909. Snakes and Ladders <medium>

***BFS***

int snakesAndLadders(vector<vector<int>>& board) {

int n=board.size();

queue<int> q; //BFS

unordered\_map<int, int> moves;

moves[1]=0; //no moves when we are in square 1

q.push(1);

while (!q.empty())

{

int curr\_sq=q.front(); // current square (val from 1..n\*n)

q.pop();

// try all the next 6 pos but not over n\*n

for(int i=1; i <= 6;++i)

{

int next\_sq=curr\_sq+i; // next pos when we move from sq

if (next\_sq > n\*n)

break;

// Determine the next sq val, on snake or ladder

// Compute row and col in the board

int row = (next\_sq-1)/n;

// the colum depends on if row is even or odd

int col = (row % 2 ==0) ? (next\_sq-1) % n : n-1 - (next\_sq-1) % n;

// the next sq can be on the snake (-1) or ladder

if(board[n-1-row][col] != -1)

next\_sq= board[n-1-row][col];

// update the moves for that square

if (moves.count(next\_sq)==0)

{

moves[next\_sq]=moves[curr\_sq] + 1; // moves to sq plus 1

if(next\_sq==n\*n) return moves[next\_sq]; // early exit

q.push(next\_sq);

}

}

}

return -1;

}

#if 0

/\*

Snake and ladder board can be thought as a directed graph with number of vertices equal to the number of cells in the board. Every vertex of the graph has an edge to next six vertices if next six vertices do not have a snake or ladder. If any of the next six vertices has a snake or ladder, then the edge from current vertex goes to the top of the ladder or tail of the snake.

This problem is basically to find the shortest path between source and destination of the Snakes and ladders board.

To simplify the problem, we will convert the Snakes and ladders board into an 1D array.

DFS is generally used for topological sorting, solving problems that require graph backtracking, detecting cycles in a graph, finding paths between two nodes, etc.

BFS is generally used for finding the shortest path between two nodes, testing if a graph is bipartite, finding all connected components in a graph, etc.

\*/

class Solution {

public:

int snakesAndLadders(vector<vector<int>>& board) {

int n = board.size();

// Convert the Snakes and ladders board into an 1D array

vector<int> oneDimensionalBoard(n \* n, 0);

// Snakes and ladders board has a zigzag pattern and we start with last row and first column.

// To simulate the zigzag pattern, we defined the 'direction' flag.

int r = n - 1;

int c = 0;

int index = 0;

int direction = 1;

while (index < n \* n) {

oneDimensionalBoard[index++] = board[r][c];

if (direction == 1 && c == n - 1) {

direction = -1;

r--;

} else if (direction == -1 && c == 0) {

direction = 1;

r--;

} else {

c += direction;

}

}

vector<int> visited(n\*n, false);

// If there's a ladder on the starting index (Generally this is not the case)

int start = oneDimensionalBoard[0] > -1 ? oneDimensionalBoard[0] - 1 : 0;

queue<int> q;

q.push(start);

visited[start] = true;

int step = 0;

while (!q.empty()) {

int size = q.size();

while (size-- > 0) {

int currentPosition = q.front();

q.pop();

// If we reach the destination, returm the number of steps taken

if (currentPosition == n \* n - 1) {

return step;

}

for (int nextPosition = currentPosition + 1; nextPosition <= min(currentPosition + 6, n \* n - 1); nextPosition++) {

int destination = oneDimensionalBoard[nextPosition] > -1 ? oneDimensionalBoard[nextPosition] - 1 : nextPosition;

if (!visited[destination]) {

visited[destination] = true;

q.push(destination);

}

}

}

step++;

}

return -1;

}

};

#endif