# Maze Simulation

Ajay Alladi , Aryan Sahoo

June 28 2021

## 1 Introduction

A man is in hospital for his covid treatment where he has to pay more.All his savings got exhausted and the hospital gave deadline that he has to pay the remaining amount by evening. So based on his situation his friend contacted several other friends and some agree to provide money.This guy knows address of each friend.Now, he has to go and collect money and come back to the hospital in minimum possible time.So we are going to help him to find his shortest path and minimum possible time.

## 2 Town as a maze

According to the above situation the man has to choose minimum path in terms of the cycle and in terms of the distance between any two points also. To find the shortest path between two points A* algorithm helps.and once we got the path and length we can use Branch and Bound method for direct travelling salesman problem

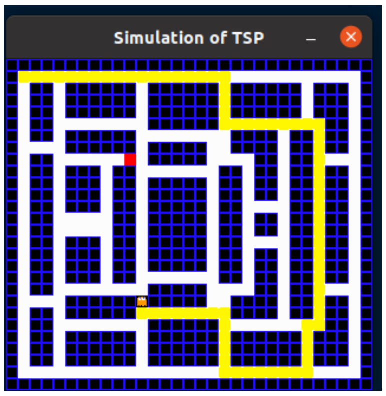


Figure 1: Town as maze

## 2.1 pseudo code for A* algorithm

```
typedef pair<int,pair<int,int>> Ppair;

vector<point*> findpath(int Maze[R][C], src, dest){

  if(src==dest){
    return;
  }

  bool obtained = false;

  bool visited[R][C]; ///maintains the visited & not visited info

  cell M[R][C];
  for(int y1 = 0; y1<R;y1++){ //initializes the cell maze
    for(int y2 =0;y2<C;y2++){
      visited[y1][y2] = false;
      M[y1][y2].Hval = NAN;
      M[y1][y2].parentx = -1;
      M[y1][y2].parenty = -1;
    }
  }

  M[src.x][src.y].Hval = 0;  // initializes src point

  set<Ppair> g;

  g.insert(make_pair(0,make_pair(src.x,src.y)));

  while(g.empty()){

    Ppair R = *g.begin(); //node with minimum H value
    int k1 = R.second.first;
    int k2 = R.second.second;

    visited[k1][k2] = true;

    //if the north to the curent node is valid
    if(inmatrix(k1,k2-1) && K[k1][k2-1]==1){

      if((k1,k2-1)==dest){ /// if node == dest

        M[k1][k2-1].parentx = k1;
        M[k1][k2-1].parenty = k2;

        vector<point*> X = getpath(M,s2); //iterates over the nodes
      through parent pointers from dest to src
        obtained = true;

        return X;
      }
      else if(!visited[k1][k2-1]){

        int d = 1 + M[k1][k2].Hval +  Hueristicvalue(k1,k2-1,s2);

        if(M[k1][k2-1].Hval > d ){
          M[k1][k2-1].Hval = d;
          M[k1][k2-1].parentx = k1;
          M[k1][k2-1].parenty = k2;
          g.insert(make_pair(d,make_pair(k1,k2-1)));
        }
      }
    }
    ///similarly for other 3 directions
    g.erase(g.begin());
  }
}
```

.

### 2.1.1 Explanation

- First check if src and dest are same or if any of the source and destination is a wall or they are in matrix or not in all these case no path exists
- So initially we initialized cell matrix with Hval as INT MAX and parent pointers as -1
- And then initialized the source pointer with Hval as zero
- And then the loop runs through the non empty set gets the fist element with least Heuristic value and checks for all 4 possible directions
- If any of the point matches with dest then loop breaks at that point and prints the path through the parent pointers of the current node to the source node which we initializes in previous conditions
- if not we will calculate Hval for the next node in that direction as sum of Heuristic value of the current node and **Manhattan distance** between(next node,current node) and **Manhattan distance** between (next node,dest)
- Now we will update the Hval if the calculated Hval is less than the already existed Hval pointer.

### 2.1.2 Time complexity

In worst case the algorithm has to run through all the nodes and it will take **O(R*C)** time

### 2.1.3 Space complexity

Extra space used is for maintaining visited array **(O(R*C)** and for set(at max **O(R*C))** so total space complexity is **O(R*C)**

### 2.1.4 Data Structures used

. Set data structure is used to get the node with least Hval in less time and struct is used for cell matrix and point* and Ppair¡int,¡int,int¿¿ is used to maintain coordinates with Hvalue

### 2.1.5 Will this gives shortest path??

Yes because we are taking node with minimum Hval which is distance to reach from src to dest through current node

## 2.2 pseudo code for branch and bound algorithm

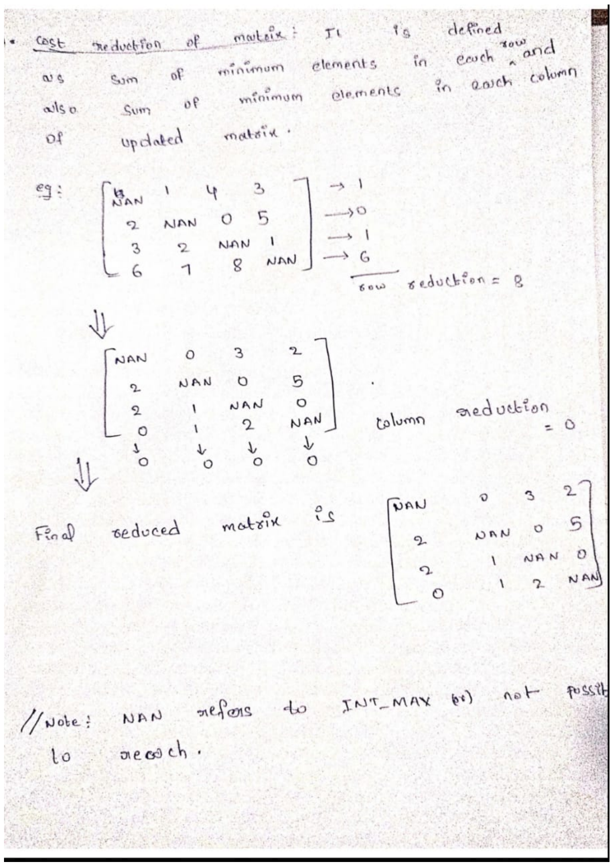
```
typedef pair<int, point*> pair;

point* newpoint(int R[n][n],vector<pair<int,int>> const &path ,int
    r, int p,int k){
  //for creating a newnode with required pointers
  point* p1 = new point;
  p1->path = path;
  int P[n][n];
  for(int k1 = 0;k1<n;k1++){
    for(int k2 =0;k2<n;k2++){
       P[k1][k2] = R[k1][k2];
    }
  }

  if(p!=0){

    p1->path.push_back(make_pair(k,r));
    makerow(P,k);
    makecol(P,r);
  }
  P[r][0] = NAN;

  for(int k1 = 0;k1<n;k1++){
    for(int k2 =0;k2<n;k2++){
      p1 -> costmatrix[k1][k2] = P[k1][k2];
    }
  }
  p1 -> ref = r;
  p1 -> position = p;
}


int* findpro(int K[n][n]){

  set<pair> list;
  vector<pair<int,int>> v;
  int b =  reducedcost(K);
  point* g;
  g -> position = 0; // position that no of nodes visited so far
  g -> level = 0; // level based on numbering to the points
  g->path = v;//this maintains the nodes visited till now

  g->costmatrix = K;//maintain costmatrix
  g->cost = b;//maintain cost

  list.insert(make_pair(b,g));

  while(!list.empty()){


    pair p3 = *list.begin();

    pairlist.erase(pairlist.begin());
    point* s = p3.second;

    int t = p3.first;

    int y1 = s->ref;

    if(s->position == n-1){
      // if the last node found
      vector<pair<int,int>> v1;
      v1 = s->path;
      static int T[n+1];
      T[0] = 0;

      // prints the order of nodes
      for(int y = 0; y< v1.size(); y++){

        T[y+1] = v1[y].second;

        cout << v1[y].first << " ---->" << v1[y].second << endl;
      }
      T[n] = 0;
      cout<<  s->cost << endl;

      return T;
    }

    for(int j =0 ; j < n ; j++){
      if(s->costmatrix[y1][j] != NAN){ /// if it can be reachable
    from current node;
        point* o = newpoint(s->costmatrix,s->path,j,s->position +
    1,s->ref);
        o->cost = reducedcost(o->costmatrix) + s->cost + s->
    costmatrix[y1][j]; // updating the cost of newly formed node
        list.insert(make_pair(o->cost,o));
      }
    }
  }
  static int T[1];
  T[0] = NAN;
  return T; // if path doesn't exist
}
```

### 2.2.1 explanation

- It is also same as the above A* algorithm but in place of manhattan distance here we use cost reduction to compare.
- Initially we start with a matrix with distances between every pair of nodes and distance between same points(i.e; M[i][i]) is to be INTMAX; and the distance between those pair of points which are not reachable from each other is INTMAX;
- At each step we will findout the cost of reduction and we will get the

minimum cost among all points and we will update our current node will update no of nodes visited until this point and when we reach the last point we will printout the path it has to travel to get minimum cost.

- We will use priority queue to get the node with minimum cost
- If this node is the last node we will return path through the path pointer to the node
- Else we will calculate the cost needed to reach the other non visited points and update the cost ,cost matrix and path pointers to these and push it into priority queue
- And the loop continues until we get the last node of the travelling salesman problem and then we will print the path and return the path vector to the next level to project it on a designed maze. **Note**



1. Cost reduction of matrix: It is defined as sum of minimum elements in each row and also sum of minimum elements in each column of updated matrix.

eg:

$$\begin{bmatrix} NAN & 1 & 4 & 3 \\ 2 & NAN & 0 & 5 \\ 3 & 2 & NAN & 1 \\ 6 & 7 & 8 & NAN \end{bmatrix} \begin{array}{l} \rightarrow 1 \\ \rightarrow 0 \\ \rightarrow 1 \\ \rightarrow 6 \end{array}$$

row reduction = 8

$$\Downarrow$$

$$\begin{array}{cccc} \left[\begin{matrix} NAN & 0 & 3 & 2 \\ 2 & NAN & 0 & 5 \\ 2 & 1 & NAN & 0 \\ 0 & 1 & 2 & NAN \end{matrix}\right] \\ \begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 0 & 0 & 0 \end{matrix} \end{array}$$

column reduction = 0

$$\Downarrow$$

Final reduced matrix is

$$\begin{bmatrix} NAN & 0 & 3 & 2 \\ 2 & NAN & 0 & 5 \\ 2 & 1 & NAN & 0 \\ 0 & 1 & 2 & NAN \end{bmatrix}$$

//Note: NAN refers to INT_MAX (or) not possib to reach.

Figure 2: Reduction Cost

- reduction cost of matrix = sum of minimum elements in each row + sum of minimum elements in each column of updated matrix.

#### 2.2.2 Time Complexity

In this we need to traverse through all nodes and to find cost matrix and to find cost we need Time Complexity O(n2) for each node and for total nodes **O(n3)**. and at each step for finding minimum in queue it would max take **O(n3logn)** and collectively total algorithm takes **O(n3 logn)**

#### 2.2.3 Space Complexity

Here we are using matrix for cost reduction related operations so it would take **O(n2)** and using priority queue it would take at max **O(n2)** but less..And all other arrays collectively require space **O(n)**.

#### 2.2.4 Data Structures used

struct pointers are used to maintain cost matrix , reduced cost, level of the node and priority queue is used to get the node with minimum cost at each step and pair is used and general data structures like array , vector, matrix are used as well.

## 3 Final Implementation

So according to the above mentioned context we need to get the minimum path so first we will find the shortest path between every pair of point and store it in a vector and its distances are stored in the form of matrix and when we give this matrix as input to Branch and bound algorithm we will get the order through which man has to travel So now, we got the path of the man we simulate with SDL 2.0 C++ Library

## 4 references

- The implementation code for above theory can be found here.
- Implementation video for above and some random screenshots can be found here.