Dávid Pukanec
xpukanecd email: ikarus12111@gmail.com

Umelá inteligencia

Zadanie 2, F

Problém 3. Algoritmom slepého prehľadávania (do hĺbky) je možné nájsť (všetky) riešenia (v bežných výpočtových – čas a pamäť – podmienkach PC) iba pri šachovniciach do veľkosti 6x6, max 7x7. Implementujte tento algoritmus pre šachovnice s rozmermi 5x5 a 6x6 a skúste nájsť prvých 5 riešení pre každú šachovnicu tak, že pre šachovnicu 5x5 aj 6x6 si vyberte náhodne 5 východzích bodov (spolu teda 10 východzích bodov) s tým, že jeden z týchto bodov je (pre každú šachovnicu) ľavý dolný roh a pre každý z týchto bodov nájdite (skúste nájsť) prvé riešenie. V prípade, že ho v stanovenom limite nenájdete, signalizujte neúspešné hľadanie. V diskusii potom analyzujte pozorované výsledky.

Opis riešenia

Riešením je prehľadávanie do hĺbky a následné ukladanie prejdených údajov do stacku. Algoritmus sa začína inicializáciou prvého(počiatočného) políčka šachovnice. Jedno políčko šachovnice je reprezentované triedou v ktorej je uložená pozícia políčka, cesta akou sa k tomuto políčku jazdec dostal a v koľkom ťahu bolo navštívené. Takéto políčko sa vloží do stacku.

Následne sa začne cyklus prehľadávania.

1. Skontroluje sa či je rada prázdna, ak áno cyklus sa zastaví(štartovná pozícia nemá riešenie).
2. Zo zásobníka sa popne posledný pridaný node(v prvom prípade to je ten ktorý sme tam vložili na začiatku).
3. Skontroluje sa, či poradie jeho navštívenia sa nerovná N*N, pričom N je rozmer šachovnice.
4. Ak sa rovná, znamená to, že jazdec navštívil všetky políčka a máme riešenie.
5. Ak sa nerovná, z tohto políčka sa následne nájdu všetky možne políčka, kde jazdec môže skočiť a tie sa pridajú do zásobníka.

Reprezentácia údajov

Cestu ktorú jazdec prešiel reprezentujem pomocou premennej ***currentPath.*** Premenná je list menších listov, v ktorých sú uložené súradnice [x,y].

Trieda ***node*** slúži na uchovávanie informácii a reprezentuje jedno políčko na šachovnici. Ukladá sa do nej cesta cez ktorú jazdec prešiel. Táto cesta slúži na overenie pri prehľadávaní iných políčok ako aj na výpis finálnej cesty

Zásobník, ***stack***, v ktorom sú uložené jednotlivé body som taktiež reprezentoval listom. Do neho som vkladal triedu ***node***. List bol veľmi užitočný keďže jednoducho sa ku nemu pripájajú a taktiež popuju údaje.

Dávid Pukanec
xpukanecd email: ikarus12111@gmail.com

Možne smery pohybu som reprezentoval tak že som si vytvoril všetky vychýlenia kam sa môže jazdec z aktuálnej pozície pohnúť

```
possibleMoves = [[1, 2], [1, -2], [-1, 2], [-1, -2],
                 [2, 1], [2, -1], [-2, 1], [-2, -1]]
```

Dôležité funkcie

randomMove()

```
63  def randomMove(node, width):
64      global moveCounter
65      global stack
66
67      # Picking every move from the list of moves.
68      for move in possibleMoves:
69          yOffset = node.yCor + move[1]
70          xOffset = node.xCor + move[0]
71          # Controlling borders of chessboard
72          if yOffset < 0 or yOffset > width - 1 or xOffset < 0 or xOffset > width - 1:
73              continue
74          # Checking if the node is not already part of the path
75          if [xOffset,yOffset] in node.currentPath:
76              continue
77          # If none of the condition above are fulfilled then it must be legal move
78          if [xOffset,yOffset] not in node.currentPath:
79              newNode = MyNode(xOffset, yOffset, node.visitable + 1)
80              newNode.currentPath = node.currentPath.copy()
81              newNode.currentPath.append([xOffset,yOffset])
82              stack.append(newNode)
```

Úlohou tejto funkcie je nájsť všetky možne políčka, na ktoré možné skočiť z node. Funkcia na vstupe dostane políčko, ktoré ide analyzovať(node) a šírku šachovnice(width).

V hlavnom cykle zoberie každý offset a pozrie či:

- nie je mimo hranice šachovnice
- nebolo už prehľadané

Ak nebola spustená ani jedna z podmienok, znamená to že toto políčko/ťah je možný, preto sa pridá na koniec rádu.

Dávid Pukanec
xpukanecd
email: ikarus12111@gmail.com

driveF()

V tejto funkcii sa nachádza hlavný cyklus ktorý spúšťa každé prehľadávanie. Taktiež sa tu nachádzajú hlavne podmienky podľa ktorých sa vie či sa cesta našla alebo nie

```
while True:

    # If stack is empty
    if not stack:
        print("Not a valid root")
        break

    # If not it pops the last node
    else:
        nextNode = stack.pop()
    # Checks if I did not finished
    if nextNode.visitable == width * width:
        print("I've done it")
        print(nextNode.currentPath, end= '\n')
        return nextNode.currentPath
    randomMove(nextNode,width)
return -1
```

Testovanie

Zadanie som testoval pomocou textových súborov kde som si vždy navolil rôzne počiatočné body ale aj rôzny čas výpočtu. Pochopiteľne, nie každé riešenie sa stihlo v čas a niektoré políčka na 5x5 šachovnici ani riešenie nemajú. Niektoré z testovacích súborov som poslal spolu s riešením.

Z testov vypláva, že pre niektoré štartovacie pozície, pri šachovnici 5x5, neexistuje riešenie. Dôvodom je, že na šachovniciach s neparným rozmerom strany, má jedna farba vždy o jedno políčko viac a na tejto farbe aj treba začať.

Zhodnotenie

Riešenie som realizoval pomocou prehľadávania do hĺbky a ukladania aktuálny nodov do zásobníka. Algoritmus by sa dal jednoducho vylepšiť napríklad vyberania políčok podľa počtu susedov do ktorých sa dá z nich isť. Ale takéto riešenie je mimo tohto zadania. Taktiež môj algoritmus ma veľkú nevýhodu keď začiatočné body sú napr. [0,0], pretože tieto body sa prehľadávajú ako prvé, uložia sa do zásobníka skôr ako body ktoré sa prehľadávajú neskôr. Z tohto dôvodu sa výberu zo zásobníka neskôr.

Podľa môjho názoru nezávisí na prostredí, v ktorom je program naprogramovaný, pretože všetky využite prostriedky sa dajú ľahko implementovať aj v iných prostrediach.