**EXPERIMENT 6 - A\* Algorithm**

**Aim:**

Write a program to implement A\* algorithm.

**Theoretical Background:**

A\* is a searching algorithm that is used to find the shortest path between an initial and a final point.

It is a handy algorithm that is often used in map traversal to find the shortest path to be taken. It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

Another aspect that makes A\* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithm can take the path with the least cost and find the best route in terms of distance and time.

A\* search algorithm is a simple and efficient search algorithm that can be used to find the optimal path between two nodes in a graph. It will be used for the shortest path finding. It is an extension of Dijkstra’s shortest path algorithm (Dijkstra’s algorithm). The extension here is that, instead of using a priority queue to store all the elements, we use heaps(binary tree) to store them. The A\* search algorithm also uses a heuristic function that provides additional information regarding how far away from the goal node we are.

**Algorithm:**

Step 1: Start.

Step 2: Start with OPEN containing only the initial node. Set g=0, h’, f’=g+h’, Set closed to empty list.

Step 3: Until a goal is found repeat:

Step 3.1: If there are no nodes on OPEN report failure, otherwise pick a node with lowest f’.

Step 3.2: Call t BESTNODE. Move it from open to closed.

Step 3.3: If BESTNODE is the goal, report a solution.

Step 3.4: Otherwise generate successors ,for each SUCCESSOR, do the following.

Step 3.4.1: Set successor point back to BESTNODE. These backwards links help to recover the path once a solution is found.

Step 3.4.2: G(SUCCESSOR)=g(BESTNODE)+cost of getting from BESTNODE to SUCCESSOR.

Step 3.4.3: If SUCCESSOR is in OPEN, call that node OLD. Throw SUCCESSOR away and add OLD to BESTNODE ‘s successors. Compare 2 paths, via its current parent to SUCCESSOR ,via BESTNODE. Set the least cost path.

Step 3.4.4: If it was not on OPEN, see if it is on CLOSED.If so, call the node on CLOSED OLD, and add OLD to the list of BESTNODE’s successors. Reset the path to least cost as in Step 3.4.3. If we get a better path, propagate the improvements to OLD’s successors.

Step 3.4.5: If SUCCESSOR was not in either OPEN or CLOSED, put it on OPEN, and add it to BESTNODE ‘s successors. Compute f’(SUCCESSOR)=g(SUCCESSOR)+h’(SUCCESSOR).

Step 4: Stop.

**Conclusion:**

Python program to implement A\* algorithm is successfully implemented.