Samuel Bubán 102879 25.10.2020

# Bláznivá križovatka

**Zadanie:**

- treba nájsť riešenie, kde červené auto prejde úplne vpravo
- hlavolam je reprezentovaný mriežkou
- vozidlá majú šírku jedna, nemôžu sa otáčať
- v jednom kroku sa môžu posunúť o 1-n políčok dopredu alebo dozadu

**Opis riešenia:**

Na vyriešenie hlavolamu som použil cyklicky sa prehlbujúce hľadanie. V princípe je to rovnaké prehľadávanie ako do hĺbky, ale je obmedzená maximálna hĺbka, ktorá sa s každým krokom zvýši o 1.

Ako jeden uzol som si určil aktuálny stav. Každý stav je reprezentovaný mriežkou obsadených (prázdnych) políčok, a zoznamom áut s ich pozíciami.

Tento problém som riešil rekurzívne – ak sa môže nejaké auto posunúť o ľubovoľnú vzdialenosť, znovu sa zavolá funkcia, ktorá skúša posúvanie všetkých áut o všetky možné vzdialenosti.

Ja som tento problém zovšeobecnil, a vyriešil som ho pre ľubovoľnú mriežku MxN. Takisto som ho vyriešil pre ľubovoľne dlhé autá (nákladiaky), a môžu byť aj ľubovoľne otočené (hore, dole, vpravo, vľavo).

Moje riešenie je optimálne vzhľadom na pamäť, teda v ľubovoľnom čase využíva minimum pamäte. V riešení som implementoval možnosť pamäťovo menej efektívneho riešenia, ktoré je ale časovo mnohonásobne rýchlejšie. Týmto pamäťovo neefektívnym riešením je možné otestovať vstupy, ktoré by dlho trvali, a potom porovnať výsledky. Nevýhodou pamäťovo neefektívneho variantu je tiež neoptimálnosť výsledného riešenia – algoritmus nemusí nájsť nakratší spôsob, ako vyriešiť hlavolam.

**Reprezentácia údajov:**

Prvotné riešenie obsahovalo množstvo tried, ktoré udržovali veľké množstvo aj statických dát. Preto bolo toto riešenie neefektívne, a výsledný program žiadne triedy neobsahuje. Vypustenie tried urýchlilo program niekoľko-násobne.

Každý stav je implementovaný ako zoznam, ktorý obsahuje mriežku a zoznam áut. Jedna mriežka je reprezentovaná tabuľkou, ktorá obsahuje údaje o tom, či je dané políčko obsadené alebo nie. Zoznam áut obsahuje len informácie o aktuálnej pozícií daného auta. Prvé auto je vždy červené, teda to, ktoré sa potrebuje dostať doprava.

Program ďalej obsahuje niekoľko statických informácii, ktoré sa nikdy nemenia. Jedná sa o dĺžku jednotlivých áut a ich otočenie. Tieto informácie sú uložené tiež v zozname.

**Spôsob testovania:**

Na testovanie som si vytvoril niekoľko rôznych vstupných máp, niektoré mali riešenie jasné, niektoré sa nedali vyriešiť. Pre všetky tieto mapy som dostal výsledok, ktorý som očakával, a preto uvažujem, že program pracuje ako má.

Svoje riešenie som tiež porovnával s kolegami, ktorí mali rovnaké zadanie. Porovnávali sme časovú zložitosť našich algoritmov, a tiež schopnosť nájsť optimálne (najkratšie) riešenie. Pre všetky vstupy sme mali rovnaké výsledky.

Niekoľko vstupných a k nim výstupných súborov som priložil aj k dokumentácií.

**Zhodnotenie.**

So svojim programom som vysoko spokojný, nakoľko sa mi podarilo vytvoriť čistý kód, ktorý sa dá jednoducho upraviť podľa potreby.

Počas riešenia tohoto zadania som vytvoril viac prototypov.

**Zapamätávanie si všetkých navštívených stavov pomocou tried**

Môj prvý prototyp využíval zapamätávanie si všetkých stavov. Tento spôsob negarantuje nájdenie optimálneho riešenia problému, a preto som ho musel upraviť. Druhý problém tohto prvého algoritmu bolo využívanie tried, čo sa mi neskôr preukázalo ako veľmi časovo neefektívne. Pre porovnanie – na základnom vstupe toto riešenie prešlo za približne 1 minútu a 20 sekúnd. Keď som sa zbavil tried, riešenie som získal za zopár sekúnd.

**Nezapamätávanie si ničoho:**

Ako druhý prototyp som skúsil riešenie, ktoré si nepamätá vôbec nič, ani len rodičovský stav. Toto riešenie by malo byť pamäťovo najefektívnejšie, nakoľko si neukladá žiadne údaje, ale po tom, ako som toto riešenie pustil na základný vstup, a nechal ho bežať 10 minút som zistil, že je prakticky nemožné toto využiť. Môj odhad, koľko by trvalo tomuto algoritmu vyriešenie základného vstupu je – viac ako 500 rokov. Toto je neprípustné, preto si treba pamätať aspoň nejaké už navštívené uzly.

**Výsledné riešenie:**

Ako výsledný algoritmus som si zvolil interpretáciu, ktorá neobsahovala žiadne triedy (mnoho-násobné urýchlenie riešenia). Ďalej som zmenil množstvo údajov, ktoré si musí každý stav pamätať, a obmedzil ho na minimum. Každý stav teda obsahuje len údaje o tom, čo sa reálne môže meniť, a informácie ako veľkosť tabuľky alebo otočenie vozidiel sú uložené v statických zoznamoch.

Nakoľko nepamätanie si žiadnych údajov bolo časovo veľmi neefektívne, rozhodol som sa, že každý stav bude mať informáciu o všetkých svojich predchodcoch. Tieto informácie sú uložené na to, aby keď algoritmus nájde riešenie, aby sa dali vypísať ako optimálna cesta, teda riešenie.

Každý uzol skúša posunúť všetky autá do všetkých smerov, ak zistí, že riešenie sa nenachádza ani v jednom z jeho podstromov, vymaže svoje údaje z navštívených stavov, a posunie sa rekurzívne o úroveň späť.

Algoritmus bude stále zväčšovať maximálnu hĺbku, až kým sa nedostane do bodu, že už hlbšie ísť nevie. Týmto som ošetril riešenia, kde sa môže výsledná cesta nachádzať ľubovoľne hlboko. Funguje to tak, že ak sa uzol, ktorý je aktuálne riešený nachádza vo väčšej hĺbke, ako bola zatiaľ dosiahnutá, zapíše sa ako nová najväčšia hĺbka. Po tom, ako prejdú všetky možnosti pre aktuálnu maximálnu hĺbku algoritmus porovná, či sa táto hĺbka aj dosiahla. Ak áno, znamená to, že sa tam ešte môžu nachádzať ďalšie neobjavené stavy, a teda sa zvýši maximálna hĺbka o jedna. Ak je ale maximálna dosiahnutá hĺbka menšia ako maximum, znamená to, že už tam žiadne ďalšie riešenia neexistujú, a teda boli prehľadané všetky stavy, preto pre daný vstup neexistuje riešenie.

Skúšanie, či je aktuálny uzol výsledný – vždy, keď sa vytvorí nový stav (uzol), je otestované, či sa jedná o výslednú pozíciu, teda či prvé auto (červené) je úplne vpravo. Ak áno, rekurzívna funkcia skončí s Flagom True, a takisto aj všetky pred ňou. Následne sa vypíše riešenie.

Svoj algoritmus som musel niekoľkokrát optimalizovať, pretože prvá verzia optimálneho riešenia trvala na základom vstupe presne 59 minút na vyriešenie. Po čiastočnom upravení podmienok sa mi podarilo tento čas znížiť na výsledných 16 minút pre základný vstup.

Vstupy, ktoré vyžadujú počet krokov do 7 sa vyriešia takmer okamžite. Následne každý ďalší krok zvyšuje časovú záťaž exponenciálne – pričom základ mocniny je počet možných krokov v danej hĺbke.

Najprv som využil jednoduchšie riešenie, ktoré počítalo len s možnosťou pohybu o jedno vpred alebo vzad. Toto znížilo základ mocniny, ktorá násobila čas, ale zato zvýšilo mocninu, čo je o dosť horšie.

**Porovnanie:**

Algoritmus s cyklicky sa zvyšujúcou hĺbkou je optimalizovaný na priestorovú zložitosť, a teda nevyžaduje takmer žiadnu pamäť. Pre porovnanie napríklad prehľadávanie do šírky – veľmi vysoké využitie pamäte, ale zato nízka časová zložitosť, stotiny sekundy.

Prehľadávanie do hĺbky – ideálna priestorová zložitosť, ale veľmi neefektívne časové prevedenie (500 rokov).

Preto cyklicky sa prehlbujúci algoritmus je spojenie týchto dvoch riešení, a teda niečo medzi tým.

**Možné úpravy, optimalizácie ... :**

Skúsil som zmeniť, aby sa maximálna hĺbka zvyšovala napríklad o 2 vždy. Toto by zabezpečilo veľké časové zrýchlenie pre náročnejšie vstupy, ale môže sa stať, že riešenie nebude optimálne (o jeden krok viac). Druhá vec, ktorá sa môže stať je, že sa bude prehľadávať o jednu hĺbku navyše, a teda sa časová zložitosť ešte zhorší (ako sa to stalo pri vstupnom súbore, ktorému to trvalo príliš dlho, a preto som ho vypol).

Žiadne ďalšie úpravy alebo optimalizácie mi nenapadnú.

Spôsob, ako by sa dalo urýchliť riešenie je, ak by sme poznali potrebný počet krokov, a teda by sme vedeli preskočiť všetky hĺbky až do tejto. Takto by sa ušetril čas, ktorý reálne na výsledný výstup nemá žiadny vplyv.

**Návod na spustenie:**

Po spustení programu treba zadať cestu ku vstupu, alebo len vstupný súbor. Stačí zadať aj bez koncovky .txt, program automaticky vyhodnotí, či ju treba pridať alebo nie.

Následne je možné vstúpiť do DEBUG módu, teda po každom novo-vytvorenom uzle bude vyžadovaná interakcia od užívateľa (enter).

Ďalej je možné spustiť neefektívny, ale časovo rýchly algoritmus, ktorý si pamätá všetky navštívené stavy (pre kontrolu).

Na koniec sa zadá názov výstupného súboru, alebo c pre výpis do konzoly.

Formát vstupných súborov:

Prvý riadok obsahuje veľkosť tabuľky – počet stĺpcov a riadkov oddelených medzerou.

Následne ide n áut, stĺpec, riadok, veľkosť, otočenie (u – up, d – down, r – right, l – left) oddelených medzerou.