# GSWEP Week 1 Homework Problems

## spiralMatrixSum\_se Solution

function spiralMatrixSum(matrix) {

function updateDirection(currentDirection) {

switch (currentDirection) {

case DIRECTION.RIGHT:

if (col + 1 >= COLS || visited[row][col + 1]) {

currentDirection = DIRECTION.DOWN;

}

break;

case DIRECTION.DOWN:

if (row + 1 >= ROWS || visited[row + 1][col]) {

currentDirection = DIRECTION.LEFT;

}

break;

case DIRECTION.LEFT:

if (col - 1 < 0 || visited[row][col - 1]) {

currentDirection = DIRECTION.UP;

}

break;

case DIRECTION.UP:

if (row - 1 < 0 || visited[row - 1][col]) {

currentDirection = DIRECTION.RIGHT;

}

break;

default:

break;

}

return currentDirection;

}

const DIRECTION = {

RIGHT: [0, 1],

DOWN: [1, 0],

LEFT: [0, -1],

UP: [-1, 0],

};

if (!matrix || !matrix.length) {

return 0;

}

const ROWS = matrix.length;

const COLS = ROWS > 0 ? matrix[0].length : 0;

let row = 0;

let col = 0;

let spiralMatrixSum = 0;

let visited = new Array(ROWS).fill().map(() => new Array(COLS).fill(false));

let currentDirection = DIRECTION.RIGHT;

while (!visited[row][col]) {

visited[row][col] = true;

spiralMatrixSum += matrix[row][col];

currentDirection = updateDirection(currentDirection);

row += currentDirection[0];

col += currentDirection[1];

}

return spiralMatrixSum;

}

## LeetCode Problem 322. Coin Exchange

function coinChange(coins, amount) {

const minCoins = new Array(amount + 1).fill(Infinity);

minCoins[0] = 0;

for (let initalAmount = 1; initalAmount <= amount; initalAmount++) {

for (const coinDenomination of coins) {

if (initalAmount - coinDenomination >= 0) {

minCoins[initalAmount] = Math.min(

minCoins[initalAmount],

1 + minCoins[initalAmount - coinDenomination]

);

}

}

}

return minCoins[amount] === Infinity ? -1 : minCoins[amount];

}

## Find the Shortest Path through Weighted Graph

/\* Time Complexity: O(V^2)

Space Complexity: O(V) \*/

function solution(graph, startingVertex) {

const verticies = graph.length;

const paths = new Array(verticies).fill(Infinity);

const visitedVerticies = new Array(verticies).fill(false);

paths[startingVertex] = 0;

for (let vertexCycle = 0; vertexCycle < verticies; vertexCycle++) {

let minDistance = Infinity;

let minVertex = -1;

// find the smallest distance between start vertex and neighboring vertex.

for (let vertex = 0; vertex < verticies; vertex++) {

if (!visitedVerticies[vertex] && paths[vertex] < minDistance) {

minDistance = paths[vertex];

minVertex = vertex;

}

}

// if vertex is isolated return paths;

if (minVertex == -1) {

return paths;

}

visitedVerticies[minVertex] = true;

// find smallest distance betwn startingVertex to minVertex through other Verticies

for (let neighbor = 0; neighbor < verticies; neighbor++) {

if (graph[minVertex][neighbor] != -1) {

let alternatePathDistance =

paths[minVertex] + graph[minVertex][neighbor];

if (alternatePathDistance < paths[neighbor]) {

paths[neighbor] = alternatePathDistance;

}

}

}

}

return paths;

}

## Staircase Distinct Ways

function solution(steps) { // Dynamic Programming = Constant Space

let prevAnswers = [1, 1]; // prev - 1, prev - 2

let prevAnswerIdx = false;

for(let step = 2; step <= steps; step++){

prevAnswerIdx = !prevAnswerIdx;

prevAnswers[Number(prevAnswerIdx)] = prevAnswers[0] + prevAnswers[1];

}

return prevAnswers[Number(prevAnswerIdx)];

}