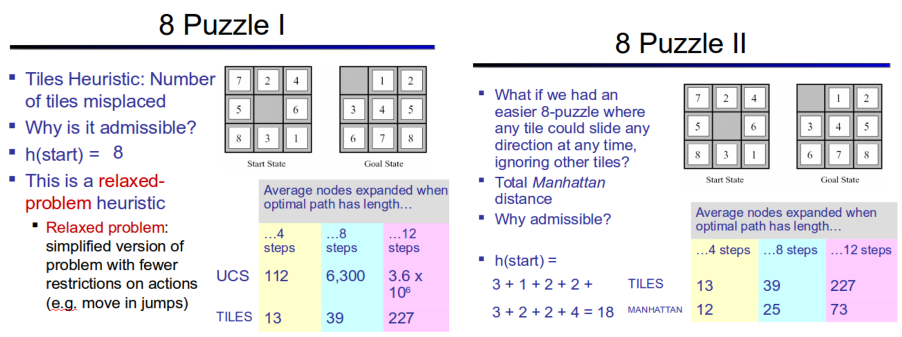
COSC 604: Techniques in Artificial Intelligence *Assignment 1*

Andreas Henschel - `andreas.henschel@ku.ac.ae`

Hand out: September 29, 2020
Submission due: November 1th, 2020

1 Introduction

This assignment comes in two parts. The first part is about implementations of search strategies, applied to the simple Routing problem discussed in the book. In the second part you are using the same generic algorithms to implement search solutions for the Tile Puzzle problem (as described during the lectures and in the book). The main goal is to reproduce the results from the slides:



8 Puzzle I

- Tiles Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a relaxed-problem heuristic
  - Relaxed problem: simplified version of problem with fewer restrictions on actions (e.g. move in jumps)

Start State		
7	2	4
5		6
8	3	1

Goal State		
	1	2
3	4	5
6	7	8

Average nodes expanded when optimal path has length…

	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6 x 10^6
TILES	13	39	227

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why admissible?
- h(start) =
  3 + 1 + 2 + 2 +
  3 + 2 + 2 + 4 = 18

Start State		
7	2	4
5		6
8	3	1

Goal State		
	1	2
3	4	5
6	7	8

Average nodes expanded when optimal path has length…

	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

Figure 1: Lecture slides regarding the performance of different heuristics in the tile set.

1.1 Provided Code

This assignment is to be implemented in Python. You are provided with sample code (myseraches.py on Blackboard), which gives you ideas where to extend code and run sample commands.
The provided files for now are

`search.py` : this file contains the search algorithms discussed in the lecture. Here you are supposed to implement the search algorithms (see instructions below) It also ontains important data structures that are necessary

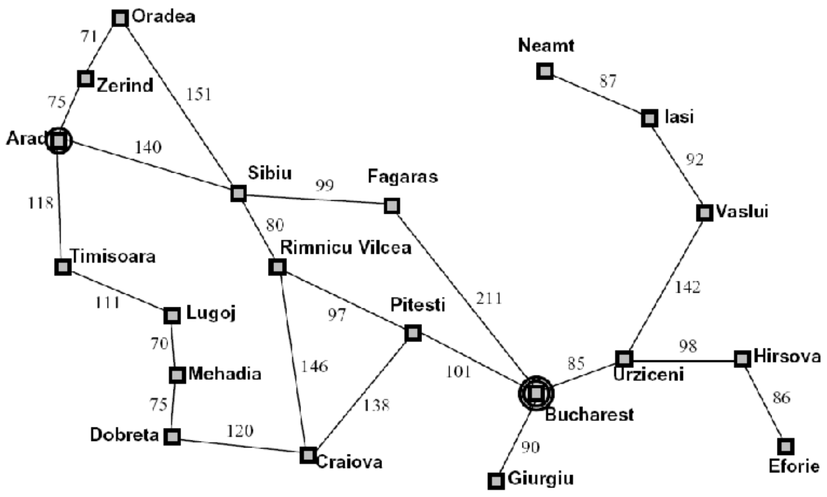


Figure 2: Romania example from the book: "Artificial Intelligence - A modern Approach"

for the search algorithms above (LIFO, FIFO, PriorityQueue, and PriorityQueueWithFunction. You are not required to modify this file except for the last task. You are also given a scaffold implementation of the Romania Route finding problem. You can look at the provided functions for a search problem (start node, goal test, successor state function).

`vacuuum.py` : An implementation of the vacuuum problem. It exemplifies how to make sure that states are hashable and thus usable in the explored set.

2 Search

2.1 Navigation in the Toy Problem

The code provides you with minimal examples to conduct different types of search, including A* search with a hard coded heuristic function.

2.2 Romania

Now, start with familiarizing yourself with the Romania problem. Look at the commented code in the main section and experiment with it. Try different initial and goal states, try analysing the returned data structures.

2.3 Iterative Deepening

Implement iterative Deepening in a generic fashion for graph search, making use of the provided depth first search function.

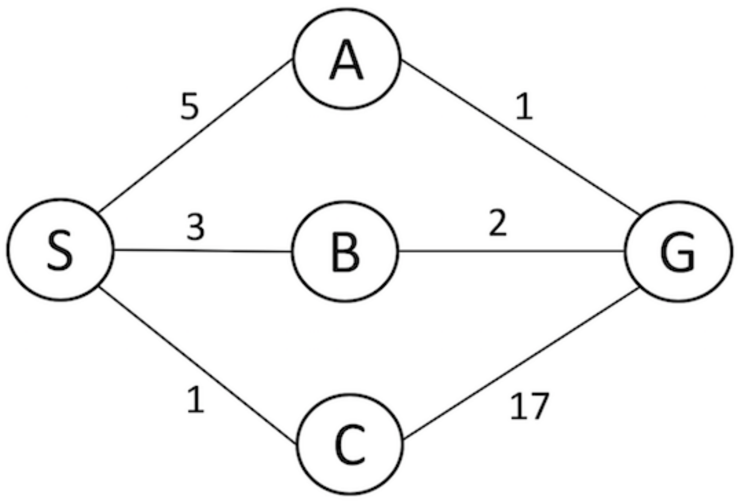


Figure 3: Toy example, helpful to understand different types of searches

2.4 A^* search with Euclidean Distance heuristics

Implement A^* search based on the Euclidean distance for the Romania problem by passing a function that can take a node as input and calculates the Euclidean distance to the goal state. It is possible to implement A^* by providing the correct frontier object. You will need a suitable heuristic function, that should be passed as an argument to PriorityQueue. The heuristic function takes as input a node (which has a state) and returns a value that will be used as a priority. For the calculation, you should make use of locations (provided as a global variable). Note that passing the heuristic function is a bit tricky, we will discuss that during the lab/lecture.

Using the Graph search algorithm, it is now straightforward to implement Depth First Search by providing the correct frontier. Also finish the implementation of the Romania Map problem in romania.py (see comments in the `romania.py` file). You can then test your search algorithm. Note that the Graph search algorithm is supposed to return a Node object, which contains the state, the path (history of actions), and the cost. In order to print out the final solution, print out the latter two.

2.5 Exploration history, frontier size and optimality

Solve the Romania problem for all possible initial states, using Depth First Search, Breadth First Search, Iterative Deepening, Uniform cost search and A^* (with graph search). Create three bar charts:

- one that plots the exploration history size for all initial states;

- one that plots solution length for all initial states and
- one that plots maximal frontier sizes for all initial states.

Argue how the plots reflect time and space complexity as well as completeness and optimality. Using the Graph search algorithm, it is now straightforward to implement Breadth First Search by providing the correct frontier.

2.6 Tree Search

~~Answer the question: what would happen if you would use tree search, and describe what change(s) would be needed in the parameters to the GraphProblem construction, to solve the problem maximal frontier sizes.~~

3 Tile Puzzle

Complete the 4x4 tile puzzle problem. You can elaborate on the provided code or you can do your own implementation. Solve it using Uniform Cost Search, with the misplaced tiles heuristic and A^* with the total Manhattan heuristic. You will need to create initial states of varying difficulty: you can either take a solution and add a certain number of random moves to it, or you take a random state and make sure that you get the difficulty level from one your optimal solution (note: all three solvers should give you an optimal solution, i.e., should agree on the number of moves).

4 Bonus

Implement the general N x N tile problem. Can you come up with a better heuristic than "total accumulated Manhattan"?

5 Submission

Submit your code by email. Write a short report containing the solutions for each search algorithm, as well as a bar diagram for the frontier sizes. Ideally you would formulate this report in the shape of a small scientific paper written in LaTeX, adhering to the Scientific Method (see Wikipedia).