Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Dátové štruktúry a algoritmy

Zadanie 2 - Prehľadávanie stavového priestoru
A) Bláznivá križovatka DFS a BFS

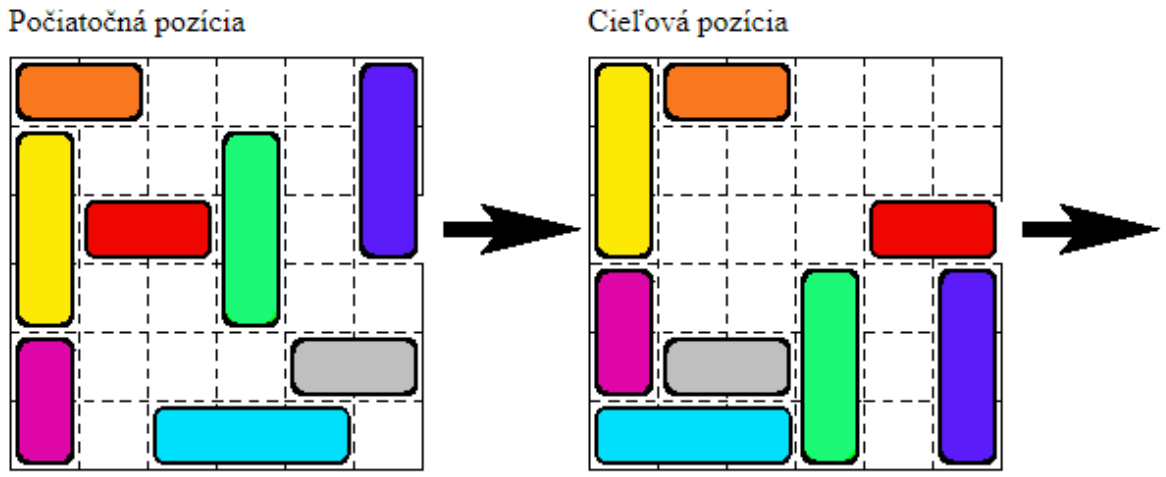
Jakub Hrnčár
AIS ID: 110 800, ak. rok 2021/2022

1. Úvod do zadania

Toto zadanie predstavuje úvod do algoritmov umelej inteligencie. Jej základom je prehľadávanie stavového priestoru, ktoré sa dá vykonať pomocou mnohých algoritmov, napríklad prehľadávanie do hĺbky a šírky. Tieto algoritmy predstavujú úplný základ a majú veľmi široké využitie, aj vďaka pomerne ľahkej implementácií.

V probléme “bláznivá križovatka” stavový priestor predstavuje simulovaná križovatka, na ktorej sa pohybujú autá a nevedia cez ňu prejst. Zameriavam sa len na jedno auto - červené. Mojou úlohou je navrhnúť taký program, ktorý toto auto dostane skrz križovatku, z ľavej strany do pravej. Prekážku mu robia iné autá, ktoré môžu byť otočené aj zvislo. Problém riešim na 6x6 mriežke, avšak je riešiteľný aj na iných veľkostiach.

Zadanie bolo vypracované v jazyku Python 3.9, v prostredí PyCharm Ultimate s hardvérom:
CPU: Intel i3-10100F 4 jadrá 8 vlákien
RAM: 16 GB DDR4 2666 MHz CL 16



Obr. 1 Počiatočná a cieľová pozícia hlavolamu Bláznivá križovatka.

2. Stručný opis algoritmu

Algoritmy sú použité dva: prehľadávanie do hĺbky a do šírky.
Pseudokód pre hľadanie do šírky (BFS):

```
1  procedure BFS(G, root) is
 2      let Q be a queue
 3      label root as explored
 4      Q.enqueue(root)
 5      while Q is not empty do
 6          v := Q.dequeue()
 7          if v is the goal then
 8              return v
 9          for all edges from v to w in G.adjacentEdges(v) do
```

```
10              if w is not labeled as explored then
11                  label w as explored
12                  Q.enqueue(w)
```

Pseudokód pre hľadanie do hĺbky (DFS):

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

Realisticky, kód týchto algoritmov je veľmi podobný, líši sa najmä v prístupe uchovávania zoznamu uzlov, ktoré ešte treba navštíviť.
BFS: pamäť je fronta Last in First Out (LIFO). Každý nový vytvorený uzol sa uloží na koniec, spracováva sa prvý uzol.
DFS: pamäť je zásobník First in First Out (FIFO). Každý nový uzol sa uloží na začiatok a prvý uzol sa spracováva.

Zadanie som vypracoval v objektovo-orientovanej paradigme, avšak pre jednoduchú organizáciu som hlavné časti programu vložil do funkcie.
Sú tu 2 objekty: Vehicle a Node. Vehicle predstavuje jedno auto v križovatke, má atribúty polohu, veľkosť a orientáciu, pričom na sebe dokáže vykonávať jednoduché metódy - operátory vľavo, vpravo, hore, dole. Node je objekt prestavujúci uzol v strome - stavovom priestore. Ukladá si križovatku vo forme 2D poľa, objekty áut v nej, rodiča a deti; má jedinú metódu, ktorá dokáže z objektov áut zostaviť 2D vizualizáciu križovatky.

Spoločný algoritmus pre BFS aj DFS je nasledovný:
Pozriem sa na pole áut v uzle, prechádzam nimi cez *for* cyklus. Zistím možný pohyb s autom na základe pozície, pričom kontrolujem skrz podmienky hranice križovatky a kolízie s autami. Ak sa dokáže auto pohnúť o viac ako jedno políčko, zistím o koľko je maximálny pohyb:

```
max_move = 1
while car.x - 1 + max_move + car.w < 6 and
current.state[car.y-1][car.x - 1 + max_move + car.w] == 0:
   max_move += 1
```

Následne pre každý pohyb generujem nový stav, ale iba v prípade, že daný stav ešte nebol vygenerovaný - vygenerované stavy ukladám do globálneho pola *states*:

```
for i in range(1, max_move + 1):
   child = Node(current, copy.deepcopy(current.cars),
current.layer + 1)
   child.cars[(car.number - 1)].right(i)
```

```
    child.log = ["right", i, child.cars[(car.number - 1)].color]
    child.set()
    if child.state not in states:
        states.append(child.state)
```

Toto je konkrétny príklad pre operáciu Vpravo, avšak ostatné operácie sú veľmi podobné.

Rozdielne časti pre oba algoritmy spočívajú v prístupe ku polu *edge*, kde sú uložené stavy na spracovanie.
BFS: vyberie sa prvý prvok v poli *edge*, pri vygenerovaní nového stavu sa nový stav pridá okamžite na koniec *edge*. Pri prechode medzi stavmi sa prvý stav odstráni.
DFS: vyberie sa prvý prvok v poli *edge*, pri vygenerovaní nového stavu sa nový stav pridá na koniec pomocného poľa *children*, ktoré pri prechode medzi stavmi nahradí pôvodný prvý prvok, z ktorého vychádzali; teda sa dostanú nové stavy na začiatok *edge.*

3. Testovanie programu

0 - prázdne miesto
1 - červené auto - cieľ
2+ - ostatné autá

a) Ľahké - 6 áut:

1.
[0, 0, 0, 0, 0, 0]
[0, 5, 0, 0, 0, 0]
[0, 5, 0, 3, 3, 3]
[0, 5, 0, 0, 6, 2]
[0, 4, 4, 0, 6, 2]
[0, 0, 1, 1, 6, 2]
BFS: number of steps: 6 Number of states: 154
BFS time: 0.06781554222106934 sec

DFS time: 0.021965980529785156 sec
DFS: number of steps: 34 Number of states: 132

2.
[0, 0, 0, 0, 0, 5]
[0, 3, 3, 0, 0, 5]
[0, 0, 0, 2, 0, 5]
[1, 1, 6, 2, 0, 0]
[0, 0, 6, 2, 0, 0]
[0, 0, 6, 4, 4, 4]
BFS: number of steps: 4 Number of states: 149
BFS time: 0.028948307037353516 sec

DFS time: 0.059812068939208984 sec
DFS: number of steps: 82 Number of states: 366

b) Stredné - 8 áut:
1. vzorové
[4, 4, 0, 0, 0, 2]
[5, 0, 0, 3, 0, 2]
[5, 1, 1, 3, 0, 2]
[5, 0, 0, 3, 0, 0]
[6, 0, 0, 0, 7, 7]
[6, 0, 8, 8, 8, 0]
BFS: number of steps: 8 Number of states: 1064
BFS time: 1.1854262351989746 sec

DFS time: 0.20252776145935059 sec
DFS: number of steps: 190 Number of states: 901
2.
[5, 0, 8, 0, 0, 0]
[5, 0, 8, 0, 0, 0]
[5, 0, 8, 2, 2, 0]
[0, 3, 6, 6, 4, 0]
[0, 3, 1, 1, 4, 0]
[0, 3, 0, 7, 7, 7]
BFS: number of steps: 9 Number of states: 877
BFS time: 0.8328008651733398 sec

DFS time: 0.3127145767211914 sec
DFS: number of steps: 171 Number of states: 863

c) Ťažké - 12 áut:

1.
[9, 9, 09, 00, 00, 2]
[7, 7, 10, 10, 10, 2]
[0, 1, 1, 03, 05, 04]
[0, 0, 6, 03, 05, 04]
[0, 0, 6, 12, 12, 12]
[0, 0, 11, 11, 08, 8]
BFS: number of steps: 18 Number of states: 1265
BFS time: 1.7189135551452637 sec

DFS time: 0.21194148063659668 sec
DFS: number of steps: 178 Number of states: 823

2.
[0, 2, 2, 6, 10, 00]
[0, 0, 0, 6, 10, 05]
[0, 1, 1, 4, 10, 05]
[9, 0, 8, 4, 12, 12]
[9, 7, 8, 0, 03, 03]
[9, 7, 11, 11, 0, 0]
BFS: number of steps: 21 Number of states: 13187
BFS time: 125.33270788192749 sec

DFS time: 4.1528000831604 sec
DFS: number of steps: 1024 Number of states: 5483

Z testovania veľmi jasne vyplýva niekoľko informácií:

- pri zložitom probléme sa oba algoritmy dostávajú do exponenciálnej zložitosti
- najväčšia limitácia je dátová štruktúra použitá na uchovávanie už nájdených stavov - vylepšiť by dala hash tabuľkou, momentálne tam je obyčajné pole
- BFS vytvára vždy viac stavov ako DFS, avšak konečné riešenie je na omnoho menší počet krokov
- problém bláznivej križovatky je veľmi volatilný - aj veľký počet áut môže znamenať jednoduchý hlavolam
- výsledné kroky DFS rastú oveľa rýchlejšie ako výsledné kroky BFS
- limitácia 6x6 tabuľky je pravdepodobne 13 áut

4. Záver

Program je funkčný a potvrdzuje poznatky z prednášky:
BFS je náročnejší avšak nájde optimálne riešenie - na menej a bez zbytočných krokov. DFS síce zbehne rýchlo, avšak vytvorí obrovské množstvo nepotrebných alebo neefektívnych stavov (napr. stavy s posunom o 1). Ďalej to potvrdzuje, že iteratívne-prehlbujúce DFS by bola naozaj “zlatá stredná cesta”, avšak skúmanie tohto algoritmu nebolo predmetom tohto zadania.

Myslím si, že mi toto zadanie dalo veľmi slušný základ do fungovania umelej inteligencie, pochopil som základným princípom prehľadávania stavého priestoru.