Travelling Salesman Challenge

Travelling Salesman Challenge

Toto je bonusová úloha. Dá sa za ňu získať nanajvýš 7 bonusových bodov.

Budeme sa zaoberať exaktným riešením problému obchodného cestujúceho vo verzii s neorientovanými (symetrickými) hranami pomocou orezávaného backtracku.

Vašou úlohou bude navrhnúť heuristiku, ktorá algoritmu umožní orezať čo najviac možností (samozrejme, bez toho, aby sme pri tom orezali aj optimálne riešenie).

Algoritmus

Problém obchodného cestujúceho riešime algoritmom, ktorý postupne generuje všetky permutácie vrcholov začínajúce vrcholom 0, pre každú z nich spočíta jej cenu (teda cenu kružnice, ktorá vrcholy navštevuje v tomto poradí) a nakoniec vyberie najmenšiu z týchto cien.

Permutácie sú generované backtrackom, ktorému zhruba zodpovedá nasledujúci pseudokód (skutočný kód nájdete v súbore `tsp.cpp`).

```
1 def vyskusaj_pokracovania(prefix, nenavstivene_vrcholy):
2   if len(prefix) == n:
3     zapocitaj_permutaciu(prefix)
4   else:
5     for nasledujuci in nenavstivene_vrcholy:
6       vyskusaj_pokracovania(prefix + [nasledujuci], nenavstivene_vrcholy - {nasledujuci})
7
8 vyskusaj_pokracovania([0], {1, 2, ..., n-1})
```

Okrem toho ale robíme ešte dva triky:

1. Orezávame neperspektívne vetvy rekurzie. Teda v prípade, že je pre nejaký prefix permutácie jasné, že všetky kružnice začínajúce týmto prefixom budú horšie, než najlepšia doteraz nájdená, tento prefix sa už nesnažíme predlžovať. Aby sme vedeli o nejakom prefixe povedať, že je neperspektívny, potrebujeme vedieť zdola odhadnúť, aká najlepšia kružnica sa začína týmto prefixom. To urobíme s pomocou heuristiky, ktorej vymyslenie a naprogramovanie bude vaša úloha.
2. Pri vetvení prechádzame vetvy od najsľubnejších. Teda keď pre nejaký prefix našej permutácie skúšame všetky možnosti nasledujúceho vrcholu (`for`-cyklus na riadku 5), najprv pre každý možný nasledujúci

vrchol odhadneme (našou heuristikou), akú najlepšiu cenu môže mať permutácia pokračujúca týmto vrcholom. Vrcholy s nižším odhadom ceny potom skúšame skôr. Cieľom tohto triku je, aby sme čím skôr našli nejaké dobré riešenie, vďaka ktorému budeme môcť orezať veľa vetiev v ďalšom behu nášho backtracku.

Technické detaily

Táto úloha je čisto praktická (programátorská). Dá sa riešiť iba v jazyku C++. V priloženom zipe by ste mali nájsť súbory `tsp.cpp`, `heuristic.h` a `heuristic.cpp`. Súbor `tsp.cpp` obsahuje hotovú implementáciu vyššie spomínaného backtracku a mal by vám slúžiť len na testovacie účely. Vašou úlohou je implementovať triedu `Heuristic` v súbore `heuristic.cpp` (táto heuristika je potom volaná z `tsp.cpp`). Súbor `heuristic.cpp` je jediný z poskytnutých súborov, ktorý potrebujete editovať.

Vaším cieľom je, aby backtrackovací algoritmus používajúc vašu heuristiku vyriešil čo najviac testovacích inštancií TSP v časovom limite.

Trieda Heuristic

Trieda `Heuristic` musí mať verejný konštruktor so signatúrou

```
Heuristic(unsigned n, std::vector<std::vector<int>> distances)
```

a (verejnú) metódu `LowerBound` so signatúrou

```
int LowerBound(unsigned from_vertex,
               unsigned to_vertex,
               const std::vector<unsigned> &through_vertices,
               int upper_bound)
```

(deklarácie oboch metód nájdete pripravené v `heuristic.cpp`). Okrem toho v nej môžete mať ľubovoľné ďalšie metódy, podľa vašej potreby.

Pri konštrukcii dostane `Heuristic` popis grafu: počet vrcholov `n` a maticu vzdialeností `distances` typu $n \times n$. Pre maticu `distances` bude platiť, že obsahuje iba nezáporné celé čísla nepresahujúce 2000, je symetrická (vzdialenosť z u do v je rovnaká, ako z v do u) a na hlavnej diagonále má samé nuly (vzdialenosť z vrcholu do neho samého je nulová).

Náš backtrackovací algoritmus bude počas svojho behu volať metódu `LowerBound`, ktorá má nasledujúci význam:

Odhadni, ako najlacnejšie sa dá dostať z vrcholu `from_vertex` do vrcholu `to_vertex`, ak pri tom musíme práve raz navštíviť každý vrchol z `through_vertices` (a nesmieme navštíviť žiaden iný vrchol).

Pritom bude platiť, že `from_vertex` a `to_vertex` sú rôzne vrcholy a žiaden z nich sa nevyskytuje v `through_vertices`. Ako napovedá aj názov metódy, má ísť o dolný odhad, teda ak `LowerBound` vráti hodnotu c, nemôže existovať cesta lacnejšia než c (ak nesplníte túto podmienku, backtrack môže orezať aj vetvu s optimálnym riešením). Zároveň chceme čo najtesnejší odhad, aby toho backtrack mohol orezať čo najviac. Argument `upper_bound` má nasledujúci význam:

Ak by mal byť výsledok väčší než (alebo rovný) ***`upper_bound`****, nie je už dôležité, o koľko presne bude väčší (lebo už vieme, že ide o neperspektívnu vetvu).*

Tento parameter možno nevyzerá užitočne, ale pri niektorých typoch heuristík sa môže hodiť (napr. ak vaša heuristika iteratívne vylepšuje svoj odhad, vďaka `upper_bound` môže niekedy prestať vylepšovať o niečo skôr).

Testovanie

Aby ste mohli spustiť svoj kód, treba ho skompilovať a zlinkovať s `tsp.cpp`, teda napr.

```
g++ heuristic.cpp tsp.cpp -o tsp
```

vám vytvorí binárku `tsp`. Tá potom očakáva na štandardnom vstupe popis grafu vo formáte ako v súbore `sample.in` a vypisuje na štandardný výstup a na stderr. Spustíte ju teda napríklad ako

```
./tsp < sample.in
```

Odovzdávanie

Na stránke https://testovac.ksp.sk/tasks/artp-2020-tsp/ odovzdajte **len súbor `heuristic.cpp`** (nie `tsp.cpp`!). Ako aj pri iných programátorských úlohách, máte viac pokusov, po každom pokuse sa do pár minút dozviete výsledky testovania.

Bodovanie

Za každý vstup (okrem `00.sample.10.in`), ktorý backtrack s vašou heuristikou vyrieši, dostanete jeden bonusový bod.