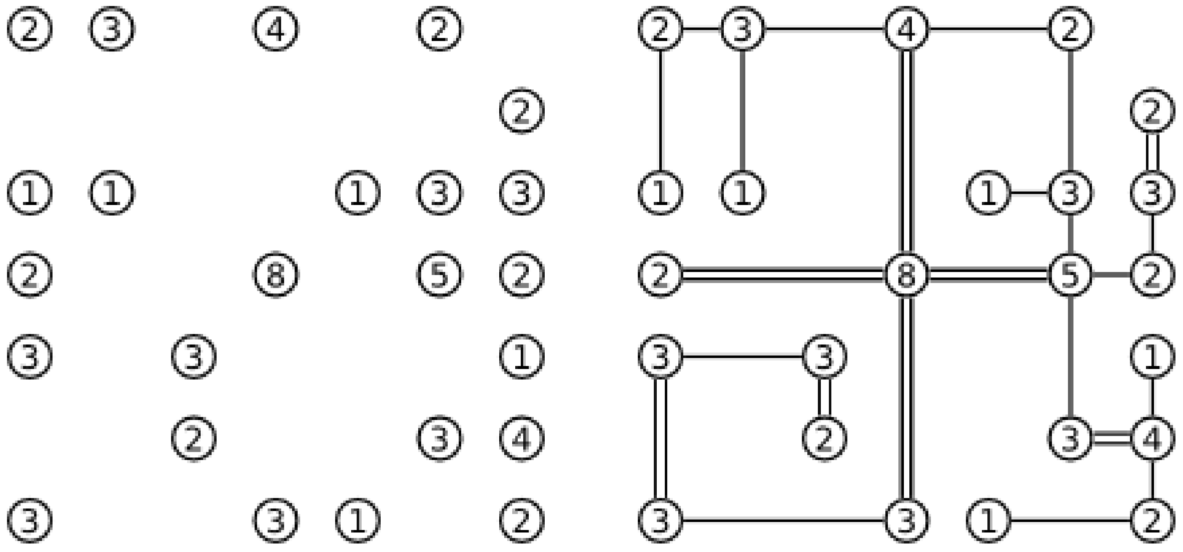
## Task A2. PUZZLE

⏳10 sec. 💾 256 MB

Alice really enjoys solving puzzles like Sudoku and others. Recently she found a new one. It is played on a rectangular grid. Some cells start out with numbers from 1 to 8 inclusive; these are the ”islands”. The rest of the cells are empty. No two islands are next to each other (i.e. no two islands share an edge). The goal is to connect all of the islands by drawing a series of bridges between the islands. The bridges must follow certain criteria:

- They must begin and end at distinct islands, travelling a straight line in between.
- They may only run orthogonally (i.e. they may not run diagonally).
- They must not cross any other bridges or islands.
- At most two bridges connect a pair of islands.
- The number of bridges connected to each island must match the number on that island.
- The bridges must connect the islands into a single connected group.

Note that the solution may not be unique. Here is an example istance of this puzzle on the left and on the right is one of its solutions (the grid is not shown):



Alice wants help with solving some puzzles of this type of various sizes. Help her by implementing a program that does this. Clearly, this problem may not have a fast solution in the worst case, so it is important to focus on “realistic” cases. To that end, you are provided a set of tests that is equivalent to the set of tests in the grading system (generated in the same way, with the same parameters, just with different seeds).

Each test will consist of one or more subtests – each of which is an instance of the puzzle. Your program will be compiled together with a grader, which will feed it with these subtests one after another and will stop when when your program produces an incorrect solution, or when your program decides to stop, or if the time limit has passed before your finishes the subtest. Your score for the test will be proportional to the number of successfully solved subtests.

### *Implementation details*

First the grader will call your function `init` to specify the number of subtests and the time limit in seconds for the test (different tests may have different time limits):

```
void init(int numSubtests, double timeLimit);
```

Then for each subtest it will call your function `solve` with an instance of the puzzle, represented as a vector of vectors (inner vectors represent rows), where empty cells are 0s. Your function must modify this structure to write out its solution. It must place a `-1` for a horizontal single bridge, `-2` for a horizontal double bridge, `-3` for a vertical single bridge and `-4` for a vertical double bridge. It must return `true`, if it wants to solve the current subtest and keep going, or `false`, if it wants to stop (i.e. not solve the current subtest). This is the function signiture:

```
bool solve(std::vector<std::vector<int>>& puzzle);
```

Finally, your functions may call the following grader implemented function, which returns the time passed in seconds:

```
double timePassed();
```

Your code must implement `init` and `solve`. Besides them, it may have any other helper functions, variables, structs, etc. However, the code cannot contain a `main` function and it must include the header `park.h`.

### *Local testing*

You are given the files `Lgrader.cpp` and `park.h`, which you can compile together with your code to test it. You are also given a representative set of sample tests.

### *Constraints*

- $2 \leq \text{numRows}, \text{numColumns} \leq 47$
- $2 \leq \text{numIslands} \leq 200$

### *Tests*

| **Tests** | numIslands | numRows, numColumns | numSubtests | timeLimit |
|---|---|---|---|---|
| 1-3 | $\leq 15$ | $\leq 7$ | $= 1$ | 2 |
| 4-6 | $\leq 15$ | $\leq 7$ | $= 2$ | 2 |
| 7-8 | $\leq 100$ | $\leq 31$ | $= 1$ | 2 |
| 9-10 | $\leq 100$ | $\leq 31$ | $= 6$ | 2 |
| 11-12 | $\leq 100$ | $\leq 31$ | $= 36$ | 2 |
| 13-14 | $\leq 200$ | $\leq 47$ | $= 1$ | 5 |
| 15-16 | $\leq 200$ | $\leq 47$ | $= 6$ | 5 |
| 17-18 | $\leq 200$ | $\leq 47$ | $= 24$ | 8 |

*All tests are scored independently and are worth the same.*

## *Scoring*

The grader will repeatedly call `solve` with instances of the puzzle. It will stop doing that and terminate the program, if any of the following happens:

- The test's time limit has passed before yout function returns.
- Your function returns an invalid solution.
- Your function decides to stop (i.e. returns `false`).

Then, you will still receive the points for all correctly solved subtests prior to this, i.e. if you solved $C$ subtests and there are $T$ subtests in total, you will get $C/T$ of the points for the test.

Note that you still need to avoid the grading system's hard time limit of 10 seconds. You may use `timePassed()` and stopping by returning `false` to do this. If your program exceeds it, it will be killed and it will get 0 points.

## *Sample communication*

First, `init(int 1, double 2);` is called. Then:

```
solve({
    { 0,  0,  2,  0,  3,  0,  3},
    { 4,  0,  0,  0,  0,  2,  0},
    { 0,  0,  0,  0,  0,  0,  0},
    { 6,  0,  0,  0,  0,  2,  0},
    { 0,  0,  0,  0,  0,  0,  0},
    { 0,  0,  0,  0,  0,  0,  0},
    { 4,  0,  0,  0,  0,  0,  4}
});
```

This function runs, returns `true` and modifies the given vector to the following:

```
{
    { 0,  0,  2, -2,  3, -1,  3},
    { 4, -2, -2, -2, -2,  2, -4},
    {-4,  0,  0,  0,  0,  0, -4},
    { 6, -2, -2, -2, -2,  2, -4},
    {-4,  0,  0,  0,  0,  0, -4},
    {-4,  0,  0,  0,  0,  0, -4},
    { 4, -2, -2, -2, -2, -2,  4}
}
```

This would score the full points for the test, if it ran in the specified time limit.