Grounded-L Grafy

Samuel Krajči

1 Úvod

Cieľom tohoto ročníkového projektu bolo vytvoriť rôzne nástroje na uľahčenie skúmania *Grounded-L* triedy prienikových grafov.

Konkrétne, boli vyvinuté tieto nástroje:

- Kreslič grounded-L reprezentácie grafu

  hľadanie grounded-L reprezentácie sme implementopvali troma spôsobmi:

  - hrubá sila (*bruteforce*)
  - eliminácia vzorov (*pattern elimination*)
  - paralelizácia
- Editor usporiadania grounded-L grafov
- Grafový editor

Všetky programy boli písané v jazyku *Python*.

2 Kreslič grounded-L reprezentácie

Dejú sa tu dve veci. **Hľadanie** a samotné **vykreslenie** grounded-L reprezentácie.

Vykreslenie

Vykresľovanie je pomerne priamočiare a nezaujímavé, deje sa pomocou knižnice *tkinter*. Implementácia sa nachádza v súbore `graph_utils.py`

Hľadanie

Hľadanie grounded-L reprezentácie je kľúčovou časťou celého projektu a je implementované viacerými spôsobmi, ktoré popíšeme nižšie.

Grounded-L reprezentácia pozostáva z troch častí:

- **poradie** vrcholov; je to permutácia vrcholov
- **výšky** (resp. hĺbky) nožičiek jednotlivých L-iek; je to permutácia vrcholov (podstatné sú iba relatívne výšky)
- **dĺžky** pätiek jednotlivých L-iek

Hrubá sila

Najpriamočiarejší spôsob hľadanie týchto parametrov je *hrubá sila*, teda pre každú kombináciu rôznych poradí, výšok a dĺžok overiť, či sa pretínajú práve tie L-ká, ktoré majú. Časová zložitosť takého algoritmu je $O(n!^3)$.

O krok rozumnejšie je pre každú kombináciu poradia a výšky vypočítať pre každé L-ko minimálnu dĺžku aby dosiahlo na najvzdialenejšieho suseda a opäť overiť nadbytočné prieniky. Časová zložitosť nám klesne na $O(n!^2)$. Presne tento algoritmus je implementovaný v súbore `bruteforce.py`.

Podobne ako sme sa vyhli prehľadávaniu všetkých možností dĺžok sa vieme vyhnúť aj prehľadávaniu všetkých možností výšok. Zľava doprava budeme L-kám priraďovať výšky a každému ďalšiemu dáme minimálnu výšku aby siahalo nižšie ako všetci jeho susedia naľavo od neho. Samozrejme, opäť aj overíme, či máme skutočne práve tie prieniky, ktoré majú byť. Takto nám časová zložitosť klesne na $O(n!)$, pretože už skúšame iba všetky poradia. Tento algoritmus je implementovaný v súbore `bruteforce_smarter.py`.

Zatiaľ čo hľadanie výšok a dĺžok sa nám darí robiť celkom efektívne, hľadanie poradia je stále veľmi neefektívne. Zistenia z článku V. Jelínka a M. Töpfera o grounded-L grafoch.

Eliminácia vzorov

Z vyššie spomínaného článku vieme, že poradie nesmie obsahovať nejaké podpostupnosti dĺžky 4. Poradia teda budeme skúšať vo vzostupnom poradí podľa čísla premutácie a pri každej permutácii overíme, či sa v nom nachádza nejaký zakázaný vzor. Ak áno, preskočíme všetky nasledujúce permutácie v ktorých sa tento vzor bude nachádzať.

Presnejšie, pre aktuálnu permutáciu budeme pre jednotlivé vrcholy zľava doprava overovať, či daný vrchol netvorí zakázaný vzor spolu s nejakými tromi vrcholmi naľavo od neho a ak áno, nepreskočíme iba na nasledujúcu permutáciu, ale rovno na najmenšiu takú, ktorá má na tomto mieste iný vrchol.

Časová zložitosť tohoto algoritmu je síce $O(n!)$, no v praxi sa ukazuje, že je rýchlejší ako predchádzajúce. Je implementovaný v súbore `pattern_elimination.py`.

Paralelizácia

Paralelizovať vieme hociktorý z vyššie popísaných algoritmov úplne priamočiaro, pretože overovania jednotlivých usporiadaní sú úplne nezávislé. Všetky poradia teda rozdelíme rovnomerne viacerým procesom (čo je počet dostupných vlakien), a skončíme buď ak prvý z nich objaví vyhovujúce poradie, alebo ak každý z nich prejde všetky poradia a zistí, že ani jedno z nich nevyhovuje.

Tento algoritmus má časovú zložitosť $O(\frac{X}{p})$, kde X je časová zložitosť pôvodného algoritmu a p je počet procesov, ktorú použijeme (typicky počet vlákien, teda na bežných počítačoch nanajvýš v ráde desiatok).

V súbore `parallelization.py` je paralelne implementovaný algoritmus *eliminácie vzorov* a teda jeho časová zložitosť je $O(\frac{n!}{p})$.

Multiprocessing je implementovaný pomocou knižnice `concurrent`.

3 Editor usporiadania grounded-L grafov

Na skúmanie Grounded-L grafov sa môže hodiť nástroj na pozorovanie lokálnych úprav poradia vrcholov, v ktorom vime jednoducho meniť poradie vrcholov a vidíme zlé prieniky. Implementácia sa nachádza v súbore `orderng_visualizer.py`.

Na vykresľovanie je opäť použitá knižnica *tkinter*.

Program jednoducho pre užívateľom zvolené poradie zvolí výšky a dĺžky ako sme vyššie popisovali a, aj napriek prebytočným prienikom, vykreslí graf a zvýrazní zlé prieniky.

4 Grafový editor

Grafový editor slúži na jednoduché vytvorenie testovacích vstupov pre bežné grafové úlohy, vrátane našej.

Hlavné funkcie sú:

- Jednoduché vytvorenie menších grafov
- Export do štandardného grafového textového formátu
- Možnosť integrovať postprocessing vytvoreného grafu (v našom prípade chceme vykresliť jeho Grounded-L reprezentáciu)

Program vieme rozčleniť na *backend* a *frontend*.

4.1 Backend

Backend je tvoreny jedným, objektom triedy `Graph`, v ktorom je uložený celý graf a obsahuje základné metódy na pridanie/odobranie vrcholu/hrany, ale aj serializáciu grafu do *štandardného grafového formátu*.

Každý vrchol má navyše svoj *label*, teda meno. To sú stále zasebouidúce čísla začínajúce nulou. Po vymazaní ľubovoľného vrcholu sa vrcholy vždy premenujú, aby boli *labely* stále zasebouidúce.

Štandardný grafový formát

Tento formát sa používa v takmer každej grafovej úlohe *kompetetívneho programovania*. V prvom riadku je počet vrcholov a počet hrán a na zvyšných riadkoch sú dvojice vrcholov medzi ktorými vedie hrana. Napríklad K_4 vyzerá takto:

```
4 6
0 1
0 2
0 3
1 2
1 3
2 3
```

4.2 Frontend

Na vykresľovanie je opäť použitá knižnica tkinter.

Frontend je tvorený objektom triedy `Screen`, ktorý si drží stav grafu v objekte `Graph` a vlastný `tkinter.canvas` a všetky potrebné parametre na vykresľovanie (ako napríklad rozmery okna, alebo veľkosť vrcholov).

Pre komunikáciu backendu a frontendu sú dôležité listy `vertex_id`, `edge_id` a `label_id`, ktoré priraďujú id objektov frontendu (jednotlivé krivky vrcholov, hrán a ich labely) ku objektom backendu.

4.3 Postprocessing grafu

Často chceme s vytvoreným grafom rovno niečo spraviť. Napríklad v našom prípade chceme vykresliť jeho Grounded-L reprezentáciu. Na to slúži optional argument `submit_fn` (a `submit_button_text` a `submit_button_tooltip`) objektu `Screen` kde môžeme špecifikovať čo sa má s grafom stať. Funkciu voláme tlačidlom na *canvase* (ak nie ej špecifikovaná, tlačidlo nebude existovať).

Funkcia musí brať v prvom parametri súbor z ktorého načíta graf. Keď objekt `Screen` túto funkciu volá, uloží graf do dočasného súboru a ten pošle do funkcie v prvom parametri.