CodeWOOT (3978)

AoPS Staff

Thursday
Sep 5, 2024 - Mar 6, 2025
7:30 - 9:30 PM ET (4:30 - 6:30 PM PT)

Homework

Homework: Lesson 1

You have **8** writing problems to complete.
Due **Sep 18**.

Readings

Lesson 1 Transcript: Thu, Sep 5 - A
Lesson 1 Transcript: Thu, Sep 5 - B

Challenge Problems

Total Score: 0

Problem 1 – Answered (46267)

Problem 1-1: Array-Based Max Heap: Report Error

Let's practice implementing a max-heap as an array that implicitly represents a binary tree, as discussed in class. As a refresher, the way the heap works is:

* The first element of the array represents the root of the tree, the second and third elements of the array represent the next layer of the tree, the fourth through seventh elements of the array represent the layer below that, and so on.
* This structure makes it easy to write formulas: given the array index of an element, what are the array indices of the element's children (if any)? And what is the array index of the element's parent (if any)?
* When adding an element to the heap, place it at the end of the array. (Conceptually, this represents using the leftmost open slot on the current bottom level of the tree, or starting a new level if that level is full.) Then, if that element is larger than its parent, propagate the element upward by swapping with its parent, and continue checking to see if it's still larger than its new parent, etc.
* When deleting the maximum element in the heap, pop the last element from the array and use it to replace the first element. Then propagate the element downward as needed if it's smaller than one or both of its children. (If it's smaller than both children, how do you choose which child to "promote"?)

Given Q add/delete operations ($2 \leq Q \leq 5000$), can you keep track of the state of the heap? All values added are in the range $[1, Q]$. For convenience -- and to make the behavior of the heap unambiguous -- no value appears more than once within a test case.

Each query consists of one integer X_i $(0 \leq X_i \leq Q)$, and has one of the following forms:

0: Delete the maximum value in the heap. (This instruction will only appear when there is at least one value in the heap.)
v: (with $v \neq 0$): Add the value v to the heap.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains Q.
Each of the next Q lines represents a query, and contains an integer X_i.

OUTPUT FORMAT (print output to the terminal / stdout):

After executing each instruction, print the elements of the array, in order, on its own line.

SAMPLE INPUT:

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

SAMPLE OUTPUT:

```
1
4 1
4 1 2
4 3 2 1
3 1 2
2 1
1

9
```

Note the blank line as part of the output, corresponding to printing the values of an empty heap. (Although blank lines in inputs and outputs are a little weird, they do sometimes show up in USACO problems!)

SCORING:

Test cases 2-10 satisfy $Q \leq 100$.
Test cases 2-5 and 11-14 have no delete instructions.
Test cases 15-20 satisfy no additional constraints.

Solution:
Unlike almost every other problem you will see in this course, this is a pure implementation exercise. But there are still details to think through:

1. In lesson 1, we came up with a formula for finding the index of the parent of an array element:
* $parent(x) = \frac{(x-1)}{2}$ (using integer division as C++ does, so, e.g., $3/2 = 1$)

We can inspect a small heap and derive formulas for the indices of an element's children (if any):
* $left\_child(x) = 2x + 1$
* $right\_child(x) = 2x + 2$

2. We need to figure out how to update the heap after deleting the maximum element and leaving a hole at the top of the tree.

A natural idea is to reverse our procedure for insertion from Lesson 1, in which we added an element at the end of the array, and then fixed the heap by swapping the element upward in the tree (if necessary) until it was either at the root or no larger than its parent.

By analogy, after deleting the max element, we can move the element at the end of the array into that empty top spot (at the beginning of the array), then swap it downward in the tree until it either has no children or is at least as large as all of its children.

There is one subtlety here: what do we do when a parent is smaller than both of its children? We need to swap with the larger of the two children, or else the child that we move upward will become a parent of the other, larger child, which violates the heap property.

Here is our solution:

```
#include <bits/stdc++.h>
using namespace std;

#define MAX_Q 5000

int arr[MAX_Q];
int arr_size = 0;

int get_parent(int idx) {
  return (idx - 1) / 2;
}

int get_left_child(int idx) {
  return 2 * idx + 1;
}

int get_right_child(int idx) {
  return 2 * idx + 2;
}

void add(int v) {
  arr[arr_size] = v;
  int idx = arr_size;
  arr_size++;
  while (true) {
    if (idx == 0) {
      return;
    }
    int parent_idx = get_parent(idx);
    if (arr[idx] < arr[parent_idx]) {
      return;
    }
    swap(arr[idx], arr[parent_idx]);
    idx = parent_idx;
  }
}

void delete_max() {
  arr[0] = arr[arr_size - 1];
  arr_size--;
  int idx = 0;
  while (true) {
    // Start this off with the parent -- we need at least one child's value
    // to exceed the parent's value if we're going to do a swap.
    int bigger_child_idx = idx;
    int lc_idx = get_left_child(idx);
    if (lc_idx < arr_size && arr[lc_idx] > arr[bigger_child_idx]) {
      bigger_child_idx = lc_idx;
    }
    int rc_idx = get_right_child(idx);
    if (rc_idx < arr_size && arr[rc_idx] > arr[bigger_child_idx]) {
      bigger_child_idx = rc_idx;
    }
    if (bigger_child_idx == idx) {
      break;
    }
    swap(arr[idx], arr[bigger_child_idx]);
    idx = bigger_child_idx;
  }
}

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int q, v;
  cin >> q;
  for (int i = 0; i < q; ++i) {
    cin >> v;
    if (v == 0) {
      delete_max();
    } else {
      add(v);
    }
    for (int j = 0; j < arr_size; ++j) {
      cout << arr[j];
      if (j != arr_size - 1) {
        cout << " ";
      }
    }
    cout << "\n";
  }
}
```

Languages: C++17

Source File:

(Practice mode: submission will not count toward your final score.)

Submission Results - 1-1-maxHeap.cpp (2024-09-10 00:17:35) - 333 / 333

1	*	3.2MB	2ms
2	*	3.204MB	2ms
3	*	3.2MB	2ms
4	*	3.204MB	2ms
5	*	3.204MB	2ms
6	*	3.204MB	2ms
7	*	3.2MB	2ms

Your Response(s):
1-1-maxHeap.cpp (2024-09-09 22:48:03) - 0 / 333
1-1-maxHeap.cpp (2024-09-09 22:52:43) - 0 / 333
1-1-maxHeap.cpp (2024-09-09 22:54:41) - 0 / 333
1-1-maxHeap.cpp (2024-09-09 23:00:08) - 0 / 333
test.cpp (2024-09-09 23:01:14) - 0 / 333

test.cpp (2024-09-09 23:02:42) - 0 / 333
test.cpp (2024-09-09 23:03:26) - 0 / 333
test.cpp (2024-09-09 23:05:12) - 0 / 333
test.cpp (2024-09-09 23:07:09) - 0 / 333
test.cpp (2024-09-09 23:07:40) - 0 / 333
test.cpp (2024-09-09 23:08:01) - 0 / 333
test.cpp (2024-09-09 23:08:28) - 0 / 333
test.cpp (2024-09-09 23:09:03) - 0 / 333
test.cpp (2024-09-09 23:11:23) - 0 / 333
1-1-maxHeap.cpp (2024-09-09 23:26:32) - 0 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-09 23:57:53) - 140 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:00:48) - 0 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:01:51) - 140 / 333 [PRACTICE]
sol.cpp (2024-09-10 00:02:50) - 333 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:03:20) - 0 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:03:43) - 140 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:06:46) - 0 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:14:57) - 140 / 333 [PRACTICE]
1-1-maxHeap.cpp (2024-09-10 00:17:35) - 333 / 333 [PRACTICE]

Problem 2 – Answered (46324)

Problem 1-2: Rectangle Overlap: Report Error
After baling hay all day, Farmer John is relaxing by making a two-dimensional drawing of N rectangles $(2 \leq N \leq 2 \cdot 10^5)$.

Each rectangle sits on the x-axis -- that is, it has its lower left corner at $(L_i, 0)$ and its upper right corner at (R_i, H_i). $(1 \leq L_i < R_i \leq 10^9, 1 \leq H_i \leq 10^9)$. Rectangles might overlap each other.

Farmer John has taken a pencil and made sure that every rectangle in the drawing is completely shaded in. (If there are overlapping parts, he doesn't need to shade them in multiple times.) Now he wants to know: what is the total shaded area?

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains N.

Each of the next N lines contains L_i, R_i, H_i, representing a rectangle.

OUTPUT FORMAT (print output to the terminal / stdout):

Print one line with the total shaded area.

SAMPLE INPUT:

```
4
2 4 2
1 5 1
8 9 1
6 7 3
```

SAMPLE OUTPUT:

```
10
```

SCORING:

Test cases 2-4 satisfy $N = 2$.
Test cases 5-9 satisfy $2 \leq N \leq 10^4$.
Test cases 10-20 satisfy no additional constraints.

Solution:
Here is a C++ priority queue based solution for this in-class problem.

```
#include <bits/stdc++.h>

using namespace std;
typedef pair<int, int> pi;
typedef long long ll;

const int MAX_X = 1e9;

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

  priority_queue <tuple<int, int, int>> events;
  priority_queue <pi> barns;

  int l, r, h;
  for (int i = 0; i < n; i++) {
    cin >> l >> r >> h;
    events.emplace(-l, -r, h);
  }

  barns.emplace(0, -MAX_X - 1);

  ll area = 0, prev_height;
  int prev_loc = 0, cur;
  while (!events.empty()) {
    auto [cur, r, h] = events.top();
    events.pop();
    prev_height = barns.top().first;

    if (h > 0) {
      // entry event
      barns.emplace(h, r);
      events.emplace(r, 0, 0);
    } else {
      // exit event, clear out stale stuff in barns PQ
      while (barns.top().second >= cur) barns.pop();
    }
    area += prev_height * (prev_loc - cur);
    prev_loc = cur;
  }

  cout << area << '\n';

  return 0;
}
```

As implied in class, if we try to maintain a sorted vector, and use the lower_bound method from the algorithm library in the STL to find where to insert into a vector, the solution is indeed too slow.

However, we can use, e.g., a set in place of a priority queue, as in this alternate solution:

```
// Uses the sorted-ness of a set in place of a priority queue.

#include <algorithm>
#include <ios>
#include <iostream>
#include <set>
#include <vector>
using namespace std;

typedef long long ll;

# define MIN_X 1
# define MAX_X 1000000000

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);
  int n;
  cin >> n;
  int status[n];
  for (int i = 0; i < n; ++i) {
    status[i] = 0;
  }
  vector<pair<int, int> > events;
  // The negative height is because the set will be sorting in reverse.
  vector<pair<int, int> > barn_info; // -height, right edge
  set<pair<int, int> > active_barn_heights;
  bool barn_seen[n];
  for (int i = 0; i < n; ++i) {
    barn_seen[i] = false;
  }

  int l, r, h;
  for (int i = 0; i < n; ++i) {
    cin >> l >> r >> h;
    barn_info.push_back(make_pair(-h, r));
    events.push_back(make_pair(l, i)); // entry event
    events.push_back(make_pair(r, i)); // exit event
  }
  // add fake height 0 "barn" to represent the ground, for simplicity
  active_barn_heights.insert(make_pair(0, MAX_X + 1)); // never expires
```

```
    sort(events.begin(), events.end());

    ll area = 0;
    int last_change_position = 0;
    int curr_x, curr_barn, prev_top;
    for (int i = 0; i < 2 * n; ++i) {
      curr_x = events[i].first;
      curr_barn = events[i].second;
      prev_top = -(active_barn_heights.begin()->first);
      if (!barn_seen[curr_barn]) { // entry event
        barn_seen[curr_barn] = true;
        active_barn_heights.insert(barn_info[curr_barn]);
      } else { // exit event, clear stale stuff in array
        while (active_barn_heights.begin()->second <= curr_x) {
      active_barn_heights.erase(active_barn_heights.begin());
        }
      }
      area += (ll)(curr_x - last_change_position) * (ll)prev_top;
      last_change_position = curr_x;
    }
    cout << area << '\n';
  }
```

Languages: C++17

Source File:

(Practice mode: submission will not count toward your final score.)

Submission Results - 1-2-rectangleOverlap.cpp (2024-09-10 20:08:38) - 333 / 333

*	*	*	*	*	*	*
1 3.128MB 4ms	2 3.128MB 2ms	3 3.128MB 2ms	4 3.124MB 3ms	5 3.252MB 15ms	6 3.252MB 16ms	7 3.252MB 17ms

Your Response(s):

1-2-rectangleOverlap.cpp (2024-09-10 16:56:01) - 0 / 333
1-2-rectangleOverlap.cpp (2024-09-10 16:56:30) - 0 / 333
1-2-rectangleOverlap.cpp (2024-09-10 16:56:56) - 0 / 333
1-2-rectangleOverlap.cpp (2024-09-10 17:07:45) - 0 / 333
sol.cpp (2024-09-10 17:17:14) - 333 / 333 [PRACTICE]
1-2-rectangleOverlap.cpp (2024-09-10 20:08:38) - 333 / 333 [PRACTICE]

Problem 3 (46325)

Problem 1-3: Long, Narrow Roads: Report Error

Farmer John has N fields ($1 \leq N \leq 10^5$) that are connected by M bidirectional roads ($1 \leq M \leq 2 \cdot 10^5$). The i^{th} of these roads directly connects fields A_i and B_i ($A_i < B_i$) and is L_i units long ($1 \leq L_i \leq 10^9$), and can accommodate any cow of size up to S_i ($1 \leq S_i \leq 10^9$). Any two fields are directly connected by at most one road.

Farmer John would like to determine the size of the largest cow that can get from field 1 to field N and the length of the shortest path that cow could take to get from field 1 to field $N.$ It is guaranteed that there exists a sequence of roads that can take a cow of size 1 from field 1 to field N.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line of each test case contains two integers, N and M.

Each of the next M lines has four integers A_i, B_i, L_i, and S_i, describing one of the roads.

OUTPUT FORMAT (print output to the terminal / stdout):

Print one line with two integers: the size of the largest cow that can reach field N, and the minimum total path length that a cow of that size would use to get to field N.

SAMPLE INPUT:

```
4 5
2 4 1 1
1 3 1 1
1 2 1 1
3 4 1 1
2 3 1 1
```

SAMPLE OUTPUT:

```
1 2
```

SAMPLE INPUT:

```
4 5
2 4 40 1
1 3 50 1
1 2 30 1
3 4 20 1
2 3 10 1
```

SAMPLE OUTPUT:

```
1 60
```

SAMPLE INPUT:

```
4 5
2 4 40 400
1 3 50 100
```

```
1 2 30 300
3 4 20 500
2 3 10 200
```

SAMPLE OUTPUT:

```
300 70
```

SCORING:

Test cases 1, 4-6 and 10-13 satisfy $L_i = 1$.
Test cases 1-2 and 4-9 satisfy $S_i = 1$.
Test cases 3 and 14-20 satisfy no additional constraints.

Languages: C++17

Source File:

Problem 4 (46745)

Problem 1-4: Pair Moover: Report Error

This is the very first IOI problem! It was the one and only problem in the 1989 contest, and let's just say that things were considerably less difficult then. Formatting conventions for programming problems have also changed a lot in the last three and a half decades, so we have paraphrased this problem into our beloved world of Farmer John. Enjoy!

N cows $(4 \leq N \leq 18, N$ even) are standing in a line. The line contains a total of $\frac{N}{2} - 1$ Guernseys and $\frac{N}{2} - 1$ Holsteins, and two open slots that are next to each other.

Farmer John just bought a new machine, called the Pair Moover, with a claw that can pick up two adjacent cows at a time -- and it has to be *exactly* two, to keep the claw balanced! The claw then moves this pair of cows, *maintaining their relative order*, and deposits them in the two open slots. This all counts as a single use of the Pair Moover.

As usual, for mysterious reasons known only to him, Farmer John wants to use the Pair Moover to reorganize the line such that all Guernseys come before all Holsteins. He does not care where in the line the block of two open slots ends up; in particular, it does *not* necessarily have to separate all of the Guernseys from all of the Holsteins.

Each case consists of T subcases $(1 \leq T \leq 4)$. In each case, find the minimum number of uses of the Pair Moover needed to accomplish the goal, or print IMPOSSIBLE if there is no way. Cases are strings with G standing for Guernsey, H standing for Holstein, and O standing for an open slot.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains T. T subcases follow, each consisting of two lines.
The first line of each subcase contains N.
The second line of each subcase contains a string of N characters, as described above.

OUTPUT FORMAT (print output to the terminal / stdout):

For each subcase, print one line with either the minimum number of uses required or IMPOSSIBLE.

SAMPLE INPUT:

```
3
8
GHHGOOGH
6
GHOOGH
4
GOOH
```

SAMPLE OUTPUT:

```
2
IMPOSSIBLE
0
```

SCORING:

In test cases 2-5, $N \leq 8$.
In test cases 6-9, $N \leq 12$.
In test cases 10-16, $N \leq 16$.
Test cases 17-20 have no additional constraints.

Languages: C++17

Source File:

Problem 5 (46482)

Problem 1-5: Muddy Roads: Report Error

Farmer John has N fields $(2 \leq N \leq 2000)$, numbered 1 through N. There are M bidirectional roads $(2 \leq M \leq 5000)$, numbered 1 through M, each of which connects a pair of fields A_i and B_i $(1 \leq A_i < B_i \leq N)$. Any two fields are directly connected by at most one road, and it is possible to get from any field to any other via some sequence of roads.

Bessie is trying to get from field 1 to field N. But today the roads are very muddy. Road 1 is the least muddy, and then road 2 is the next least muddy, and so on up to road M, which is the muddiest of all.

When Bessie is trying to choose between two paths, she first sees whether either uses the muddiest road, M, and if exactly one of the paths does, she picks the other path. Otherwise, she then sees whether either path uses road $M-1$, and so on.

Find the path that Bessie will actually take.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains N and M.
The i-th of the next M lines represents road i, and contains values A_i, B_i.

OUTPUT FORMAT (print output to the terminal / stdout):

Print one line with the values of the fields visited, in order.

SAMPLE INPUT:

```
4 5
2 3
2 4
3 4
1 3
1 2
```

SAMPLE OUTPUT:

```
1 3 2 4
```

SCORING:
Test cases 2-5 satisfy $M \leq 10$.
Test cases 6-10 satisfy $M \leq 60$.
Test cases 11-20 satisfy no additional constraints.

Languages: C++17

Source File:

Problem 6 (46499)

Problem 1-6: Dual Maze: Report Error
Oh no! Farmer Nhoj has trapped Bessie and Essie in separate mazes, and they will have to work together to escape!

Each maze is an $N \times N$ square grid ($4 \leq N \leq 100$); each cell in the maze is either empty (denoted by a dot) or has a wall (denoted by a pound sign). All of the outer border cells of each maze (i.e., cells for which the row is 1 or N, and/or the column is 1 or N) are walls. The cell $(2, 2)$ is the start, and the cell $(N-1, N-1)$ is the exit. These two cells do not have walls, and they are the upper-left-most and lower-right-most cells without walls.

A cow can move one cell in any of the four orthogonal directions, as long as her move would not take her into a cell wall. However, Farmer Nhoj has added another fiendish twist!

The mazes are stacked one on top of the other, so that Essie's maze is directly above Bessie's. The cows are connected with a magical rope that prevents them from getting too far apart. Specifically, the cows' positions cannot differ by more than K $(1 \leq K \leq \lceil \frac{N}{4} \rceil)$ total unit steps at any time (even if one of the cows is already at the exit!) That is, if we denote Bessie's position by (r_B, c_B) and Essie's position by (r_E, c_E), then at all times we must have $|r_E - r_B| + |c_E - c_B| \leq K$.

The magical rope also requires that only one cow can make a move at a time, and the other cow must stand still as this happens. (The cows do not necessarily have to alternate making moves, though.)

Determine the minimum total number of moves needed for both of the cows to reach their respective exit cells. It is guaranteed that a solution exists.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains N and K.
Each of the next N lines is a string of N characters (each of which is . or #, as described above), representing Bessie's maze.
Each of the next N lines is a string of N characters, in the same format, representing Essie's maze.

OUTPUT FORMAT (print output to the terminal / stdout):

Print one line with the smallest total number of moves needed.

SAMPLE INPUT:

```
7 2
#######
#..#..#
##....#
#.#..##
#.....#
#.....#
#######
#######
#.#...#
#...#.#
#.##..#
#.#..##
#.....#
#######
```

SAMPLE OUTPUT:

```
20
```

SCORING:
Test cases 2-4 satisfy $K = 1$.
Test cases 2-10 satisfy $N \leq 20$.
Test cases 11-20 satisfy no additional constraints.

Languages: C++17

Source File:

Problem 7 (46268)

Problem 1-7: Digit Walk: Report Error
Consider an undirected graph with N vertices $V_1 \ldots V_N$ $(2 \leq N \leq 10^5)$. The vertices are connected by M undirected edges $(1 \leq M \leq 5 \cdot 10^5)$; the i^{th} of these connects vertices A_i and B_i $(1 \leq A_i < B_i \leq N)$. It is guaranteed that every vertex is reachable from every other vertex, and that any pair of vertices is directly connected by at most one edge.

Your goal is to find the minimum possible cost of a path that connects V_1 to vertex V_N . The cost of a path is calculated by summing up the costs of all the edges that make up the path.

Now, what are the edge costs? There are two versions of this problem, type 1 and type 2, as given by a parameter T. In both types, each vertex V_i is labeled with a digit D_i ($0 \le D_i \le 9$).

In type 1, using an edge to travel from a vertex V_x to another vertex V_y costs D_x followed by D_y. For example, to go from vertex V_a with $D_a = 3$ to vertex V_b with digit $D_b = 5$, it costs 35. To go from V_b to V_a costs 53. (If there is a leading zero, it should be dropped.)

In type 2, using an edge to travel from a vertex V_x (that you reached by traveling from vertex V_w) to another vertex V_y costs D_w followed by D_x followed by D_y. (If there is no vertex V_w -- i.e., if V_x is the starting vertex -- then treat D_w as 0.) For example, suppose that vertices V_a, V_b, V_c are all connected to vertex V_c, and $D_a = 1, D_b = 0, D_c = 2$. Then to go from V_b to V_c (having arrived at V_b from V_a on the previous move) costs 102. To go from V_a to V_c (having arrived from V_b on the previous move) costs 12.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains $T, N, M.$
The next line contains $D_1, ..., D_N.$
Each of the next M lines represents an edge and contains two integers $A_i, B_i.$

OUTPUT FORMAT (print output to the terminal / stdout):

Print one line with the answer.

SAMPLE INPUT:

```
1 7 8
3 5 0 0 9 2 4
6 7
1 4
3 4
4 5
3 6
1 2
5 7
2 7
```

SAMPLE OUTPUT:

```
56
```

SAMPLE INPUT:

```
2 7 8
3 5 0 0 9 2 4
6 7
1 4
3 4
4 5
```

```
3 6
1 2
5 7
2 7
```

SAMPLE OUTPUT:

```
356
```

SCORING:

$T = 1$ in test cases 3-10 and $T = 2$ in test cases 11-20.
Test cases 3 and 11 satisfy $N \leq 8, M \leq 28$.
Test cases 4-5 and 12-13 satisfy $N \leq 10^3, M \leq 10^3$.
In test cases 6 and 14, each digit is 1.
The other test cases satisfy no additional constraints.

Languages: C++17

Source File:

Problem 8 (47043)

Problem 1-8: Moo Zoom: Report Error
Bessie is about to drive one loop around her favorite racetrack, starting at the top (degree 0) and going clockwise until she reaches her starting point again.

The track is also currently being used by C other cows $(1 \leq C \leq 5 \cdot 10^5)$ who are less talented drivers. At the time that Bessie starts driving, the i^{th} such cow is at degree $D_i (1 \leq D_i \leq 359)$, numbered clockwise from the top, and moving clockwise at a constant speed such that they complete one full revolution in M_i minutes $(1 \leq M_i \leq 10^9)$. These cows never change their behavior.

Bessie is an expert driver and has perfect control over her speed -- she can change it at any time. It can be any nonnegative (not necessarily integer) value at any time; in particular, she is not restricted to the range of speeds of the other cows.

If we think of Bessie and the other cows as points moving around a circle, Bessie would prefer to have as few meetings with these other cows as possible. Bessie can meet multiple cows at once, or the same cow multiple times, and all of these meetings count individually. Even a meeting at the exact instant Bessie finishes counts.

Find the minimum number of encounters that Bessie could possibly have.

INPUT FORMAT (input arrives from the terminal / stdin):

The first line contains C.
Each of the next lines represents another cow, and contains D_i, M_i.

OUTPUT FORMAT (print output to the terminal / stdout):

Print one line with the answer.

SAMPLE INPUT:

```
3
90 10
270 10
180 5
```

SAMPLE OUTPUT:

```
1
```

SCORING:
Test cases 2-4 satisfy $C \leq 2$.
Test cases 5-9 satisfy $C \leq 100$.
Test cases 10-20 satisfy no additional constraints.

Languages: C++17

Source File: