Ex.No.5B
Date:11/3/2021

A* ALGORITHM

Aim: To implement A* search using python

Methodology:
1. Initialize the open list, put the starting node on the open list (you can leave its f at zero).
2. Initialize the closed list
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.

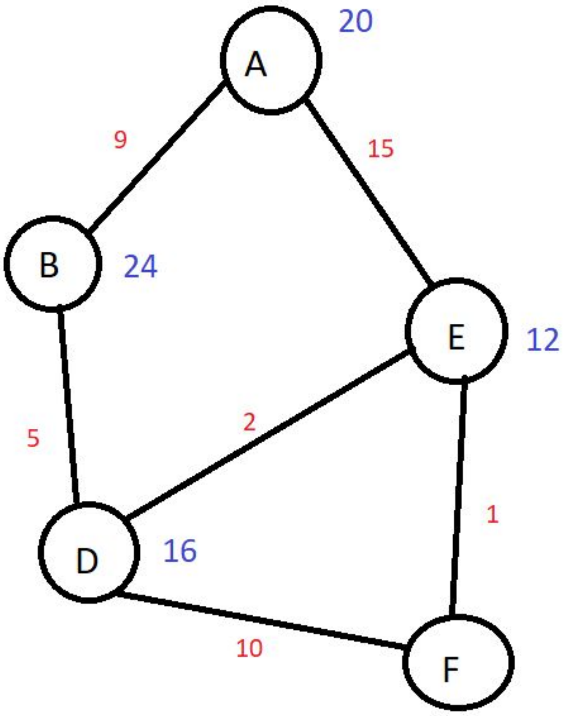
      iii) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list end (for loop)

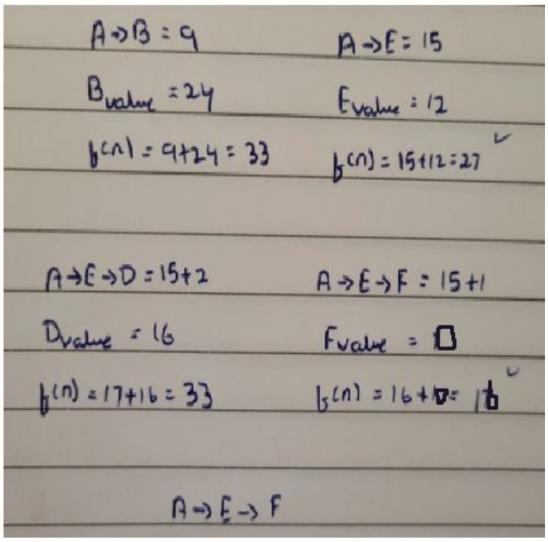
   e) push q on the closed list.
      end (while loop)

Code:
```
tree = {'A': [['B', 9], ['E', 15]],
      'B': [['A', 9], ['D', 5]],
      'D': [['B', 5], ['E', 2], ['F', 10]],
      'E': [['A', 15], ['D', 2], ['F', 1]],
      'F': [['E', 1], ['D', 10]]
      }

heuristic = {'A': 20, 'B': 24, 'D': 16, 'E': 12, 'F': 0}
cost = {'A': 0}
def AStarSearch():
  global tree, heuristic
  closed = [] # closed nodes
```

```
  opened = [['A', 20]] # opened nodes
  '''find the visited nodes'''
  while True:
    fn = [i[1] for i in opened] # fn = f(n) = g(n) + h(n)
    chosen_index = fn.index(min(fn))
    node = opened[chosen_index][0] # current node
    closed.append(opened[chosen_index])
    del opened[chosen_index]
    if closed[-1][0] == 'F': # break the loop if node G has been found
     break
    for item in tree[node]:
      if item[0] in [closed_item[0] for closed_item in closed]:
        continue
      cost.update({item[0]: cost[node] + item[1]}) # add nodes to cost dictionary
      fn_node = cost[node] + heuristic[item[0]] + item[1] # calculate f(n) of current node
      temp = [item[0], fn_node]
      opened.append(temp) # store f(n) of current node in array opened
  '''find optimal sequence'''
  trace_node = 'F' # correct optimal tracing node, initialize as node G
  optimal_sequence = ['F'] # optimal node sequence
  for i in range(len(closed)-2, -1, -1):
    check_node = closed[i][0] # current node
    if trace_node in [children[0] for children in tree[check_node]]:
      children_costs = [temp[1] for temp in tree[check_node]]
      children_nodes = [temp[0] for temp in tree[check_node]]
      if cost[check_node] + children_costs[children_nodes.index(trace_node)] ==
cost[trace_node]:
        optimal_sequence.append(check_node)
        trace_node = check_node
  optimal_sequence.reverse()
  return closed, optimal_sequence
if __name__ == '__main__':
  visited_nodes, optimal_nodes = AStarSearch()

  print('Ideal-Nodes', 'Cost')
  for i, j in visited_nodes:
    print('  ', i,'      ',j)

  #print('visited nodes: ' + str(visited_nodes))
  print('Traversal: -')
  for i in optimal_nodes:

    print(i, end=' -> ')
```

Graph:





Output:

```
Ideal-Nodes Cost
   A        20
   E        27
   F        16
Traversal: -
A -> E -> F ->
```

Result: We have successfully studied and implemented A*algorithm.