PRACTICE CERTIFICATION NEW COMPETE JOBS LEADERBOARD Search f20180084

All Contests > DSA (2021) Lab 8 > Find the Path

Find the Path

Problem Submissions Leaderboard Discussions

You are given a table, a, with n rows and m columns. The top-left corner of the table has coordinates $(0, 0)$, and the bottom-right corner has coordinates $(n - 1, m - 1)$. The i^{th} cell contains integer $a_{i,j}$.

A path in the table is a sequence of cells $(r_1, c_1), (r_2, c_2), \ldots, (r_k, c_k)$ such that for each $i \in \{1, \ldots, k - 1\}$, cell (r_i, c_i) and cell (r_{i+1}, c_{i+1}) share a side.

The weight of the path $(r_1, c_1), (r_2, c_2), \ldots, (r_k, c_k)$ is defined by $\sum_{i=1}^{k} a_{r_i,c_i}$ where a_{r_i,c_i} is the weight of the cell (r_i, c_i).

You must answer q queries. In each query, you are given the coordinates of two cells, (r_1, c_1) and (r_2, c_2). You must find and print the minimum possible weight of a path connecting them.

Note: A cell can share sides with at most 4 other cells. A cell with coordinates (r, c) shares sides with $(r - 1, c)$, $(r + 1, c)$, $(r, c - 1)$ and $(r, c + 1)$.

Input Format

The first line contains 2 space-separated integers, n (the number of rows in a) and m (the number of columns in a), respectively.
Each of n subsequent lines contains m space-separated integers. The j^{th} integer in the i^{th} line denotes the value of $a_{i,j}$.
The next line contains a single integer, q, denoting the number of queries.
Each of the q subsequent lines describes a query in the form of 4 space-separated integers: r_1, c_1, r_2, and c_2, respectively.

Constraints

- $1 \le n \le 7$
- $1 \le m \le 5 \times 10^3$
- $0 \le a_{i,j} \le 3 \times 10^3$
- $1 \le q \le 3 \times 10^4$

For each query:

- $0 \le r_1, r_2 < n$
- $0 \le c_1, c_2 < m$

Output Format

On a new line for each query, print a single integer denoting the minimum possible weight of a path between (r_1, c_1) and (r_2, c_2).

Sample Input

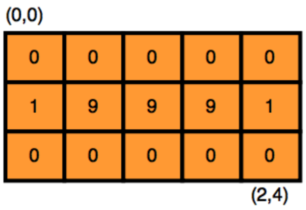
```
3 5
0 0 0 0 0
1 9 9 9 1
0 0 0 0 0
3
0 0 2 4
0 3 2 3
1 1 1 3
```

Sample Output

```
1
1
18
```

Explanation

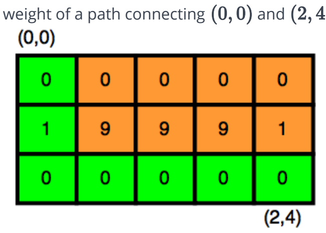
The input table looks like this:



0	0	0	0	0
1	9	9	9	1
0	0	0	0	0

The first two queries are explained below:

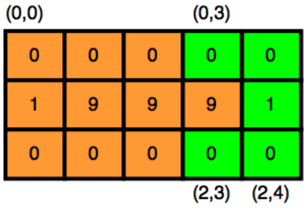
1. In the first query, we have to find the minimum possible weight of a path connecting $(0, 0)$ and $(2, 4)$. Here is one possible path:



0	0	0	0	0
1	9	9	9	1
0	0	0	0	0

The total weight of the path is $0+1+0+0+0+0+0=1$.

2. In the second query, we have to find the minimum possible weight of a path connecting $(0,3)$ and $(2,3)$. Here is one possible path:



0	0	0	0	0
1	9	9	9	1
0	0	0	0	0

The total weight of the path is $0+0+1+0+0=1$.

Submissions: 24
Max Score: 100
Difficulty: Hard

Rate This Challenge:
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

C

```c
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'shortestPath' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. 2D_INTEGER_ARRAY a
 *  2. 2D_INTEGER_ARRAY queries
 */

/*
 * To return the integer array from the function, you should:
 *     - Store the size of the array to be returned in the result_count variable
 *     - Allocate the array statically or dynamically
 *
 * For example,
 * int* return_integer_array_using_static_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     static int a[5] = {1, 2, 3, 4, 5};
 *
 *     return a;
 * }
 *
 * int* return_integer_array_using_dynamic_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     int *a = malloc(5 * sizeof(int));
 *
 *     for (int i = 0; i < 5; i++) {
 *         *(a + i) = i + 1;
 *     }
 *
 *     return a;
 * }
 *
 */
int* shortestPath(int a_rows, int a_columns, int** a, int queries_rows, int queries_columns, int** queries, int*
result_count) {

}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char** first_multiple_input = split_string(rtrim(readline()));

    int n = parse_int(*(first_multiple_input + 0));

    int m = parse_int(*(first_multiple_input + 1));

    int** a = malloc(n * sizeof(int*));

    for (int i = 0; i < n; i++) {
        *(a + i) = malloc(m * (sizeof(int)));

        char** a_item_temp = split_string(rtrim(readline()));

        for (int j = 0; j < m; j++) {
            int a_item = parse_int(*(a_item_temp + j));

            *(*(a + i) + j) = a_item;
        }
    }

    int q = parse_int(ltrim(rtrim(readline())));

    int** queries = malloc(q * sizeof(int*));

    for (int i = 0; i < q; i++) {
        *(queries + i) = malloc(4 * (sizeof(int)));

        char** queries_item_temp = split_string(rtrim(readline()));

        for (int j = 0; j < 4; j++) {
            int queries_item = parse_int(*(queries_item_temp + j));

            *(*(queries + i) + j) = queries_item;
        }
    }

    int result_count;
    int* result = shortestPath(n, m, a, q, 4, queries, &result_count);

    for (int i = 0; i < result_count; i++) {
        fprintf(fptr, "%d", *(result + i));

        if (i != result_count - 1) {
            fprintf(fptr, "\n");
        }
    }

    fprintf(fptr, "\n");

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}

```

Line: 1 Col: 1

Upload Code as File ☐ Test against custom input Run Code Submit Code