Prehľadávanie stavového priestoru

Zadanie č. 1

Toto zadanie sa venuje niekoľkým základným algoritmom prehľadávania stavového priestoru. Základná štruktúra algoritmu by mala byť podľa možností vyjadrená jednou funkciou.

V dokumentácii je potrebné uviesť použitý algoritmus a opísať vlastnosti tohto algoritmu – teoretické aj skutočné – koľko naozaj rozvíja uzlov, koľko mu to trvá a aké riešenie nájde. Použite *aspoň* dva rôzne príklady na hľadanie – vzdialenosť do cieľa napríklad 6 a 10 krokov. Porovnajte dosiahnutý výsledok pri zámene začiatočného a koncového uzla, ak to má zmysel (problém 2).

Základom hodnotenia je funkčnosť programu podľa špecifikácie a osobitná dokumentácia k programu. Dokumentácia musí obsahovať:

- riešený problém – názov špecifického zadania (a meno autora!)
- stručný opis riešenia a jeho podstatných častí
    - reprezentáciu údajov problému, použitý ***konkrétny*** algoritmus
- spôsob testovania a zhodnotenie riešenia
    - možnosti rozšírenia, prípadné optimalizácie, výhody a nevýhody konkrétnej implementácie (aj či sú závislé alebo nezávislé na programovacom prostredí)
    - porovnanie vlastností použitých metód pre rôznu dĺžku riešenia (u eulerovho koňa rôzne veľkosti šachovnice)

Do systému AIS je treba odovzdať elektronickú verziu dokumentácie plus okomentované zdrojové kódy. Najlepšie zabalené do jedného zip súboru.

Ďalej sa hodnotí:

- efektívna a prehľadná implementácia algoritmu (včítane komentárov)
- vhodné otestovanie činnosti algoritmu
- možnosť spracovania hlavolamu m*n (iný než 3x3, problém 2)

Nezabudnite, že ***hlavným výstupom programu*** je postupnosť krokov od začiatku k cieľu! (má byť reprezentovaná postupnosťou operátorov)

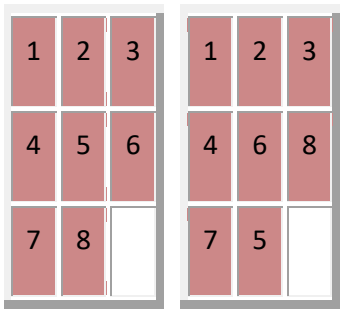
K programu je potrebné dodať sadu testovacích príkladov! Ak je program interpretovaný, môžu byť v tom istom súbore ako zdrojový kód.

Definovanie problému 2

Našou úlohou je nájsť riešenie 8-hlavolamu. Hlavolam je zložený z 8 očíslovaných políčok a jedného prázdneho miesta. Políčka je možné presúvať hore, dole, vľavo alebo vpravo, ale len ak je tým smerom medzera. Je vždy daná nejaká východisková a nejaká cieľová pozícia a je potrebné nájsť postupnosť krokov, ktoré vedú z jednej pozície do druhej.

Príkladom môže byť nasledovná začiatočná a koncová pozícia:

Začiatok: **Koniec:**



Im zodpovedajúca postupnosť krokov je: **VPRAVO, DOLE, VĽAVO, HORE**.

Implementácia 2

Keď chceme túto úlohu riešiť algoritmami prehľadávania stavového priestoru, musíme si konkretizovať niektoré pojmy:

STAV

Stav predstavuje aktuálne rozloženie políčok. Počiatočný stav môžeme zapísať napríklad

```
((1 2 3)(4 5 6)(7 8 m))
```

alebo

```
(1 2 3 4 5 6 7 8 m)
```

Každý zápis má svoje výhody a nevýhody. Prvý umožňuje (všeobecnejšie) spracovať ľubovoľný hlavolam rozmerov m*n, druhý má jednoduchšiu realizáciu operátorov.

Vstupom algoritmov sú práve dva stavy: začiatočný a cieľový. Vstupom programu však môže byť aj ďalšia informácia, napríklad výber heuristiky.

OPERÁTORY

Operátory sú len štyri:

```
VPRAVO, DOLE, VLAVO a HORE
```

Operátor má jednoduchú úlohu – dostane nejaký stav a ak je to možné, vráti nový stav. Ak operátor na vstupný stav nie je možné použiť, výstup nie je definovaný. V konkrétnej implementácii je potrebné výstup buď vhodne dodefinovať, alebo zabrániť volaniu nepoužiteľného operátora. **Všetky operátory pre tento problém majú rovnakú váhu.**

Príklad použitia operátora DOLE:

Vstup:

```
((1 2 3)(4 5 6)(7 8 m))
```

Výstup:

```
((1 2 3)(4 5 m)(7 8 6))
```

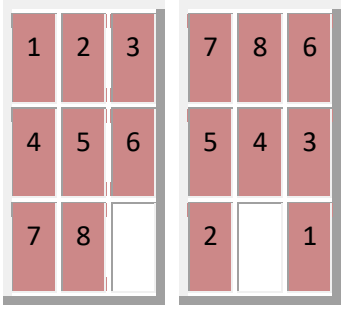
HEURISTICKÁ FUNKCIA

Niektoré z algoritmov potrebujú k svojej činnosti dodatočnú informáciu o riešenom probléme, presnejšie odhad vzdialenosti od cieľového stavu. Pre náš problém ich existuje niekoľko, môžeme použiť napríklad

1. Počet políčok, ktoré nie sú na svojom mieste
2. Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
3. Kombinácia predchádzajúcich odhadov

Tieto odhady majú navyše mierne odlišné vlastnosti podľa toho, či medzi políčka počítame alebo nepočítame aj medzeru. Započítavať medzeru však nie je vhodné, lebo taká heuristika nadhodnocuje počet krokov do cieľa.

Príklad:
Heuristika č. 2, bez medzery, odhaduje vzdialenosť nasledujúcich dvoch stavov na




```
4 + 3 + 1 + 1 + 1 + 1 + 2 + 2 = 15
```

UZOL

Stav predstavuje nejaký bod v stavovom priestore. My však od algoritmov požadujeme, aby nám ukázali cestu. Preto musíme zo stavového priestoru vytvoriť graf, najlepšie priamo strom. Našťastie to nie je zložitá úloha. Stavy jednoducho nahradíme uzlami.

Čo obsahuje typický uzol?
Musí minimálne obsahovať

- **STAV** (to, čo uzol reprezentuje) a
- **ODKAZ NA PREDCHODCU** (pre nás zaujímavá hrana grafu, reprezentovaná čo najefektívnejšie).

Okrem toho môže obsahovať ďalšie informácie, ako

- **POSLEDNE POUŽITÝ OPERÁTOR**
- **PREDCHÁDZAJÚCE OPERÁTORY**
- **HĹBKA UZLA**
- **CENA PREJDENEJ CESTY**
- **ODHAD CENY CESTY DO CIEĽA**
- Iné vhodné informácie o uzle

Uzol by však nemal obsahovať údaje, ktoré sú nadbytočné a príslušný algoritmus ich nepotrebuje. Pri zložitých úlohách sa generuje veľké množstvo uzlov a každý zbytočný bajt v uzle dokáže spotrebovať množstvo pamäti a znížiť rozsah prehľadávania algoritmu. Nedostatok informácií môže zase extrémne zvýšiť časové nároky algoritmu. ***Použité údaje zdôvodnite***.

ALGORITMUS

Každé zadanie používa svoj algoritmus, ale algoritmy majú mnohé spoločné črty. Každý z nich potrebuje udržiavať informácie o uzloch, ktoré už kompletne spracoval a aj o uzloch, ktoré už vygeneroval, ale zatiaľ sa nedostali na spracovanie. Algoritmy majú tendenciu generovať množstvo stavov, ktoré už boli raz vygenerované. S týmto problémom je tiež potrebné sa vhodne vysporiadať, zvlášť u algoritmov, kde rovnaký stav neznamená rovnako dobrý uzol.

Činnosť nasledujúcich algoritmov sa dá z implementačného hľadiska opísať nasledujúcimi všeobecnými krokmi:

1. Vytvor počiatočný uzol a umiestni ho medzi vytvorené a zatiaľ nespracované uzly
2. Ak neexistuje žiadny vytvorený a zatiaľ nespracovaný uzol, skonči s neúspechom – riešenie neexistuje
3. Vyber najvhodnejší uzol z vytvorených a zatiaľ nespracovaných, označ ho aktuálny
4. Ak tento uzol predstavuje cieľový stav, skonči s úspechom – vypíš riešenie
5. Vytvor nasledovníkov aktuálneho uzla a zaraď ho medzi spracované uzly
6. Vytrieď nasledovníkov a ulož ich medzi vytvorené a zatiaľ nespracované
7. Choď na krok 2.

Uvedené kroky sú len všeobecné a pre jednotlivé algoritmy ich treba ešte vždy rôzne upravovať a optimalizovať.

Problém 2. Použite A^* algoritmus, porovnajte výsledky heuristík 1. a 2.