Backtracking
(Assignment Solutions)

Solution 1:

Hint : To track which cell has or not been visited, create a NxN vector called visited.
This vector will be initialized with false values for all cells & make the value for a particular cell to true when you have visited it.

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

void  solveMazeUtil(int  maze[][4],  int  x,  int  y,  string  sol,  int  N,
vector<vector<bool>> vis) {
      if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
          cout << sol << "\n";
          return;
      }



       //up
       if(x-1 >= 0 && !vis[x-1][y] && maze[x-1][y] == 1) {
           vis[x][y] = true;
           solveMazeUtil(maze, x-1, y, sol+"U", N, vis);
           vis[x][y] = false;
       }
       //down
       if(x+1 < N && !vis[x+1][y] && maze[x+1][y] == 1) {
           vis[x][y] = true;
           solveMazeUtil(maze, x+1, y, sol+"D", N, vis);
           vis[x][y] = false;
       }

       //right
       if(y+1 < N && !vis[x][y+1] && maze[x][y+1] == 1) {
           vis[x][y] = true;
           solveMazeUtil(maze, x, y+1, sol+"R", N, vis);
```

```
            vis[x][y] = false;
        }

        //left
        if(y-1 >= 0 && !vis[x][y-1] && maze[x][y-1] == 1) {
            vis[x][y] = true;
            solveMazeUtil(maze, x, y-1, sol+"L", N, vis);
            vis[x][y] = false;
        }

}



void solveMaze(int maze[][4], int N) {
        string sol = "";
        vector<vector<bool>> vis(N, vector<bool> (N, false));

        if(maze[0][0] == 1) { // only move if the initial position allows
            solveMazeUtil(maze, 0, 0, sol, N, vis);
        }
}

int main(){
    int n = 4;
    int maze[4][4] = { { 1, 0, 0, 0 },
                       { 1, 1, 0, 1 },
                       { 1, 1, 0, 0 },
                       { 0, 1, 1, 1 } };



    solveMaze(maze, n);
}
```

Solution 2:

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

void bfs(int pos, int len, string ans, string Dig, vector<vector<char>> L) {
      if (pos == len){
          cout << ans << endl;
      } else {
          vector<char> letters = L[Dig[pos]-'0'];
          for (int i = 0; i < letters.size(); i++)
              bfs(pos+1, len, ans + letters[i], Dig, L);
      }
}

void letterCombinations(string Dig, vector<vector<char>> L) {
      int len = Dig.size();
      if (len == 0) {
          cout << "";
          return;
      }

      string ans = "";
      bfs(0, len, ans, Dig, L);
}

int main(){
       vector<vector<char>> L = {{},{},{'a','b','c'},{'d','e','f'},{'g','h','i'},
                        {'j','k','l'},{'m','n','o'},{'p','q','r','s'},
                        {'t','u','v'},{'w','x','y','z'}};

      string digits = "23";
      letterCombinations(digits, L);
}
```

Solution 3 :

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

bool isSafe(int x, int y, int sol[][8], int N){
      return (x >= 0 && x < N && y >= 0 && y < N
              && sol[x][y] == -1);
}



void printSolution(int sol[][8], int N) {
      for (int x = 0; x < N; x++) {
          for (int y = 0; y < N; y++)
              cout << sol[x][y] << " ";
          cout << endl;
      }
}

bool solveKTUtil(int x, int y, int movei, int sol[][8],
                                   int xMove[],int yMove[], int N) {
      int k, next_x, next_y;
      if (movei == N * N)
          return true;


      for (k = 0; k < 8; k++) {
          next_x = x + xMove[k];
          next_y = y + yMove[k];
          if (isSafe(next_x, next_y, sol, N)) {
              sol[next_x][next_y] = movei;
              if (solveKTUtil(next_x, next_y, movei + 1,
                              sol, xMove, yMove, N))
                  return true;
              else
                  sol[next_x][next_y]
                      = -1; // backtracking
          }
      }
```

```
        return false;
}

bool solveKT(int N) {
        int sol[8][8];
        for (int x = 0; x < N; x++)
            for (int y = 0; y < N; y++)
                sol[x][y] = -1;


        int xMove[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
        int yMove[] = { 1, 2, 2, 1, -1, -2, -2, -1 };


        //As the Knight starts from cell(0,0)
        sol[0][0] = 0;


        if (!solveKTUtil(0, 0, 1, sol, xMove, yMove, N)) {
            cout << "Solution does not exist\n";
            return false;
        }
        else
            printSolution(sol, N);


        return true;
}

int main(){
        int N = 8;
        solveKT(N);
}
```