1. Algorithm Design

Based on backtracking strategy, first, we need to initial our matrix and fulfill with the number pairs. Second, we need to define our goal test, which is touched on each cell in the matrix with some path. Then, we need to implement our successor function, for the first step, we find a path of one pair of those number points and we have four different direction to move (up, down, right, left), if we find the correct direction from one point to another just print it and choice the next move step. However, if not, we go back to change another direction. After we found one pair of number points we go to next one. And when we finish this function, we can find the solution we want. For this project, we do not need to consider our path cost.

2. Improvement

To deal with large number of input text, we cannot just use simple backtracking strategy. So we need to change our code based on DFS algorithm. However, we need to avoid looping to add cycle detection, I have some trouble with deal with move cycle free, so I cannot implement this project with DFS.

Prolog implement on DFS

```
solve(Input, Solution):-
    Input = [N, K|Pairs],
    length(Pairs,K),
    generate_Matrix(N,N,Solution),
    initMatrix(Pairs, Solution),
    findSolution(Pairs, Solution).

%2D array for Matrix
generate_Matrix(Cols, Rows, Matrix):-
    length_list(Rows, Matrix),
    maplist(length_list(Cols), Matrix).
length_list(N, List) :- length(List, N).

initMatrix([],_).
initMatrix([Top|Pairs], Pos):-
    Top = [Num, [X1,Y1],[X2,Y2]],
    nth1(X1,Pos1,Num),
    nth1(X2,Pos2,Num),
    nth1(Y1,Pos,Pos1),
    nth1(Y2,Pos,Pos2),
    initMatrix(Pairs,Pos).

move(N,Cell,NewCell):-
    Cell = [Num, [X,Y],[X2,Y2]],
    X1 is X+1,
    NewCell = [Num, [X1,Y],[X2,Y2]],
    validCell(N,NewCell).

move(N,Cell,NewCell):-
    Cell = [Num, [X,Y],[X2,Y2]],
    X1 is X-1,
    NewCell = [Num, [X1,Y],[X2,Y2]],
    validCell(N,NewCell).

move(N,Cell,NewCell):-
    Cell = [Num, [X,Y],[X2,Y2]],
    Y1 is Y+1,
    NewCell = [Num, [X,Y1],[X2,Y2]],
    validCell(N,NewCell).

move(N,Cell,NewCell):-
    Cell = [Num, [X,Y],[X2,Y2]],
    Y1 is Y-1,
    NewCell = [Num, [X,Y1],[X2,Y2]],
    validCell(N,NewCell).

validCell(N,Cell):-
    Cell = [_,[X,Y],[_,_]],
    X>0,Y>0,
    X =< N, Y =<N.

goal(Node):-
```

```
    Node = [_,[X1,Y1],[X2,Y2]],
    X1 =X2, Y1 = Y2.

findSolution([],_).
findSolution([Top|Pairs], Solution):-
    solve_dfs(Top,_,Solution),
    findSolution(Pairs, Solution).

solve_dfs(Node,Path, Pos):-
    depthfirst([Node],Node,RevPath,Pos),
    reverse(RevPath, Path).

depthfirst(Visted, Node, Visted, _) :- goal(Node).
depthfirst(Visted, Node, Path, Pos):-
    move_cyclefree(Visted,Node, NextNode,Pos),
    depthfirst([NextNode|Visted],NextNode,Path,Pos).

move_cyclefree(Visted,Node,NextNode,Pos):-
    length(Pos,N),
    move(N,Node, NextNode),
    \+member(NextNode,Visted),
       move_cyclefree([NextNode|Visted],Node, NextNode, Pos).
```