Computer Assignment 1
Informed and Uninformed Search

In this computer assignment we will implement three search algorithms, BFA, IDS, and A*.

We will try to model a problem witch our agent must collect all parts and reach to end point. But there are some special parts in the map that will create another agent in upper left corner of the map. We cannot move more than one agent at the same time and all agents must be in upper right corner of the map at the end.

BFS

BFS (Breath First Search) is an uninformed search algorithms witch will check all possible actions to find the answer.

This algorithm is very memory hungry and relatively slow.

This is the definition of our states: position of each agent, position of parts need to be collected and position of special parts.

We define our initial state like this: one agent in (0,0), list of parts, and list of special parts.

We define our goal state like this: all agents in (n-1,m-1) and empty list of parts.

Each action is move up, down, left, and right and if there is any part our special part we will collect them and take necessary actions like creating another agent or remove from list.

```python
import enum
import time


class Cell(enum.Enum):
    empty = 0
    wall = 1
    potion = 2
    double = 3
    start = 4
    end = 5


temple = []
m, n = 0, 0


class State:
    def __init__(self, doubles, potions, doctors, parent):
        self.doubles = doubles
        self.potions = potions
        self.doctors = doctors
        self.parent = parent

    def print_path(self):
        doc = [[] for x in range(int(len(self.doctors)))]
        current = self
        while current is not None:
            for i in range(len(current.doctors)):
                if len(doc[i]) == 0:
                    doc[i].append(current.doctors[i])
                elif current.doctors[i] != doc[i][-1]:
                    doc[i].append(current.doctors[i])
            current = current.parent
        result = ""
        for index in range(len(doc)):
            doc[index].reverse()
            result += f"Doctor {index + 1} : {doc[index]} Length = {len(doc[index]) -
        return result

    def is_done(self):
        done = True
        if len(self.potions) != 0:
            return False
        for doc in self.doctors:
            if doc != (int(n) - 1, int(m) - 1):
                return False
        return done

    def __str__(self):
        result = ""
        result += str(self.doubles)
        result += " "
        result += str(self.potions)
        result += " "
        for i in range(len(self.doctors)):
            result += f"{i} : {self.doctors[i]} "
        return result
```

```python
60        def __hash__(self):
61            return self.__str__().__hash__()
62
63        def __eq__(self, other):
64            return self.__hash__() == other.__hash__()
65
66
67    move = ((1, 0), (0, 1), (-1, 0), (0, -1))
68
69
70    def next_state(frontier, visited):
71        ns = []
72        for i in range(len(frontier)):
73            for j in range(len(frontier[i].doctors)):
74                for adj in move:
75                    pos = frontier[i].doctors[j]
76                    new_pos = (pos[0] + adj[0], pos[1] + adj[1])
77                    if pos == (int(n) - 1, int(m) - 1):
78                        continue
79                    if new_pos[0] >= int(n) or new_pos[0] < 0 or new_pos[1] >= int(m) or n
80                        continue
81                    if temple[new_pos[0]][new_pos[1]] == Cell.wall:
82                        continue
83                    new_doubles = list(frontier[i].doubles)
84                    new_potions = list(frontier[i].potions)
85                    new_doctors = list(frontier[i].doctors)
86                    new_doctors[j] = new_pos
87                    new_s = State(new_doubles, new_potions, new_doctors, frontier[i])
88                    new_s.doctors[j] = new_pos
89                    new_s.parent = frontier[i]
90                    if new_s not in visited:
91                        ns.append(new_s)
92                        visited.add(new_s)
93
94        return ns
95
96
97    def bfs(initial):
98        visited = set()
99        queue = [initial]
100       while queue:
101           for f in queue:
102               for doc in f.doctors:
103                   x, y = doc
104                   if temple[x][y] == Cell.potion and (x, y) in f.potions:
105                       f.potions.remove((x, y))
106                   elif temple[x][y] == Cell.double and (x, y) in f.doubles:
107                       f.doctors.append((int(n) - 1, 0))
108                       f.doubles.remove((x, y))
109                   if f.is_done():
110                       return f
111           queue = next_state(queue, visited)
112
113
114   result = []
115   test_time = []
116   for test in 1, 2, 3:
117       test_time.append([])
118       for r in 1, 2, 3:
119           file = open(f'Tests/test{test}.in')
120           n, m = file.readline().split()
```

```
121            c, k = file.readline().split()
122
123            temple = [[Cell.empty for x in range(int(n))] for y in range(int(m))]
124
125            temple[0][0] = Cell.start
126            temple[int(n) - 1][int(m) - 1] = Cell.end
127
128            potion = []
129            for i in range(int(c)):
130                x, y = file.readline().split()
131                temple[int(x)][int(y)] = Cell.potion
132                potion.append((int(x), int(y)))
133
134            double = []
135            for i in range(int(k)):
136                x, y = file.readline().split()
137                temple[int(x)][int(y)] = Cell.double
138                double.append((int(x), int(y)))
139
140            d = file.readline()
141
142            for i in range(int(d)):
143                x, y = file.readline().split()
144                temple[int(x)][int(y)] = Cell.wall
145
146            initial_state = State(double, potion, [(0, 0)], None)
147            begin = time.time()
148            res = bfs(initial_state)
149            test_time[test - 1].append(time.time() - begin)
150            if len(result) == test - 1:
151                result.append(res)
152        print(f"Average execution time of test{test}.in is : {sum(test_time[test - 1]) / l
153        print("Optimal path is :")
154        print(result[test - 1].print_path())
```

Average execution time of test1.in is : 0.23353727658589682 seconds
Optimal path is :
Doctor 1 : [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3)] Length =
6
Doctor 2 : [(3, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3)] Length = 5

Average execution time of test2.in is : 21.773528655370075 seconds
Optimal path is :
Doctor 1 : [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 3)] Length =
6
Doctor 2 : [(3, 0), (2, 0), (2, 1), (2, 2), (3, 2), (3, 3)] Length = 5

Average execution time of test3.in is : 20.616196791330974 seconds
Optimal path is :
Doctor 1 : [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (1, 2), (1, 3), (1, 4),
(1, 5), (2, 5), (3, 5), (4, 5), (5, 5)] Length = 12
Doctor 2 : [(5, 0), (5, 1), (5, 2), (5, 3), (4, 3), (5, 3), (5, 4), (5, 5)]
Length = 7