**DESIGN AND ANALYSIS OF ALGORITHMS**

HOME ASSIGNMENT- DESIGN

**A\* Algorithm**

****

**Group No : TY-25**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll No.** | **Gr No.** |
| **Shreya Hirapurkar** | **22** | **11810820** |
| **Vinayak Kokane** | **58** | **11810499** |
| **Dhanraj Kore** | **60** | **11810970** |
| **Vaishnavi Kulkarni** | **63** | **11920122** |

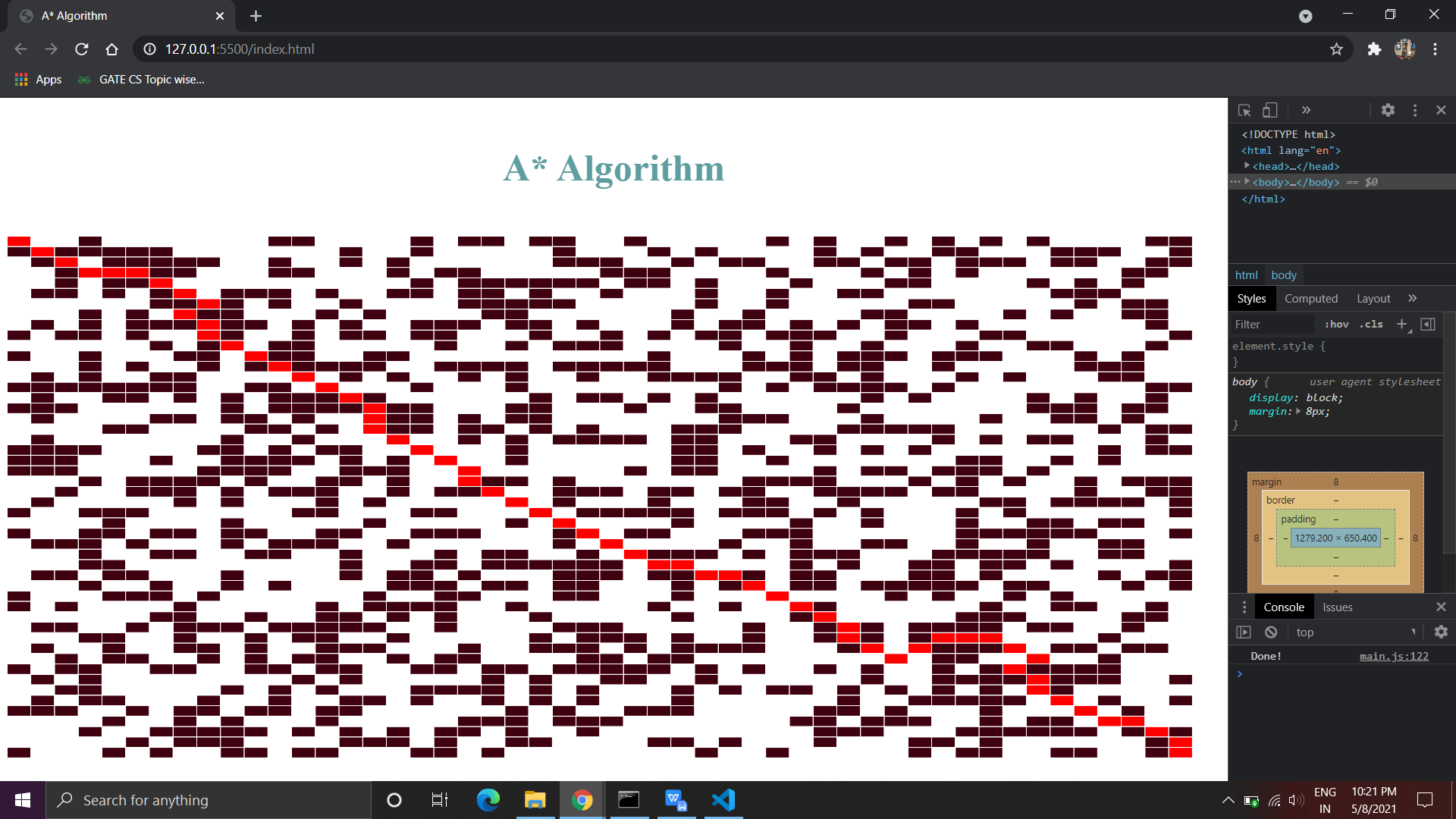
**HA Design Implementation link : [A\* Algorithm Implementation](https://priceless-galileo-42985b.netlify.app/)**

* **Introduction**:

A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. Informally speaking, A\* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. Motivation behind this algorithm is to approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances. And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently.

We can consider a 2D Grid having several obstacles and we start from a source cell (upper left corner) to reach towards a goal cell (bottom right corner).

* **Visualization of Algorithm**:



* **Approach**:
* Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A\* Search Algorithm comes to the rescue.
* What A\* Search Algorithm does is that at each step it picks the node according to a value ‘**f**’ which is a parameter equal to the sum of two other parameters ‘**g**’ and ‘**h**’. At each step it picks the node or cell having the lowest ‘**f**’, and process that node or cell.
* We define ‘**g**’ and ‘**h**’ as follows, ‘**g’** is the movement cost to move from the starting point to a given square on the grid, following the path generated to get there and ‘**h’** is the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of obstacles can be in the way. There can be many ways to calculate this h.
* **Ways to calculate heuristic**:

We can do two things.  
A) Either calculate the exact value of h (which is certainly time consuming).  
B ) Approximate the value of h using some heuristics (less time consuming).

Let’s See both the methods:

* **Exact Heuristics:** We can find exact values of h, but that is generally very time consuming.

Below are some of the methods to calculate the exact value of h.

1) Pre-compute the distance between each pair of cells before running the A\* Search Algorithm.

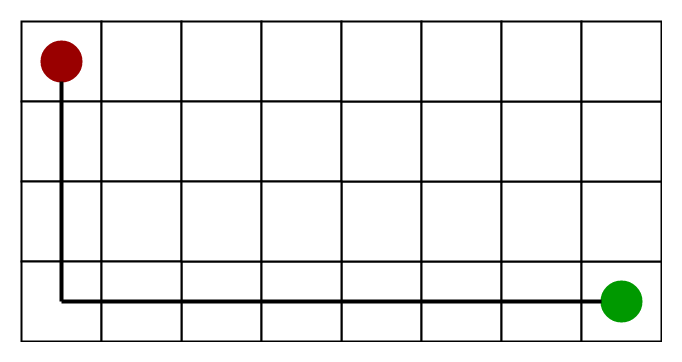
2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the [distance formula/Euclidean Distance](https://en.wikipedia.org/wiki/Euclidean_distance).

* **Approximation Heuristics**: There are generally three approximation heuristics to calculate h.

1. Manhattan Distance:It is nothing but the sum of absolute values of differences in the goal’s x and y coordinates and the current cell’s x and y coordinates respectively, i.e.,

*h = abs (current\_cell.x - goal.x) + abs (current\_cell.y - goal.y)*

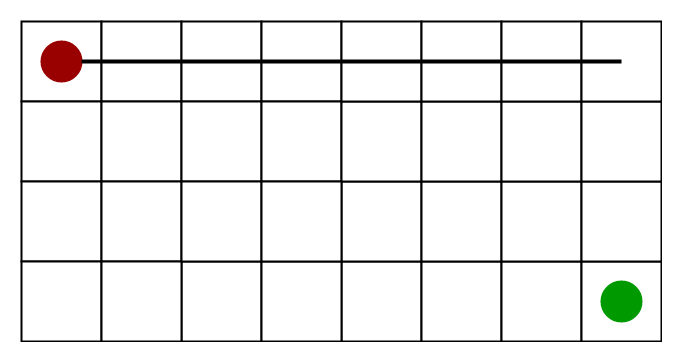
When to use this heuristic? – When we are allowed to move only in four directions only (right, left, top, bottom).



1. Diagonal Distance: It is nothing but the maximum of absolute values of differences in the goal’s x and y coordinates and the current cell’s x and y coordinates respectively, i.e.,

*h = max {abs(current\_cell.x – goal.x),abs(current\_cell.y – goal.y) }*

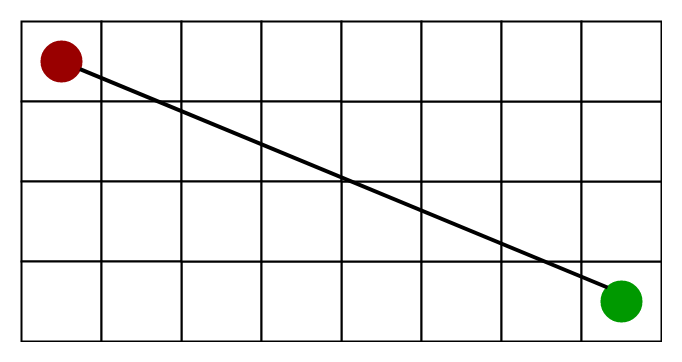
When to use this heuristic? – When we are allowed to move in eight directions only (similar to a move of a King in Chess).



1. Euclidean Distance-: It is nothing but the distance between the current cell and the goal cell using the distance formula

*h = sqrt ( (current\_cell.x – goal.x)^2 + (current\_cell.y – goal.y)^2 )*

When to use this heuristic? – When we are allowed to move in any directions.



* **Algorithm**:

We create two lists – Open List and Closed List.

1.Initialize the open list

2.Initialize the closed list, put the starting node on the open list (we can set its ‘f’ as 0.)

3. while the open list is not empty

a) find the node with the least ‘f’ on the open list, call it ‘q’.

b) pop ‘q’ off the open list.

c) generate q's 8 successors and set their parents to q.

d) for each successor

i) if successor is the goal, stop search.

successor.g = q.g + distance between successor and q

successor.h = distance from goal to successor

successor.f = successor.g + successor.h

ii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor.

iii) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor, otherwise, add the node to the open list.

end (for loop)

e) push q on the closed list

end (while loop)

* **Time Complexity**: Considering a graph, it may take us to travel all the edge to reach the destination cell from the source cell. For example, consider a graph where source and destination nodes are connected by a series of edges, like – 0(source) –>1 –> 2 –> 3 (target). So the worst case time complexity is O(E), where E is the number of edges in the graph.
* **Space Complexity**: In the worst case we can have all the edges inside the open list, so required auxiliary space in worst case is O(V), where V is the total number of vertices.
* **Limitations**: Although being the best pathfinding algorithm around, A\* Search Algorithm doesn’t produce the shortest path always, as it relies heavily on heuristics / approximations to calculate – h.