Archipelago

CMIMC New Language Round

April, 2025

"One can hardly call it civilization when the fruits of the earth become the cause of strife. Perhaps the true lesson lies not in what we fight over, but why we fight at all."

Montesquieu

1 About

The Archipelago of Carnegie lives off a precious resource. Melons. The contention of these melons have lead to wars between islands, until a wise computer scientist (you) arrived. The Archipelago has invited you to walk around the islands and requested you to plan out shipping lanes and set up instructions to redistribute the melons to satisfy the needs of the islands.

2 Language Specifications

See provided Javascript visualizer for an archipelago program that demonstrates execution.

2.1 Program

An archipelago program consists of a rectangular grid of dimensions $H \times W$ zero-indexed by row, column. All squares by default contain water. Any square not on the boundary of the grid may instead contain land. A contiguous orthogonally adjacent collection of land squares is an island. Each land square may additionally have a special command, a letter from `i,d,a,s,p,o,j,f`. The letters `j,f` may additionally be augmented with one of `<,=,>`.

Each island has a stack of integers. There must be an input island that contains the square $(1, 1)$. and an output island that contains the square $(H - 2, W - 2)$. The input and output island must be distinct. The stack on the input island is initialized to the input of the program. Upon halting, the output is the stack on the output island. All stacks, aside from that on the input island, are initialized to be empty.

2.2 Execution

The program execution involves an IP (instruction pointer) moving along the boundary of islands. At any point of execution, the IP has a location, a direction, and a handedness. In the absence of a special command on the square, the IP travels along the boundary of the island, as follows.

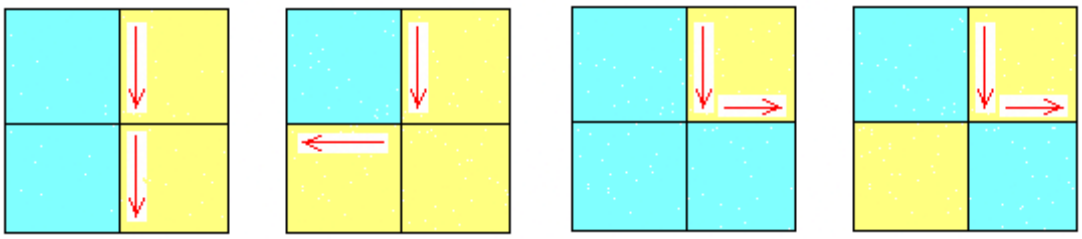


Figure 1: IP starts on left of top-right square travelling down with island to its left. Yellow is land.

At any point of execution, the opposite island of the IP is the island that is seen if one hypothetically stands at the IP and looks off of the boundary of its current island. That is, consider a hypothetical ray emitted from the square of the IP normal to IP's current edge. The ray finds its first intersection with an island, and this is the opposite island. It may be the same as the island the IP is originally on. The square of intersection is the opposite square.

If there is a special command on the square, then the following actions are instead performed.

- If command `i` or `d`, then the top element of the current island is incremented or decremented. Then IP moves one step along the boundary.
- If command `a`, then the top element of the current island popped off and added to the second top element. The IP moves one step along the boundary.
- If command `s`, then the top element of the current island popped off and subtracted from the second top element. The IP moves one step along the boundary.
- If command `o`, then the top element of the current island is popped off and disappears. The IP moves one step along the boundary.
- If command `p`, then a copy of the top element of the current island is pushed on the opposite island. The IP moves one step along the boundary.
- If command `j`, then the IP jumps to the opposite island, gets attatched to the opposite square and keeps the same direction. Note that the handedness flips. If the current island is on the left of the IP, then after jumping the new island is on the right of the IP.
- If command `f`, then the IP jumps to the opposite island, and the direction is flipped. The handedness remains the same.
- If command `j<,j=,j>,f<,f=,f>`, then a conditional jump is performed. The top element of the current island is compared with the top element of the opposite island, and the jump happens only if the inequality is true. Otherwise, the IP moves one step along the boundary like normal. The elements are not consumed.

Whenever a stack does not have enough elements for `i,d,a,s,o,p`, a zero will be pushed on the stack to complete the operation. For `a,s`, if the stack has no elements, two zeros will be pushed on, and if the stack has one element, that element is the top element and the zero pushed is treated as the second top element. Whenever a stack does not have enough elements for a comparison `<,=,>` is involved, then a zero is used, but the stack does not gain the zero.

The program starts with IP begins on the left edge of square $(1, 1)$ traveling downwards with the initial island to its left. The program terminates when the IP reches the square $(H-2, W-2)$, on any edge and handedness.

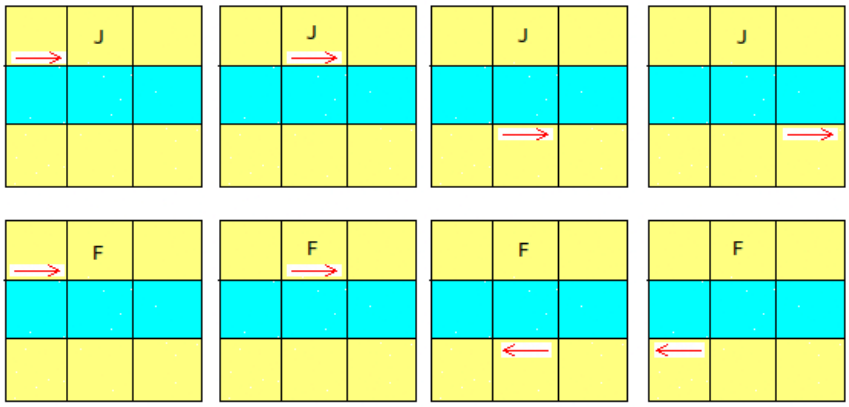


Figure 2: Jump and flip command. Original handedness is left.

2.3 Errors

The program raises an error if an opposite island is not found when executing a `p,j,f` command.

3 Coding in Archipelago

An archipelago program is formatted as a comma delimited csv file. Within each cell, a dash `-` or hash `#` indicates that a square is land. A square with a special command is denoted instead with the special command, one of `i,d,a,s,o,p`, `j,j<,j=,j>,f,f<,f=,f>`, and no dash or hash is needed. Any square left empty or with another command is treated as as water. Any squares on the boundary of the grid is treated as water.

It is suggested that one uses a spreadsheet tool to write their code, and export the spreadsheet to a csv file. The dimensions of the csv file should match the size of the rectangular grid one intends, so the location of the output matches with what the interpreter expects.

3.1 Limitations

For our interpreter, your code is required to satisfy $4 \leq H, W < 512$. Your code is allowed a maximum of 1,000,000 steps of execution, which should be more than enough for the size of the test cases.

4 Problems

Each normal problem is worth 10 points. You should also aim to reduce the number of islands in your program, hereinafter referred to size of program. This number is statically determined by your code. Submissions with smaller size will be awarded extra points, with rules to be determined.

Given N successful submissions, all successful submissions with size ranked at most $\lceil 0.5N \rceil$ smallest will be awarded 3 extra points, and all successful submissions size ranked at most $\lceil 0.2N \rceil$ smallest will be awarded another extra 3 points.

The input and desired output stacks are described in input and output, and the output stack must match exactly with no extra elements. The top of the stack is the right-most element of

the list.

4.1 Problems

Task 1. Implement an archipelago program that adds two numbers

Input: $[x, y]$ where x, y are integers
Output: $[x + y]$

Task 2. Implement an archipelago program that takes the max of two numbers

Input: $[x, y]$ where x, y are integers
Output: $[z]$ where $z = \max(x, y)$

Task 3. Implement an archipelago program that multiplies two numbers

Input: $[x, y]$ where x, y are integers
Output: $[x \cdot y]$

Task 4. Implement an archipelago program that inverts the input

Input: $[x_1, x_2, \ldots, x_{n-1}, x_n, n]$ where $n > 0$, x_i are integers
Output: $[x_n, x_{n-1}, \ldots, x_2, x_1, n]$

Task 5. Implement an archipelago program that outputs the Fibonacci sequence

Input: $[n]$ where $n > 0$
Output: $[z_1, z_2, \ldots, z_n]$ where z_n is the n-th Fibonacci number

Task 6. Implement an archipelago program that finds the max of the input

Input: $[x_1, x_2, \ldots, x_n, n]$ where $n > 0$, x_i are integers
Output: $[z]$ where $z = \max_{1 \le i \le n} x_i$

Task 7. Implement an archipelago program that finds the median of the input

Input: $[x_1, x_2, \ldots, x_n, n]$ where $n > 0$ is odd, x_i are integers
Output: $[z]$ where $z = \text{median}(x_1, x_2, \ldots, x_n)$

Task 8. Implement an archipelago program that performs inclusive scan

Input: $[x_1, x_2, \ldots, x_n, n]$ where $n > 0$, x_i are integers
Output: $[z_1, z_2, \ldots, z_n]$ where $z_i = \sum_{j=1}^{i} x_i$

Task 9. Implement an archipelago program that computes the discrete Laplacian of a 2D grid of numbers, padding zeros as necessary

Input: $[x_{1,1}, x_{1,2}, \ldots, x_{n,m-1}, x_{n,m}, m, n]$ where $m, n, > 0$, x_i are integers
Output: $[z_{1,1}, z_{1,2}, \ldots, z_{n,m-1}, z_{n,m}]$ $z_{i,j} = x_{i+1,j} + x_{i-1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j}$, $x_{i,j}$ outside the grid are treated as zeros

4.2 Bonus Problem

Bonus Task 10. Implement a program that visually looks interesting

Input:	You provide
Output:	You provide
Goal:	Make a program that visually looks interesting
Constrains:	All islands in the program must be visited

The extra task encourages you to write a program that visually looks interesting. You need not compress the number of islands, and you can use as many as you want, but each island must be visited during program execution. The program also must run within reasonable time.

This extra task will be submitted separately. Your submissions will be entered in a contest where all contestants will vote for the most interesting programs. Any submission that performs a non-trivial task (should be no simpler than task 3 multiplication), as judged by the CMIMC staff, will recieve 5 points. The top 3 voted programs will recieve 10 points each.