|  |  |
| --- | --- |
| **Mã số sinh viên** | **DTH195423** |
| **Họ và tên** | **Nguyễn Thành Đặng** |
| **Lớp** | **DH20TH1** |

**BÀI TẬP CHƯƠNG 2**

1. Explain why problem formulation must follow goal formulation.

*Answer:*

In problem formulation, we decide how to manipulate the important aspects, and ignore the others. So, without doing goal formulation, if we do the problem formulation, we would not know what to include in our problem and what to leave, and what should be achieved. So problem formulation must follow goal formulation.

1. Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall.
2. Formulate this problem. How large is the state space?

*Answer:*

To formulate the problem we should start off by setting up a coordinate system with x and y vertices, so start at center of a “maze” and set it as (0,0).

The entire maze can be a square of (-1,1) to (1,1).

The way to test this will be as long as the x and y are greater than 1 to the current location. Successor function can be moving forward any amount of distance and the cost function can be the total distance moved.

The state space can be infinite due to robot position as infinite.

1. In navigating a maze, the only place we need to turn is at the intersection of two or more corridors. Reformulate this problem using this observation. How large is the state space now?

*Answer:*

If the navigation is going to change to only needing turn at intersection of two or more corridors, we need to have an exit node at the end of each corridor. The initial state will now be facing north in the center of the maze as before. The test will be to get to an exit node. The successor function is to move past the intersection if there is one in front of us, and the cost function is just the total distance moved as before. The state space will have changed due to the number if intersections and that can be 4, therefore the state space is now 4N with N being the number of intersections.

1. From each point in the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using these actions. Do we need to keep track of the robot’s orientation now?

*Answer:*

Changing the navigation to only being able to move in any four directions till we reach the wall is what this is asking for. To do this, the initial state will be at the center of the maze (direction won’t matter due to the fact that we’re just looking to reach a wall). The test will be to an exit node once again, and the successor function will be to move to the next intersection (lets say you originally got to the north wall, now we need to go to either the east,west, or south wall). The total cost function is the total distance moved as before.

1. In our initial description of the problem, we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

*Answer:*

Three simplifications that we could have made for this are the following:

1) We assumed that the robot can only face 4 directions, what if it could move in other ways?

2) We ignored the other variables about the world including the temperature, wind that may move the robot and change orientation, and other natural causes.

3) We also ignored possibility of other robots in the same area or other items in the way of the robot moving in its “space”.

1. Suppose two friends live in different cities on a map, such as the Romania map shown in Figure 3.2. On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city *i* to neighbor *j* is equal to the road distance *d(i, j)* between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the first on his/her cell phone) before the next turn can begin. We want the two friends to meet as quickly as possible.

Diagram

Description automatically generated

1. Write a detailed formulation for this search problem. (You will find it helpful to define some formal notation here.)

*Answer:*

We can start the search problem by defining the state space. States are all city pairs (i,j). The successor function would be the successors of (I,j) lets call them, (x,y), and adjacent pairs (x,i), (y,j). The goal will be to be at some (i,i) such that both people are in the exact same location together. The cost function in this will be to going from (i,j) to (x,y).

1. Let *D(i, j)* be the straight-line distance between cities *i* and *j.* Which of the following heuristic functions are admissible? (i) *D(i, j)*; (ii) 2 · *D(i, j)*; (iii) *D(i, j)*/2.

*Answer:*

The function that’s admissible would be function (iii). This is because this way it is evenly spread out between each friend.

1. Are there completely connected maps for which no solution exists?

*Answer:*

Yes it is possible to have connected maps for which no solution exists. This is due to a possibility such as if the two friends start at an odd number of steps apart, they will never be together in the same spot.

1. Are there maps in which all solutions require one friend to visit the same city twice?

*Answer:*

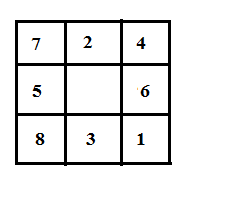
This is not a possibility due to you finding a path in which the two friends meet together. It is not possible to have them double back and go to the same city twice if you are trying to find the most efficient way in which two people can meet.

1. Show that the 8-puzzle states are divided into two disjoint sets, such that any state is reachable from any other state in the same set, while no state is reachable from any state in the other set. (Hint: See Berlekamp et al. (1982).) Devise a procedure to decide which set a given state is in, and explain why this is useful for generating random states.

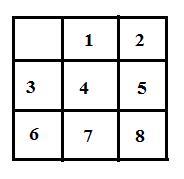
*Answer:*

Define the order of counting from the upper left corner to the lower right corner.

Start state is



Let N denote the number of lower numbers following a number, called “inversions”. Then, goal state is



The goal state has the numbers in a certain order where counting starts at the upper left corner and proceeds from left to right. When reached to the end of a row then move to the left most square in the next row. For any other configuration besides the goal, whenever a tile with a greater number on it precedes a tile with a smaller number, the two tiles are said to be inverted.

Proposition:

For a given puzzle configuration, let N denote the sum of the total number of inversions and the row number of the empty square.

Then (N mod 2) is invariant under any legal move. Means, after a legal move an odd N remains odd whereas an even N remains even.

Proof:

Sliding a tile horizontally changes neither the total number of inversions nor the row number of an empty square. Therefore let us consider sliding a tile vertically.

Assume that the tile 1 is located directly over the empty square. Sliding it down changes the parity of row number of an empty square. Now consider the total number of inversions. The move only affects relative positions of tiles 1, 2, 3, and 4. If none of the 2, 3, 4 caused an inversion relative to 1 then after sliding one gets three of additional inversions.

If one of the three is smaller than 1, then before the move greater than 2, 3, and 4 contributes a single inversion. After the move they’ll be contributing two inversions - a change of 1, also an odd number. Two additional cases obviously lead to the same result. Thus the change in the sum N is always even. This is precisely what we have set out to show.

So before we solve a puzzle, we should compute the N value of the start and goal state and make sure they have the same parity, otherwise no solution is possible.

1. Give a complete problem formulation for each of the following. Choose a formulation that is precise enough to be implemented.
2. Using only four colors, you have to color a planar map in such a way that no two adjacent regions have the same color.

*Answer:*

The problem formulation would the following. The initial state would be no regions colored. The test would be all regions are colored with no adjacent regions with same color. The successor function would be to assign a different color to a region next to one you’ve already colored. The cost function would depend on the size of the planar map.

1. A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.

*Answer:*

The initial state is the monkey standing in a room with the banana’ hanging from the ceiling. The goal will be the monkey getting the banana’s. The successor function will be to use the crate to get a banana then to move the crate and to grab another banana. The cost function is simply the number of actions preformed.

1. You have a program that outputs the message “illegal input record” when fed a certain file of input records. You know that processing of each record is independent of the other records. You want to discover what record is illegal.

*Answer:*

The initial state for this will be all the input records. The goal will be to have it put out “illegal message” to a record that’s illegal. The successor function will be to test all of the records again. The cost function will be the total number of runs made.

1. You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.

*Answer:*

The initial state that jugs will all be empty. The goal will be to measure out exactly one gallon. The successor function will be to set calculate one gallon by emptying each of the three jugs in a mathematical way. You can do this by assigning x y z values to each of the jugs. The cost function will be the number of total actions preformed.

1. Consider the problem of finding the shortest path between two points on a plane that has convex polygonal obstacles as shown in Figure 3.31. This is an idealization of the problem that a robot has to solve to navigate in a crowded environment.

Shape, polygon

Description automatically generated

1. Suppose the state space consists of all positions *(x, y)* in the plane. How many states are there? How many paths are there to the goal?

*Answer:*

There will be infinite number of states and paths when the state space is (x,y).

1. Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. How large is this state space?

*Answer:*

We start this off by knowing that the shortest path from one polygon vertex to any other is a straight line. If it is not possible to make a straight line due to things in the way, then we need to make a few lines that are as straight as possible. Make the line go straight till the obstacle, then deviate past the obstacle then make another straight line. The obstacle is polygonal so the lines must go from start to tangent point and the tangent points will be vertices of the obstacle. The state space will be the 35 set of vertices.

1. Define the necessary functions to implement the search problem, including an ACTIONS function that takes a vertex as input and returns a set of vectors, each of which maps the current vertex to one of the vertices that can be reached in a straight line. (Do not forget the neighbors on the same polygon.) Use the straight-line distance for the heuristic function.
2. Apply one or more of the algorithms in this chapter to solve a range of problems in the domain, and comment on their performance.