jsou zakódované zprávy pro zločince. Určité slovo, představující zprávu, je do textu vložené jako vybraná podposloupnost – to znamená, že slovo získáme vybráním určitých znaků z textu článku (ale nesmíme změnit jejich pořadí). Celý systém je poměrně složitý a záleží i na tom, kolikrát je slovo do textu vloženo.

Obdržíte text článku a slovo a musíte najít všechny výskyty slova v článku. Pro jednoduchost předpokládáme, že se ve slově neopakují znaky. Na výstup vypište pozice písmen, které vyberete z článku a dají dohromady hledané slovo.

Ukázkový vstup:	*Ukázkový výstup:*
`11`	`1 5 7`
`aanubbcahoj`	`1 6 7`
`3`	`2 5 7`
`abc`	`2 6 7`



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

Netrvalo dlouho a ocitl jsem se v první akci. V nočních hodinách jsme vykrádali bankovní trezor. Ostatní členové se dostali dovnitř a já hlídal, zda se nikdo nepovolaný neblíží. Povedlo se a já s úsměvem poslouchal zprávy o bezradných vyšetřovatelích, když jsem si z tajného místa odnášel podíl z loupeže.

Pak jsem povýšil a učil jsem se, jak odemknout který zámek, jaký drát přestřihnout k vypnutí zabezpečovacího systému nebo jak správně omráčit nočního hlídače.

Odloučil jsem se od rodiny, pořídil si vlastní byt a užíval si toho, že mi stačí jednou za několik měsíců jít do nebezpečné akce a pak dlouho odpočívat a dělat, cokoliv se mi zlíbí. Ať už ve městě, nebo na ostrově v Karibiku. „Vydělaných" peněz přibývalo a ani jsem nevěděl, za co všechno je utratit.

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

Několikrát jsem se setkal s Ronnym. Vždy byl ve skvělé náladě, vždy byl skvěle oblečený a za celou dobu neshodil ani kilo. Věděl jsem, že je ve skutečnosti nekompromisní a dokáže být tvrdý ke svým nepřátelům. Ale říkal, že ve mě od samého začátku věřil a že se nemám ničeho bát. To, že by mě jen tak podrazil, mě nenapadlo ani ve chvíli, kdy trochu propadl megalomanii a začal plánovat vloupání se do několika bank současně.

28-2-4 Útěk z města — 10 bodů

Zločinecký gang plánuje velkou akci. Chtějí se simultánně vloupat do všech poboček banky ve městě, využít chaosu a utéct z města pryč do bezpečí. Poslední část není tak jednoduchá, jak se zdá: na místě, kterým proběhne lupič, se shromáždí policejní hlídky a přes ně už nikdo další neproběhne.

Město je reprezentováno čtvercovou sítí. Na každém poli je buď prázdné místo, jímž se dá proběhnout, nebo budova. Také máte zadané souřadnice jednotlivých poboček. Na každou pobočku připadá jeden zločinec.

Najděte pro každého z nich cestu z banky kamkoliv na okraj mapy tak, aby cesta vedla jen po průchozích polích a každé pole bylo použito maximálně jednou (nepočítejte s tím, že dva zločinci mohou jedním polem proběhnout současně – tak dobře se nemají šanci zkoordinovat). Není možné se pohybovat po diagonálách.

Já a ještě jeden člen gangu jsme dostali za úkol se postarat o centrální banku v samém srdci města. Nebezpečná akce začala tím, že jsme pronikli do honosné dvorany budovy. Odsud to bylo jen několik chodeb k lákavému obsahu, skrytému za kovovými dveřmi.

Samotné otevření trezoru a vyzvednutí peněz proběhlo až podezřele bez potíží. Teď bylo třeba utéct přes propojovací chodbu do vedlejší budovy. Vrátili jsme se do dvorany. Uprostřed rozlehlé, potemnělé místnosti se můj partner náhle zastavil. Z jeho tváře jsem vyčetl strach a zklamání. A já najednou cítil to samé.

„Je mi to líto," řekl. Sáhl po revolveru, ale pak ruku zase svěsil a potřásl hlavou. A takto celá melodramatická scéna skončila, protože těsně poté se ve dvoraně rozsvítila světla a já viděl početný zástup policistů, rozestoupených v rozích a mířících na nás zbraněmi.

* * *

Podivné období života s lehce vydělanými penězmi skončilo. Procitl jsem ze snu, uvědomil si, že ne všechno bylo takové, jak to vypadalo. Ale bylo pozdě. Ronny Tlouštík nás dva zradil a anonymně oznámil vloupání do centrální banky na policii. Policisté se na toto místo zaměřili a díky tomu měli lupiči z ostatních poboček volnou cestu k útěku.

Prošedivělý soudce mi oznámil dobu trestu a moje další cesta vedla do vězení. První rok v novém prostředí byl krušný. Těžko jsem si zvykal na stísněnou celu a osamělost. Vězeňští bachaři od nás vyžadovali naprostou poslušnost, ačkoliv sami byli poměrně vybíraví. Dlouhou dobu se například dohadovali, kdo bude hlídat který vězeňský blok.

28-2-5 Hlídání věznice — 10 bodů

Věznice je rozdělena na mnoho menších bloků. Blok hlídá právě jeden bachař. Pracuje se na dvě směny, denní a noční, a každý hlídač pracuje v obou z nich.

Na začátku roku se rozpis hlídání mění a bachaři přišli se svými požadavky. Každý přinesl seznam svých oblíbených bloků, které je ochoten hlídat. A protože by se jen v jednom nudil, je třeba, aby blok přidělený ve dne a v noci byl odlišný.

Na základě požadavků přiřaďte každému bloku dva bachaře, z nichž jeden jej bude hlídat ve dne a druhý v noci. Nezapomeňte, že hlídač může v jednu chvíli střežit jen jeden blok.

Formát vstupu: Na prvním řádku dostanete čísla celá čísla B a P, udávající po řadě počet bloků/bachařů (musí být stejný, jinak by řešení neexistovalo) a počet preferencí. Následuje P řádků popisujících preference bachařů. Každý z nich obsahuje dvě čísla h_i a b_i ($0 \leq h_i, b_i \leq B-1$) a znamená „bachař h_i je ochoten hlídat blok b_i".

Platí $2 \leq B \leq 30\,000$ a $4 \leq P \leq 125\,000$, ale spousta vstupů je mnohem menší.

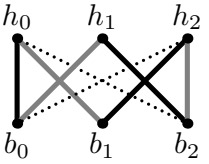
Formát výstupu: Na výstup vypište B řádků popisujících přiřazení jednotlivých bachařů. i-tý řádek popisuje i-tého bachaře a obsahuje dvě čísla d_i a n_i udávající po řadě blok, který bachař hlídá v denní a v noční směně (tedy záleží na pořadí těchto čísel).

Ukázkový vstup:

```
3 8
0 0
0 1
0 2
1 0
1 2
2 0
2 1
2 2
```

Ukázkový výstup:

```
0 1
2 0
1 2
```



Na obrázku silné čáry značí přiřazené směny (černé denní, šedé noční) a tečkované nepoužité preference. Pro jeden vstup existuje více správných výstupů. Například pokud všechny denní směny prohodíte za noční a naopak, dostanete opět platné řešení.

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

Vlastně vůbec nevím, co by se se mnou stalo, kdyby jednoho dne do mé cely nepřiřadili nového spoluvězně. Byl starší než já, očividně ve vězení strávil už pěkně dlouhou dobu a vypadalo to, že tu zůstane ještě déle. O své minulosti nikdy moc nemluvil. Zdálo se mi však, že se smířil se svým osudem a že je z něho cítit neobyčejná vyrovnanost, jakou jsem předtím nikdy nezažil. I hlídači se k němu chovali s větší úctou než k ostatním.

„Já se odsud už nedostanu," říkal mi, „ale tebe jednou pustí, tak přemýšlej o tom, co budeš tam venku dělat!" Apeloval na mě, ať využiji každé možnosti pracovat a zjistit, jaká legální činnost mě na svobodě bude živit.

A tak jsem se dostal jako pomocník do vězeňské kantýny. Vaření mě začalo bavit a postupně jsem se zlepšoval. Stal jsem se kuchařem vařícím pro celé vězení a odsud byl jen krok k předčasnému propuštění za dobré chování. Personál

kuchyně mi gratuloval a daroval mi nástroj, který jsem si oblíbil: velkou litinovou pánev.

Ale kam se mám teď vydat? přemýšlel jsem za branou věznice. K rodině jsem se jít styděl. Ke zločineckému gangu jsem se vrátit nemohl, i kdybych snad chtěl. Většina jeho členů byla zadržena při nepovedené loupeži, o kterou se pokusili, když jsem byl za mřížemi. Ronny Tlouštík byl za nespočetné množství loupeží a několik vražd odsouzen na doživotí a jeho sbírka dvaceti obleků pro každou příležitost se prodala v nějaké státní aukci.

Rozhodl jsem se jet někam, kde mě nikdo nezná, a odjel jsem do velkoměsta na druhé straně země. Zpočátku jsem tam zažíval krušné chvíle – než jsem si dokázal vydělat na nájem, nezbylo mi nic jiného, než přespávat v nočních linkách městské hromadné dopravy.

28-2-6 Cesta MHD — 12 bodů

Máme k dispozici kompletní jízdní řády všech vozidel městské hromadné dopravy. Trasa každého vozidla je popsána jako seznam dvojic zastávka-čas (předpokládáme, že čas odjezdu je stejný jako čas příjezdu). V zastávce lze mezi vozidly přestupovat, ale abychom měli jistotu, že vše stíháme, musí být mezi časem příjezdu prvního spoje a časem odjezdu druhého spoje rozdíl alespoň λ minut, jež také obdržíte na vstupu.

Najděte nejdelší možnou cestu v síti (délku měříme celkovým časem stráveným ve vozidle) při dodržení času na přestup.

Lehčí varianta (za 6 bodů): Řešte stejnou úlohu za předpokladu, že $\lambda = 0$.

A tím jsme se dostali až do současnosti. Vařením jsem se ve městě dokázal uživit, až jsem sehnal dostatek peněz na zakoupení své vlastní restaurace. A teď v té restauraci stojím a překvapeně poslouchám Konráda, který moji minulost zná...

⋆⋆⋆

Pomalu se vrátím do reality a vrhám na podivného hosta tázavý pohled.

„Já... pracuji v archivu...“ začal Konrád vysvětlovat. „Narazil jsem na... váš případ. Na slyšení před soudem... kdy jste vysvětloval, jak jste se prolomil do té poslední banky a otevřel trezor.“

Chvíli mi trvá, než si vzpomenu. „Jistě,“ odpovím. „Uzavřené pro veřejnost. Popisoval jsem chyby v jejich zabezpečení. Jak jste na to přišel?“ ptám se ho ostře.

Konrád se lekne mého zvýšeného tónu a znovu začne koktat. „Zvědavost, no, vždyť víte... V archivu, tam, tam se člověk nudí, začne, no, začne si číst... a pak nemůže přestat...“

Potřesu hlavou a ptám se: „A proč jste za mnou vlastně přišel?“

Odpovídá: „Můj blízký přítel v té bance... víte, pracuje tam. Je tam... říkal, že je tam pořád stejné zabezpečení... pořád... byste se tam uměl dostat.“

Je to, co říká, možné? Uvažuji o tom, ale pak mi dojde, co má na mysli. „Vy chcete, abych se tam vloupal znovu!“ šokovaně odpovídám.

„Ten... kamarád by nám s tím pomohl. Vy máte zkušenosti, on nám pomůže... a já to zorganizuji.“

Vyjeveně se na něj dívám. Ale najednou mi začnou do mysli pronikat vzpomínky. Bankovky, spousta bankovek a jejich typická vůně. Nadšení při každé povedené akci a každém otevřeném zámku. Slunné dny na Havaji, strávené nicneděláním. Podívám se po restauraci, kterou jsem si vlastníma rukama vybudoval, a najednou necítím žádný pocit z dobře odvedené práce.

„Přijďte sem později,“ zašeptám.

⋆⋆⋆

Je večer a já sedím v kuchyni svého podniku. Zbytek dne po rozhovoru jsem strávil celý nesvůj. Předtím jsem se nechtěl vrátit mezi kriminálníky, ale teď? Co se to se mnou vlastně děje? Z přemítání mě vyruší zaklepání na dveře. Je to Konrád. Tváří se napjatě a v ruce drží kufřík.

„Mám pro vás nějaké podklady,“ řekne mi. Položí kufřík na desku stolu. Na jeho boku se nachází displej a červeně na něm svítí tři čísla. „Nejnovější technika,“ usměje se a začne na malé klávesnici vedle displeje zadávat kód k otevření. Asi třikrát se splete, než trefí správné číslo.

28-2-7 Otevření kufříku — 10 bodů

Firma vyrábějící kufříky s přístupem na kód chce zajistit, aby číselné heslo nebylo konstantní. Typ kufříku, který používá Konrád, zobrazí na svém displeji několik čísel. Ty slouží jako parametry určité funkce a pro otevření je třeba zadat jejich výsledek.

Displej Konrádova kufříku zobrazí čísla A, B a K. Abyste kufřík otevřeli, zjistěte, kolik se mezi čísly A a B vyskytuje čísel, jejichž binární zápis obsahuje právě K jedniček.

Lehčí varianta (za 5 bodů): Předpokládejte, že $A = 0$ a $B = 2^n - 1, n \in \mathbb{N}$.

Zvědavě jsem nakoukl dovnitř. Na vrchu kufříku jsem uviděl několik složek. Vypadalo to, že obsahují nákresy vnitřních prostor banky a popis zabezpečení. To ale Konráda zatím nezajímalo. Odložil vrchní obsah na stranu a na dně kufru jsem spatřil – co je tohle za déjà vu? – ony zelené papírky, po kterých lidi tolik touží.

Na chvíli strnu a nasávám onu opojnou vůni peněz, jež mě zavedla do tolika problémů. Pak se podívám na Konráda, jenž má ve tváři tázavý výraz. „Jdete do toho?“

Nadechnu se a vztáhnu ruku po bankovkách. A v okamžiku, kdy se jich mám dotknout, se to stane.

Najednou slyším výkřik, pád a sypání krabic. Vyděsím se. Je tu někdo jiný! Odkud ten zvuk jde? Zaslechnu klení a další zvláštní zvuky. Vychází ze skladu, odděleného od kuchyně dveřmi. Než stihnu cokoliv udělat, dveře jsou s neskutečnou silou vyraženy a já se dívám do očí podivnému člověku. Má na sobě podivné černé oblečení a jeho tvář je rozezlená. Začne se rozhlížet po kuchyni.

„Tohle si vezmu,“ procedí mezi zuby a ze zdi něco sundá. Co to dělá? To je má památeční pánev, kterou jsem si odnesl z vězení! Chci mu ji vytrhnout z ruky, ale on rychle uskočí a pánví se po mně ožene. „Na tohle nemám čas,“ zamumlá a vyběhne přes restauraci ven do ulice.

Nemohu uvěřit tomu, co se právě stalo. Ohlížím se po Konrádovi, ale ten zbabělec je pryč. Včetně kufříku a jeho lákavého obsahu. Nevěřícně se dívám do ulice a najednou mi do duše padne příjemný klid.

⋆⋆⋆

Asi jste nečekali takový konec, že ano? Doufal jsem, že celá záležitost s tím mužem se nějak vysvětlí. Ale nevysvětlilo se nic. To, jak se dostal do skladu, zůstalo záhadou. Jediným vchodem do té místnosti, vyjma dveří do kuchyně, je mřížka ventilace. Přes ni by neprokouzl. Nebo že by přišel přes kuchyň ještě předtím, než jsem se tam dostal já?

Ale co tam pohledával? Že by si chtěl něco odnést a pak se takhle hloupě prozradil? A proč potřeboval právě pánev. . . ?

Podobné otázky mě dlouho nenechávaly spát, a když jsem někdy večer zůstal sám doma, bál jsem se, co by na mě mohlo vylézt ze skříně. Alespoň že vím, že ten neznámý byl skutečně člověk.

Zem se slehla i po Konrádovi. Už nikdy nepřišel na oběd a ani jinde jsem ho nezahlédl. Zajímalo by mě, co se s tím mužem, jenž chtěl rychle přijít k penězům, vlastně stalo. Nemyslím si, že svůj plán někdy úspěšně dokonal. Chyběla mu jedna základní vlastnost každého zločince: drzost. Nejspíš zůstane pracovat v archivu a myšlenka na vloupání zůstane jenom snem.

Ale nenechte se mýlit: za to, co se nakonec stalo, jsem ve skutečnosti vděčný. Nechápu, jak jsem mohl uvažovat o tom, že bych se vrátil do světa zločinu. V každém případě, ukradení pánve vyvolalo vzpomínky na mého skvělého spoluvězně a na to, že ve mě věřil. Snad to bude dostatečná vzpruha do dalších let.

Pocit z dobře odvedené práce? Až se někdy půjdete najíst do mé restaurace a uvidíte mě stát u baru, zeptejte se, zda ho stále cítím. A kdybych se na vás tvářil rozpačitě a díval se po vaší peněžence? Pak mě raději rychle něčím přetáhněte.

Kuba Maroušek

28-2-8 Genetika vs. procházení krajiny 16 bodů

V prvním díle seriálu jsme si představili genetické algoritmy, jejich operátory a základní funkčnost. V tomto díle se postupně dostaneme k verzi algoritmů, které pracují s jedinci tvořenými reálnými čísly, a získáme aspoň základní porozumění, proč by vůbec takový algoritmus měl fungovat.

Než se však dostaneme přímo k těmto otázkám, představíme si základní postupy optimalizačního prohledávání prostoru řešení.

Všechny optimalizační problémy se dají formulovat tak, že máme zadanou n-rozměrnou funkci f, která jako vstupní parametry dostane n reálných čísel a odpoví jedním reálným číslem. Funkci f se pak snažíme maximalizovat, resp. minimalizovat. To jest hledáme takové vstupní parametry, pro které bude výsledek funkce největší, resp. nejmenší možný.

Takovou funkcí může být například:

$$f(x, y, z) = 3x^4 + 2(y + 4)^2(z - 2)^2$$

Pokud ji chceme minimalizovat, optimálním řešením jsou hodnoty $x = 0$, $y = -4$, $z = 2$, pro které $f(0, -4, 2) = 0$.

To je jednoduché, ne? Bohužel ale jen v tomto případě. My obvykle nemáme žádný takto jasný předpis a často ani nevíme, jaká je optimální hodnota funkce. Většinou jen známe počet vstupních parametrů a pro jejich konkrétní hodnoty umíme spočítat hodnotu funkce.

Příkladem takových funkcí mohou být všechny fitness funkce z minulého dílu seriálu a také všechny cílové funkce, které se objeví v tomto díle.

Metoda horolezení (hill climbing)

Budeme pracovat s funkcemi reálných proměnných, které popisují, jak dobré je řešení nějakého problému. Takové funkce jsou obvykle rozumně spojité. To znamená, že kdybychom si graf funkce nakreslili, tak nám její povrch bude připomínat krajinu. Budou na ní kopce, údolí, nadmořská výška bude plynule přecházet a zřídkakdy narazíme na svislý útes nebo propadliště. Prostě taková obyčejná krajina, kterou všichni známe.

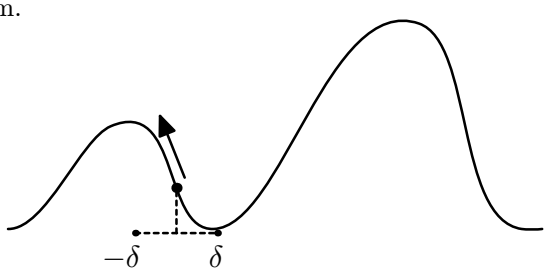
Nadále si dovolíme předpokládat, že všechny funkce mají tvar nějaké takové krajiny. Pak si pod optimalizační úlohou pro takovou funkci si můžeme představit hledání nejvyšší hory (v případě maximalizace) nebo nejhlubšího údolí (v případě minimalizace).

My se budeme věnovat hledání nejvyšších hor a ukážeme si metodu, která se nazývá *hill climbing* (česky metoda horolezení). Metoda si na začátku náhodně vybere start (náhodný bod v krajině) a z něj začne šplhat nahoru na kopec, dokud to jde.

Šplhání probíhá tak, že se náhodně zvolí bod z okolí místa, kde právě stojíme, spočítáme v něm hodnotu funkce, a pokud je stejná nebo vyšší než aktuální, přesuneme se tam. Celý postup opakujeme po daný počet iterací.

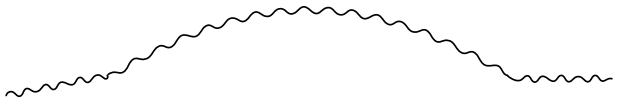
Bod přesunu vybíráme tak, že ke každé souřadnici přičteme náhodnou hodnotu z rozmezí $\langle -\delta, \delta \rangle$, kde δ je námi určená konstanta. Ta se pro začátek algoritmu volí trochu větší (povolujeme velké skoky a hledáme kopec) a v závěrečné fázi naopak hodně malá (už jsme na kopci a jen se přibližujeme vrcholu).

Tento algoritmus má však jednu značnou nevýhodu: skončíme na prvním kopci, který najdeme, a vůbec nevíme, zda třeba někde není další a ještě vyšší. Řečí matematiky najdeme nějaký lokální extrém, o kterém nevíme, jestli je i globálním.



To můžeme napravit tak, že metodu pustíme vícekrát za sebou a ze všech pokusů vybereme ten nejlepší. To už vypadá lépe – když vylezeme na více kopců, tak tím zvýšíme pravděpodobnost, že jsme se alespoň jednou ocitli na tom úplně nejvyšším. Ale. . .

Co když naše krajina bude vypadat jako zorané pole? Tam pak jsou všude samé malé kopečky, a ať začneme na jakémkoliv místě, hned na některém z nich skončíme. Tomu bychom chtěli nějak zamezit.



Simulované žíhání

Problém „malých kopečků“ řeší další metoda, která se nazývá *simulované žíhání*. Ta dělá přesně to, co metoda horolezení, jen navíc dovoluje s určitou pravděpodobností přejít i do bodu nižšího než aktuální.

Pravděpodobnost, s jakou si dovolíme přejít níž, se řídí dvěma faktory. Prvním je velikost změny od aktuální hodnoty (tu budeme značit Δf) a druhým je takzvaná *teplota* (značená T). Čím vyšší teplota, tím vyšší pravděpodobnost, že změnu přijmeme.

Na začátku algoritmu nastavíme teplotu vysokou a v průběhu ji pomalu snižujeme až skoro na nulu. Snižováním

teploty snižujeme toleranci na velikost poklesu. Pro danou velikost snížení Δf a danou teplotu T má pravděpodobnost přijetí přesně hodnotu $e^{-\Delta f/T}$.

Snižování teploty můžeme provádět přenásobením konstantou α, kterou zvolíme z intervalu $(0, 1)$. Tomu se říká chladící schéma. Následuje pseudokód algoritmu simulovaného žíhání pro jednorozměrnou funkci (funkci jedné proměnné).

1. $T = T_0, \delta = \delta_0$
2. Vyber počáteční bod $x = x_0$.
3. Urči maximální počet iterací M a číslo iterace $i = 0$.
4. Dokud $i < M$
5. $\quad y = x + \text{rand}(-\delta, \delta)$ *(náhodný posun)*
6. $\quad$ Pokud $f(y) > f(x)$
7. $\quad\quad x = y$ *(je vyšší, hned přijmi)*
8. $\quad$ Jinak
9. $\quad\quad$ Vygeneruj náhodně hodnotu $r \in \langle 0, 1 \rangle$.
10. $\quad\quad$ Pokud $r < e^{\Delta f/T}$, pak $x = y$ *(přijmeme s danou pravděpodobností)*
11. $\quad$ Pokud x je zatím nejlepší řešení, zapamatuj si jej.
12. $\quad T = \alpha T$
13. $\quad$ Volitelně můžeme snížit δ.

Algoritmus pro více proměnných je naprosto shodný, jen s proměnnou x pracujeme jako s vektorem a operace provádíme zvlášť na všechny jeho složky. Celý algoritmus stejně jako u horolezení pouštíme vícekrát a ze všech řešení vybereme to nejlepší.

Na závěr této sekce ještě poznamenáme, že pro výběr dalšího bodu se často používá posun podle Gaussova normálního rozdělení. Protože to se ale na středních školách často nevyučuje, zvolili jsme jednoduší postup, který také funguje dobře.

Úkol 1 [7b]: Zkuste pomocí metody horolezení, simulovaného žíhání nebo nějakým vlastním způsobem vyřešit následující úlohu.

V rovině máme rozmístěných dohromady p bodů. Chtěli bychom tam přidat k centrálních stanic tak, aby součet vzdáleností všech bodů do jejich nejbližší stanice byl minimální.

Pro zadané vstupní body[2] řešte postupně pro $k = 1, 3, 5$. Na prvním řádku najdete počet bodů p. Na dalších p řádcích jsou vždy dvě čísla udávající souřadnice jednoho z bodů.

Na úlohu můžete vyzkoušet i algoritmy z dalšího textu. Napište, co jste zkoušeli a jak vám to fungovalo. Který z přístupů vám fungoval nejefektivněji?

Společně s popisem řešení pošlete i průběh vašeho algoritmu společně s nejlepšími dosaženými řešeními.

Explorace versus exploatace

Nyní odhalíme, proč jsme vůbec simulované žíhání a metodu horolezení probírali v souvislosti s genetickými algoritmy. Prvním důvodem je, že všechny tyto algoritmy mají společný cíl, totiž maximalizaci či minimalizaci cílové funkce. Druhým důvodem pak je, že oběma těmto skupinám se pokusíme porozumět pomocí pojmů explorace a exploatace.

Pojem *explorace* zastřešuje objevování nových částí prostoru řešení. To jest pokud se náš algoritmus podívá na hodně míst krajiny funkce, tak hodně exploroval.

Pojem *exploatace* naproti tomu zastřešuje lokální prohledávání a využívání informací, které jsme již objevili. Tedy hledání kopce na nějakém lokálním místě v krajině patří do exploatace.

Při návrhu optimalizačního algoritmu se snažíme o vyvážení explorace a exploatace. Pokud algoritmus bude málo explorovat a hodně exploatovat, tak bude mít tendence nalézat lokální extrémy a držet se jich. Naopak pokud bude hodně explorovat a málo exploatovat, tak se bude blížit náhodnému tipování bodů. Sice jich hodně vyzkouší, ale nevyužije pořádně informace o tvaru jejich okolí.

Metoda horolezení i simulované žíhání v prvním kroce explorují (vyberou náhodný bod) a pak už jen exploatují (zkoumají aktuální okolí). Tento nedostatek explorace se pak snaží dohnat opakovaným spuštěním celého algoritmu.

Nyní pojďme rozebrat genetické algoritmy. Generování počáteční populace a opakované spouštění algoritmu patří jednoznačně do explorace. Operátor selekce se na druhou stranu řadí do exploatace (z aktuálních jedinců ponecháváme jen ty nejlepší). Zbývá nám zařadit operátory, které s jedinci přímo manipulují: křížení a mutaci.

Mutace je v genetickém algoritmu považována za představitele exploračního operátoru – přinášíme do jedince náhodnou novou informaci. Křížení se naopak považuje za operátor exploatační, protože pouze skládá dohromady informace, které již v jedincích máme.

Pohled na křížení jako na exploatační operátor může na první pohled vypadat neintuitivně. Vždyť přece křížením dvou jedinců se najednou dostaneme na úplně nové místo v krajině… To je sice pravda a z tohoto pohledu křížení může vypadat trochu exploračně, ale stále platí fakt, že jsme využili jen informace, které jsme již měli. Takže křížení v jistém smyslu funguje lokálně, ale naprosto jiným způsobem a v jiném rozsahu než například metoda horolezení nebo simulované žíhání.

Důvodem, proč např. genetické algoritmy vnímáme nadějně, je právě dobré zastoupení jak explorace, tak exploatace. Algoritmus prohledává okolí hned na několika místech najednou, tyto informace kombinuje dohromady a při tom se zároveň snaží objevovat nová místa.

Genetické algoritmy v reálných číslech

Nyní se podíváme na to, jak bychom genetickým algoritmem mohli řešit problém vyžadující reálné jedince (vektor reálných čísel). Pak jej budeme moci porovnat s metodami výše. Jednotlivé hodnoty jedinců budeme nazývat složky.

Takový genetický algoritmus bude fungovat naprosto stejně jako ten z prvního dílu seriálu, jen pro něj musíme navrhnout operátory křížení a mutace, které dokáží s reálnými jedinci pracovat.

Mutaci můžeme realizovat obdobně – náhodně pohneme s jednou či více složkami jedince. Realizujeme přičtením

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

náhodné hodnoty z intervalu $\langle -\delta, \delta \rangle$. Další možností je vygenerovat novou hodnotu z daného rozsahu.

Křížení můžeme dělat jednobodové, stejně jako v minulém díle, anebo s jedinci můžeme pracovat více jako s vektory a počítat například jejich průměr. Případně místo průměru můžeme počítat konvexní kombinaci, která pro jedince x a y vypadá následovně:

$$z = ax + (1 - a)y \; ; \; a \in \langle 0, 1 \rangle \, ,$$

kde hodnotu a můžeme mít fixní, volit náhodně, nebo volit na základě fitness obou jedinců.

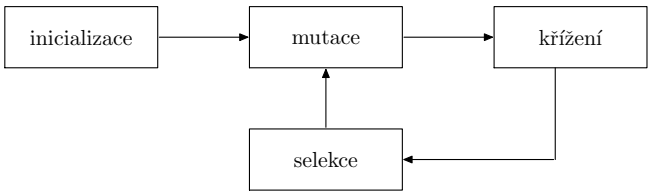
Oba operátory se dají uchopit ještě mnoha dalšími způsoby. Mně se například na operátoru křížení nelíbí to, že opakovaným průměrováním hodnot si jedinci budou navzájem čím dál podobnější. Časem budou všichni téměř stejní, blízko průměru původních hodnot. Přesto bych ale chtěl pracovat s jedinci jako s vektory hodnot a nějakým způsobem využívat informace, které v sobě uchovávají, a dostávat z nich nová, doposud nepoznaná řešení. Práci s vektory hojně využívá diferenciální evoluce.

Diferenciální evoluce

Diferenciální evoluce je specifická verze genetického algoritmu, ve které se s jedinci pracuje jako s vektory reálných čísel. Také využívá operátory selekce, křížení a mutace, ale přistupuje k nim jiným způsobem než genetické algoritmy.

Průběh diferenciální evoluce vypadá následovně:

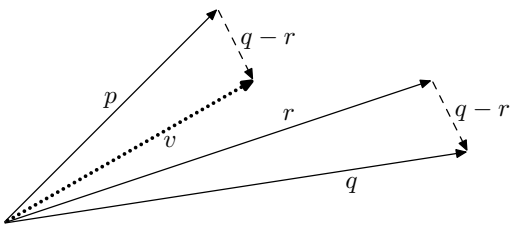
1. Inicializuj populaci n náhodnými jedinci o velikosti d.
2. Opakuj následující:
3. Proveď mutace.
4. Proveď křížení.
5. Proveď selekci.



Nejdříve aplikujeme mutaci, ta probíhá tak, že pro každého jedince vytvoříme takzvaného dárce (donor) z dalších tří náhodných jedinců. Pro jedince x vytvoříme donora v z náhodných jedinců p, q, r:

$$v = p + F \cdot (q - r),$$

kde F je reálný parametr z intervalu $[0, 2]$, kterému se říká *diferenciální váha*. Sice se teoreticky povoluje váha až 2, ale v praxi se vyplatí používat hodnoty jen do 1.



Operace mutace probíhá s celými vektory po složkách. Vezmeme směr vektoru jednoho jedince, přičteme k němu rozdíl směrů dalších dvou jedinců a získáme našeho dárce, kterého využijeme v dalších fázích.

Dále je na řadě křížení. To nastává s pravděpodobností $C \in \langle 0, 1 \rangle$. Křížíme původního jedince x s jeho dárcem v a vytvoříme tak výsledného jedince u. To provedeme tak, že vygenerujeme náhodné číslo $r \in \langle 0, 1 \rangle$. Pokud $r < C$, tak položíme $u = v$, jinak $u = x$, až na jednu náhodnou složku j, kterou vezmeme z v.

Neboli:

$$u_i = \begin{cases} v_i & \text{pokud } r < C \text{ nebo } i = j \\ x_i & \text{pokud } r \geq C \text{ a } i \neq j \end{cases}$$

Pro hodnotu C je většinou dobrá první volba $C = 0.5$.

Jako poslední v sérii operací je selekce. V té pouze porovnáme fitness původního jedince x s fitness výsledného jedince u a do další generace vezmeme jen lepšího z nich.

Takto vypadá celá jedna iterace. Náhodní jedinci pro dárce se určí na základě tří náhodných permutací jedinců. To znamená, že během jedné iterace se každý z jedinců použije právě jednou jako p, právě jednou jako q a právě jednou jako r. Navíc existuje i verze algoritmu, kdy se za jedince p vždy dosadí aktuálně nejlepší jedinec z populace (tím všichni dárci vychází ze směru stejného, nejlepšího jedince, což nemusí být vždy výhodné).

To je celý algoritmus. Má výhodu i v tom, že se díky rovnicovému zápisu dá naprogramovat o něco lépe než například klasický genetický algoritmus. Všimněte si, že je nám dokonce i jedno, zda fitness funkci maximalizujeme či minimalizujeme.

Závěrem okomentujeme, jak volit velikost populace. Ta z logiky algoritmu musí být alespoň 4. Avšak většinou se n volí velikostí mezi $5d$ a $10d$, kde d je velikost (počet složek) jedince. Je to ale pouze doporučení, není žádný důvod, proč nezkusit třeba fixní hodnotu mezi 40 a 100.

Úkol 2 [9b]: Pomocí genetického algoritmu v reálných číslech a diferenciální evoluce zkuste řešit následující úlohu.

Máme zadaný velký obdélník o rozměrech $W \times H$ a sadu k malých obdélníků o rozměrech $w_1 \times h_1, \ldots, w_k \times h_k$. Naskládejte malé obdélníky do velkého tak, aby se celkově co nejméně překrývaly. Celkový překryv je součtem překryvů všech dvojic obdélníků. Za překryv se navíc počítá i vybočení ven z velkého obdélníka.

Úlohu řešte pro data, která najdete na stránce seriálu.[3] Na prvním řádku jsou čísla W a H, na druhém řádku pak počet obdélníků k a na dalších k řádcích jsou vždy dvě čísla: w_i, h_i – rozměry obdélníka i.

Opět vyzkoušejte různé kombinace parametrů. Úlohu můžete zkusit vyřešit i jiným, neevolučním způsobem, váš výsledek do evoluce uměle dosadit a zkusit jej ještě zlepšit.

Při řešení můžete využívat šablonu genetického algoritmu z minulého dílu nebo novou šablonu pro diferenciální evoluci. Obě najdete na stejné stránce jako vstupní data.

Společně s popisem řešení pošlete i průběh vašeho algoritmu a nejlepší řešení, jakého jste dosáhli.

Karel Tesař

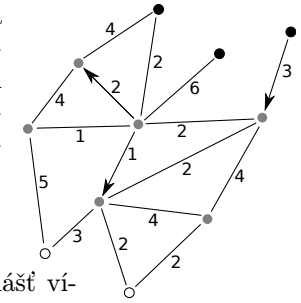
[2] `http://ksp.mff.cuni.cz/viz/evoluce`

Recepty z programátorské kuchařky: Toky v sítích

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.

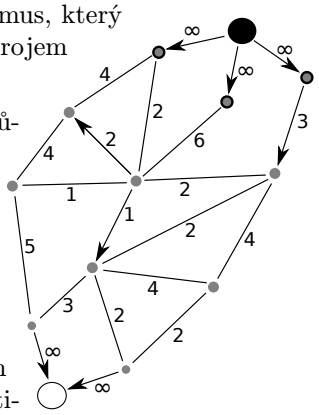
Naleziště jsou bezedná a mohou posílat neomezená množství ropy. Odběratelé také dokáží neomezená množství ropy z koncových bodů odebírat. Petrobaron čelí problému, jak protlačit danou distribuční sítí co nejvíce ropy za hodinu ze zdrojů k odběratelům.

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.

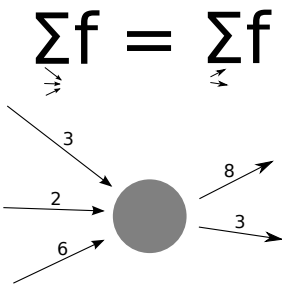
Podobně se zbavíme neorientovaných hran.

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.

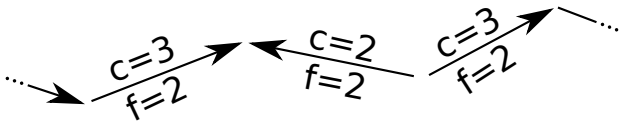
Představme si graf, na kterém počítáme tok, a dejme tomu, že už nějaký tok máme – třeba prázdný. Představme si, že jsme ropný magnát a každý rozdíl mezi kapacitou potrubí a jejím využitím (tokem) nás stojí miliony dolarů.

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. Odvoláme 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.[3]

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 kapitole o tocích[4] Medvědových skriptíček o algoritmech a datových strukturách, ukázka druhého přístupu k řešení hledání maximálního toku je tam také.

K zamyšlení

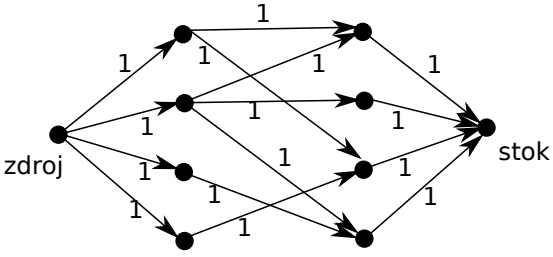
- Důležitou vlastností algoritmu je, že když dostane celočíselné kapacity, vrátí celočíselný tok. Bude se nám to hodit v aplikacích. Dokážete to?
- Rozdíl mezi Fordem-Fulkersonem, který hledá cesty obecným způsobem, a takovým, který to dělá prohledáváním do šířky, je ze složitostního hlediska docela 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?



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

[4] http://mj.ucw.cz/vyuka/ads/41-toky.pdf

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.

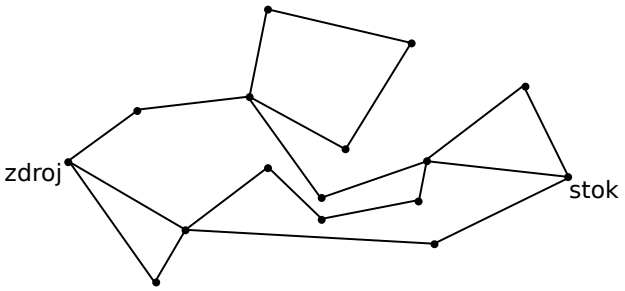
Hledání hranově a vrcholově disjunktních cest

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),[5] 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

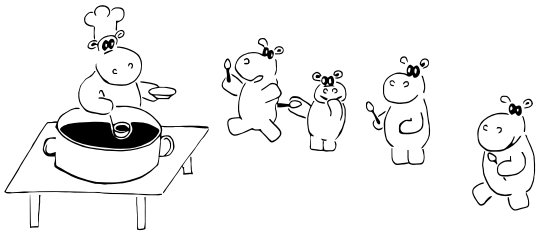
Lukáš Lánský

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

28-1-1 Jízda na biomotorce

Naše mapa města je představována neorientovaným grafem, v němž hledáme nejkratší cestu mezi počátečním a cílovým vrcholem. Délka cesty je určena počtem hran, které musíme projít. Toho jste se takřka všichni správně chytli a usoudili jste, že řešením úlohy bude modifikace prohledávání do šířky.[6] Tento algoritmus, zvaný také BFS (podle anglického *breadth-first search*) nám najde vzdálenosti všech vrcholů grafu od nějakého počátečního vrcholu.

Použijeme frontu, což je datová struktura, do které můžeme přidávat prvky a zase je odebírat. Důležité je, aby prvky byly vyjmuty v pořadí, v jakém byly do fronty přidány – stačí si představit frontu lidí na poště. V naší frontě si budeme uchovávat vrcholy grafu.



U každého vrcholu bude uložené číslo odpovídající jeho vzdálenosti od počátečního vrcholu. Ten bude mít nastavenou vzdálenost na nulu (nemusíme jít přes žádnou hranu). Ostatním vrcholům na začátku přiřadíme vzdálenost *nekonečno*. Počáteční vrchol vložíme do fronty.

Samotný algoritmus probíhá tak dlouho, dokud je fronta neprázdná: vyjmeme z fronty vrchol U a podíváme se na jeho sousední vrcholy (ty, které jsou s ním spojeny hranou). Pokud nějaký sousední vrchol má nekonečnou vzdálenost, pak ji nastavíme na vzdálenost U zvětšenou o jedna. Navíc vložíme takový vrchol do fronty.

Všechny vrcholy, jež jsou z toho počátečního dosažitelné (je mezi nimi cesta složená z hran), mají na konci správnou vzdálenost. Jak je to možné? Všimněte si, že vrcholy jsou ve frontě seřazeny v pořadí dle vzdálenosti – nejprve uložíme počáteční vrchol se vzdáleností nula, následují jeho sousední vrcholy se vzdáleností rovnou jedné, následně sousední vrcholy těchto vrcholů atd. Pokud tedy do nějakého vrcholu vede nejkratší cesta složená z H hran, bude na této cestě nejprve počáteční vrchol, pak nějaký jeho soused, dále jeho soused… a vzdálenost každého vrcholu se bude po jedné zvětšovat, až nakonec ve vzdálenosti H bude vrchol, do kterého jsme hledali cestu.

Nedostupným vrcholům zůstane nekonečná vzdálenost, nicméně v naší úloze jsme předpokládali, že celý graf je souvislý (mezi každými dvěma vrcholy vede nějaká cesta).

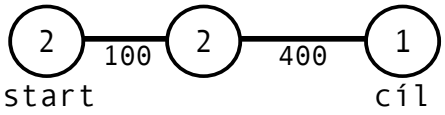
Pokud jako N označíme počet vrcholů a M počet hran (zapamatujte si toto označení, používá se poměrně často), celý algoritmus má časovou složitost $\mathcal{O}(N + M)$. Hlavní cyklus se opakuje tolikrát, kolik máme v grafu vrcholů – a každý z nich vložíme do fronty jen jednou. V každé iteraci cyklu prozkoumáme hrany vedoucí z určitého vrcholu, v celém průběhu algoritmu se ale na každou hranu takto podíváme jen dvakrát (z jednoho a druhého vrcholu, které spojuje).

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

Že by tedy stačilo spustit na startovní křižovatce BFS a vypsat vzdálenost cílového vrcholu? Nikoliv, věc nám totiž komplikují pomeranče. Při cestě mezi sousedními vrcholy (nazvěme je V a W) je třeba zkontrolovat, zda jich máme dostatek na cestu. To je však stále snadné, prostě srovnáme počet pomerančů, jež máme ve V, a délku hrany (vydělenou stem, abychom dostali číslo odpovídající spotřebě).

Pokud je počet pomerančů větší nebo roven, můžeme se přesunout do W. Zde se stav nádrže může změnit – vypočteme jej tak, že od počtu pomerančů ve V odečteme pomeranče spotřebované na cestě a naopak přičteme ty, jež obdržíme na křižovatce W. Jen nesmíme zapomenout na limit deseti pomerančů v nádrži. Pokud je počet pomerančů ve V menší, pak W není, alespoň prozatím, dostupné.

Někteří řešitelé však zapomněli na ještě jeden důležitý a na první pohled nepříliš viditelný fakt. Protože některé hrany nelze projet z důvodu nedostatku pomerančů, může se stát, že do některých vrcholů-křižovatek budeme chtít zajet víckrát než jednou. Jeden obrázek nám řekne více než odstavec slov:



Do prostředního vrcholu se dostaneme s počtem tří pomerančů, což nám pro pokračování do cíle nestačí. Je ale možné se otočit a vrátit se do startovního vrcholu, kde opět dostaneme pomeranče, tudíž máme čtyři. Pak cestujeme opět do prostředního vrcholu, kde přistaneme s pěti pomeranči – a to už je dostatek pro překonání hrany dlouhé 400 metrů. Do cíle tedy dojedeme, kvůli oklice budeme potřebovat čtyři hrany.

Při ignorování této možnosti vám často program nefungoval na čtvrtém testovacím vstupu generujícím strom, tedy graf, kde se nenacházejí cykly. V takovém případě vedla mezi startem a cílem jen jedna cesta a pro nalezení řešení bylo často nutné se popsaným způsobem „vracet“.

Jak tedy vypadá kompletně správné řešení? Abychom odlišili různé možnosti, které ve vrcholu máme v závislosti na stavu nádrže, pohybujeme se mezi *stavy*. To jsou v tomto případě dvojice (V, P), kde V je vrchol, v němž se nacházíme, a P počet pomerančů, které v něm máme. Všechny možné dvojice dohromady tvoří *stavový prostor*. Všimněte si, že počet dvojic je konečný (vrcholů máme omezeně mnoho a počet pomerančů je shora omezen). Prostor lze reprezentovat jako graf – jednotlivé stavy jsou vrcholy a hrana vede ze stavu (V, P_1) do (W, P_2) právě tehdy, když z křižovatky V s P_1 pomeranči v nádrži můžete dojet na křižovatku W a mít zde P_2 pomerančů.

Takový graf už nebude neorientovaný, ale prohledávání do šířky můžeme spustit i zde: začneme ve stavu odpovídajícímu naší výchozí situaci (startovní vrchol a tolik pomerančů, kolik jsme zde sebrali) a procházíme do té doby, než dojdeme do stavu, jehož vrchol je cílový (počet pomerančů nás zde nezajímá, chceme se dostat co nejrychleji do cíle bez závislosti na tom, kolik nám jich zbyde). Správnou odpovědí je pak vzdálenost do tohoto cílového stavu.

Jak se nám změní časová složitost? Vrcholů tohoto grafu je desetkrát více než předtím, počet hran se může také takto zvětšit. Desítka je ale stále rozumná konstanta, která se „vejde do óčka“, a tak složitost paměťová i časová je stále $\mathcal{O}(10N + 10M) = \mathcal{O}(N + M)$, kde N je počet vrcholů a M počet hran.

Zbývá jen doplnit, že při implementaci algoritmu není nutné si explicitně takový graf stavět. Stačí si udržovat původní síť představující mapu města a ke každému vrcholu připojit pole indexované od 0 do 10, což odpovídá počtu pomerančů – prvky pole jsou vzdálenosti příslušných stavů.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-1-1.c`

Kuba Maroušek

28-1-2 Zapalování kostek

Se soupeřem se střídáme v zapalování kostek v pyramidě a chtěli bychom být tím, kdo zapálí poslední kostku. Jak něco takového udělat?

Zadání sice mělo lehčí variantu, ale pojďme rovnou vyřešit úlohu bez dalších omezení, strategie pro konkrétní K nám z toho vypadnou samy.

Kdyby byly pyramidy dvě, obě stejné a každá pro jednoho hráče, nemá začínající hráč šanci. Jeho soupeř se totiž může „opičit“, tedy zahrát na své pyramidě vždy to, co hrál první hráč na té své. Tím pádem dokud by měl první hráč co hrát, měl by co hrát i druhý hráč, až by druhý hráč nakonec vyhrál.

To ale znamená, že kdyby se nám prvním tahem podařilo rozdělit pyramidu na dvě stejné menší, můžeme se my opičit po soupeři a tím vyhrát. A přesně to zvládneme. Pro lichá K je postup přímočarý – zapálíme prostřední kostku v prvním patře. Pro sudá nám stále stačí rychlé zamyšlení a zjistíme, že chceme pálit prostřední kostku ve druhém patře.

Tím nám vzniknou dvě stejné, izolované pyramidy (pro sudá K se jejich spodní řady dotýkají, ale to nevadí, protože sousední kostky se nemohou navzájem zapálit). Bez ohledu na to, jaké kostky zapálíme v jedné z nich, druhá zůstane nedotčena.

Po rozdělení tedy kopírujeme soupeřovy tahy. Soupeř sice může hrát v kterékoliv z menších pyramid, ale to my také. Vybereme si tedy vždy tu opačnou, takže po našem tahu budou opět obě pyramidy vypadat úplně stejně.

Soupeř musí dříve nebo později zapálit poslední nehořící kus jedné pyramidy. V té chvíli my zapálíme stejný kus druhé pyramidy, čímž vyhrajeme. A jelikož kostek je konečně mnoho a v každém tahu se zapálí alespoň jedna kostka, dojde i k naší výhře v konečném čase.

Karolína „Karryanna“ Burešová

28-1-3 Bourání komínu I

28-1-4 Bourání komínu II

Jak seřadit bomby, aby celková energie potřebná na zbourání komínu byla co nejmenší?

Vyřešme nejdříve lehčí variantu úlohy, kdy bomby dohromady přesně vystačí na zbourání komínu. Představme si, že už máme nějaké (libovolné) pořadí bomb zvolené. Zaměřme se nyní na dvě bomby, které použijeme jako poslední (označme si je A a B). Ty váží w_A a w_B a zničí d_A, resp. d_B metrů komínu. Před jejich použitím je komín vysoký $d_A + d_B$.

Pokud bomby použijeme v původním pořadí, tedy nejdřív A a potom B, spotřebujeme $w_A \cdot (d_A + d_B) + w_B \cdot d_B$ jednotek energie. Kolik energie spotřebujeme, pokud je prohodíme? Přece $w_B \cdot (d_A + d_B) + w_A \cdot d_A$.

Prohození se tedy vyplatí, pokud platí následující nerovnost:

$$w_A \cdot (d_A + d_B) + w_B \cdot d_B \leq w_B \cdot (d_A + d_B) + w_A \cdot d_A$$

Závorky si roznásobíme, odečteme stejné členy z obou stran a zbude nám $w_A \cdot d_B \leq w_B \cdot d_A$ (1).

To si upravíme na $w_A/d_A \leq w_B/d_B$. Číslu w_A/d_A říkejme *nákladnost* bomby A. Už víme, že z posledních dvou bomb se vyplatí použít nejdříve tu méně nákladnou.

Tato úvaha ovšem neplatí jen pro poslední dvě bomby. Podíváme-li se na libovolné dvě po sobě následující bomby A a B, dostaneme nerovnost podobnou té předchozí. Jen pokud po jejich použití zbude místo holé země komín výšky D, musíme obě vynést do výšky o D vyšší než v minulém případě. Tedy na obou stranách nerovnosti přibude člen $(w_A + w_B) \cdot D$, který ale můžeme hned odečíst. Opět tedy platí, že pokud je w_A/d_A větší než w_B/d_B, vyplatí se A a B prohodit.

Takovéto prohazování bychom mohli opakovat, dokud je někde v naší posloupnosti nákladnější bomba těsně před méně nákladnou. To ale není nic jiného než bublinkové třídění![7] Z tohoto příměru jsou hned jasné dvě věci: prohazování se časem zastaví a výsledná posloupnost bomb bude setříděná vzestupně podle nákladnost. Ta je tedy také správným řešením, protože kdykoli posloupnost není setříděná, existuje dvojice, kterou můžeme prohodit a řešení zlepšit.

My samozřejmě nemusíme třídit bublinkově, ale můžeme použít nějaký lepší algoritmus (např. MergeSort). Tak získáme řešení s časovou složitostí $\mathcal{O}(N \log N)$, kde N je počet bomb.

Drobná poznámka na okraj: nerovnici (1) jsme mohli zrovna tak upravit na $d_B/w_B \leq d_A/w_A$. Poměru d_A/w_A, vyjádřenému v „metrech na kilo“, říkejme třeba *efektivita* bomby A. V tomto případě naopak platí, že bomby s největší efektivitou chceme použít jako první, tedy musíme třídit sestupně.

Těžší varianta

Co ale uděláme, když jsme si pořídili bomb víc a všechny naráz by byly schopné zbourat komín větší než náš?

Nejprv si rozmyslete, že bez ohledu na to, které bomby si vybereme, opět je chceme seřadit nejvýhodnějším způsobem, tedy vzestupně podle nákladnosti. Tak si bomby setřiďme už na začátku, a pak teprve vybírejme, které použít.

[7] `http://cs.wikipedia.org/wiki/Bubblesort`

Ukážeme si nejprve „hloupé“ řešení, které zkouší všechny možnosti, a pak ho zkusíme vylepšit. Bomby procházíme postupně od nejméně nákladné. U každé se můžeme rozhodnout, jestli ji použít, nebo ne. Pro obě z těchto možností pokračujeme rekurzivně v procházení zbylých bomb. Průběžně si hlídáme, aby celková síla vybraných bomb nepřekročila výšku komínu, a počítáme spotřebovanou energii. Na konci jen vybereme nejlevnější variantu. Takové řešení určitě funguje, ale možností, jak vybrat bomby, je celkem 2^N. To je moc na to, abychom je stihli vyzkoušet všechny.

Když se na průběh vybírání podíváme podrobněji, zjistíme, že se často opakovaně dostaneme do podobné situace. Například se může více způsoby stát, že po m krocích výběru skončíme s bombami, které sníží komín na výšku h. Pro každou z těchto variant pak zkoušíme všechna možná pokračování, přestože zřejmě stačí uvažovat jen tu nejlevnější a ostatní zahodit.

Neboli pro každé m a h bychom chtěli spočítat, jak nejlevněji snížit komín na výšku h za použití některých z m nejefektivnějších bomb. K tomu se nám bude hodit pozorování, že výška komínu v průběhu bourání bude vždycky mezi 0 a K.

Tyto hodnoty si tedy můžeme ukládat do dvourozměrného pole P velikosti $(N+1) \times (K+1)$. Budeme je počítat postupně pro $m = 0, \ldots, N$. Na začátku víme, že $P[0][K] = 0$ (když nic neuděláme, „zbouráme“ komín na původní výšku K s nulovou cenou), a na ostatních políčkách $P[0]$ bude „nekonečno“, tedy nějaká nekřesťanská cena znamenající, že zbourat komín na takovouto výšku zatím neumíme.

Když zpracováváme m-tou bombu, už známe všechny hodnoty v $P[m-1]$ a rádi bychom vyplnili $P[m]$. To uděláme tak, že budeme procházet pole $P[m-1]$ pro h od nuly do K, a když na výšce h narazíme na cenu bourání z minula (což je minimální cena, jakou jsme schopni vydat za zbourání komínu do výšky h pouze s využitím bomb před m), tak do pole $P[m]$ zapíšeme nové ceny dvou bourání.

Jedna cena bude za bourání, u kterého jsme bombu m nepoužili (tedy $P[m][h] = P[m-1][h]$, cena zůstává stejná), druhá, když ji zkusíme použít. Do pole $P[m][h - d_m]$ zapíšeme $P[m-1][h] + h \cdot w_m$, tedy cenu bourání s předchozími bombami plus cenu za vynesení bomby m na vršek komínu.

Psali jsme ale, že bombu m zkusíme použít. Ono se to nemusí povést, třeba když je moc silná a $h - d_m < 0$. Pak to samozřejmě zapisovat nebudeme. Nebo jsme se s předchozími bombami dostali na stejnou výšku levněji. Pak bombu m zahodíme. (Ale opatrně, bude se ještě hodit.)

A to je celé. Když přežijete zahazování nepoužitých bomb, můžete si na konci na políčku $P[N][0]$ přečíst cenu nejlevnějšího postupu na zbourání, nebo tu nekřesťanskou cenu za ruční rozebrání, protože bombami to nejde.

Ještě si všimněte, že nemusíme udržovat v paměti celé P. V každém kroku pracujeme vždy jen s posledními dvěma řádky (z jednoho čteme a do druhého zapisujeme), stačí si tedy pamatovat ty.

To můžeme udělat například tak, že pole P bude velké jen $2 \times (K+1)$ a místo $P[m]$ budeme používat $P[m \bmod 2]$, kde $m \bmod 2$ je parita m (zbytek po dělení dvěma). Vzhledem k tomu, že dvě po sobě jdoucí m mají opačnou paritu, ve chvíli, kdy zapisujeme do jednoho řádku P, bude ten druhý obsahovat právě čísla zapsaná v minulém kroku.

Paměťová složitost je $\mathcal{O}(N+K)$, časová $\mathcal{O}(N \cdot \log N + NK)$.

Známe cenu, ale kterých bomb?

Takto jsme tedy získali nejlevnější cenu. Kdybychom chtěli vědět i to, jaké bomby jsme použili, pak můžeme buďto použít pole čísel 0 až K pro každou bombu, a ne jen dvě, a ke každé ceně si zapisovat i předchozí použitou bombu (podobně jako u hledání nejkratší cesty). Tím se ale zhorší paměťová složitost na $\mathcal{O}(NK)$.

Nebo můžeme mít pole jen dvě, ale použité bomby si udržovat ve spojových seznamech vedle ceny. V nejhorším případě ovšem bude potřeba také $\mathcal{O}(NK)$ paměti.

A co s těmi nepoužitými bombami? Odneste si je v batohu! Máte batoh s nosností K, bomby mají váhy w_i a na černém trhu za ně dostanete v_i. Odneste si takové bomby, aby jejich cena byla co největší a abyste nepřekročili nosnost (a mohli do letadla ;)). To je známý kombinatorický „problém batohu“. Pokud je K tak malé, že si můžete dovolit mít v paměti takto dlouhé pole, pak se dá problém řešit dynamickým programováním. Je to velmi podobné jako naše úloha. A to už přece umíte!

Program (C):
`http://ksp.mff.cuni.cz/viz/28-1-4.c`

Dominik Macháček

28-1-5 Likvidace plísně

Úloha byla vskutku jednoduchá a neskrýval se za ní žádný složitý trik. Pokud jste si uvědomili několik jednoduchých pozorování, napsání programu již bylo hračkou.

Prvním krokem je uvědomit si, že se nám vždy vyplatí přesunovat jenom na úplně nové (prázdné) hromádky. Pokud bychom chtěli přesunout nějaké vrstvy na existující hromádku, tak přesunem vrstev na novou můžeme jen zkrátit celkovou dobu odstraňování („nezaplácneme“ si druhou hromádku dalšími vrstvami).

Druhým pozorováním je, že všechno odstraňování plísně má smysl provádět až po všech přesunech. Použijeme klasický postup důkazu a budeme uvažovat, že by existovalo nějaké optimální řešení, které by po nějakých krocích odstraňování plísně provádělo ještě přesuny. Ale u takového řešení můžeme modifikovat přesuny tak, aby za každý krok odstraňování braly o jednu vrstvu více (o tu, kterou by odstraňování odmazalo), a přemístíme všechna odstraňování na konec.

Díky tomu odmažeme minimálně stejný počet vrstev, takže jsme tím převedli libovolné jiné řešení na alespoň stejně dobré splňující naší podmínku. Nějaké z optimálních řešení má tedy všechna mazání až na konci.

Stačí nám jen postupně pro všechny délky mazací části (označme si ji k) zkusit spočítat, kolik kroků potřebujeme na rozdělení hromádek tak, aby byly všechny maximálně k vrstev vysoké. To spočítáme jednoduchým cyklem (počet přesunů na rozdělení hromádky vysoké h je $\lceil h/k - 1 \rceil$). Ze všech výšek k vybereme tu nejlepší.

Pokud N udává počet hromádek a V maximální výšku plísně, tak časová složitost algoritmu je $\mathcal{O}(V \cdot N)$.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-1-5.c`

Jirka Setnička

28-1-6 Úloha z ovladače

Úlohu si můžeme představit tak, že posouváme okénkem délky K po nekonečně dlouhé posloupnosti čísel a hlásíme, jaké je zrovna minimum uvnitř okénka.

Kdybychom minimum pokaždé počítali znovu, trvalo by to $\mathcal{O}(K)$. Jak to udělat lépe? Rozmysleme si, jak se posunutím okénka může minimum změnit. Označme původní prvky $x_0, \ldots, x_{K-1}$ (x_m byl nejmenší) a nové $x_1, \ldots, x_K$.

- Pokud $x_K \leq x_m$, je novým minimem x_K.
- Pokud $x_K > x_m$ a $m > 0$ (staré minimum dosud okénko neopustilo), zůstává minimem x_m.
- Pokud $x_K > x_m$ a $m = 0$ (staré minimum právě vypadlo), nevíme, kde minimum leží.

V posledním případě tedy musíme všechny prvky okénka znovu projít, což zase trvá $\mathcal{O}(K)$. A co hůř, může se to stát pokaždé: představte si případ, kdy zadané prvky tvoří rostoucí posloupnost.

Mohli bychom si pomoci haldou (nebo vyhledávacím stromem), kde bychom si pamatovali celý obsah okénka. Pak by jedno posunutí trvalo $\mathcal{O}(\log K)$. My to ale umíme lépe, poslyšte, jak.

Přemýšlejme nad tím, jak přesně vypadá situace, kdy jsme o minimum právě přišli. Čím ho nahradíme? Nejmenším z prvků, které leží napravo od něj. Prvek s touto vlastností si opět můžeme průběžně udržovat, jenže co když pak vypadne i ten? Tak si pojďme pamatovat náhradu i za něj, atd.

Přehledněji řečeno, budeme si udržovat následující posloupnost:

m_0 := poloha aktuálního minima,

m_1 := poloha nejmenšího z prvků ležících vpravo od m_0,

$\vdots$

m_r := poloha nejmenšího z prvků vpravo od m_{r-1}.

Při každém posunutí okénka nyní provedeme toto:

- Pokud m_0 vypadlo z okénka, smažeme ho z posloupnosti.
- Dokud je nový prvek menší než x_{m_r}, smažeme m_r.
- Přidáme na konec posloupnosti pozici nového prvku.
- Nahlásíme prvek x_{m_0} jako minimum nového okénka.

Snadno si rozmyslíme, že tím vznikne posloupnost požadovaných vlastností pro nové okénko.

Opravdu jsme si tím pomohli? V nejhorším případě přeci můžeme mazat až K hodnot m_i, takže časová složitost je stále $\mathcal{O}(K)$! V této temnotě se ale mihotá světélko naděje: ke smazání všech prvků nemůže docházet často, protože prvky se doplňují jen po jednom.

Dokážeme, že posunutí okénka má *konstantní amortizovanou složitost.* Tím se myslí, že provedeme-li n posunutí, trvají dohromady $\mathcal{O}(n)$. Všechny operace totiž buďto přidávají prvky do posloupnosti, nebo je odebírají. Těch přidávacích je nejvýše n, neboť pokaždé přidáme právě jeden prvek. A odebíracích je nejvýše tolik, kolik je přidávacích, protože žádný prvek nemůžeme smazat víckrát.

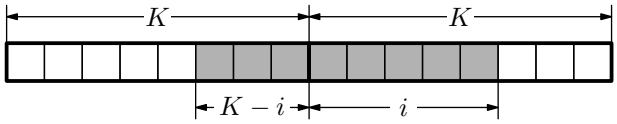
Program (C):
`http://ksp.mff.cuni.cz/viz/28-1-6-amort.c`

Řešení rozkladem na bloky

Ukážeme si ještě jedno amortizovaně konstantní řešení, založené na úplně jiné myšlence.

Vstup budeme dělit na bloky o K prvcích. Pro každý blok $x_1, \ldots, x_K$ si budeme průběžně počítat *prefixová minima* $\min(x_1, \ldots, x_i)$. Jakmile blok skončí, dopočítáme i *suffixová minima* $\min(x_j, \ldots, x_K)$.

Jak je vidět na následujícím obrázku, okénko délky K můžeme vždy rozdělit na prefix aktuálního bloku a suffix toho předchozího:



Minimum aktuálního okénka tedy můžeme spočítat v konstantním čase z předpočítaných hodnot.

Časová složitost vyjde opět amortizovaně konstantní, protože předvýpočet nás stojí $\mathcal{O}(K)$ na konci bloku o velikosti K. Stačí tedy, aby každý prvek přispěl časem $\mathcal{O}(1)$ na budoucí zpracování hotového bloku.

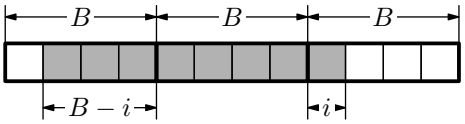
Čistokrevné konstantní řešení

Ačkoliv to může znít neuvěřitelně, existuje i řešení, kterému stačí konstantní čas na posunutí okénka i v nejhorším případě. Zkusíme „deamortizovat“ trik s rozkladem na bloky (mimochodem, první řešení s posloupností náhradních prvků deamortizovat neumíme).

Místo toho, abychom postupně střádali prvky a pak najednou zpracovali celý blok, zkusíme ho zpracovávat postupně: s každým novým prvkem kousek (nejdřív suffix délky 1, pak délky 2, atd.). To ovšem nemůže vyjít, protože hned u prvního prvku následujícího bloku potřebujeme nejdelší suffix toho předchozího.

Zachráníme to elegantním trikem: zvolíme jinou velikost bloku než K, konkrétně $B = K/2$ (pro jednoduchost předpokládejme, že K je sudé; lichá K dořešíme později).

Jak ukazuje následující obrázek, okénko velikosti K zasahuje do tří bloků: z aktuálního používá prefix, minulý pokrývá celý a z předminulého používá suffix:



Tím jsme si sice výpočet trochu zkomplikovali, ale teď mezi ukončením bloku a okamžikem, kdy potřebujeme jeho suffixová minima, leží alespoň jeden další blok, během kterého můžeme minima počítat.

Přesněji řečeno: pokud jsme právě dostali i-tý prvek aktuálního bloku B_t, provedeme následující:

1. Spočítáme i-té prefixové minimum bloku B_t.
2. Spočítáme i-té suffixové minimum bloku B_{t-1}.
3. Vrátíme jako výsledek minimum z těchto hodnot:
4. i-té prefixové minimum bloku B_t,
5. minimum celého bloku B_{t-1} (to je také prefixové minimum, takže už je spočítané),
6. i-té suffixové minimum bloku B_{t-2}.

Toto vše spočítáme v čase $\mathcal{O}(1)$, na udržování prefixových a suffixových minim spotřebujeme $\mathcal{O}(B) = \mathcal{O}(K)$ buněk paměti.

Zbývá dořešit okénka liché délky: pro ta postačí nastavit $B = \lceil K/2 \rceil$. Podle obrázku si rozmyslíme (dobře se to představuje třeba pro $K = 7$, kdy bude $B = 4$), že vše vyjde správně. Pokud blok právě začal ($i = 0$), potřebujeme celý minulý blok a suffix předminulého. Pokud se blok právě chystá skončit ($i = K - 1$), stále minulý blok pokrýváme celý, takže se nestane, že bychom potřebovali suffixové minimum, které jsme dosud nespočítali.

(Mimochodem, i kdyby nám to pro lichá K takhle hezky nevyšlo, mohli bychom si pomoci oklikou: okénko bychom zmenšili na $K - 1$ prvků a navíc bychom si pamatovali seznam posledních K prvků. Pak bychom jako výsledek vraceli minimum z minima okénka a nejstaršího prvku seznamu.)

Gratulujeme Jirkovi Sejkorovi, který jako jediný vymyslel řešení sice složitější než naše vzorové, ale také pracující v konstantním čase.

Program (C):
`http://ksp.mff.cuni.cz/viz/28-1-6-worst.c`

Martin „Medvěd“ Mareš & Václav Končický

28-1-7 Vyříznutý kus mříže

Máme mnohoúhelník s N vrcholy v mřížových bodech a chceme spočítat, kolik mřížových bodů se nachází na jeho obvodu. Vrcholy máme na vstupu v pořadí, v jakém jsou na obvodu. Naše řešení postavíme z krabičky, která umí počítat mřížové body na jedné straně. Pustíme ji na každém páru po sobě následujících bodů na obvodu. Kdybychom ale sečetli počet mřížových bodů na všech stranách, nedostali bychom správný výsledek: každý vrchol je mřížový bod, který leží na dvou stranách, proto musíme ještě od součtu odečíst N.

Jak spočítáme počet mřížových bodů na straně? Nechť naše strana vede z bodu (a, b) do bodu (c, d). Označme si $c - a$ jako Δx a $d - b$ jako Δy, a odteď jenom počítejme počet mřížových bodů na úsečce $(0, 0) \ldots (\Delta x, \Delta y)$. Tím jsme jenom posunuli úsečku do počátku – to počet mřížových bodů nijak nezmění.

Vodorovné a svislé úsečky, tj. ty, kde Δx nebo Δy je 0, můžeme ošetřit zvlášť (počet mřížových bodů na nich je absolutní hodnota toho Δx nebo Δy, které není nulové).

Jeden ze způsobů, jak spočítat počet mřížových bodů na úsečce, je zkoušet všechny hodnoty x od 0 do Δx. Ke každému takovému x na naší úsečce leží právě jeden bod (x, y) a pro něj platí $y = x \cdot \Delta y / \Delta x$. Pokud nám pro nějaké x vyjde y celočíselné, započítáme si najitý mřížový bod. Protože se chceme vyhnout nepřesnému dělení čísel s plovoucí čárkou, test na celočíselnost můžeme napsat třeba jako `(deltaY * x) % deltaX == 0`.

Takové řešení funguje, ale má jednu nevýhodu: každou stranu projíždíme celou od 0 do Δx, a proto je jeho složitost přímo úměrná obvodu mnohoúhelníka (respektive součtu Δx přes všechny strany). Počítat mřížové body na úsečce $(0, 0) \ldots (1, 1)$ bude mnohem rychlejší, než kdyby končila v $(100, 100)$.

Podívejme se na ten mřížový bod, který má z vnitřních mřížových bodů nejmenší obě souřadnice. Ať je to bod (p, q).

Použijeme dvě zajímavé vlastnosti bodu (p, q). Zaprvé pokud si vybereme jakýkoliv jiný mřížový bod (s, t), tak musí být souřadnice (s, t) násobek (p, q).

Zkusme si představit, že (s, t) není násobek (p, q). Když je (s, t) mřížový bod na úsečce, tak určitě i $(s - p, t - q)$ je mřížový bod na úsečce. Odečítejme od (s, t) násobky (p, q) tak dlouho, dokud to už nejde bez toho, abychom šli do záporných souřadnic. Dostaneme z toho mřížový bod $(s', t') = (s, t) - k \cdot (p, q)$. Protože další odečtení (p, q) by šlo do záporných souřadnic, je jedna ze souřadnic (s', t') menší než souřadnice (p, q), a to je zase spor s volbou (p, q). Proto je určitě (s, t) násobek (p, q).

Tohle je super: když najdeme správně bod (p, q), jsou všechny mřížové body na úsečce jeho násobky, takže nám stačí podělit $\Delta x/p$, přičíst jedničku za mřížový bod v počátku a máme počet mřížových bodů na celé úsečce.

Jak ale najdeme (p, q)? K tomu použijeme druhou vlastnost: p a q musí být nesoudělná čísla. Měla-li by společného dělitele r, pak by i $(p/r, q/r)$ byl mřížový bod, ale to by byl spor: my jsme si přece vybrali (p, q) tak, aby první souřadnice byla co nejmenší a $p/r < p$.

Čísla p a q jsou tedy nesoudělná a $p/q = \Delta x/\Delta y$, protože leží na úsečce. K nalezení p a q nám vlastně stačí zkrátit zlomek $\Delta x/\Delta y$ do základního tvaru!

Krácení zlomku se dělá tak, že najdeme největšího společného dělitele Δx a Δy. Po vydělení Δx a Δy jejich největším společným dělitelem dostaneme p a q. Na hledání největšího společného dělitele použijeme Euklidův algoritmus, který doběhne v čase $\mathcal{O}(\log \min\{\Delta x, \Delta y\})$. Jeho detaily si můžete dohledat v naší kuchařce o teorii čísel.[8] To je podstatné zlepšení proti posouvání se po všech potenciálních mřížových bodech, které stojí $\Theta(\Delta x)$.

Trochu si ještě zjednodušíme počítání. Určení souřadnic (p, q) vlastně vůbec není potřeba: poté, co spočítáme p jako $\Delta x / \operatorname{nsd}(\Delta x, \Delta y)$, ho používáme na určení počtu mřížových bodů: $1 + (\Delta x/p)$. Stačí to trochu upravit, a vidíme, že výsledek je rovný $1 + \operatorname{nsd}(\Delta x, \Delta y)$ a p nikde nepotřebujeme. Nakonec se vyhneme nehezkému odečítání N v posledním kroku: místo $\operatorname{nsd}(\Delta x, \Delta y) + 1$ budeme sčítat jenom $\operatorname{nsd}(\Delta x, \Delta y)$, což N od součtu automagicky odečte.

Celý algoritmus doběhne v čase $\Theta(N \cdot \log \min\{\Delta x, \Delta y\})$. Navíc protože si nepotřebujeme pamatovat pozice všech bodů, můžeme pro všechny úsečky na vstupu spočítat počet mřížových bodů a úsečku rovnou zahodit. Stačí nám konstantní paměť.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-1-7.py`

Michal „Prvák“ Pokorný

28-1-8 Programování podle Darwina

Úloha 1

V této úloze bylo cílem vyzkoušet si genetický algoritmus, jaké má vlastnosti a jak jednotlivé parametry ovlivňují jeho chování. Z mnohých věcí jste mohli vypozorovat například následující:

Mutace: Pokud zvýšíme pravděpodobnost mutace, tak se algoritmus chová více náhodně. Ze začátku má sice rychlý nástup, ale pak pro náš problém platí, že čím máme lepšího jedince, tím je menší pravděpodobnost, že jej vylepšíme a větší pravděpodobnost, že jej zhoršíme. Pak pokud nastavíme pravděpodobnost na změnu jednoho bitu příliš velkou, tak mutace dělá větší změny a tím se i chová méně konzistentně.

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

Selekce: Při řešení jedničkového problému by použití ruletové vs. turnajové selekce nemělo mít velký vliv na výsledek. Ruletová má výhodu v tom, že více preferuje lepší jedince, zatímco turnajová je celkově jednodušeji implementovatelná a paměťově i časově efektivnější.

Ti bystřejší z vás si navíc všimli, že pokud předem víme, kolik jedinců pomocí ruletové selekce chceme vybrat, tak ji celou můžeme implementovat efektivněji. To jest v čase $O(P + N \log N)$, kde P je velikost populace a N je počet vybíraných jedinců.

Křížení a velikost populace: Tyto dva parametry spolu implementačně příliš nesouvisí, ale fakticky spolu souvisí hodně. Velikost populace určuje variabilitu dat, která na začátku v jedincích máme. Čím více jedinců, tím více různých dat. Křížení pak během algoritmu tato data dává různě dohromady a díky selekci tvoří lepší jedince. Při malé populaci se nám stane, že za chvíli nemáme co nového mezi jedinci křížit a musíme čekat na štěstí v mutaci, zatímco ve velkých populacích se informace zbytečně opakují a výpočet nám to akorát zpomalí.

Velikost jedince a velikost populace: Tyto dva parametry spolu také souvisí. Velikost populace bychom měli volit v závislosti na velikosti jedince. Tj. aby byla dost velká šance, že na začátku budou jedinci dohromady obsahovat správná data pro všechny bity a že je jednou nešťastnou mutací či křížením zase neztratíme.

Fitness funkce: Ta v prvním případě počítala jen počet jedniček. Zajímavější byl druhý případ. Pokud se snažíme vyvinout jedince, kde se jedničky a nuly střídají, dostaneme fitness funkci se dvěma možnými maximy, které jsou k sobě inverzní. Pak v průběhu algoritmu nám „jdou proti sobě“ a jedinci obou frakcí spolu „soupeří“. Jedni se snaží mít na sudých pozicích nuly, na lichých jedničky a druzí naopak.

Na tom jste si měli hlavně vyzkoušet, jak genetický algoritmus (ne)funguje dobře, pokud máme fitness funkci s více rozdílnými maximy, které jsou velmi odlišného tvaru. V tomto případě to lze vyřešit třeba fintou, že postavíme fitness funkci tak, aby preferovala jedince s jedničkami na lichých pozicích. Obecně ale do fitness funkce tolik nevidíme, abychom podobné finty mohli dělat.

Úloha 2

V této úloze již byla potřeba jistá modifikace řešení a kreativní náhled na problém.

Celý genetický algoritmus jednoduše přepíšeme tak, aby namísto s binárními jedinci fungoval s jedinci celých čísel 0 až 6. Křížení zůstane nezměněno a mutace místo překlopení generuje náhodnou novou hodnotu.

Fitness funkci naprogramujeme podle zadání. Ohodnotíme jedince podle toho, jak dobře rozděluje zadané věci mezi sedm loupežníků. Pokud používáme ruletovou selekci, tak navíc hodnotu překlopíme na $1/(x + 1)$, abychom mohli maximalizovat a ne minimalizovat.

Tak to by bylo. Pustíme algoritmus a ono to moc dobře nefunguje. Proč?

Zkusme se zamyslet, co fakticky znamenají křížení a mutace v rámci tohoto problému. Jednobodové křížení vezme dvě různá řešení a v náhodném bodě je zkříží. Proč by takové křížení mělo jakkoliv směřovat ke stejným součtům různých hromádek? Třeba díky fitness funkci, ale…

Stejně jako v případě střídajících se nul a jedniček, i tady máme více různých nejlepších řešení, které jdou „proti sobě“. Těch je alespoň 7! a faktoriál sedmi je 5 040, k tomu obsahuje ještě víc lokálních neoptimálních maxim, ze kterých se nedá jednoduše dostat. Křížení teda vypadá, že nám moc nepomůže.

A co mutace? Ta zas jen provede malou náhodnou změnu. Ta nám sice občas může přinést něco dobrého, ale samotná určitě nestačí, chce to něčím doplnit.

My ji doplníme chytrou mutací, to jest takovou, která využívá znalost řešeného problému. Ta si pro daného jedince spočítá velikosti jednotlivých hromádek, pak vybere náhodný předmět z nejtěžší hromádky a přendá jej na nejlehčí hromádku.

Taková mutace uměle cílí na část problému, která by měla pozitivně ovlivnit fitness funkci. To má výhodu v tom, že se na začátku algoritmu budeme efektivně blížit k relativně dobrému řešení. Bohužel nevýhodou je, že můžeme lehce uvíznout v nějakém suboptimálním řešení, ze kterého se přehozením jednoho předmětu nedostaneme. Jak ale uvidíme dále, nám tento operátor bude stačit.

V celém řešení použijeme pouze chytrou mutaci a normální mutaci. Křížení vůbec používat nebudeme. Populaci nepotřebujeme příliš velkou, bude stačit 50 jedinců, protože jedinci mezi sebou stejně neinteragují. Na druhou stranu jich bereme 50, abychom procházeli více náhodných změn najednou a ne jen jednu.

Celé to necháme běžet velké množství iterací, třeba 10 000, abychom určitě zkonvergovali a případně měli dost času se dostat z lokálních minim. To celé pustíme opakovaně v několika bězích, abychom několikrát začali s jinak nagenerovanými jedinci.

Na výsledcích nejtěžšího vstupu pak můžeme vidět, že za 10 000 iterací k optimálnímu řešení obvykle dojdeme. V nejlepších případech se tak stane řádově po stovkách iterací, jindy řádově po tisících iterací a jindy k optimu vůbec nedojdeme. Záleží na tom, jak se nám zrovna nagenerovala vstupní data a kam se výpočet nasměroval. Proto také nevsázíme jen na jeden výpočet a pouštíme algoritmus opakovaně s různě nagenerovanými počátky.

Toto řešení je implementované ve vzorovém kódu.

Program (C++):
`http://ksp.mff.cuni.cz/viz/28-1-8.cpp`

Karel Tesař