Castle on the Grid

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'minimumMoves' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. STRING_ARRAY grid
 *  2. INTEGER startX
 *  3. INTEGER startY
 *  4. INTEGER goalX
 *  5. INTEGER goalY
 */

int minimumMoves(vector<string> grid, int startX, int startY, int
goalX, int goalY) {
    int n=grid.size();
    vector<vector<int>> dist(n,vector<int>(n,-1));
    queue<pair<int,int>> q;
    q.push({startX,startY});
    dist[startX][startY]=0;
    int dx[4]={1,-1,0,0};
    int dy[4]={0,0,1,-1};
    while(!q.empty()){
        auto [x,y]=q.front();q.pop();
        for(int d=0;d<4;d++){
            int nx=x+dx[d],ny=y+dy[d];
            while(nx>=0&&ny>=0&&nx<n&&ny<n&&grid[nx][ny]=='.'){
                if(dist[nx][ny]==-1){
                    dist[nx][ny]=dist[x][y]+1;
                    q.push({nx,ny});
                }
                nx+=dx[d];ny+=dy[d];
            }
        }
    }
    return dist[goalX][goalY];
```

```
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    vector<string> grid(n);

    for (int i = 0; i < n; i++) {
        string grid_item;
        getline(cin, grid_item);

        grid[i] = grid_item;
    }

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

    int startX = stoi(first_multiple_input[0]);

    int startY = stoi(first_multiple_input[1]);

    int goalX = stoi(first_multiple_input[2]);

    int goalY = stoi(first_multiple_input[3]);

    int result = minimumMoves(grid, startX, startY, goalX,
goalY);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);
```

```
    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int,
int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```