CS4006 - Intelligent Systems, CSIS/UL
A-star(A*)

Lecturer: Dr Malachy Eaton
Project Coordinator: Davi Monteiro
`davi.monteiro@ul.ie`

March 2020

1 Introduction

A* is an optimisation (search) algorithm that is used to find the optimum path between a start **state** and one of the goal **states** (if there are multiple-goal states) with minimum cost. The term *state* may refer to two cities when the optimum (shortest) path between two cities is of interest to be found, or may refer to two different states of a 15 or 8 puzzle when going from state 1 to state 2 is of interest.

Normally there are a large number of paths between two points and testing all possible paths in order to find the optimum path is nearly impossible or it is hugely time-consuming, therefore some smart solutions have to be employed.

In order to illustrate the problem more graphically, we can consider the trip from city S to the city E in Figure 1. To have the map of all cities, one can create the *Tree* structure of the map in order to illustrate the complexity of the problem (See Figure 2).

As you can see there are 12 paths
$\{1, 5, 13\}$, $\{1, 6, 13\}$, $\{1, 7, 14\}$, $\{1, 8, 14\}$, $\{1, 8\}$, $\{2, 8, 14\}$, $\{2, 9, 15\}$, $\{3, 10, 15\}$, $\{4, 11, 16\}$, $\{4, 12, 16\}$, $\{1, 13\}$, $\{2, 8\}$.

The question is which path is the best to choose?

2 A*

A* has two functions called g and h. The input of these two functions is the next node on the path. Function $g(n)$ simply calculates the cost of the path from the start point, therefore the value for $g(n)$ is 0 at the beginning. $h(n)$ is a function that estimates the cost from point n the destination. The sum of these two functions indicates the total cost of the current path $f(n) = g(n) + h(n)$.

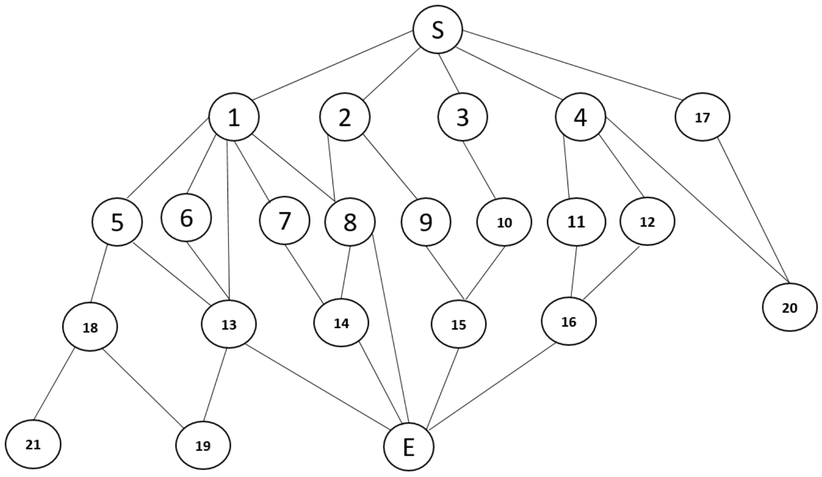

Figure 1: Travelling from S to E

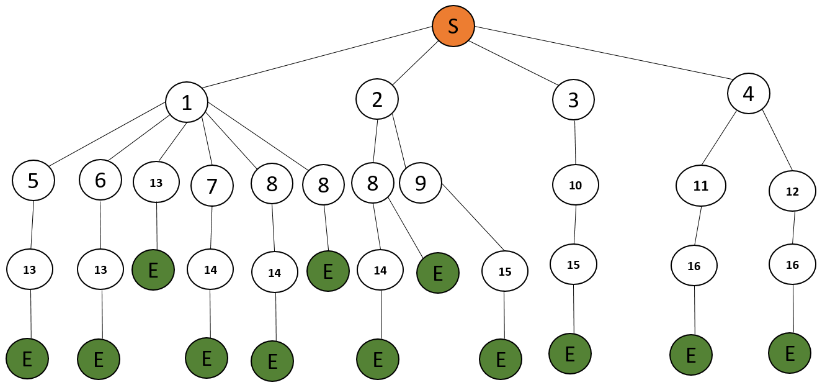

Figure 2: All possible paths from S to E

2.1 Algorithm

In this section, the A* algorithm is presented. This algorithm can be then generalised for any type of problems from different contexts.

1. Declare three parameters, the source, the destination and the current point, S, E, C.
2. Declare a function called $mem(n)$ for each single point n.
3. $C := S$
4. Initiate two lists call them *closed* and *open*.
5. Identify all the points (m points) that are directly/simply (without any intermediate point) reachable from C: $X = \{X_1, X_2, ..., X_m\}$.
6. Those members of X that are not in *closed* are added into the *open* list.
7. Add C into *closed*.
8. Calculate $g(n)$ for all the members of *open*.
9. Estimate $h(n)$ for all the members of *open*.
10. Estimate the value of $f(n)$ for all the members of *open*.
11. Select a point from *open*, O_i, where $O_i \notin$ *closed* and $f(O_i)$ is minimum; then assign O_i to C; $C := O_i$.
12. $mem(O_i) = C$
13. Add O_i into *closed* and remove O_i from *open*.
14. If $C = E$, terminate the algorithm otherwise go back to step 4.

Following the aforementioned algorithm above, one can find the shortest path in Figure 1 as follows:

1. $C = S$
2. $X = \{1, 2, 3, 4\}$
3. *open* $= \{1, 2, 3, 4\}$
4. *closed* $= \{S\}$
5. $f(1) = 3$, $f(2) = 3$, $f(3) = 4$, $f(4) = 4$
6. Either $f(1)$ or $f(2)$ is selected to be assigned into O_i. (we choose $O_i = f(1)$
7. $C = 1$

8. $mem(C) = S$; $mem(1) = S$
9. $open = \{2, 3, 4\}$: Current members of *open*.
10. $closed = \{S, 1\}$
11. $X = \{5, 6, 7, 8, 13, S\}$
12. $open = \{2, 3, 4, 5, 6, 7, 8, 13\}$: Updating *open*
13. $f(2) = 3$, $f(3) = 4$, $f(4) = 4$,$f(5) = 4$, $f(6) = 4$, $f(7) = 4$, $f(8) = 4$, $f(13) = 3$: Either 2 or 13 is chosen, we choose 13.
14. $C = 13$
15. $mem(C) = 1$; $mem(13) = 1$
16. $open = \{2, 3, 4, 5, 6, 7, 8\}$: Current members of *open*.
17. $closed = \{S, 1, 13\}$
18. $open = \{2, 3, 4, 5, 6, 7, 8, E\}$: Updating *open*
19. $f(2) = 3$, $f(3) = 4$, $f(4) = 4$,$f(5) = 4$, $f(6) = 4$, $f(7) = 4$, $f(8) = 4$, $f(E) = 2$: E is chosen.

3 Project

3.1 The shortest path problem

In this project, we are going to use the A* algorithm [1] to solve a problem of finding a path between two vertices or nodes in a graph such that the sum of the weights of its constituent edges is minimised. This problem is known as the shortest path problem [2] and can be defined for undirected, directed, or mixed graphs. For this project, we are going to consider the undirected graph definition expressed on grid maps.

The grid map used in this project should be an 8×8 matrix that allows 4 directions of movement (e.g., left, right, up, and down). Diagonal movements are not allowed. In addition to the grid map, an obstacle needs to be generated randomly, including its position on the map and its shape. For generating the obstacle, a minimum of 3 or a maximum of 5 connected blocks should be considered and in the shape of a letter (e.g., T, L, or I). Figure 3 illustrates an example of a grid map with an obstacle.

The generated grid map with an obstacle must be displayed using a console or graphical interface. Subsequently, the end-user must be asked for the start and goal positions. User input must be validated. If the input provided by the user is invalid (e.g., start or goal positions are outside of the grid map), new

[1] `https://en.wikipedia.org/wiki/A*_search_algorithm`
[2] `https://en.wikipedia.org/wiki/Shortest_path_problem`

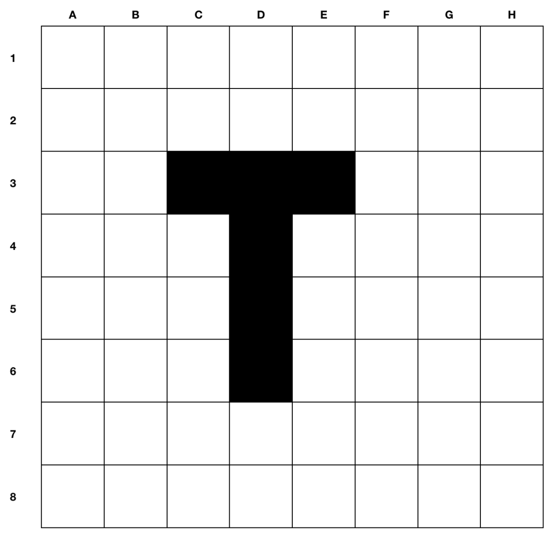

Figure 3: A grid map with obstacle

input needs to be provided. If the input is valid, an updated version of the grid map with the start and goal positions should be shown, as illustrated in Figure 4.

3.2 Manhattan distance

In the A* algorithm, h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. To this end, we are going to use *Manhattan distance* [3] as a heuristic function. The Manhattan distance is calculated by $d(i, j) = |x1 - x2| + |y1 - y2|$. For more information regarding how to calculate the Manhattan distance, use the following resource [4]. Finally, after finding the shortest path, you must print it out on the grid map.

4 Submission

You are required to write a Java program in order to solve the shortest path problem using the A* algorithm.

1. The deadline of the submission is (Saturday) 25th of April 2020 (end of week 12).

2. Only one .java file has to be submitted by one of the members of the group.

[3] `https://en.wikipedia.org/wiki/Taxicab_geometry`

[4] `http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html`

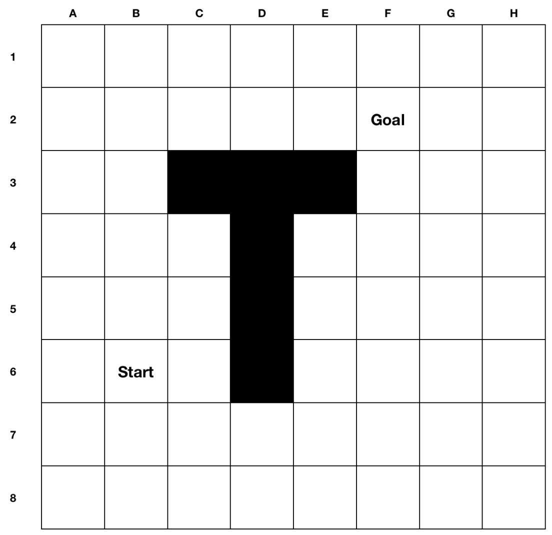

Figure 4: A grid map with obstacle and start and goal positions

3. The name of the .java file has to be *is*12345678.*java* where 12345678 is the student ID of the person who submits the project.
4. The first few lines of the .java file has to contain the full name and the student ID of all the member in the group.
5. The submission must be made by email to *davi.monteiro@ul.ie* and the full name and student ID of all members have to be listed in the email. Please CC all the members of the group in the submission email as well.
6. Any submission that fails to run and/or compile will be graded 0.
7. Submissions after the deadline will not be evaluated.
8. The submission is marked out of 20 marks.