Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Umelá Inteligencia

Zadanie č.1

d) 8-Hlavolam – lačné hľadanie

Akademický rok 2022/2023

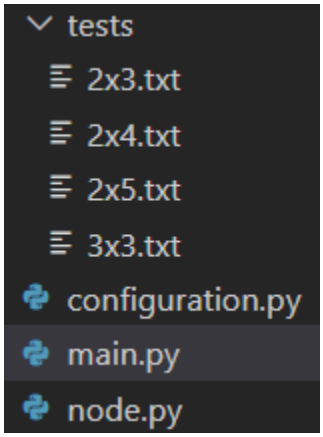
Meno: Irina Makarova **Dátum:** 11.10.2022
Cvičiaci: Ing. Martin Komák, PhD.

Opis zadania

8-hlavolam predstavuje m*n maticu, kde je jedno políčko prázdne a ostatné sú očíslované od 1 do m*n-1. Na začiatku sú dané dve konfigurácie – počiatočná a cieľová. Problém spočíva v nájdení postupnosti operácií, po vykonaní ktorých sa dostaneme do cieľovej konfigurácie. Na vyriešenie hlavolamu sa mal použiť algoritmus lačného hľadania, ktorý v každom kroku vyberá konfiguráciu s najmenšou hodnotou podľa heuristickej funkcie.

Organizácia projektu

Program sa spúšťa zo súboru *main.py*, ktorý zároveň obsahuje aj implementáciu algoritmu lačného prehľadávania. V súbore *node.py* sa nachádza trieda *Node* reprezentujúca jednotlivé uzly v strome stavového priestoru; jej súčasťou je aj funkcia *generateChildren()* na vytvorenie dcérskych uzlov. Súbor *configuration.py* obsahuje triedu *Configuration* slúžiacu na ukladanie a vytváranie nových konfigurácií; taktiež sa ňom nachádzajú funkcie *heuristic_2()* a *heuristic_1()* na počítanie heuristickej hodnoty konfigurácie. V adresári *tests* sú uložené konfigurácie na testovanie programu, pričom prvé číslo v názve súboru je počet riadkov a druhé predstavuje počet stĺpcov konfigurácie. Jeden riadok testovacieho súboru obsahuje vstupnú a cieľovú konfigurácie, ktoré sú oddelené podčiarkovníkom. Samotná konfigurácia je uložená ako reťazec znakov, kde každý znak je číslo matice. Matica sa na reťazec prepisuje zadávaním hodnôt v smere zhora dole a zľava doprava.



Slovný opis algoritmu

Reťazce zo vstupného súboru sa pretransformujú na dvojrozmerné pole. Postupnosť znakov pred podčiarkovníkom je počiatočná konfigurácia, za ním – cieľová. Tieto polia spolu s použitou heuristikou potom slúžia ako vstup pre hlavnú funkciu *greedySearch()*. V nej sa vytvoria dve dátové štruktúry – pole na ukladanie a vyhľadávanie vytvorených konfigurácií a prioritný rad obsahujúci vygenerované a ešte nerozvité uzly zoradené podľa heuristickej hodnoty.

```
def greedySearch(initial, goal, heuristic):
    allConfigurations = list() #vsetky vytvorene konfiguracie
    initState = Configuration(len(initial), len(initial[0]))
    initState.configuration = initial #ak zadava pouzivatel
    initState.findZero() #ak zadava pouzivatel
    currentNode = Node(initState, initState.chooseHeuristic(heuristic, goal))

    generatedNotExpandedNodes = PriorityQueue()
    generatedNotExpandedNodes.put((currentNode.heuristic, currentNode)) #vlozi
    allConfigurations.append(currentNode.state.configuration) #ulozime pociato
```

Do poľa sa uloží počiatočná konfigurácia a do rady sa vloží koreňový uzol. Ďalšie kroky prebiehajú v cykle while, ktorý sa ukončí buď ak sa nájde riešenie, alebo sa prehľadá celý prioritný rad a k riešeniu sa nebude dať prísť. V každej slučke cyklu sa pre uzol, ktorý momentálne rozvíjame vytvoria posunom prázdneho políčka dcérske uzly. Následne sa pre novovytvorené uzly skontroluje, či už neexistuje iný uzol s rovnakou konfiguráciou. Tie uzly, ktoré obsahujú nové konfigurácie, sa potom vložia do prioritnej rady a taktiež sa do poľa uložia aj konfigurácie. V prípade, že rozvitím uzla nevznikne ani jedna nová konfigurácia, vyberie sa z prioritnej rady prvý uzol, a ten sa bude rozvíjať v ďalšom cykle.

```
#hlavny cyklus
while True:
    if generatedNotExpandedNodes.empty(): #uz nie su nerozvite uzly, riesenie neexistuje
        return None

    if currentNode.foundGoal(goal): #nasiel sa cielovy stav
        return currentNode

    children = currentNode.generateChildren(heuristic, goal) #vygeneruje potomkov aktualne
    children.sort(key=lambda x: x.heuristic) #a usporiada ich vo vzostupnom poradi

    onlyNewNodes = [] #sem sa vyfiltruju nove konfiguracie
    for c in range(len(children)):
        if children[c].state.configuration not in allConfigurations: #ak konfiguracia este
            onlyNewNodes.append(children[c])

    if len(onlyNewNodes) == 0: #ak nevznikli ziadne nove konfiguracie
        currentNode = generatedNotExpandedNodes.get()[1] #vyberieme prvy uzol z prioritneh
    else: #ak vznikla aspon jedna nova konfiguracia
        currentNode = onlyNewNodes[0] #uzol obsahujuci konfiguraciu s najmensou heuristick
        allConfigurations.append(currentNode.state.configuration) #prida aktualnu konfigur
        for c in range(1, len(onlyNewNodes)): #ostatne uzly odlozime do prioritneho radu a
            generatedNotExpandedNodes.put((onlyNewNodes[c].heuristic, onlyNewNodes[c]))
            allConfigurations.append(onlyNewNodes[c].state.configuration)
```

Porovnanie heuristík 1 a 2

Veľkosť vstupnej matice	Heuristika 1	Heuristika 2
2x3	30	15
3x3	503	35
2x4	383	100
2x5	793	117

pri rovnakých vstupoch generovala prvá heuristika v priemere viac uzlov ako druhá. Druhá heuristika je pre hľadanie efektívnejšia, lebo sa pri nej zohľadňuje počet aj krokov do cieľa, kým pri prvej sa len počítajú políčka, ktoré nie sú na správnom mieste