© 2025, Jongwook Kim. All rights reserved.

ECE 36800 Assignment #8

```
Original Due: 1:00 PM, Tuesday,  November 18
    Extended: 1:00 PM, Thursday, November 20
```

Programming task

You will write a program that finds the fastest way for a visitor to enter a grid from the top row and exit from the bottom row. Each grid cell stores a non-negative short integer representing the time the visitor must spend in that cell upon arrival. After serving that time, the visitor may move to one of the four orthogonally adjacent cells, when such a neighbor exists. Rows are indexed top to bottom starting at 0, columns left to right starting at 0.

	0	1	2	3	4
0	4	2	1	0	5
1	2	0	4	1	0
2	4	2	2	2	7
3	1	7	2	9	2

Cost Grid (C)

For example, in the grid above, cell (2, 1) stores 2. If the visitor arrives at time t, the earliest departure to an adjacent cell is t + 2. If a cell stores 0, the visitor may leave immediately upon arrival. Adjacent means same row and neighboring column, or same column and neighboring row; boundary cells have fewer neighbors. Entrances are any cells in row 0; exits are any cells in row m−1.

Example

In these examples we use the pay-on-depart convention: an edge u→v has weight C[u], so T does not include the cost of the destination cell. The time to exit from a bottom cell b = (3, j) is T[3, j] + C[3, j]. Any optimal path is acceptable.

Assuming we use the above 4x5 Cost Grid and we start from cell (0,0). The table below shows earliest arrival times T[r, c] from (0, 0). These numbers are not the input costs, They are the costs to go from (0,0) to each of the cells in C.

0	4	6	7	7
4	6	6	7	8
6	6	8	8	8
10	8	10	10	15

Arrival Times T_{00} (start = (0,0))

The best bottom-row exit is (3, 0) with exit time T[3, 0] + C[3, 0] = 10 + 1 = 11.
One fastest path is (0, 0) → (1, 0) → (2, 0) → (3, 0).

© 2025, Jongwook Kim. All rights reserved.

If you start from (0,1), Earliest arrival times T[r, c] from (0, 1) would be:

2	0	2	3	3
2	2	2	3	4
4	2	4	4	4
8	4	6	6	11

Arrival Times T_{01}(start = (0,1))

The best bottom-row exit is (3, 2) with exit time T[3, 2] + C[3, 2] = 6 + 2 = 8.
One fastest path is (0, 1) → (1, 1) → (2, 1) → (2, 2) → (3, 2).

In the above cost grid, the best path will be obtained if we start at (0,3). If we start at (0,3) the Arrival times will be as follows:

3	1	0	0	0
3	3	1	0	1
5	3	3	1	1
9	5	5	3	8

Arrival Times T_{03}(start = (0,3))

Among all these entry points, the entry point at the (0, 3)-location allows a visitor to enter the grid at the top and exit the grid at the bottom at the (3, 2)-location the earliest in 7 time units.

One fastest path is therefore (0, 3) → (1, 3) → (2, 3) → (2, 2) → (3, 2).

Executable and Input/Output Format

The executable a8 should be run as follows:

```
./a8  binary_grid_file  text_grid_file  fastest_times_file  fastest_path_file
```

Given a 2-dimensional grid stored in the (binary) input file binary_grid_file, the program produces three output files: text_grid_file stores the input grid in text form, fastest_times_file stores the fastest time to exit the grid at the bottom for each entry location at the top of the grid, and fastest_path_file stores the fastest path that allows a visitor to enter a location in the top row of the grid and exit at a location in the bottom row of the grid the earliest possible. Both fastest_times_file and fastest_path_file should be binary.

© 2025, Jongwook Kim. All rights reserved.

Binary grid file format

The binary_grid_file argv[1] is an input file in binary format. Given a grid of m rows and n columns, 0 < m, n <= SHRT_MAX, the first two short's stored in binary grid file are m and n, respectively. Here, SHRT_MAX is the largest short integer (see limits.h).

After those two short's, the file is followed by a total of m x n shorts. All m x n short's are within the range of [0, SHRT_MAX]. Out of these m x n shorts, the first n shorts are in row 0. Out of these n shorts, the first short is at column 0 and the last short is at column n-1. The next n shorts are in row 1. The last group of n shorts are in row m-1.

The total size of the file is (2+mn) x sizeof(short) bytes. You may assume that all input files are all correct.

In general, the sample files given to you are named as m_n.b where m is the number of rows and n is the number of columns. However, you should always determine the true dimensions of a given grid from the first two shorts of the input file. The file 4_5.b provided to you for this assignment contains the example given earlier.

Suppose we ran a8 with the following command:

./a8 4_5.b 4_5.t 4_5.f 4_5.p

The file 4_5.t should store the 2-dimensional grid in text form. The file 4_5.f should store the fastest times to exit the grid for all entry locations in binary form. The file 4_5.p should store the fastest path in binary form.

Text grid file format

The first output file argv[2] is simply a conversion of the binary grid file to a text form. Again, we assume that there are m rows and n columns in the binary file. The first line is printed with the format "%hd %hd \n", where the first short is the number of rows, m, and the second short is the number of columns, n.

After that, there should be m lines. Each line should print n shorts, where each short is printed with the format "%hd". There should be a space (' ') character between a pair of consecutive shorts. There should be a newline character ('\n') immediately after the last short in each line (row). There should not be a space character after the last short in a line.

Fastest times file format

The second output file argv[3] stores the fastest time to exit the grid for each entry location in the top row in binary form. The file first stores the number of entry locations in the top row of the input grid. This should correspond to the number of columns in the given input grid. The number of columns should be stored as an short.

Let n be the number of columns in the given grid. The file next stores n int. Assuming that we label the int from 0 to n-1. The ith int, 0 ≤ i ≤ n-1, is the fastest time to exit at the bottom of the grid when the entry point is the (0, i)-location.

The total size of the fastest-times file is (sizeof(short) + n x sizeof(int)) bytes.

© 2025, Jongwook Kim. All rights reserved.

Fastest path file format

The third output file argv[4] stores the fastest path information in binary form.

The file first stores the fastest time it would take for a visitor to enter at a location in the top row of the input grid and exit at a location in the bottom row of the input grid. The fastest time should be stored as an int.

Then, we store in the file an int for the number of locations that constitute the path. Let l be the number of locations that constitute the path, the output file next contains coordinates of the l entries that form the path. Each location is stored as a pair of shorts, the (row, column) coordinates of the location, with the row coordinate followed by the column coordinate. The order in which the coordinates appear in the output file should follow the order in which the corresponding locations appear in the path. The path should start with a location in the top row of the grid and end with a location in the bottom row of the grid.

The total size of the path file is (2 x sizeof(int) + 2l x sizeof(short)) bytes.

Return value of main function

The main function should simply return EXIT_FAILURE if the argument count is incorrect.

If the given input file cannot be opened, your program should terminate and return EXIT_FAILURE.

Now, assume that the given input file can be opened. If in the process of determining the fastest path, your program encounters a failure in memory allocation or a failure in writing to the output files, your program should gracefully exit and return EXIT_FAILURE. We will test your program with valid input files of reasonable sizes. Therefore, it is unlikely that you will have to return EXIT_FAILURE when you could open an input file.

Of course, your program should return EXIT_SUCCESS otherwise.

Grading

The assignment will be graded based on the three tasks performed by your program. The first output file accounts for 10%, the second output file accounts for 60%, and the third output file accounts for 30%.

It is important that all files are closed and all allocated memory is freed before the program exits. Memory leaks or memory errors reported by valgrind will result in a 50% penalty.

Be aware that a time limit is set for each test case based on its size. If your program does not complete its execution before the time limit for a test case, it will be deemed to have failed that test case.

Submission Instructions

You must submit the following files to Gradescope under Assignment 8:

1. All source files and header files used in your implementation.
2. A Makefile that compiles your code to an executable named a8.

© 2025, Jongwook Kim. All rights reserved.

Initial Testcases and Helpful Information

You are given 4 sample binary grid files (4 5.b, 5 4.b, 5 5.b, 7 7.b) and the corresponding text grid files (4 5.t, 5 4.t, 5 5.t, 7 7.t), fastest times files (4 5.f, 5 4.f, 5 5.f, 7 7.f), and path files (4 5.p, 5 4.p, 5 5.p, 7 7.p). It is possible that a binary grid file may contain multiple paths that allow a visitor to enter and exit a grid in the shortest time possible. Your program should not try to match the path exactly. However, the shortest time that it took to enter and exit a grid should match.

To help you to understand the fastest times files and path files, we also provide the text version of these files (4 5.ft, 5 4.ft, 5 5.ft, 7 7.ft and 4 5.pt, 5 4.pt, 5 5.pt, 7 7.pt). In each .ft file, the first line stores the number of columns in the given grid file (printed with the format "%hd\n"). Let n be the number of columns. The second line has n int, where each int is printed with the format "%d". There is a space (’ ’) character between a pair of consecutive ints. There is a newline character (’\n’) immediately after the last int. Assuming that we label the int from 0 to n − 1. The ith int, 0 ≤ i ≤ n − 1, is the fastest time to exit at the bottom of the grid when the entry point is the (0, i)-location.

In each .pt file, the first line stores the (fastest) time (printed with the format ”%d\n”). The second line stores the number of locations in the path (printed with the format ”%d\n”). Each subsequent line stores the (row, column) coordinates of an entry (printed with the format ”%hd %hd\n”).