A* and BEST FIRST SEARCH

Team:

Aditya Arandhara(RA1911003010376)

Arunabh Kalita(RA1911003010375)

Parth Moghekar(RA1911003010368)

Saachi Almeida(RA1911003010370)

Shubham Goel(RA1911003010379)

A* Search Algorithm

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Given a MxN matrix where each element can either be 0 or 1. We need to find the shortest path between a given source cell to a destination cell. The path can only be created out of a cell if its value is 1.

Expected time complexity is O(MN).

ALGORITHM

1. **We start from the source cell and calls BFS procedure.**
2. **We maintain a queue to store the coordinates of the matrix and initialize it with the source cell.**
3. **We also maintain a Boolean array visited of same size as our input matrix and initialize all its elements to false.**
4. **We LOOP till queue is not empty**
5. **Dequeue front cell from the queue**
6. **Return if the destination coordinates have reached.**
7. **For each of its four adjacent cells, if the value is 1 and they are not visited yet, we enqueue it in the queue and also mark them as visited.**

```
# set (i, j) cell as visited
  visited[i][j] = 1

 # go to the bottom cell
  if isSafe(mat, visited, i + 1, j):
     min_dist = findShortestPath(mat, visited, i + 1, j, dest, min_dist, dist + 1)

 # go to the right cell
  if isSafe(mat, visited, i, j + 1):
     min_dist = findShortestPath(mat, visited, i, j + 1, dest, min_dist, dist + 1)

  # go to the top cell
  if isSafe(mat, visited, i - 1, j):
     min_dist = findShortestPath(mat, visited, i - 1, j, dest, min_dist, dist + 1)

  # go to the left cell
  if isSafe(mat, visited, i, j - 1):
     min_dist = findShortestPath(mat, visited, i, j - 1, dest, min_dist, dist + 1)

 # backtrack: remove (i, j) from the visited matrix
  visited[i][j] = 0

  return min_dist

print("Destination cannot be reached from source")
```

CODE

```
import sys

def isSafe(mat, visited, x, y):
    return 0 <= x < len(mat) and 0 <= y < len(mat[0]) and \
      not (mat[x][y] == 0 or visited[x][y])
def findShortestPath(mat, visited, i, j, dest, min_dist=sys.maxsize, dist=0):



    if (i, j) == dest:
        return min(dist, min_dist)



    visited[i][j] = 1
```

```
if isSafe(mat, visited, i + 1, j):
        min_dist = findShortestPath(mat, visited, i + 1, j, dest, min_dist, dist + 1)


    if isSafe(mat, visited, i, j + 1):
        min_dist = findShortestPath(mat, visited, i, j + 1, dest, min_dist, dist + 1)


    if isSafe(mat, visited, i - 1, j):
        min_dist = findShortestPath(mat, visited, i - 1, j, dest, min_dist, dist + 1)


    if isSafe(mat, visited, i, j - 1):
        min_dist = findShortestPath(mat, visited, i, j - 1, dest, min_dist, dist + 1)


    visited[i][j] = 0

    return min_dist
```

```
def findShortestPathLength(mat, src, dest):


 i, j = src


 x, y = dest


 if not mat or len(mat) == 0 or mat[i][j] == 0 or mat[x][y] == 0:
   return -1


 (M, N) = (len(mat), len(mat[0]))


 visited = [[False for _ in range(N)] for _ in range(M)]

 min_dist = findShortestPath(mat, visited, i, j, dest)

 if min_dist != sys.maxsize:
     return min_dist
 else:
     return -1
```

```
if __name__ == '__main__':


    mat = [
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
        [0, 1, 1, 1, 1, 1, 0, 1, 0, 1],
        [0, 0, 1, 0, 1, 1, 1, 0, 0, 1],
        [1, 0, 1, 1, 1, 0, 1, 1, 0, 1],
        [0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
        [1, 0, 1, 1, 1, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
        [0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
        [0, 0, 1, 0, 0, 1, 1, 0, 0, 1]
    ]
```

```
src = (0, 0)
    dest = (7, 5)
    min_dist = findShortestPathLength(mat, src, dest)
    if min_dist != -1:
        print("The shortest path from source to destination has length", min_dist)
    else:
        print("Destination cannot be reached from source")
```

OUTPUT

```
dest = (7, 5)
min_dist = findShortestPathLength(mat, src, dest)
if min_dist != -1:
    print("The shortest path from source to destination has length", min_dist)
else:
    print("Destination cannot be reached from source")
```

```
The shortest path from source to destination has length 12
```

In [11]:
```
#BFS
from queue import PriorityQueue
```

Best First Search

Best first search is a traversal technique that decides which node is to be visited next by checking which node is the most promising one and then check it. For this it uses an evaluation function to decide the traversal.

This best first search technique of tree traversal comes under the category of heuristic search or informed search technique.

The cost of nodes is stored in a priority queue. This makes implementation of best-first search is same as that of breadth First search. We will use the priorityqueue just like we use a queue for BFS.

Algorithm

```
1) Create an empty PriorityQueue
        PriorityQueue pq;
    2) Insert "start" in pq.
        pq.insert(start)
    3) Until PriorityQueue is empty
        u = PriorityQueue.DeleteMin
        If u is the goal
        Exit
        Else
        Foreach neighbor v of u
                If v "Unvisited"
                Mark v "Visited"
                pq.insert(v)
        Mark u "Examined"
End procedure
```

CODE

```
from queue import PriorityQueue
v = 5
graph = [[] for i in range(v)]
def best_first_search(source, target, n):
    visited = [0] * n
    visited[0] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()
```

```
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
addedge(0, 1, 5)
addedge(0, 2, 1)
addedge(2, 3, 2)
addedge(1, 4, 1)
addedge(3, 4, 2)
source = 0
target = 4
best_first_search(source, target, v)
```

OUTPUT

```
addedge(3, 4, 2)
source = 0
target = 4
best_first_search(source, target, v)
```

```
0 2 3 4
```

THANK YOU!