CSCI 4964/6963 Advanced Robotics, Spring 1999
Assignment 4: out Friday March 19, due Monday April 5

Implementation of the A* graph search algorithm

In this assignment, you will implement the A* graph search algorithm and use it to determine the shortest path in visibility graphs.

There are four easy steps to this assignment:

1. *Implement the algorithm.* You can use your favorite programming language. I have provided some C/C++ code to get you started, as well as a few simple examples to use for debugging.
2. *Run your program on the three data sets.*
3. *Answer the questions about the A* algorithm.*
4. You must turn in:
   - A printout of your code
   - A printout of the output generated by your program on the three data sets. (For details, see below.)
   - Written answers to the questions.

More details are included in the following sections.

Visibility graphs

The input to your program is a graph, which I have described in two separate files. One file contains the adjacency matrix; the other contains the locations of each nodes. The adjacency matrix is a square matrix that describes which nodes in the graph are connected. The graphs I have generated for the data sets are visibility graphs from a two dimensional configuration space. I have provided illustrations of the configuration spaces that gave rise to these visibility graphs for your convenience.

Visibility graphs are generated in a configuration space where the obstacles are polygonal. Any vertex of the world or of a polygon that is not inside another C-space obstacle is a node in the visibility graph. Two vertices are connected if there is a straight line between them that does not intersect any C-space obstacles. The illustrations show both the original obstacles and the "enlarged" C-space obstacles. Nodes in the visibility graph are plotted with a square.

Files & file formats

I have provided two debugging data sets, `debug{1,2}{adj,nodes}`. The three data sets for this assignment are: `prob{1,2,3}{adj,nodes}`. You can get these files from the course web pages.

Each file begins with a line such as:

```
nnodes: 9
```

which gives the number of nodes in the graph. I have adopted the C numbering convention of the first node being node number 0.

The adjacency matrix file contains one line for each row, hence there are $nnodes$ lines, each with $nnodes$ numbers. An entry a_{ij} is 1 if node i is connected to node j and is 0 otherwise. Since this is an acyclic graph, the matrix is symmetric. Also, $a_{ii} = 1$.

The file of node locations contains one line for each node, also starting with node 0. Each line contains the x and y coordinate of the node in configuration space. The coordinates are floating point numbers. The start node is always node 0. The goal node is always the last node (node $nnodes - 1$).

Compare the debugging files with the illustration of the graph for an example.

The A* algorithm

Recall from class that the basic A* algorithm is as follows:

- place the start node in a queue
- while the queue is not empty and the first element of the queue isn't the goal:
  - remove the first element from the queue
  - explore the paths formed by traversing one edge from this node:
    - if a path reaches a node that we haven't reached before, add it to the queue
    - if a path reaches a node that we have already reached, see if the current path to that node is shorter than the best path to that node
  - sort the queue based on the estimated cost of each partial path — the length of the best path to a node plus an underestimate of the length remaining to the goal
- if the first element of the queue is the goal, report success; otherwise report failure.

I suggest that you use the straight line distance from a node to the goal as an underestimate of the distance remaining to the goal. Note that in order to report a path after this algorithm has run, you will have to do some "bookkeeping".

Your program

You can implement the A* algorithm in your favorite programming language. Feel free to modify my data files as you wish — the important thing is to get the same graph into your program.

I have provided some skeleton C/C++ code which has routines to load the data files into global variables. This code is available from the course web pages. You can feel free to modify this code as you wish, or you can ignore it altogether.

Program output & sample output

At each iteration, your program should print out which node it is exploring. After a path has been found, it should print out the path. Your program does not have to produce output exactly like the sample output below, but it must contain the above information.

```
[whuang@localhost assign4]$ ./astar debug1adj debug1nodes
Exploring paths from node 0
Exploring paths from node 5
Exploring paths from node 1
Exploring paths from node 3
Exploring paths from node 2
Exploring paths from node 6
Exploring paths from node 7

A path was found to the goal:
  from the start node at (1.00, 1.00),
  go to node 6 at (1.00, 4.00)
  go to node 7 at (3.00, 4.00)
  go to node 8 at (4.00, 4.00)
  and that's the goal!

[whuang@localhost assign4]$ ./astar debug2adj debug2nodes
Exploring paths from node 0
Exploring paths from node 5
Exploring paths from node 1
Exploring paths from node 3
Exploring paths from node 2
Exploring paths from node 6
Exploring paths from node 4

No path found to the goal.
```

Academic honesty & sharing code

Giving code to another student or receiving code from someone else is inappropriate. However, if you are working in a language other than C/C++, you may share code for reading in the data files only — such code should not contain any data types other than those for the adjacency matrix and the node coordinates.

Note

My solution (the file containing code that would have to be added to the available C/C++ skeleton) is 102 lines, which took me less than a half hour to write. It will probably take you longer since you did not design the assignment, but this should give you general idea of the size of the programming part of this assignment.

Questions

1. The A* algorithm requires an underestimate of the distance remaining to the goal. What happens if you use 0 as the underestimate for all nodes? Does the algorithm still work correctly? How is the search performed by the algorithm affected?

2. What happens if you use an overestimate of the distance remaining to the goal? Does the algorithm still work correctly? How is the search performed by the algorithm affected?