

JULY 2021

UNIVERSITY OF SCIENCE, VNU - HCM

A* Algorithm

Shortest pathfinding algorithm

Prepared by

- 19127216 - Đặng Hoàn Mỹ
- 19127544 - Nguyễn Hoàn Hoài Tâm
- 19127609 - Đinh Quang Tú

Instructed by

Teacher Đinh Bá Tiến

Teacher Hồ Tuấn Thanh

Teacher Nguyễn Lê Hoàng Dũng

Teacher Trương Phước Lộc

Table of Content

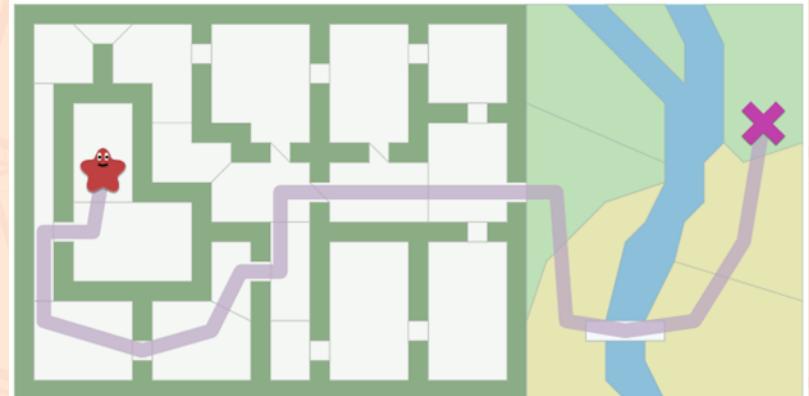
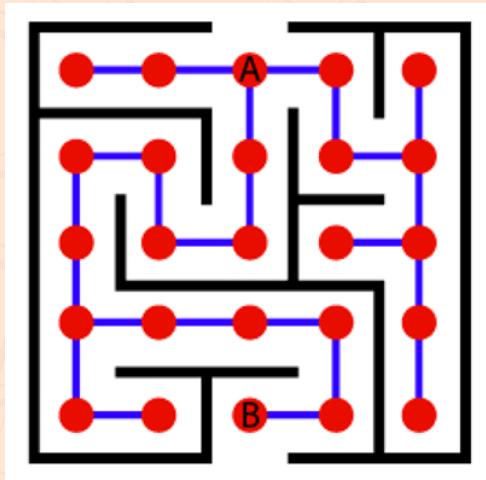
Introduction.....	3
History.....	4
Application.....	5
Data Structure.....	6
Implement Algorithm.....	7
Terminology.....	8
Complexity.....	8
Graph Example.....	9
List of questions for Quiz.....	10
Work Assignments.....	12
References.....	13

Introduction

The maze has become a classic game in the gaming world. There are thousands of games inspired by the maze. And the main task in which is probably mostly the player finding the way out of the maze in the shortest amount of time. There are many ways to complete this game. Many people choose to move forward without pre-calculating the path. However, in this way, when encountering an obstacle, the player will have to waste time to return and find a new path. Some people use pre-calculation of steps to find the most optimal path to the destination. Although pre-calculation can take quite a while, it will help players avoid obstacles.

From the 2 play above we can see that it is important to make a plan before we move. Although they are usually time-consuming initially they are not trapped whereas moving first can be faster but fall into a trap.

The A* algorithm was born for that reason. In games, we often want to find paths from one location to another. We're not only trying to find the shortest distance; we also want save as much time as possible.



History

A achieves better performance by using heuristics to guide its search.*

Created as part of the Shakey project aimed to build a mobile robot that has artificial intelligence to plan its actions, A* was initially designed as a general graph traversal algorithm. Originally Nills Nilsson suggested using the Graph Traverser algorithm for Shakey's path planning. Graph Traverser is guided by a heuristic function $h(n)$, the estimated distance from node n to the goal node: it entirely ignores $g(n)$, the distance from the start node to n . Different from Nills Nilsson, Raphael suggested using the sum, $g(n) + h(n)$. Peter Hart invented the concepts we now call admissibility and consistency of heuristic functions. A* was originally designed for finding least-cost paths when the cost of a path is the sum of its costs, but it has been shown that A* can be used to find optimal paths for any problem satisfying the conditions of a cost algebra. Because of its flexibility and versatility, it can be used in a wide range of contexts.

A* was developed in 1968 to combine heuristic approaches like Greedy Best-First-Search and formal approaches like Dijkstra's Algorithm. It's a little unusual in that heuristic approaches usually give you an approximate way to solve problems without guaranteeing that you get the best answer. However, A* is built on top of the heuristic, and although the heuristic itself does not give you a guarantee, A* can guarantee the shortest path.



Shakey the Robot in its display case at the Computer History Museum.

Application

- A* algorithm is an advanced BFS algorithm that searches for shorter paths first rather than longer paths. A* is optimal as well as a complete algorithm.
- A* Algorithm is one of the best and popular techniques used for pathfinding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently. One example of this is the very popular game- Warcraft III.
- It finds applications in diverse problems, including the problem of parsing using stochastic grammars in NLP.



(a) Navigating from A to B using waypoint graph.

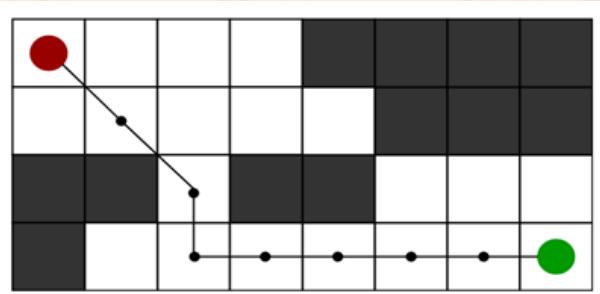


(b) Navigating from A to B on NavMesh.



The Pathfinder: GameplayKit Pathfinding Basics

This game automatically creates mazes then uses GameplayKit to solve them, highlighting the shortest path through each maze.



2D Grid having several obstacles and we start from the source red cell to reach towards a goal green cell

Data Structure

I decided to approach this algorithm by using frontier.

This frontier is a min-heap for a priority queue - it keeps the smallest value at the top. It is suitable for the frontier to keep track of the node that may be traversed in the next steps.

This min-heap priority queue uses the min-heap data structure which supports operations such as insert, minimum, extract-min, decrease-key. In this implementation, the weight of the edges is used to decide the priority of the vertices. Lower the weight, higher the priority and higher the weight, lower the priority.

A* uses the heuristic to reorder the nodes so that it's more likely that the goal node will be encountered sooner. And the priority queue will rearrange that to get the best node that we can visit next.

Implement Algorithm

The main technique for this algorithm is using frontier as a priority queue to put the nodes while waiting. Then get the smallest cost of a path from that and put in the visited to mark up what has been visited. When we catch the goal, we start to reverse the path (the parents of visited nodes) that we saved in the frontier. Then, we return the path if that node if there is no goal, we traverse all the children of that node and put it in the frontier with the formula $f(n) = g(n) + h(n)$.

visited = [] // save the visited node in this, some other paper could mention it as expanded states

path = [] // the path will return

parents = [] // save the parents of the child node every time it finds the path

frontier = <priority_queue> // priority queue which has the smallest on the top

push [heuristics[START], START, -1] // the f value, the state, its parent

while frontier: // loop until the frontier is NOT empty

 heuristic, node, parent = top of frontier // get the smallest value $f(node)$

 pop the top

 if node not in visited:

 visited add node

 parents [node] = parent

 if node is GOAL: // if goal is reached, get the path and return

 while parents [node] ≠ 1:

 path add node

 node = parents [node]

 path add start

 reverse the path

 return path

 traverse all the child of the node using for

 calculate the $f(\text{child}) = g(\text{child}) + h(\text{child})$

 frontier push [$f(\text{child})$, child, node]

Terminology

A is a best-first search starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost.*

Heuristic value is the estimated cost of moving from the current state to the goal state.

- f is the parameter of A* which is the sum of the other parameters G and H and is the least cost from one node to the next node. This parameter is responsible for helping us find the most optimal path from our source to destination. $f = g + h$
- + g is the cost of moving from one node to the other node. This parameter changes for every node as we move up to find the most optimal path.
- + h is the heuristic/estimated path between the current code to the destination node. This cost is not actual but is, in reality, a guess cost that we use to find which could be the most optimal path between our source and destination.

Complexity

One major practical drawback is its $O(b^d)$ space complexity, as it stores all generated nodes in memory.

Time Complexity

The time complexity of A* depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: $O(b^d)$, where b is the branching factor (the average number of successors per state).

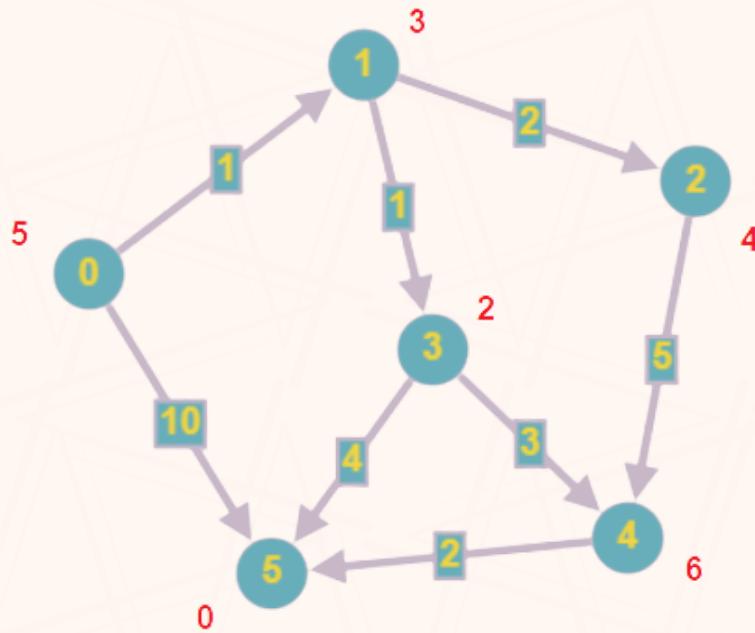
Space Complexity

The space complexity of A* is roughly the same as that of all other graph search algorithms, as it keeps all generated nodes in memory.

$O(|V|) = O(b^d)$, V is the number of vertices in a graph. The worst case is traversing all of the vertices in the graph.

Graph Example

We start from 0 to 5 as a goal.



The formula is $f(n) = g(n) + h(n)$, then we apply to the source: $f(0) = 5 + 0$ and put it to the frontier.

Next, there are 2 ways from 0, we put 0 to the visited and the frontier is 1 5.

- $0 \rightarrow 1$ is $f(1) = 1 + 3 = 4$
- $0 \rightarrow 5$ is $f(5) = 10 + 0 = 10$

Because 1 is on the top of the frontier, we put it in visited, then check the children of 1 because 1 is not the goal, the frontier will be 3 2 5.

- $1 \rightarrow 2$ is $f(2) = (\text{cost from } 0 \text{ to } 2) 3 + 4 = 7$
- $1 \rightarrow 3$ is $f(3) = (\text{cost from } 0 \text{ to } 2) 2 + 2 = 4$

Next, 3 is the top of the frontier, we put it in visited, then check the children because 3 is not the goal, the frontier now will be 5 ($0 \rightarrow 1 \rightarrow 3 \rightarrow 5$), 2, 5 ($0 \rightarrow 5$), 4.

- $3 \rightarrow 4$ is $f(4) = (\text{cost from } 0 \text{ to } 4) 5 + 6 = 11$
- $3 \rightarrow 5$ is $f(5) = (\text{cost from } 0 \text{ to } 5) 6 + 0 = 6$

Now, goal 5 is on the top of the frontier, we reverse the path after getting the parents of the nodes when we approach goal 5. Then we have the path which is **0, 1, 3, 5**.

List of questions for Quiz

1. A* Algorithm does ...

- a. **path-finding and graph traversals, calculate from heuristics and path cost**
- b. finds shortest paths in a directed weighted graph with positive or negative edge weights (Floyd-Warshall Algorithm)
- c. finds the least-cost path from a given initial node to any goal node (B* Algorithm)
- d. explores as far as possible along each branch before backtracking (BFS)

2. In which project and in what year was the A* Algorithm first published?

- a. **Shakey project (1968)**
- b. Plankalkül programming language (1945) - BFS
- c. Richard Korf (1985) - IDA*
- d. Harpy Speech Recognition System (1976) - Beam

3. What is heuristic value?

- a. **the estimated cost from the current vertex to the goal vertex**
- b. the smallest cost from the start vertex to the goal vertex
- c. the cost from the root node to the current node
- d. the value for the best solution to a given problem

4. The A* Algorithm is developed based on:

- a. **Dijkstra's Algorithm**
- b. BFS Algorithm
- c. IDA* Algorithm
- d. Floyd-Warshall Algorithm

List of questions for Quiz

4. What is the TIME and SPACE complexity of A* Algorithm?

- a. **O(b^d)**
- b. O(h)
- c. O(h) - O(|V|)
- d. O(V) - O(h)

5. What is NOT an application of A* Algorithm?

- a. **Google Maps**
- b. Warcraft III
- c. Graph Traversal
- d. n-puzzle Solving

6. What formula does the A* Algorithm calculate the path to the next node?

- a. **f(n) = g(n) + h(n)**
- b. f(n) = g(n) / h(n)
- c. g(n) = f(n) + h(n)
- d. h(n) = g(n) + g(n)

Work Assignments

- Find out the introduction (history...)
- Some terminologies (heuristic...)
- Application

Quang Tú

- Notes for implementation, time/ space complexity
- Write source code (Graph)
- Graph demo (draw -> result)
- Find the questions for Quiz and put the questions on Kahoot
- Report composer and PowerPoint arranger
- Template for PowerPoint and report

Hoàn Mỹ

Hoài Tâm

References

Being grateful to these websites, videos and our courage to complete acknowledging A Algorithm.*

- A. (n.d.). GitHub - aliwalker/graph_search_algorithm: Some graph search algorithm. An assignment of Introduction to AI. GitHub. Retrieved July 31, 2021, from https://github.com/aliwalker/graph_search_algorithm
- Chatterjee, M. (2021, June 11). A* Search Algorithm in Artificial Intelligence (AI). GreatLearning Blog: Free Resources What Matters to Shape Your Career! <https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/>
- de Souza, J. V. A. (2014, August 28). Inteligent Control - A* Algorithm. Dudevictor GitHub IO. <https://jsouza.dev/en/projects/controle-inteligente/?hcb=1>
- Edpresso Team. (2021, July 30). What is the A* algorithm? Eduative: Interactive Courses for Software Developers. <https://www.educative.io/edpresso/what-is-the-a-star-algorithm>
- Elgabry, O. (2018, May 28). Path Finding Algorithms - OmarElgabry's Blog. Medium. <https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40>
- GameplayKit Programming Guide: Pathfinding. (2016, March 21). Apple Developer. https://developer.apple.com/library/archive/documentation/General/Conceptual/GameplayKit_Guide/Pathfinding.html?hcb=1
- GeeksforGeeks. (2021, July 9). A* Search Algorithm. <https://www.geeksforgeeks.org/a-search-algorithm/>
- Tim, T. W. (2020, July 16). A* Pathfinding Visualization Tutorial - Python A* Path Finding Tutorial. YouTube. <https://www.youtube.com/watch?v=JtiK0DOel4A&feature=youtu.be>

References

Being grateful to these websites, videos and our courage to complete acknowledging A Algorithm.*

- Leo Willyanto Santoso, Alexander Setiawan, Andre K. Prajogo. (2010). Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path. http://fportfolio.petra.ac.id/user_files/04-021/MICEEI2010.pdf
- Red Blob Games: Introduction to A*. (n.d.). Red Blob Games. Retrieved July 31, 2021, from <https://www.redblobgames.com/pathfinding/a-star/introduction.html>
- Wikipedia contributors. (2021a, May 29). A* search algorithm. Wikipedia. https://en.wikipedia.org/wiki/A*_search_algorithm
- Wikipedia contributors. (2021b, July 26). Priority queue. Wikipedia. https://en.wikipedia.org/wiki/Priority_queue