Automatizované riešenie úloh o dláždení polyominami

Ročníkový projekt, zimný semester 2020/2021

Dávid Mišiak
školiteľ doc. RNDr. Ján Mazák, PhD.

1 Zadanie

Rovinný útvar, ktorý dostaneme spojením niekoľkých rovnakých štvorčekov pozdĺž ich hrán, je známy ako polyomino[1] („zovšeobecnené domino“). V zábavnej matematike sa stretávame s triedou úloh, kde má riešiteľ dokázať alebo vyvrátiť existenciu dláždenia zadanej štvorčekovej mriežky zadanými polyominami, napríklad:

> Dá sa mriežka 10×10 vydláždiť 25 tetrominami 4×1?

Cieľom projektu je vytvoriť program, ktorý bude vedieť riešiť tento typ úloh (teda nájsť dláždenie alebo dokázať, že neexistuje). Súčasťou projektu bude definovanie vhodných vstupných a výstupných formátov, implementácia viacerých metód riešenia (backtracking, SAT solver, ...) a ich porovnanie.

2 Pojmy

Zaveďme si niekoľko pojmov:

Oblasť je geometrický útvar, ktorý vznikne spojením niekoľkých jednotkových štvorčekov pozdĺž ich hrán. Tento útvar nemusí byť súvislý. Dve oblasti, ktoré sa líšia otočením či prevrátením, sa považujú za rôzne.

Dlaždica predstavuje konkrétny tvar polyomina. Jedna dlaždica teda obsahuje niekoľko rôznych *oblastí* – jedna pre každé unikátne otočenie a prevrátenie oblasti, ktorou je definovaná (teda napríklad dlaždica reprezentujúca polyomino v tvare obdĺžnika 3×1 bude obsahovať dve rôzne oblasti). Ak nie je povedané inak, mala by byť súvislá a nemala by obsahovať „diery“.

Doska je *oblasť*, ktorú máme vydláždiť danými *dlaždicami* (čiže v príklade v Zadaní je doskou mriežka 10×10 štvorčekov).

Prvý štvorček oblasti je najľavejší štvorček v najvrchnejšom riadku štvorčekov danej *oblasti*. V istom zmysle je to teda „ľavý horný roh“ oblasti.

3 Realizácia

Všetky zdrojové súbory sa nachádzajú v repozitári Tiler na adrese `https://gitlab.com/puding/tiler`.

3.1 Kostra projektu

Aplikáciu som sa rozhodol napísať v jazyku C++ kvôli výkonu, dobrej interoperabilite so SAT solvermi či iným softvérom a tiež preto, že si chcem vyskúšať spraviť nejaký väčší projekt v tomto jazyku. Pre nedostatok skúseností s C++ som prácu na projekte začal štúdiom základných a pokročilejších princípov C++. Okrem nutných pravidiel syntaxe som sa snažil počas programovania držať aj dobrej praxe[2].

Keďže výstup z tohto ročníkového projektu by mal slúžiť ako základ bakalárskej práce, veľké množstvo pozornosti bolo venované kvalite projektu z vývojárskeho hľadiska. Oboznámil som sa s často používanými nástrojmi a rozhodol sa, že použijem

[1] `https://en.wikipedia.org/wiki/Polyomino`
[2] napr. príručka `https://google.github.io/styleguide/cppguide.html`

- **Gitlab** ako hosting repozitára a Gitlab CI na continuous integration,
- **Conan** package manager na správu knižníc,
- **ClangFormat** na formátovanie kódu a
- **Cppcheck** na statickú analýzu kódu.

Buildovanie je pod správou nástroja **CMake**. Práve rozbehanie CMake so všetkými zaujímavými funkciami, ktoré podporuje (správne nastavenie kompilátora, integrácia s Conanom, ...) bolo najnáročnejšie[3], ale momentálna konfigurácia by mala byť solídna a skompilovanie projektu na novom zariadení je jednoduché (detaily v `README.md`). Jedna z veľkých výhod CMake je jeho podpora rôznych platforiem a kompilátorov. Pri zostavovaní konfiguračných súborov som sa snažil toto neobmedziť – momentálne mám síce buildovanie otestované len na Linuxe s kompilátorom GCC, ale malo by fungovať aj inde, resp. malo by byť pomerne jednoduché projekt prispôsobiť iným kompilátorom či platformám, ak by to bolo potrebné.

Aplikácia má byť plne používateľná z príkazového riadka (poskytovať tzv. *command-line interface*, *CLI*). Na to som vybral knižnicu **CLI11**[4], ktorá je relatívne obľúbená a poskytuje dostatočné API pre tento projekt.

Na automatizované testy je použitá knižnica **Catch2**, ktorej výhodami sú popularita a jednoduchosť použitia.

3.2 Vstup

Vstupom pre aplikáciu je zadanie úlohy – definícia dosky a dlaždíc. V zložke `src/problem/` sa nachádzajú definície tried reprezentujúcich oblasť (`Region`, samotný tvar má uložený v matici `bool`-ov), dlaždicu (`Tile`, pričom jej súčasťou je aj počet kusov tejto dlaždice), dosku (`Board`) a úlohu (`Problem`, pozostáva z inštancie dosky a zoznamu inštancií dlaždíc).

Používateľ vie zadať dosku a dlaždice ako zoznam reťazcov – buď ako command-line argumenty (oddelené medzerou) alebo v textovom súbore (oddelené prázdnym riadkom).

Reťazec definujúci *oblasť* môže byť uvedený v niekoľkých rôznych formátoch:

- Názov. Všetky tvary oblastí o veľkosti 1 až 5 štvorčekov (tj. 29 rôznych tvarov, rozlišujú sa prevrátenia, ale nie otočenia) majú vlastný krátky názov pozostávajúci z veľkosti a písmena, ktorého tvar daná oblasť pripomína. Teda napríklad oblasť 3×1 má názov `3I` či oblasť pozostávajúca zo štyroch štvorčekov usporiadaných do tvaru písmena „T“ má názov `4T`.
- Rozmery. Oblasti v tvare štvorca alebo obdĺžnika môžu byť zadané jednoducho dĺžkami svojich strán, teda napríklad oblasť 4×10 vieme zadať ako `4x10`.
- Mapa. Oblasť je zadaná ako mapa štvorčekov, ktoré do nej patria a ktoré nie – ak štvorček patrí do oblasti, je zaznačený písmenom `x`, inak medzerou. Teda napríklad oblasť s názvom `4T` (viď prvý formát) vieme v tomto formáte zapísať nasledovne:

  ```
  xxx
   x
  ```

  Predchádzajúce dva formáty zachytávajú len úzku triedu všetkých možných tvarov, v tomto formáte je však možné definovať ľubovoľnú oblasť. Je však vhodný len pri vstupe zo súboru (kvôli použitiu znaku nového riadku).

Reťazec definujúci *dosku* je jednoducho reťazec definujúci oblasť.

Reťazec definujúci *dlaždicu* je taktiež reťazec definujúci jej oblasť, avšak môže mať prefix v tvare „N:“, kde N určuje počet kusov tejto dlaždice. Ak tento prefix nie je uvedený, predpokladá sa, že máme k dispozícii neobmedzený počet kusov danej dlaždice.

Otáčanie dlaždíc má aplikácia vždy dovolené, čiže ak zadáme dlaždicu jej oblasťou, automaticky sa dopočítajú všetky rotácie. S prevracaním dlaždíc je to zložitejšie – v praxi sa vyskytujú dláždiace problémy, kde je prevracanie zakázané, ale aj také, kde je dovolené. Aplikácia preto na nastavenie požadovaného režimu poskytuje CLI prepínač (ak nie je uvedený, je prevracanie zakázané).

Kvôli prehľadnosti kódu má každá trieda svoj parser vstupu, ktorý sa vie odvolávať na primitívnejšie parsery (napr. parser dlaždice oddelí „N:“ od definície oblasti a na ňu zavolá parser oblasti).

[3] veľkou pomocou boli projektové šablóny, napr. `https://github.com/lefticus/cpp_starter_project`
[4] `https://github.com/CLIUtils/CLI11`

3.3 Jednoduchý solver

Prvý z algoritmov na samotné riešenie dláždiaceho problému, definovaný v zložke `src/solvers/`. Berie inštanciu triedy `Problem` a vracia `bool` hodnotu – či existuje riešenie.

Tento solver funguje na princípe backtrackingu. V každom rekurzívnom volaní iteruje cez všetky oblasti všetkých dlaždíc (ktoré sa ešte neminuli). Zakaždým nájde *prvý štvorček* dosky (definícia viď vyššie) aj aktuálnej oblasti a skúsi aktuálnu oblasť položiť na dosku tak, že sa ich prvé štvorčeky zhodujú. Ak sa to dá (čiže oblasť je celá na nepokrytej časti dosky), funkcia sa rekurzívne zavolá s doskou zmenšenou o túto oblasť. Ak sa nepodarí nájsť žiadna oblasť žiadnej dlaždice, ktorá by sa dala priložiť, táto vetva prehľadávania možných riešení sa skončila neúspechom.

Správnosť fungovania tohto algoritmu je pomerne zjavná – pri pohľade na aktuálne nepokrytú časť dosky je isté, že jej *prvý štvorček* bude skôr či neskôr musieť byť pokrytý niektorou oblasťou niektorej dlaždice. Nech to bude ktorákoľvek, vieme, že tento štvorček musí byť pokrytý prvým štvorčekom danej oblasti (v opačnom prípade by sme sa dostali do sporu s definíciou prvého štvorčeka), môžeme teda skúsiť vyskúšať všetky možné pokrytia aktuálneho prvého štvorčeka dosky a ak sa nám to nepodarí, môžeme sa bezpečne vrátiť o jednu úroveň rekurzie vyššie. Ak riešenie existuje, dá sa zostrojiť týmto algoritmom (to, že už ho máme, zistíme jednoducho tak, že nepokrytá časť dosky je prázdna).

Táto technika prikladania dlaždíc cez prvé políčko je veľmi prirodzená (dalo by sa povedať, že dokonca elegantná), vďaka čomu je jej implementácia pomerne jednoduchá a jej beh transparentný, čo sú vítané vlastnosti najjednoduchšieho solvera. Jej nevýhodou je samozrejme pomalý beh na väčších doskách.

4 Súčasný stav

Aplikácia je kompilovateľná bez chýb aj varovaní. Po spustení s prepínačom `--help` vypíše stručný návod na použitie. Momentálne obsahuje príkaz na vypísanie všetkých pomenovaných oblastí, príkaz na vypísanie ukážkových vstupov, a samozrejme príkaz na riešenie problému. Ak dostane vstup v nesprávnom formáte alebo sú nesprávne použité command-line prepínače, vypíše chybovú hlášku (vrátane konkrétneho problému) a skončí. Ak je vstup platný, vypíše rekapituláciu problému (to sa dá potlačiť prepínačom `--quiet`), problém vyrieši a vypíše výsledok (`TRUE` alebo `FALSE`).

5 Ďalšie kroky

Najurgentnejším nedostatkom je absencia testov – iba trieda `Region` má napísané nejaké testy a je možné, že v zdrojovom kóde ostatných tried sa ukrývajú chyby, ktoré sa odhalia vďaka kvalitným testom. V letnom semestri preto začnem práve písaním testov. Taktiež sa pokúsim mierne sprehľadniť kód a skvalitniť pomocné hlášky a `README.md`.

Okrem toho by v lete mala pribudnúť nová funkcionalita:

- Štvrtý parser vstupu, ktorý bude brať oblasti definované obvodom – teda napríklad `DRDRURULLL` bude ekvivalent oblasti `4T` (pohybujeme sa po stranách štvorčekov, pričom `D` znamená dole, `U` hore, `L` doľava a `R` doprava; a „nakreslíme“ celý obvod oblasti).
- Ak existuje riešenie, okrem vypísania `TRUE` sa vytvorí obrázok vyhovujúceho dláždenia.
- Ďalší algoritmus na riešenie problému – bude využívať SAT solver.

Ak sa podarí, chcel by som tiež zostaviť nejakú množinu vstupných problémov, na ktorých by sa dal porovnať výkon jednotlivých algoritmov.