Computer and Information Sciences
University of Strathclyde

Week 3 – The MIU System, part 2 (worth 5% of your overall grade)

Aim

This this lab is a continuation of your previous lab, using the next_states(s) function. If you have not completed the previous lab, you should complete that first. All the rules from last week apply this week, and you should refer to the previous week if needed.

The MIU System

Last week you started to develop a program for the MIU Puzzle, and implemented the next_states() function, to receive an input and generate all the potential next states that fit within the rules. This week, you will use this function to implement breadth first search, iterative deepening, and then you will be able to compare the performance of these two algorithms. This practical is worth 5% of your total CS310 grade.

Computing the path – extendpath(p)

You should use your next_states(s) function to create a function called extend_path(p) which extends an existing path. A path is a list of states, starting with the start state and progressing towards the goal state in single steps.

- The input to extend_path(p) is a path from the start state to some intermediate state, **s**, this being the last state on the path. You should call next_states(s) and then return a list of all possible paths which extend the input path by a single step.

- For example, if the input path is ["MI","MII"], then the intermediate state is "MII", this being the last state on the input path.

- Calling next_states on "MII" gives ["MIIU", "MIIII"].

- For each element of this result, we create a new path from the input path by adding the new element on the end, and then return the resulting list of paths:

- extend_path(["MI","MII"]) = [["MI","MII","MIIU"], ["MI","MII","MIIII"]]

- Note that the resulting paths must be in the same order as the result returned by next_states(s)

Breadth-First Search – breadth_first_search (goalString)

You can now use your extend_path(p) function to implement breadth-first search. You will find the lecture notes from week 2 to be useful here for designing the algorithm. Create a function named breadth_first_search(goalString).

- The agenda is a queue of paths, initially set to [["MI"]]. At each iteration, set currentpath to be the first item at the start of the queue and remove it from the queue. If the last string of currentpath is equal to goalState, then print out the length of the path, the number of times extend_path(p) got called and the maximum size of the agenda, and return the currentpath list (see examples below).

- Otherwise, call extend_path(currentpath), add the new paths to end of the agenda and repeat.

- You'll also need add a limit to the number of times extend_path(p) can be called (in case the string that you're searching for turns out to be impossible to derive from "MI"). In such cases, **return [0,0,0]** as the result. The limit can be quite large if needed, say, 5000.

- For submission, you can assume that working extend_path and next_states functions are present.

- When submitting, you should not have any print values, but can return the values you need:
  `return currentPath, extendCount, agendaMaxLen`

Some BFS outputs:

```
breadth_first_search("MUIU")
Number of Expansions: 15
Max Agenda: 20
Solution Length: 5
Solution: ['MI', 'MII', 'MIIII', 'MIIIIU', 'MUIU']

breadth_first_search("MIUIUIUIU")
Number of Expansions: 6
Max Agenda: 6
Solution Length: 4
Solution: ['MI', 'MIU', 'MIUIU', 'MIUIUIUIU']
```

Depth-First Search with iterative deepening

Depth first search (i.e. treating the agenda as a stack rather than a queue) does not work well for the MIU problem. However, we can make use of modified methods to search. Here, you will need to create two functions, depthlimited_dfs(goalString, limit), and dfs_iter(goalString). You may also need to create a global variable.

- Depth-first search (i.e. treating the agenda as a stack rather than a queue) doesn't work well for MIU. The search can go on forever on a single branch of the search tree using the doubling rule (i..e double everything after M):

```
"MI" → "MIU" → "MIUIU" → "MIUIUIUIU" → "MIUIUIUIUIUIUIUIU" → ...
```

- If we impose an arbitrary limit on the depth of the search (say not expanding any paths which have reached a length of 5), then we won't run into this problem. So, first, implement this function:

```
def depthlimited_dfs(goalString, limit)
```

- Perform depth-first search using the same algorithm as breadth-first search but when you call extend_path(p), add the new paths to the front of the agenda. And when considering

currentPath for expansion, if it has already reached the length limit, don't expand it.

- Note that doing this may lead to the agenda becoming empty, so you will need to check for this.

- depthlimited example:

```
depthlimited_dfs("MIUIUIUIU",4)
Number of Expansions: 4
Max Agenda: 2
Solution Length: 4
Solution: ['MI', 'MIU', 'MIUIU', 'MIUIUIUIU']
```

- Depth-limited search has two problems. First, if the limit is not set to be big enough, then you won't find the goal. Second, if the limit is set to a value greater than the length of the optimal path, then it can return a path which is not optimal, because it finds this deeper path before the shorter one.

- Iterative deepening is the solution to this: set the depth-limit to be 2 and apply depthLimited_dfs. If this yields no solution, increase the limit by 1 and try again. Keep doing this until depthlimited_dfs returns a solution. When it does, this solution is guaranteed to be optimal.

- Implement the `dfs_iter(goalString)` function. When it finds the solution, it should print out the length of the path, the total number of calls to extend_path(p), the maximum size of the agenda, and then return the path.

- For submission, you can assume that working extend_path and next_states functions are present.
- When submitting, you should not have any print values, but can return the values you need:
- `return currentPath, extendCount, agendaMaxLen`

- Some dfs_iter outputs

```
dfs_iter("MUIU")

Number of Expansions: 19

Max Agenda: 6

Solution Length: 5

Solution: ['MI', 'MII', 'MIIII', 'MIIIIU', 'MUIU']


dfs_iter("MUIIU")

Number of Expansions: 40

Max Agenda: 24

Solution Length: 6
```

```
Solution: ['MI', 'MII', 'MIIII', 'MIIIIIIII', 'MUIIIII', 'MUIIU']
```

Submission Instructions

- This practical is worth 5% of the mark for the class. There will be a quiz on MyPlace for each of the 3 stages. For the depth first and breadth first search, you can assume that a working version of next_states and extend_path is already present. Make sure your functions return values and do not use any print statements.