UI first assignment

Peter Biely, cvicenie stvrtok o 12:00

c)

Problém 2. Použite algoritmus obojsmerného hľadania.

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:

1	2	3
4	5	6
7	8	

Koniec:

1	2	3
4	6	8
7	5	

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

Riesenie som implementoval v jazyku python. Kod mam rozdeleny na pomocne funkcie a jeden main loop v ktorom volam pomocne funkcie a riesim problem.

Prva cast kodu je classa Node ktoru pouzivam na ukladanie postupu algoritmom a mojich stavo po pohyboch. Mam v nej ulozeny stav a kroky ktore sa uz vykonali.

```
You, 8 minutes ago | 1 author (You)
class Node:
    def __init__(self, state, steps):
        self.state = state
        self.steps = list(steps)
```

Na zadanie prvotneho stavu a konecneho stavu na zaciatku filu su dve premenne nazvane start and end a potom na specifikaciu poctu stlpcov a riadkov su dalsie dve premenne col a row.

```
You, 13 minutes ago | 1 author (You)
start = "4 X 3 2 1 8 7 6 5"
end = "1 2 3 4 5 6 7 8 X"
row=3
col=3
```

Prva pomocna funckia je directions. Directions je pomocna funkcia ktora ma na startosti zistit kam sa moze posunut X (Blank), tato pomocna funkcia je potrebna pre pohyb.

```
def directions(state):
    state = state.split(" ")
    possibleDirections = []
    xIndex = state.index("X")
    #movement up
    if ((xIndex - 3) > 0 ):
        possibleDirections.append("Hore")
    #movement down
    if ((xIndex + 3) < len(state)):
        possibleDirections.append("Dole")
    #movement lef
    lavaStrana = 0
    for x in range(col):
        if (xIndex == row*x):
            lavaStrana = lavaStrana + 1
    if (lavaStrana == 0):
        possibleDirections.append("Vlavo")
    #movement right
    pravaStrana = 0
    for x in range(col):
        if (xIndex == (row*x)-1) or xIndex == (len(state) - 1):
            pravaStrana = pravaStrana + 1
    if (pravaStrana == 0):
        possibleDirections.append("Vpravo")
    state = " ".join(state)
    return possibleDirections;
```

Ake moznosti ma X na pohyb zistujem pomocou indexou. Na pohyb hore a dole pozeram ci ked sa pohnem hore alebo dole nevyjdem z velkosti pola a tym padom by som sa dostal “out of bounds” a left a right pozeram podla toho ci sa nachadza bud v uplne lavom stlpci pre pohyb do lava a uplne v pravom pre pohyb do prava.

Dalsia pomocna funkcia je funckia s nazvom movement tato funkcia zodpoveda za pohyb v tejto funkcie pre kazdu node ktoru do nej poslem si naprv zistime pomocou Directions do akych smerov sa moze hybat a potom nasledne podla tohto volam 4 pomocne funkcie pre pohyb kazdym smerom.

```
def movement(node, nodes):
    dir =  directions(node.state)
    for x in dir:
        if (x == "Hore"):
            childNode1 = Node(node.state, node.steps)
            moveUp(childNode1)
            childNode1.steps.append("Hore")
            nodes.append(childNode1)
        if (x == "Dole"):
            childNode2 = Node(node.state, node.steps)
            moveDown(childNode2)
            childNode2.steps.append("Dole")
            nodes.append(childNode2)
        if (x == "Vpravo"):
            childNode3 = Node(node.state, node.steps)
            moveRight(childNode3)
            childNode3.steps.append("Vpravo")
            nodes.append(childNode3)
        if (x == "Vlavo"):
            childNode4 = Node(node.state, node.steps)
            moveLeft(childNode4)
            childNode4.steps.append("Vlavo")
            nodes.append(childNode4)
```

Pomocne funkcie moveRight, moveLeft , moveDown, moveUp su fukcie ktore zabezpecuju pohyb funguju tak ze najpr si ziskam index X a podla neho zistim ze kam sa mam pohnut ked vykonam ten pohyb , potom do pomocnej premennej zapisem cislo z destinacie X a pomocou tejto pomocnej premennej vymenim X za Cislo na mieste kam sa X pohlo. Jediny rozdiel medzi funkciami je len manipulacia s indexami, ta sa meni podla smeru pohybu.

```
def moveRight(node):
    state = node.state.split(" ")
    indexOfX = state.index("X")
    tempNumber = state[indexOfX + 1]
    state[indexOfX + 1]  = state[indexOfX]
    state[indexOfX] = tempNumber
    node.state = " ".join(state)

def moveLeft(node):
    state = node.state.split(" ")
    indexOfX = state.index("X")
    tempNumber = state[indexOfX - 1]
    state[indexOfX -1] = state[indexOfX]
    state[indexOfX] = tempNumber
    node.state = " ".join(state)

def moveUp(node):
    state = node.state.split(" ")
    indexOfX = state.index("X")
    dif = indexOfX - row
    tempNumber = state[indexOfX - row]
    state[dif] = state[indexOfX]
    state[indexOfX] = tempNumber
    node.state = " ".join(state)

def moveDown(node):
    state = node.state.split(" ")
    indexOfX = state.index("X")
    tempNumber = state[indexOfX + row]
    state[indexOfX + row] = state[indexOfX]
    state[indexOfX] = tempNumber
    node.state = " ".join(state)
```

Potom tam mam inicializacie premennych a pomocnych premennych a array a prvych nodov.

```
startNode = Node(start, [])
endNode = Node(end, [])
startNodes = []
tempStartNodes = []
endNodes = []
tempEndNodes = []
success = 0
layers = 0
endSteps = []
startNodes.append(startNode)
endNodes.append(endNode)
```

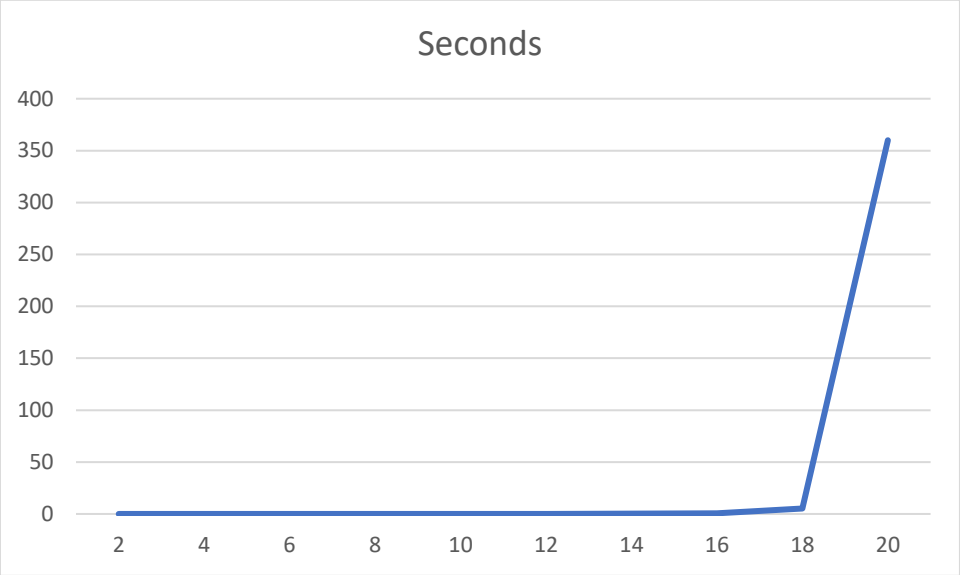
A nakoniec je moj main loop v ktorom volam vsetky funkcie a zostavujem posledny string pohybou ktory nakoniec vypisem, jedinna zaujimava vec na tomto loope je zostavenie posledneho listu pohybov ktory zostavujem tak ze najprv appendnem pohyby smerom od startu az k najdenej zhode a potom appenduje pohyb ktory sa vykonal od koncoveho stavu ale ten apendujem odzadu lebo som isiel odzadu a este otoceny lebo z pohladu startovneho stavu su tie pohyby vykonane opacne ako z pohladu kocoveho stavu.

Nakoniec len vypisem vysledok.

```
while (success == 0 or layers == 6):
    for x in startNodes:
        movement(x, tempStartNodes)

    for y in endNodes:
        movement(y, tempEndNodes)

    for x in tempStartNodes:
        for y in tempEndNodes:
            if (x.state == end and success == 0):
                success = 1
                for j in x.steps:
                    endSteps.append(j)
            if (x.state == y.state and success == 0):
                success = 1
                for j in x.steps:
                    endSteps.append(j)
                for i in reversed(y.steps):
                    if (i == "Vlavo"):
                        endSteps.append("Vpravo")
                    elif (i == "Vpravo"):
                        endSteps.append("Vlavo")
                    elif (i == "Hore"):
                        endSteps.append("Dole")
                    elif (i == "Dole"):
                        endSteps.append("Hore")
        for z in endNodes:
            if (z.state == x.state and success == 0):
                success = 1
                for j in x.steps:
                    endSteps.append(j)
                for i in reversed(z.steps):
                    if (i == "Vlavo"):
                        endSteps.append("Vpravo")
                    elif (i == "Vpravo"):
                        endSteps.append("Vlavo")
                    elif (i == "Hore"):
                        endSteps.append("Dole")
                    elif (i == "Dole"):
                        endSteps.append("Hore")

    startNodes = tempStartNodes
    endNodes = tempEndNodes
    tempStartNodes = []
    tempEndNodes = []
    layers = layers + 1 #Prevention of infinite loop

print(endSteps)
```

Testovanie:

3x3 testovanie na pocet pokynov casova zlozitost



Exponencialne stupa trvanie programu zo stupanim poctu instrukcii potrebnych k zhode.

Casova zlozitost stupa exponencialne s poctom instrukcii na velkosti puzzle zalezi menej ale vacsie puzzle napriklad 4x4 je viacej nachylne na potrebu viacej instrukcii na dostaniu sa k vysledku.

Testovanie inych rozmerov:

4x2 zaciatok: “X 1 2 3 4 5 6 7” koniec: “3 2 5 4 7 6 1 X”

Cas: 0.1599 sec

Instrukcie: ['Dole', 'Vpravo', 'Hore', 'Vlavo', 'Dole', 'Dole', 'Vpravo', 'Hore', 'Vlavo', 'Dole', 'Vpravo', 'Dole', 'Vlavo', 'Hore', 'Vpravo', 'Dole']

4x3 zaciatok: “X 1 2 3 4 5 6 7 8 9 10 11” koniec: “1 2 X 3 4 5 6 7 8 9 10 11”

Cas: 0.0

Instrukcie: Vpravo Vpravo

4x3 a 5x2 mi zbehli len velmi elementarne zmeny v podobe 2-3 krokov ostatne trvali prilis dlho ci to bolo pythonom alebo optimalizaciou mojho algoritmu ci uz hardwearom je mozne ze to bolo vsetkymi vyssie uvedenemi. Zaver je ale taky ze casova zlozitost pre toto puzzle a s pouzitim bidirectional algoritmu na najdenie vysledku je exponencialna pri kazdej velkosti.

4x3 a 5x2 ine pocti instrukcia ako 2 – 3 trvali pribline 3-5 minut a potom som bud program zastavil manualne alebo sa trigerol moj check pre prilis velke mnozstva poschodi a vytvorenych nodov v “strome”.