AI - LAB Experiment -2

- ASHISH KUMAR
- 2K18/SE/041

Ques : Write a program to implement A* search algorithm.

CODE:

```
#include<bits/stdc++.h>
#include<iostream>
#include<algorithm>
using namespace std;

#define ROW 9
#define COL 10


typedef pair<int, int> Pair;

typedef pair<double, pair<int, int>> pPair;

struct cell
{

    int parent_i, parent_j;
    // f = g + h
    double f, g, h;
};

// A Utility Function to check whether given cell (row, col)
// is a valid cell or not.
bool isValid(int row, int col)
{
    // Returns true if row number and column number
    // is in range
    return (row >= 0) && (row < ROW) &&
           (col >= 0) && (col < COL);
}
```

```
// A Utility Function to check whether the given cell is
// blocked or not
bool isUnBlocked(int grid[][COL], int row, int col)
{
    // Returns true if the cell is not blocked else false
    if (grid[row][col] == 1)
        return (true);
    else
        return (false);
}

// A Utility Function to check whether destination cell has
// been reached or not
bool isDestination(int row, int col, Pair dest)
{
    if (row == dest.first && col == dest.second)
        return (true);
    else
        return (false);
}

// A Utility Function to calculate the 'h' heuristics.
double calculateHValue(int row, int col, Pair dest)
{
    // Return using the distance formula
    return ((double)sqrt ((row-dest.first)*(row-dest.first)
                          + (col-dest.second)*(col-dest.second)));
}

// A Utility Function to trace the path from the source
// to destination
void tracePath(cell cellDetails[][COL], Pair dest)
{
    printf ("\nThe Path is ");
    int row = dest.first;
    int col = dest.second;

    stack<Pair> Path;

    while (!(cellDetails[row][col].parent_i == row
             && cellDetails[row][col].parent_j == col ))
    {
        Path.push (make_pair (row, col));
        int temp_row = cellDetails[row][col].parent_i;
        int temp_col = cellDetails[row][col].parent_j;
```

```
        row = temp_row;
        col = temp_col;
    }

    Path.push (make_pair (row, col));
    while (!Path.empty())
    {
        pair<int,int> p = Path.top();
        Path.pop();
        printf("-> (%d,%d) ",p.first,p.second);
    }

    return;
}

// A Function to find the shortest path between
// a given source cell to a destination cell according
// to A* Search Algorithm
void aStarSearch(int grid[][COL], Pair src, Pair dest)
{
    // If the source is out of range
    if (isValid (src.first, src.second) == false)
    {
        printf ("Source is invalid\n");
        return;
    }

    // If the destination is out of range
    if (isValid (dest.first, dest.second) == false)
    {
        printf ("Destination is invalid\n");
        return;
    }

    // Either the source or the destination is blocked
    if (isUnBlocked(grid, src.first, src.second) == false ||
            isUnBlocked(grid, dest.first, dest.second) == false)
    {
        printf ("Source or the destination is blocked\n");
        return;
    }

    if (isDestination(src.first, src.second, dest) == true)
    {
        printf ("We are already at the destination\n");
        return;
```

```
}

bool closedList[ROW][COL];
memset(closedList, false, sizeof (closedList));


cell cellDetails[ROW][COL];

int i, j;

for (i=0; i<ROW; i++)
{
   for (j=0; j<COL; j++)
   {
      cellDetails[i][j].f = FLT_MAX;
      cellDetails[i][j].g = FLT_MAX;
      cellDetails[i][j].h = FLT_MAX;
      cellDetails[i][j].parent_i = -1;
      cellDetails[i][j].parent_j = -1;
   }
}

// Initialising the parameters of the starting node
i = src.first, j = src.second;
cellDetails[i][j].f = 0.0;
cellDetails[i][j].g = 0.0;
cellDetails[i][j].h = 0.0;
cellDetails[i][j].parent_i = i;
cellDetails[i][j].parent_j = j;


set<pPair> openList;

openList.insert(make_pair (0.0, make_pair (i, j)));


bool foundDest = false;

while (!openList.empty())
{
   pPair p = *openList.begin();

         openList.erase(openList.begin());

   i = p.second.first;
   j = p.second.second;
```

```
    closedList[i][j] = true;



    double gNew, hNew, fNew;

//----------- 1st Successor (North) ------------


    if (isValid(i-1, j) == true)
    {
       // If the destination cell is the same as the
       // current successor
       if (isDestination(i-1, j, dest) == true)
       {
          // Set the Parent of the destination cell
          cellDetails[i-1][j].parent_i = i;
          cellDetails[i-1][j].parent_j = j;
          printf ("The destination cell is found\n");
          tracePath (cellDetails, dest);
          foundDest = true;
          return;
       }
       else if (closedList[i-1][j] == false &&
                isUnBlocked(grid, i-1, j) == true)
       {
          gNew = cellDetails[i][j].g + 1.0;
          hNew = calculateHValue (i-1, j, dest);
          fNew = gNew + hNew;

          if (cellDetails[i-1][j].f == FLT_MAX ||
                cellDetails[i-1][j].f > fNew)
          {
             openList.insert( make_pair(fNew,
                                make_pair(i-1, j)));

             cellDetails[i-1][j].f = fNew;
             cellDetails[i-1][j].g = gNew;
             cellDetails[i-1][j].h = hNew;
             cellDetails[i-1][j].parent_i = i;
             cellDetails[i-1][j].parent_j = j;
          }
       }
    }
```

```
//----------- 2nd Successor (South) ------------

      // Only process this cell if this is a valid one
      if (isValid(i+1, j) == true)
      {
         // If the destination cell is the same as the
         // current successor
         if (isDestination(i+1, j, dest) == true)
         {
            // Set the Parent of the destination cell
            cellDetails[i+1][j].parent_i = i;
            cellDetails[i+1][j].parent_j = j;
            printf("The destination cell is found\n");
            tracePath(cellDetails, dest);
            foundDest = true;
            return;
         }


         else if (closedList[i+1][j] == false &&
                  isUnBlocked(grid, i+1, j) == true)
         {
            gNew = cellDetails[i][j].g + 1.0;
            hNew = calculateHValue(i+1, j, dest);
            fNew = gNew + hNew;


            if (cellDetails[i+1][j].f == FLT_MAX ||
                  cellDetails[i+1][j].f > fNew)
            {
               openList.insert( make_pair (fNew, make_pair (i+1, j)));
               // Update the details of this cell
               cellDetails[i+1][j].f = fNew;
               cellDetails[i+1][j].g = gNew;
               cellDetails[i+1][j].h = hNew;
               cellDetails[i+1][j].parent_i = i;
               cellDetails[i+1][j].parent_j = j;
            }
         }
      }
```

```
//----------- 3rd Successor (East) ------------


      if (isValid (i, j+1) == true)
      {
         // If the destination cell is the same as the
         // current successor
         if (isDestination(i, j+1, dest) == true)
         {
            // Set the Parent of the destination cell
            cellDetails[i][j+1].parent_i = i;
            cellDetails[i][j+1].parent_j = j;
            printf("The destination cell is found\n");
            tracePath(cellDetails, dest);
            foundDest = true;
            return;
         }

         else if (closedList[i][j+1] == false &&
                  isUnBlocked (grid, i, j+1) == true)
         {
            gNew = cellDetails[i][j].g + 1.0;
            hNew = calculateHValue (i, j+1, dest);
            fNew = gNew + hNew;

            if (cellDetails[i][j+1].f == FLT_MAX ||
                  cellDetails[i][j+1].f > fNew)
            {
               openList.insert( make_pair(fNew,
                                 make_pair (i, j+1)));

               // Update the details of this cell
               cellDetails[i][j+1].f = fNew;
               cellDetails[i][j+1].g = gNew;
               cellDetails[i][j+1].h = hNew;
               cellDetails[i][j+1].parent_i = i;
               cellDetails[i][j+1].parent_j = j;
            }
         }
      }
```

```
//----------- 4th Successor (West) ------------

        if (isValid(i, j-1) == true)
    {
        // If the destination cell is the same as the
        // current successor
        if (isDestination(i, j-1, dest) == true)
        {
            // Set the Parent of the destination cell
            cellDetails[i][j-1].parent_i = i;
            cellDetails[i][j-1].parent_j = j;
            printf("The destination cell is found\n");
            tracePath(cellDetails, dest);
            foundDest = true;
            return;
        }

        // If the successor is already on the closed
        // list or if it is blocked, then ignore it.
        // Else do the following
        else if (closedList[i][j-1] == false &&
                isUnBlocked(grid, i, j-1) == true)
        {
            gNew = cellDetails[i][j].g + 1.0;
            hNew = calculateHValue(i, j-1, dest);
            fNew = gNew + hNew;


            if (cellDetails[i][j-1].f == FLT_MAX ||
                    cellDetails[i][j-1].f > fNew)
            {
                openList.insert( make_pair (fNew,
                                make_pair (i, j-1)));

                // Update the details of this cell
                cellDetails[i][j-1].f = fNew;
                cellDetails[i][j-1].g = gNew;
                cellDetails[i][j-1].h = hNew;
                cellDetails[i][j-1].parent_i = i;
                cellDetails[i][j-1].parent_j = j;
            }
        }
    }
```

```
//----------- 5th Successor (North-East) ------------

    {
        // If the destination cell is the same as the
        // current successor
        if (isDestination(i-1, j+1, dest) == true)
        {
            // Set the Parent of the destination cell
            cellDetails[i-1][j+1].parent_i = i;
            cellDetails[i-1][j+1].parent_j = j;
            printf ("The destination cell is found\n");
            tracePath (cellDetails, dest);
            foundDest = true;
            return;
        }

        // If the successor is already on the closed
        // list or if it is blocked, then ignore it.
        // Else do the following
        else if (closedList[i-1][j+1] == false &&
                isUnBlocked(grid, i-1, j+1) == true)
        {
            gNew = cellDetails[i][j].g + 1.414;
            hNew = calculateHValue(i-1, j+1, dest);
            fNew = gNew + hNew;

            if (cellDetails[i-1][j+1].f == FLT_MAX ||
                    cellDetails[i-1][j+1].f > fNew)
            {
                openList.insert( make_pair (fNew,
                            make_pair(i-1, j+1)));

                // Update the details of this cell
                cellDetails[i-1][j+1].f = fNew;
                cellDetails[i-1][j+1].g = gNew;
                cellDetails[i-1][j+1].h = hNew;
                cellDetails[i-1][j+1].parent_i = i;
                cellDetails[i-1][j+1].parent_j = j;
            }
        }
    }
```

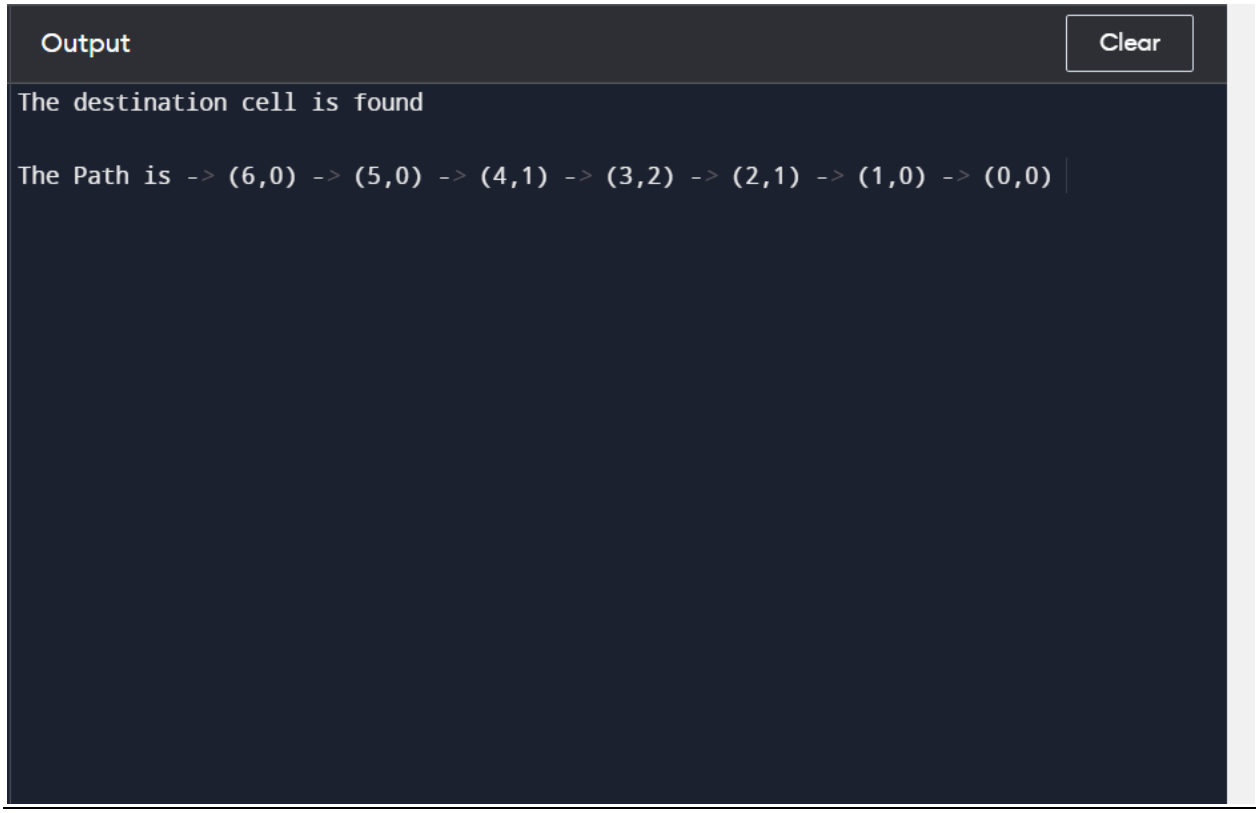
```
//----------- 6th Successor (North-West) ------------


        if (isValid (i-1, j-1) == true)
        {
            // If the destination cell is the same as the
            // current successor
            if (isDestination (i-1, j-1, dest) == true)
            {
                // Set the Parent of the destination cell
                cellDetails[i-1][j-1].parent_i = i;
                cellDetails[i-1][j-1].parent_j = j;
                printf ("The destination cell is found\n");
                tracePath (cellDetails, dest);
                foundDest = true;
                return;
            }

            // If the successor is already on the closed
            // list or if it is blocked, then ignore it.
            // Else do the following
            else if (closedList[i-1][j-1] == false &&
                    isUnBlocked(grid, i-1, j-1) == true)
            {
                gNew = cellDetails[i][j].g + 1.414;
                hNew = calculateHValue(i-1, j-1, dest);
                fNew = gNew + hNew;


                if (cellDetails[i-1][j-1].f == FLT_MAX ||
                        cellDetails[i-1][j-1].f > fNew)
                {
                    openList.insert( make_pair (fNew, make_pair (i-1, j-1)));
                    // Update the details of this cell
                    cellDetails[i-1][j-1].f = fNew;
                    cellDetails[i-1][j-1].g = gNew;
                    cellDetails[i-1][j-1].h = hNew;
                    cellDetails[i-1][j-1].parent_i = i;
                    cellDetails[i-1][j-1].parent_j = j;
                }
            }
        }
```

```
//----------- 7th Successor (South-East) ------------

        if (isValid(i+1, j+1) == true)
{
    // If the destination cell is the same as the
    // current successor
    if (isDestination(i+1, j+1, dest) == true)
    {
        // Set the Parent of the destination cell
        cellDetails[i+1][j+1].parent_i = i;
        cellDetails[i+1][j+1].parent_j = j;
        printf ("The destination cell is found\n");
        tracePath (cellDetails, dest);
        foundDest = true;
        return;
    }

    // If the successor is already on the closed
    // list or if it is blocked, then ignore it.
    // Else do the following
    else if (closedList[i+1][j+1] == false &&
            isUnBlocked(grid, i+1, j+1) == true)
    {
        gNew = cellDetails[i][j].g + 1.414;
        hNew = calculateHValue(i+1, j+1, dest);
        fNew = gNew + hNew;


        if (cellDetails[i+1][j+1].f == FLT_MAX ||
                cellDetails[i+1][j+1].f > fNew)
        {
            openList.insert(make_pair(fNew,
                            make_pair (i+1, j+1)));

            // Update the details of this cell
            cellDetails[i+1][j+1].f = fNew;
            cellDetails[i+1][j+1].g = gNew;
            cellDetails[i+1][j+1].h = hNew;
            cellDetails[i+1][j+1].parent_i = i;
            cellDetails[i+1][j+1].parent_j = j;
        }
    }
}
```

```
//----------- 8th Successor (South-West) ------------

        //
        if (isValid (i+1, j-1) == true)
        {
            // If the destination cell is the same as the
            // current successor
            if (isDestination(i+1, j-1, dest) == true)
            {
                // Set the Parent of the destination cell
                cellDetails[i+1][j-1].parent_i = i;
                cellDetails[i+1][j-1].parent_j = j;
                printf("The destination cell is found\n");
                tracePath(cellDetails, dest);
                foundDest = true;
                return;
            }

            // If the successor is already on the closed
            // list or if it is blocked, then ignore it.
            // Else do the following
            else if (closedList[i+1][j-1] == false &&
                    isUnBlocked(grid, i+1, j-1) == true)
            {
                gNew = cellDetails[i][j].g + 1.414;
                hNew = calculateHValue(i+1, j-1, dest);
                fNew = gNew + hNew;


                if (cellDetails[i+1][j-1].f == FLT_MAX ||
                        cellDetails[i+1][j-1].f > fNew)
                {
                    openList.insert(make_pair(fNew,
                                    make_pair(i+1, j-1)));

                    // Update the details of this cell
                    cellDetails[i+1][j-1].f = fNew;
                    cellDetails[i+1][j-1].g = gNew;
                    cellDetails[i+1][j-1].h = hNew;
                    cellDetails[i+1][j-1].parent_i = i;
                    cellDetails[i+1][j-1].parent_j = j;
                }
            }
        }
    }
```

```
    // When the destination cell is not found and the open
    // list is empty, then we conclude that we failed to
    // reach the destiantion cell. This may happen when the
    // there is no way to destination cell (due to blockages)
    if (foundDest == false)
        printf("Failed to find the Destination Cell\n");

    return;
}


int main()
{
    /* Description of the Grid-
     1--> The cell is not blocked
     0--> The cell is blocked    */
    int grid[ROW][COL] =
    {
        { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
        { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
        { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
        { 0, 0, 1, 0, 1, 0, 0, 0, 0, 1 },
        { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
        { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
        { 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 },
        { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
        { 1, 1, 1, 0, 0, 0, 1, 0, 0, 1 }
    };

    // Source is the left-most bottom-most corner
    Pair src = make_pair(6, 0);

    // Destination is the left-most top-most corner
    Pair dest = make_pair(0, 0);

    aStarSearch(grid, src, dest);

    return(0);
}
```

OUTPUT:


Output
Clear
The destination cell is found
The Path is -> (6,0) -> (5,0) -> (4,1) -> (3,2) -> (2,1) -> (1,0) -> (0,0)