Middle East Technical University | Department of Computer Engineering

CENG 462

Artificial Intelligence

Fall '2020-2021

Homework 1

Due date: 29 November 2020, Sunday, 23:55

1 Objectives

This assignment aims to assist you to expand your knowledge on informed search in the cases of A* and Iterative Deepening A* algorithms.

2 Problem Definition

In this assignment, you are going to solve a Tower of Hanoi puzzle *given in any valid state* by using A* and Iterative Deepening A* algorithm.

As computer scientist, we love this puzzle and use it everywhere! So, there is a great chance that you've seen Tower of Hanoi already as an example for recursion or time complexity etc. However, let's remember the original definition again, to refresh your memory about the terms and rules of Tower of Hanoi puzzle.

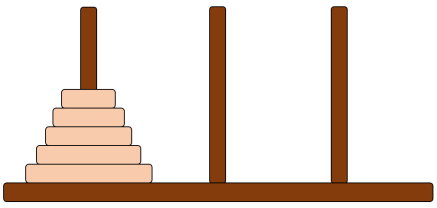

Figure 1: Tower of Hanoi with 5 disks replaced to left-rod initially

Tower of Hanoi is a game consisting of a 3 `rods` with N `disks` with varying sizes. Initially, all disks are placed onto a rod in the ascending order of their sizes, so it looks like a neat, canonical shape (see Figure 1). The purpose of the game is moving whole stack of disks to another rod without violating the following rules:

- Only one disk can be moved at a time.
- A disk can be moved only if it is on the top of its rod.
- No larger disk can be placed of a smaller one at any time. In other words, the ascending order regarding the size of disks should not be disturbed on any rod during the entire game.

Here is an example of initial configuration for a Tower of Hanoi puzzle with 3 disks:

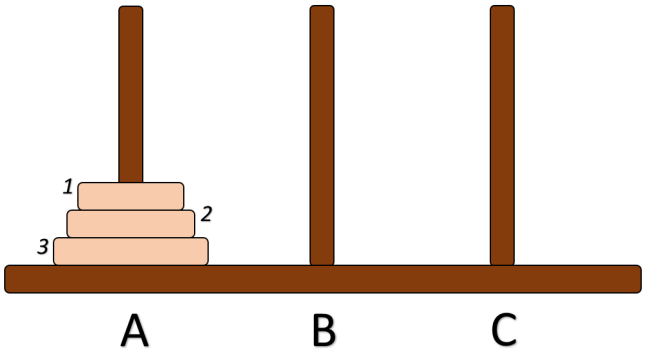

Figure 2: Example configuration as initial state for Tower of Hanoi with 3 disks

where the rods are labeled as "A", "B" and "C" left to right, and the disk are enumerated from 1 to 3 matching with their sizes (greater number means greater size). In this example, all disks are placed onto the rod "A". Assuming that the goal rod is "C", the final state should look like following:

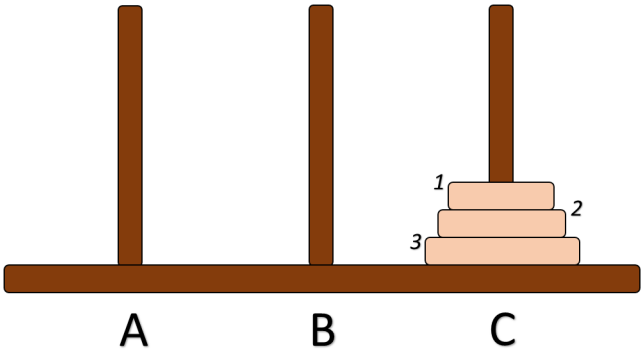

Figure 3: Example configuration as final state for Tower of Hanoi with 3 disks

Unlike the original puzzle, in this assignment, you will also be given initial configurations which represent a valid intermediate state for the original Tower of Hanoi puzzle. Therefore, the initial state given in the input may look like below:

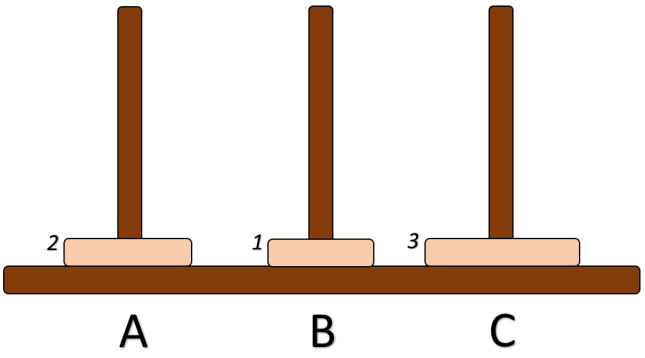

Figure 4: Another example configuration as initial state for Tower of Hanoi with 3 disks

3 Specifications

- You are going to implement A* and Iterative Deepening A* algorithms (you can refer the lecture slides for pseudocodes.) in python.
- You will use the following function as the heuristic function:

$$f(x, y) = x + 2 \times y \tag{1}$$

  where x is the total number of disks in the "non-goal" rods and y is the number of disks in the goal rod such that their sizes are smaller than any other disk in the other rods.

- The task will be given from standard input and the result will be printed to standard output.
- Input will consist of:
  - The method you are going to use for the given task ("A*" or "IDA*" for Iterative Deepening A*)
  - M as the maximum total estimated cost allowed in the method
  - N defining the number of disks,
  - The goal rod where the disks are placed in the correct order at the final state,
  - The following 3 lines will include the placement of the disks with their enumerated ids in the rod "A", "B" and "C" at the beginning (**bottom to top and separated with ","**). If no disk placed onto the related rod, a blank line will be given.
- Output will consist of "SUCCESS" or "FAILURE" stating whether the goal is reached or not respectively. If the solution exists with given constraint of total estimated cost, all configurations on the path from the initial configuration to the goal configuration will be printed as well, following a blank line after the indicator of "SUCCESS".
- Output of a configuration must look like:

```
A->[2,1]    B->[]    C->[3]
```

  Content of a rods as lists (bottom to top), separated with the tab character, and a new line character at the end is included. To ease to understand and comply this format, here is the pattern I used with the `format()` method in the python:

```
"A->{}\tB->{}\tC->{}\n"
```

- While expanding a node you are required to try to move a disk in the **lexical order** (First "A" to "B", then "A" to "C", then "B" to "A" and so on.)
- When searching for the state with the minimum cost you will select the **first** one for tie-breaking.

IMPORTANT NOTE: Complying the output format and the last 2 items is crucial for the black-box evaluation. Please avoid any violation of them, in order not to lose any points redundantly.

4 Sample I/O

Input 1:

```
A*
4
3
C
2
1
3
```

Output 1:

```
SUCCESS

A->[2]   B->[1]   C->[3]


A->[]  B->[1]   C->[3,  2]


A->[]  B->[]  C->[3,  2 ,  1]
```

Input 2:

```
A*
3
5
B
5 ,3 ,2 ,1

4
```

Output 2:

```
FAILURE
```

Input 3:

```
IDA*
10
3
A

3 ,2 ,1
```

Output 3:

```
SUCCESS

A->[] B->[3, 2, 1]  C->[]


A->[1]  B->[3, 2] C->[]


A->[1]  B->[3]  C->[2]


A->[] B->[3]  C->[2, 1]


A->[3]  B->[] C->[2, 1]


A->[3]  B->[1]  C->[2]


A->[3, 2] B->[1]  C->[]


A->[3, 2, 1]  B->[] C->[]
```

Input 4:

```
IDA*
5
3
A

3,2,1
```

Output 4:

```
FAILURE
```

5 Regulations

1. **Programming Language:** You must code your program in Python3. Your submission will be tested in inek machines. So make sure that it can be executed on them.

2. **Implementation:** You have to code your program by only using the functions in standard module of python. Namely, you **cannot** import any module in your program. The only exception for this rule is the `copy` module which may be useful with the `deepcopy` function to cope with aliasing problem in python.

3. **Late Submission:** No late submission is allowed.

4. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

5. **Discussion:** You must follow the OdtuClass for discussions and possible updates on a daily basis.

6. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. A reasonable timeout will be applied according to the complexity of test cases. This is not about the code efficiency, its only purpose is avoiding infinite loops due to an erroneous code.

6 Submission

Submission will be done via OdtuClass system. You should upload a **single** python file named in the format <**your-student-id**>_**hw1.py** (i.e. e1234567_hw1.py).