

Programming theory: Solve a maze

Asked 14 years, 6 months ago Modified 8 years, 5 months ago

Viewed 100k times



What are the possible ways to solve a maze?

Ive got two ideas, but I think they are not very elegant.

76



Base situation: We have a matrix, and the elements in this matrix are ordered in a way that it represents a maze, with one way in, and one out.



My first idea was to send a robot through the maze, following one side, until it's out of the maze. I think this is a very slow solution.

The second one passes through every successive item marked with 1, checks where it can go (up, right, down, left) chooses one way and it continues its path there. This is even slower than the first one.

Of course it's a bit faster if I make the two bots multi-threaded at every junction, but thats also not the best way.

There needs to be better solutions to send a bot through a maze.

EDIT

First: Thanks for the nice answers!

The second part of my question is: What to do in the case if we have a multi-dimensional graph? Are there special practices for that, or is the answer of Justin L. usable for that too?

I think it's not the best way for this case.

The third question:

Which of these maze solver algorithms is/are the fastest? (Purely hypothetically)

algorithm

maze

Share

edited Sep 15, 2014 at 15:51

Improve this question

Follow

community wiki

7 revs, 6 users 67%

Max

7 I usually start from the END and work my way backwards. It works almost every time! – [Joe Phillips](#) Jun 22, 2010 at 22:21

1 You could read cut-the-knot.org/ctk/Mazes.shtml which is a nice intro to mazes – [Dr. belisarius](#) Jun 22, 2010 at 22:26

6 The computer doesn't know what is the start or what is the end, starting from the end doesn't help at all. – [Dominique](#) Jun 22, 2010 at 23:14

1 @Dominique Just a subjective observation: Human maze designers tend to draw only one branch at the maze exit. That's why is usually a little bit faster to search from the finish

- Not the case with Justin L. ascii art, certainly – [Dr. belisarius](#)

Jun 23, 2010 at 10:01 

- 1 Multi-dimensional graphs are also representable by trees =)
– [Justin L.](#) Jun 24, 2010 at 5:17

14 Answers

Sorted by:

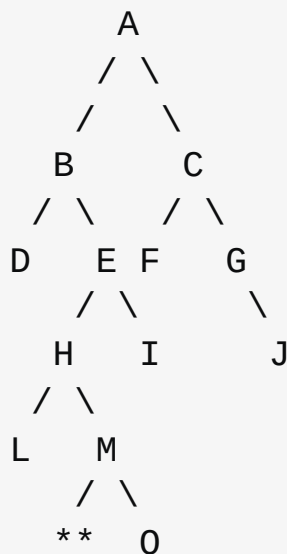
Highest score (default)



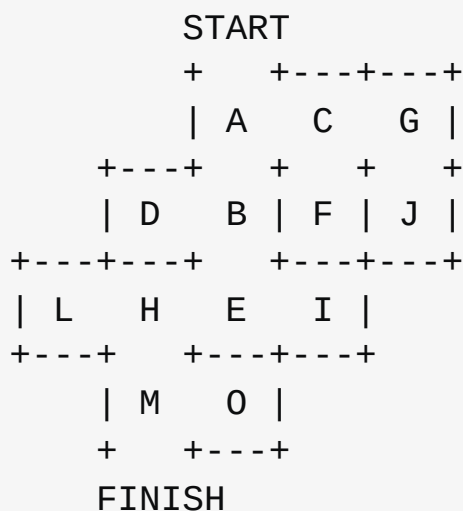
171



You can think of your maze as a tree.



(which could possibly represent)



(ignoring left-right ordering on the tree)

Where each node is a junction of paths. D, I, J, L and O are dead ends, and ** is the goal. Of course, in your actual tree, each node has a possibility of having as many as *three* children.

Your goal is now simply finding what nodes to traverse to find the finish. Any ol' tree search algorithm will do.

Looking at the tree, it's pretty easy to see your correct solution by simply "tracing up" from the ** at the deepest part of the tree:

A B E H M **

Note that this approach becomes *only slightly* more complicated when you have "loops" in your maze (i.e., when it is possible, without backtracing, you re-enter a passage you've already traversed through). Check the comments for one nice solution.

Now, let's look at your first solution you mentioned, applied to this tree.

Your first solution is basically a [Depth-First Search](#), which really isn't that bad. It's actually a pretty good recursive search. Basically, it says, "Always take the rightmost approach first. If nothing is there, backtrack until the first place you can go straight or left, and then repeat.

A depth-first search will search the above tree in this order:

```
A B D (backtrack) E H L (backtrack) M **  
(backtrack) O (backtrack thrice) I  
(backtrack thrice) C F (backtrack) G J
```

Note that you can stop as soon as you find the **.

However, when you actually code a depth-first search, using **recursive programming** makes everything much easier. Even iterative methods work too, and you never have to explicitly program how to backtrack. Check out the linked article for implementations.

Another way of searching a tree is the [Breadth-First](#) solution, which searches through trees by depth. It'd search through the above tree in this order:

```
A (next level) B C (next level) D E F G (next  
level)  
H I J (next level) L M (next level) ** O
```

Note that, due to the nature of a maze, breadth-first has a much higher average amount of nodes it checks.

Breadth-first is easily implementing by having a queue of paths to search, and each iteration popping a path out of a queue, "exploding it" by getting all of the paths that it can turn into after one step, and putting those new paths at the end of the queue. There are no explicit "next level" commands to code, and those were just there to aid in understanding.

In fact, there is a whole [expansive list of ways to search a tree](#). I've just mentioned the two simplest, most

straightforward way.

If your maze is very, very long and deep, and has loops and crazies, and is complicated, I suggest the [A*](#) algorithm, which is the industry standard pathfinding algorithm which combines a Breadth-First search with heuristics...sort of like an "intelligent breadth-first search".

It basically works like this:

1. Put one path in a queue (the path where you only walk one step straight into the maze). A path has a "weight" given by its current length + its straight-line distance from the end (which can be calculated mathematically)
2. Pop the path with the lowest weight from the queue.
3. "Explode" the path into every path that it could be after one step. (i.e., if your path is Right Left Left Right, then your exploded paths are R L L R R and R L L R L, not including illegal ones that go through walls)
4. If one of these paths has the goal, then Victory!
Otherwise:
5. Calculate the weights of the exploded paths, and put all of them back into the queue (not including the original path)
6. Sort the queue by weight, lowest first. Then repeat from Step #2

And that's **A***, which I present specially highlighted because it is more or less the industry standard pathfinding algorithm for *all* applications of pathfinding, including moving from one edge of the map to another while avoiding off-road paths or mountains, etc. It works so well because it uses a *shortest possible distance heuristic*, which gives it its "intelligence". A* is so versatile because, given any problem, if you have a shortest possible distance heuristic available (ours is easy -- the straight line), you can apply it.

BUT it is of great value to note that A* is *not* your only option.

In fact, the [wikipedia category of tree traversal algorithms](#) lists 97 alone! (the best will still be on [this page](#) linked earlier)

Sorry for the length =P (I tend to ramble)

Share Improve this answer

edited Jun 23, 2010 at 6:37

Follow

answered Jun 22, 2010 at 22:40



Justin L.

13.6k ● 5 ● 49 ● 83

3 heh, added ascii maze for fun. hope it aids in understanding how to get a tree from a maze. – Justin L. Jun 22, 2010 at 23:13

5 @Justin: Good answer. If the maze has loops then it becomes a graph. You can still traverse it like a tree if,

instead of recursion, you iterate and use a separate stack structure and check the stack for a node before you visit it to avoid looping. – [Andre Artus](#) Jun 22, 2010 at 23:14

5 Yeah and it's so long nobody will read down to my answer boo hoo – [willoller](#) Jun 23, 2010 at 0:06

2 FWIW, automatic program deadlock checkers are actually equivalent to depth first searches of graphs, where branches represent nondeterminism. It's just that the encoding of the graph – a program – is rather deeply non-trivial.
– [Donal Fellows](#) Jun 23, 2010 at 7:48

1 @JAB a Priority Queue is definitely a good implementation, as the abstraction would take care of sorting, and picking the next node. What I am describing is, basically, a queue used as a Priority Queue. – [Justin L.](#) Jun 24, 2010 at 19:26



Lots of maze-solving algorithms exist:

14

http://en.wikipedia.org/wiki/Maze_solving_algorithm



<http://www.astrolog.org/labyrnth/algrithm.htm#solve>



For a robot, [Tremaux's algorithm](#) looks promising.



Share Improve this answer

answered Jun 22, 2010 at 22:21

Follow



[willoller](#)

7,312 ● 1 ● 37 ● 63



12

An interesting approach, at least I found it interesting, is to use cellular automata. In short a "space" cell surrounded by 3 "wall" cells turns into a "wall" cell. At the



end the only space cells left are the ones on route to the exit.



If you look at the tree Justin put in his answer then you can see that leaf nodes have 3 walls. Prune the tree until you have a path.

Share Improve this answer

edited Jun 23, 2010 at 0:53

Follow

answered Jun 22, 2010 at 23:22



Andre Artus

1,890 ● 15 ● 23

I rather like this elegant solution, and it reminds me a lot of the [Dead End Filling Algorithm](#) posted by willoller – [Justin L.](#)
Jun 23, 2010 at 2:12



5



This is one of my favorite algorithms ever....

- 1) Move forward
- 2) Are you at a wall?
 - 2a) If yes, turn left
- 3) Are you at the finish?
 - 3a) If no, go to 1
 - 3b) If yes, solved

Share Improve this answer

answered Jun 22, 2010 at 22:47

Follow



Nate Noonan

1,371 ● 9 ● 19

8 What happens in this algorithm if the entire maze was a hallway with one right turn? :) – [Mike Sherov](#) Jun 22, 2010 at 23:15

Ahh! you would be trapped forever!! – [willoller](#) Jun 23, 2010 at 0:04

@Mike S: "Marching up and down the square!" – [Andre Artus](#) Jun 23, 2010 at 0:17

2 I think the author was trying to describe "wall following", where you basically just follow the wall on your left (or right) side. – [Gabe](#) Jun 23, 2010 at 4:48

@Gabe, yeah, Intro to CS class, took me 4 hours to come up with the solution on my own. Wall following is not without its caveats or failures... – [Nate Noonan](#) Jun 23, 2010 at 14:18



4

How about building a graph out of your Matrix and using Breadth First Search, Depth First Search or Dijkstras Algorithm?



Share Improve this answer
Follow

answered Jun 22, 2010 at 22:21



[Kungi](#)

1,536 ● 1 ● 19 ● 35



4

I had a similar problem in one of my University Comp. Sci. courses. The solution we came up with was to follow the left hand wall (right hand wall will work just as well). Here is some pseudocode



```
While Not At End
  If Square To Left is open,
    Rotate Left
    Go Forward
  Else
    Rotate Right
  End If
Wend
```

That's basically it. The complex part is keeping track of which direction your facing, and figuring out which grid position is on your left based on this direction. It worked for any test case I put up against it. Interesting enough the Professors solution was something along the lines of:

```
While Not At End
  If Can Go North
    Go North
  ElseIf Can Go East
    Go East
  ElseIf Can Go South
    Go South
  ElseIf Can Go West
    Go West
  EndIf
Wend
```

Which will work well for most simple mazes, but fails on the a maze that looks like the following:

```
SXXXXXXXXXXXXX
  X           X
  X           X
  X           X
XXX          X
X X          X
X XXXXXXXXXX XXXE
```

```
X                      X
XXXXXXXXXXXXXXXXXXXXX
```

With S and E being the start and end.

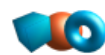
With anything that doesn't follow the wall, you end up having to keep a list of the places you have been, so that you can backtrack if necessary, when you fall into a dead end, and so that you don't get caught in a loop. If you follow the wall, there's no need to keep track of where you've been. Although you won't find the most optimal path through the maze, you will always get through it.

Share Improve this answer

edited Jun 23, 2010 at 1:34

Follow


answered Jun 23, 2010 at 1:26



Kibbee

66.1k ● 28 ● 144 ● 184

-
- 2 I suspect that you did not give the full algorithm, because this one spins/oscillates at the entrance. If you are trying the Wall Follower keep in mind that if the exit is inside an island (surrounded by a path) it will just loop. – [Andre Artus](#) Jun 23, 2010 at 11:41

If you add a "ElseIf Can go Forward, Go Forwards" before your last Else, it will be much faster. – [Coder](#) Aug 27, 2023 at 12:25 



This is a very simple representation to simulate maze in C++ :)

3



```
#ifndef vAlgorithms_Interview_graph_maze_better_h
#define vAlgorithms_Interview_graph_maze_better_h

static const int kMaxRows = 100;
static const int kMaxColumns = 100;

class MazeSolver
{
private:
    char m_matrix[kMaxRows][kMaxColumns]; //matrix
representation of graph
    int rows, cols; //actual rows and columns

    bool m_exit_found;
    int m_exit_row, m_exit_col;
    int m_entrance_row, m_entrance_col;

    struct square //abstraction for data stored in
every verex
    {
        pair<int, int> m_coord; //x and y co-
ordinates of the matrix
        square* m_parent; //to trace the path
backwards

        square() : m_parent(0) {}
    };

    queue<square*> Q;

public:
    MazeSolver(const char* filename)
        : m_exit_found(false)
        , m_exit_row(0)
        , m_exit_col(0)
        , m_entrance_row(0)
        , m_entrance_col(0)
    {}
};
```

Share Improve this answer

answered [Sep 18, 2014 at 0:26](#)

Follow

what is the format of this file

/Users/vshakya/Dropbox/private/graph/maze.txt – [gaurus](#)

Dec 3, 2015 at 10:03



2



Just an idea. Why not throw some bots in there in the monte carlo fashion. Let's call the first generation of bots gen0. We only keep the bots from gen0 that have some continuous roads in this way:

- from the start to some point
- or -from some point to the end



We run a new gen1 of bots in new random dots, then we try to connect the roads of the bots of gen1 with those of gen0 and see if we get a continuous road from start to finish.

So for genn we try to connect with the bots from gen0, gen1, ..., genn-1.

Of course a generation lasts only a feasible finite amount of time.

I don't know if the complexity of the algorithm will prove to be practical for small data sets.

Also the algorithm assumes we know start and finish points.

some good sites for ideas:

<http://citeseerx.ist.psu.edu/>

<http://arxiv.org/>

Share Improve this answer

edited Jun 23, 2010 at 0:13


Follow

answered Jun 23, 2010 at 0:05



19021programmer

435 ● 5 ● 13

-
- 1 Monte Carlo simulation is best suited for problems where computing an optimal solution is not feasible given time constraints, but partial solutions are acceptable. Maze search is solvable in finite time, so unless the maze is 1,000,000 by 1,000,000 squares, I wouldn't recommend this solution for this problem. – [IceArdor](#) Sep 15, 2014 at 15:59 
-



2



If the robot can keep track of its location, so it knows if it has been to a location before, then depth-first search is the obvious algorithm. You can show by an adversarial argument that it is not possible to get better worst-case performance than depth-first search.



If you have available to you techniques that cannot be implemented by robots, then breadth-first search may perform better for many mazes, as may Dijkstra's algorithm for finding the shortest path in a graph.

Share Improve this answer

Follow

answered Jun 23, 2010 at 1:56



Norman Ramsey

202k ● 62 ● 371 ● 541



1

Same answer as all questions on stack-overflow ;)

Use vi!



<http://www.texteditors.org/cgi-bin/wiki.pl?Vi-Maze>



It's truly fascinating to see a text editor solve an ascii-maze, I'm sure the emacs guys have an equivalent ..



Share Improve this answer

answered Jun 23, 2010 at 6:08

Follow



schemathings

47 ● 1 ● 5



1

there are many algorithms, and many *different settings* that specify which algorithm is best. this is just one idea about an interesting setting:



let's assume you have the following properties...



- you move a robot and you want to **minimize its movement**, not its CPU usage.
- that robot can either inspect only its neighbouring cells or look along **corridors** either seeing or not seeing cross-ways.
- it has **GPS**.
- it knows the coordinates of its destination.



then you can design an A.I. which...

- draws a map – every time it receives new information about the maze.
- calculates the minimal known **path** lengths **between** all **unobserved positions** (and itself and the destination).
- can prioritize unobserved positions for inspection based upon **surrounding structures**. (if it is impossible to reach the destination from there anyway...)
- can prioritize unobserved positions for inspection based upon **direction and distance** to destination.
- can prioritize unobserved positions for inspection based upon **experience about collecting information**. (how far can it see on average and how far does it have to walk?)
- can prioritize unobserved positions to **find possible shortcuts**. (experience: are there many **loops**?)

Share Improve this answer

answered Dec 3, 2013 at 19:14

Follow

community wiki
[comonad](#)



This azkaban algorithm might also help you,
<http://journals.analysisofalgorithms.com/2011/08/efficient->

0

maze-solving-approach-with.html

Share Improve this answer

answered Oct 7, 2011 at 9:59

Follow

community wiki
[raghavankl](#)

0

The best way to solve a maze is to use a connectivity algorithm such as union-find which is a quasi-linear time algorithm assuming path compression is done.



Union-Find is a data structure that tells you whether two elements in a set are transitively connected.



To use a union-find data structure to solve a maze, first the neighbor connectivity data is used to build the union-find data structure. Then the union find is compressed. To determine whether the maze is solvable the entrance and exit values are compared. If they have the same value, then they are connected and the maze is solvable. Finally, to find a solution, you start with the entrance and examine the root associated with each of its neighbors. As soon as you find a previously unvisited neighbor with the same root as the current cell, you visit that cell and repeat the process.

The main disadvantage of this approach is that it will not tell you the shortest route through the maze, if there is more than one path.

Share Improve this answer

answered Jul 7, 2016 at 22:36

Follow

community wiki

[Tyler Durden](#)



0



Not specifically for your case, but I've come across several programming contest questions where I found the [Lee's algorithm](#) quite handy to code up quickly. Its not the most efficient for all cases, but is easy to crank out.

Here's [one](#) I hacked up for a contest.



Share Improve this answer

answered Jul 15, 2016 at 18:52



Follow

community wiki

[Jubin Chheda](#)

Please include some sample code, instead of just links.

Thanks! – [JoelC](#) Jul 15, 2016 at 19:17

Actually the sample code is on the github link I posted :)

– [Jubin Chheda](#) Jul 15, 2016 at 20:08

Thanks Jubin. The general idea with Stack Overflow is that links can become invalid, so it's nice to have a more complete/self contained answer available here for posterity! :) Either way, welcome to SO! – [JoelC](#) Jul 15, 2016 at 20:58
