# Korespondenční Seminář z Programování

**33. ročník** **KSP** **Únor 2021**

## Komentáře k třetí sérii třicátého třetího ročníku KSP

### 33-3-4 Obsazování území

Pětice nejlepších řešitelů kompetitivní úlohy s obsazováním Prahy se s námi podělila o svá řešení. Za to jim velmi děkujeme a jejich řešení zde po drobném zpracování uveřejňujeme. Jak jsme již psali u orgovských řešení,[1] tak tato úloha je těžká, neznáme na ní nejlepší řešení a tak ke slovu přišly různé heuristické přístupy.

Detailní výsledky naleznete v tabulce výsledků.[2]

### Kristýna Petrlíková – Velká pražská kolonizace

Plné řešení včetně obrázků naleznete na webu.

**Generující řešení**

*Deterministický postup*

Prvním nástřelem bylo setřídit si domy podle ceny, a dokud nebylo všechno pokryto, vybrat nejlevnější dům, který se nepřekrýval s ostatními dosud zvolenými domy. Celková cena byla **1 121 672 za 524 domů**. Zkolonizovat takto Prahu trvalo asi 2 minuty.

*"Deterministický" postup*

Dále bylo dobré vzít v potaz počet dosud nepokrytých domů, které se v okolí konkrétního domu nacházely. Jednotlivé domy se pořád vybíraly primárně podle ceny, ale na překrytí s ostatními tentokrát už nezáleželo. Místo toho byl vybrán každý dům, v jehož okolí bylo více než `threshold` nepokrytých domů. Jakmile žádný takový dům neexistoval, `threshold` se snížil, a domy se začaly procházet odznovu.

Výsledky se různily v závislosti na zvoleném `thresholdu` a míře jeho zmenšování. Mohlo se od něj buď odčítat, nebo ho podělit (řekněme proměnnou `div`), což se ukázalo jako efektivnější způsob.

Bohužel výstup byl na tyto dva parametry hodně citlivý, a pokud se `threshold` posunul o jednotku nebo se `div` změnil o tisícinu, výsledná cena se mohla lišit v řádu tisíců.

Relativně dobré řešení dala konfigurace `threshold=60950` a `div=1.2`: celkem **526 domů za cenu 937 709**. Tomuto řešení trvalo doběhnout kolem 14 minut (samozřejmě čím blíže je `div` jedničce, tím déle trvá řešení spočítat.)

Dále se mohly shora omezit ceny vybíraných domů. Pokrytí, které neobsahovalo domy dražší než 6 533, nebylo možné sestavit. Horní hranicí `lim` bylo tedy zvoleno ze začátku 6 535, v pozdějších řešeních až 7 000 (ale občas nebyla použita vůbec). S takovým omezením vyšlo řešení kolem 10 minut s **528 domy za cenu 922 437**.

*Deterministický postup?*

Jelikož se ale zvolené parametry chovaly tak chaoticky, že se člověku ani nechtělo jejich zkoušení automatizovat, muselo se nakonec od této pseudorandomizace upustit. Místo toho se domy začaly vybírat čistě podle počtu nepokrytých sousedů v jejich okolí. Tedy pokaždé se vybral dům, který měl toto číslo co největší, pak se hodnoty v domech jeho sousedů přepočítaly, a to do té doby, než se takto zkolonizovalo celé město.

Takové řešení je ale samo o sobě dost špatné, neboť nijak nepracuje s cenou, kterou je ve výsledku potřeba minimalizovat. Bude se ale hodit ještě později.

Prozatím stačí srovnávací funkci (tedy operátor $<$, podle kterého se následně vybírá minimum) upravit takto:

$$\texttt{dům}_a \to \texttt{volné} \;>\; \texttt{dům}_b \to \texttt{volné} \quad \text{(Původní srovnávání)}$$

$$\frac{\texttt{dům}_a \to \texttt{volné}}{\texttt{dům}_a \to \texttt{cena}} \;>\; \frac{\texttt{dům}_b \to \texttt{volné}}{\texttt{dům}_a \to \texttt{cena}} \quad \text{(Nové srovnávání)}$$

Výsledkem pro tuto funkci je řešení s **557 domy za cenu 814 413**, které se spočítá během tří minut.

Tento postup je opět možné randomizovat, třeba přidáním `or random() mod <náhodný_parametr> == 0` do srovnávací funkce, případně různého umocňování či násobení čitatelů a jmenovatelů hodnot srovnávaných domů. Tím se výsledná cena sníží o pár desítek tisíc, ale pak se již moc nepohne.

**Zlepšující řešení**

Obrovské zlepšení přineslo pozorování, že ve výsledku jsou některé domy zakoupeny zbytečně, tedy i bez nich by celé území bylo pokryto. Kromě toho můžou být některé domy nahrazeny svými levnějšími sousedy (myšleno okolními domy ve čtverci $1\,000 \times 1\,000$), přičemž ale musí pořád dodržet 100% pokrytí.

Zlepšování je prováděno hladově: nejdříve jsou odstraněny nejdražší možné domy, teprve potom dochází k případnému nahrazení s co největším rozdílem cen.

Výsledek 814 413 je tedy zredukován na cenu **621 778 při 416 domech** a limitu ceny domu 7 000. Řešení doběhne za 14 minut.

Pokud spustíme generující řešení s původním srovnáváním (tedy pouze dle počtu neobsazených domů), vyjde nám **419 domů s cenou 1 757 160**. Tohle skóre dokáže zlepšovátko zredukovat na řešení s **348 domy za cenu 686 868**. To se sice nevyplatí tolik jako předchozí řešení, ale dochází zde k mnohem většímu zlepšení.

Posledním krokem bylo postupné zlepšování dosud nejlepšího řešení. Nejdříve se z něj náhodně vybere $K$ domů, následně se prázdné oblasti znovu vyplní s jednou ze srovnávacích funkcí, a nakonec je řešení prohnáno zlepšovátkem.

První srovnávací funkci vycházelo zlepšování mnohem lépe (fungovalo pro $K = 25 \pm 10$), druhá sice průměrně doplňovala levněji, ale zlepšovátko se pak nedokázalo příliš mnoha domů zbavit.

Občas pomohlo dosud nejlepší nalezené řešení smazat a vrátit se k nějakému dražšímu s více domy, což výsledek mohlo zlepšit až o pár tisíc.

[1] `http://ksp.mff.cuni.cz/viz/33-3-4/reseni`

[2] `https://ksp.mff.cuni.cz/h/ulohy/33/33-3-4/vysledky`

Finální řešení dospělo na **cenu 532 824 pro 329 domů**.

**Implementační detaily**

Programy jsou psané v C++.

Pro rychlé přepočty neobsazených domů v okolí se používá $17 \times 17$ polí o velikostech většinou $1\,000 \times 1\,000$ (pole na krajích jsou trochu menší). Na těch se následně počítají prefixové součty (pro množství neobsazených domů). Při obsazení domu je tedy třeba přepočítat pouze 4 pole v jeho okolí (nebo 9, pokud se člověk nechce zaobírat konkrétní polohou domu; samotný výpočet to ale příliš nezpomalí).

## Robert Jaworski

Procházíme mapu po řádcích. Když narazíme na zastavěné a nezpřístupněné políčko, vyhledáme nejvýhodnější políčko v okolí, které by ho zpřístupnilo. Funkce určující výhodnost políčka může mít různé verze. Nejvýhodnější bylo nakonec (co jsem zkoušel) pro dané pole sečíst ceny všech dosud nepřístupných polí, které koupě daného pole zpřístupní, minus cena daného pole krát nějaký koeficient (který postupně upravujeme, aby bylo řešení co nejlepší).

Jelikož ale počítáme výhodnost pro velké množství polí blízko sebe, sčítali bychom ceny stejných políček dokola. Stačí nám tedy sečíst ceny okolo prvního pole, a pro vedlejší políčko použijeme tento součet, ke kterému přičteme součet území, které přibude, a odečteme součet území, které ubude. Díky tomu je řešení dostatečně rychlé, abychom mohli upravovat koeficient ceny.

Tímto postupem lze dosáhnout ceny **811 315 nákupem 411 domů**.

Program (C++):
`http://ksp.mff.cuni.cz/viz/33-3-4-Robert-Jaw.cpp`

## Ondřej Skácel

Nejdříve jsem zkusil najít dům, který pokryje nejvíc domů. Potom jsem všechny domy, které pokryl, a jeho samotného označil jako pokryté. Poté jsem znovu hledal dům, který by zabral nejvíce domů a nebyl už pokrytý. Takto jsem pokračoval, dokud nebyly všechny domy pokryté. Tím jsem pokryl celou plochu 579 domy, jenže cena za tyto domy byla celkem vysoká (2 568 892). Výpočet tohoto řešení trval asi 1 hodinu a 30 minut.

Poté mě napadlo, že bych si plochu mohl rozdělit na menší části a v nich spočítat řešení. Plochu jsem si rozdělil na tabulku o 33 sloupcích a 33 řádcích, tj. každý sloupec a řádek (až na poslední) je široký 500 políček. V každé buňce ať koupím jakýkoliv dům, tak mám zaručeno, že pokryji všechny domy v ní. V každé buňce jsem našel dům s nejnižší cenou a ten koupil. Samozřejmě toto není efektivní řešení, protože některé domy jsou pokryty vícekrát. Tímto postupem jsem koupil 959 domů za cenu 1 707 759. Poté jsem zkusil kritérium pro výběr domu ke koupi měnit např.: podle počtu domů, poměr mezi cenou domu a počtem domů, které pokryje (čím nižší tím lepší). To, ale nedávalo lepší výsledky. Výpočet tohoto řešení trval pár minut.

Následně jsem se vrátil k mému prvnímu postupu a trochu ho vylepšil. Místo vybírání domů, který pokryje nejvíc domů, tak jsem vybíral dům, podle poměru mezi cenou domu a počtem domů, které pokryje. Tímto způsobem jsem koupil 520 domů za cenu 1 123 293. Výpočet trval stejně dlouho, jako u prvního postupu.

Nakonec jsem tento postup vylepšil tím, že místo toho, abych po koupi domu vyškrtl pokryté domy z dalšího hledání, tak jsem po koupi domu každému domu v jeho okolí upravil počet domů, které pokryje (okolí tj. čtverec $2\,000 \times 2\,000$ se středem v koupeném domě).

Postup poté pokračoval stejně. Našel jsem dům s nejnižším poměrem cena domu/počet domů, které pokryje, upravil jeho okolí a poté jsem tento postup opakoval dokud nebyly všechny domy pokryté. Takhle samozřejmě vzroste počet operací. Abych zrychlil přepočet pokrytí domů v okolí koupeného domu, tak jsem si pamatoval pro každé políčko kolik domů se nachází nalevo od něho včetně něho. Průměrně potom 1 iterace trvala asi minutu. Po každých 15 koupených domech jsem si souřadnice již koupených domů uložil do textového souboru, abych měl uložené mezivýsledky. Celý výpočet zabral asi 10 hodin. Tímto postupem jsem dosáhl celkové ceny **814 413 a 557 domů**. Určitě by se dalo provést pár optimalizací, které by výpočet zrychlily.

## Dan Skýpala

Nejdřív začneme s triviálním řešením. Když si celý vstup rozdělíme na mřížku $500 \times 500$ a v každé buňce mřížky postavíme jeden dům, tak určitě získáme validní řešení. Pro každou buňku si tedy najdeme nejmenší dům v ní a ten zakoupíme. Pokud tam žádný není, tak nic v buňce nekupujeme. Tohle nás dostane na cenu 1 711 755.

Program (Python 3) – mřížka:
`http://ksp.mff.cuni.cz/viz/33-3-4-Dan-mrizka.py`

Můžeme si ale všimnout, že pokrýváme spoustu domů víckrát. Na to použijeme pro řádky následující trik: Když koupíme dům na pozici $[x, y]$, tak další buňku budeme prohledávat od souřadnice $x + 500$. Když například vybereme dům na pozici $[250, 0]$, tak další buňka, ve které budeme hledat, je čtverec s protějšími body $[750, 0]$ a $[1250, 500]$ (místo původního $[500, 0]$, $[1000, 500]$). Takto dosáhneme ceny 1 244 506.

Program (Python 3) – plavoucí mřížka X:
`http://ksp.mff.cuni.cz/viz/33-3-4-Dan-pmrizka.py`

Před tím jsem prošel jednou iterací, kdy jsem se snažil posouvat pro souřadnici y všechny buňky naráz, ale to moc nefungovalo.

Program (Python 3) – plavoucí mřížka XY:
`http://ksp.mff.cuni.cz/viz/33-3-4-Dan-ppmrizka.py`

Nyní posouváme souřadnici $x$ docela dobře, nicméně ještě bychom mohli chtít posouvat po souřadnici $y$. To můžeme udělat takto: Budeme si udržovat intervaly pro předchozí „řádek“ a pro každý si budeme pamatovat, jaký je jeho dosah v souřadnici $y$. Když budeme dělat buňku v rozsahu $x$ až $x + 500$, podíváme se na intervaly, které se toho týkají. a vybereme z nich nejmenší $y$ jako začáteční souřadnici $y$ buňky. Tohle nás nedostane ještě samo o sobě pod milion, ale můžeme použít heuristiku. Budeme preferovat domy, které jsou v buňce vpravo dole, protože zaberou víc nového území a jsou tedy výhodnější. To nás posune pod milion a skončíme na finálních **971 803**.

Program (Python 3) – dvojplavoucí mřížka:
`http://ksp.mff.cuni.cz/viz/33-3-4-Dan-dpmrizka.py`

## David Holas

Kompetitivní úloha byla zajímavá, hlavně tím, že ti nic neodpustí. Myslím, že jde o variantu problému pokrytí množiny výběrem z podmnožin, což je podle Wikipedie těžká

úloha, na kterou není žádný rychlý algoritmus se zaručeně nejlepším výsledkem. Dělají se jen nějaké odhadované dostatečně dobré výsledky.

Docela výzva je nějaké paměťové uspořádání. V mapě je přes 21 milionů domů, každý může pokrýt asi milión ostatních (rekordní dům jich pokrývá asi 470 tisíc). Vytvářel jsem si pomocnou mapu 16K×16K s prvky `uint32` o velikosti 1 GB, což je tak maximum paměti co jsem si mohl dovolit. Včetně zdrojové mapy cen zabíral program asi 1.6 GB paměti.

Zkoušel jsem různé přístupy, zde je jejich přehled.

**Hladový algoritmus podle ceny pokrytých domů**

Hladový algoritmus, který se pro každý dům řídí součtem cen domů, které zkoumaný dům pokryje, versus cena tohoto domu. V každém kroku algoritmus najde dům, který má největší rozdíl mezi cenou jím pokrytých domů mínus cena tohoto domu krát konstanta. Když je pokryto všechno, algoritmus končí. Na mém PC v C++ algoritmus běží pár minut na jeden pokus. Zkoušel jsem různé hodnoty konstanty, nejlepší výsledek byl asi 1.1 milionu.

**Hladový algoritmus podle počtu pokrytých domů**

Hladový algoritmus, který se pro každý řídí počtem domů, které zkoumaný dům pokryje, versus cena tohoto domu. Princip je úplně stejný, jen pracuje s jinou mapou (místo ceny je prvek mapy počet). Pro různé hodnoty konstanty byl výsledek těsně přes milion (přesněji **1 011 260 s 486 domy**), ten jsem odevzdal

**Eliminační algoritmus**

Eliminační algoritmus, který se mi bohužel nepodařilo dotáhnout do konce. Princip vychází z 1 GB mapy pokrytí (v každém bodě mapy je číslo, které udává počet pokrytých jiných domů). Úvaha je taková, že pokud dům A pokrývá dům B, tak i dům B pokrývá dům A (dvojice se pokrývají vzájemně). Takže jsem nejprve seřadil všechny domy podle jejich ceny. Pak jsem postupně bral jednotlivé domy a zkoušel je vyškrtávat.

Čísla v 1 GB mapě jsem chápal jako zbývající počet dosud nevyškrtnutých domů, které tento dům pokrývá. Vyškrtnuté domy jsem si označil příznakem v mapě (maximální počet použije jen 25 bitů, tak do 32 ještě nějaké místo na příznaky je). Jeden příznak byl pro vyškrtnutí, další příznak byl, že tento dům je již pokrytý. Dům jde v pohodě vyškrtnout, pokud nemá ve svém okolí (čtverec ±500) žádný dům s pokrytím rovným jedná. Hodnota jedna znamená, že zpracovávaný dům je jediný, kdo může tento dům pokrýt (on je ta jednička).

Pokud má dům v okolí nějakou jedničku s příznakem vyškrtnutí, tak tento dům musíme použít do výsledku, protože on je jediná možnost, jak tuto vyškrtnutou jedničku ještě pokrýt. Pokud všechny nalezené jedničky jsou ještě nevyškrtnuté, tak se můžeme rozhodnout, zda použijeme aktuálně zpracovávaný dům, nebo místo něj všechny domy s jedničkou v jeho okolí (podle toho, co je levnější). Musíme nejprve ale zkontrolovat, zda některý z těchto domů nemá ve svém okolí také nevyškrtnutou jedničku. Pak by byla povinná, protože pro tuto jedničku je to jediná možnost.

Zbylé nepovinné jedničky porovnáme s cenou aktuálního domu a použijeme lepší variantu. Vyškrtnutí domu znamená označit ho jako vyškrtnutý (už se nikdy nepoužije pro pokrytí) a snížit všechna čísla v jeho okolí o jedna (počet nevyškrtnutých domů, které příslušné políčko mohou pokrýt).

Algoritmus končí, když jsou všechny domy pokryté (stačí připočítávat všechny nově pokryté domy). Tenhle algoritmus vypadal na malých zkušebních datech hodně slibně, ale pro celou Prahu jsem ho bohužel nedotáhl. Vždycky jsem někde zapomněl na nějakou variantu a výsledek nebyl dobře. Nejslibnější výsledek (i když ne zcela správný) byl někde kolem 650 tisíc. Ten by se třeba dal doopravit :) Běželo to asi 16 hodin.

**Náhodné zkoušení**

Uvažoval jsem nad náhodným zkoušením, prostě vygenerovat náhodné pokrytí a to zkoušet nějakými náhodnými změnami vylepšovat (zkoušet něco ubrat nebo posunout na levnější vedlejší), to by mohlo postupně zlepšovat celkovou cenu pokrytí. Když už by to dál nevedlo, tak by se to trochu víc zamíchalo. Mohlo by se takových náhodných pokrytí zkoušet víc a pak ty nejlepší kousky z nich nějak kombinovat. Tuhle variantu jsem neprogramoval.