Data Structures and Algorithms (Spring 2022)
Assignment №2

IT University Innopolis

Contents

1 About this homework

1.1 Coding part

For practical (coding) part you have to provide you program as a single Java class file or C++ file that contains all necessary definitions and a `main` method.

The program should read from standard input and write to standard output unless mentioned otherwise.

1.1.1 Preliminary tests

For some coding parts a CodeForces system will be used for preliminary tests. You will have to specify a submission number in those systems as a proof that your solution is correct. Only correct solutions will be assessed further by TAs.

1.1.2 Code style and documentation

The source code should contain adequate internal documentation in the form of comments. Internal documentation should explain how the code has been tested, e.g. the various scenarios addressed in the test cases.

Do not forget to include your name in the code documentation!

All important exceptions should be handled properly.

Bonus code style points might be awarded for elegant solutions, great documentation and the coverage of test cases. However, these bonus code style points will only be able to cancel the effects of some penalties.

1.2 Theoretical part

Solutions to theoretical exercises have to be elaborate and clear. Write all solutions in a single document, then save or compile it as a PDF for submitting.

Do not forget to include your name in the document!

1.3 Submission

You are required to submit your solutions to all parts via Moodle.

For this assignment you will need to submit:

- Java or C++ class file(s) for the coding parts;
- link(s) to accepted submission(s) in CodeForces as a proof of correctness; please verify that your link has the following format: `https://codeforces.com/group/.../contest/.../submission/<submission_id>`
- a PDF file with solutions for the theoretical part.

Submit files as a single archive with your name and group number. For example, if your name is *Ivanov Ivan* and your group is *B21-05* then your archive should be named `B21-05_Ivanov_Ivan.zip`.

2 Coding part (60 points)

2.1 Simple fraud detection

A Bank has a simple policy for warning clients about possible fraudulent account activity. If the amount spent by a client on a particular day is greater than or equal to 2 times the client's median[1] spending for a trailing number of days, they send the client a notification about potential fraud. The bank doesn't send the client any notifications until they have at least that trailing number of prior days' transaction data. The bank can send multiple notifications per day.

Example 2.1.1. Alice is a client of the Bank and has the following history of spendings:

1. 2022-01-14 — $ 30.00
2. 2022-01-15 — $ 25.00
3. 2022-01-16 — $ 5.00
4. 2022-01-16 — $ 10.00
5. 2022-01-17 — $ 20.00
6. 2022-01-17 — $ 35.00
7. 2022-01-18 — $ 20.00
8. 2022-01-19 — $ 40.00

Suppose that the Bank is tracking 3 trailing days. Then, after first three days, we collect information about total daily spendings, with median total of $ 25.00 (median of $ 30.00, $ 25.00, and $ 15.00). On day 4 (2022-01-17) we have two spendings. The first one ($ 20.00) is below threshold of $ 50.00 ($2 \times$ $ 25.00). The second one brings the daily spendings for that day to $ 55.00 (= $ 20.00 + $ 35.00), triggering the fraud alert, so a notification will be sent to Alice.
On day 5 (2022-01-18), the median for the trailing 3 days remains $ 25.00 (median of $ 25.00, $ 15.00, and $ 55.00). However, here we have no alerts, since $ 20.00 is less than $ 50.00.
On day 6 (2022-01-19), the median for the trailing 3 days is $ 20.00 (median

[1]The *median* of a sequence of numbers can be found by first sorting the numbers in ascending order. If there is an odd number of values, the middle one is picked. If there is an even number of values, the median is then defined to be the average of the two middle values.

of $ 15.00, $ 55.00, and $ 20.00). Since Alice spends $ 40.00 on this day, which is equal to the alert threshold, the Bank is again issuing the notification. So, in total, Alice receives two notifications for these days.

Given the number d ($1 \leq d \leq 10^3$) of trailing days and a client's history of spendings for a period of n days ($1 \leq n \leq 10^7$), determine the number of times the client will receive a notification over all days.

Requirements

To solve this task, you need to implement two sorting algorithms:

1. One algorithm should be used to sort the input data (by date);
2. Another algorithm should be used to sort the data corresponding to the trailing days (by amount spent).

One of the implemented sorting algorithms must be comparison-based. Another one must be a linear based sorting algorithm. For the moving window of trailing days, you can use any queue implementation, including one from a standard library. You can also use standard arrays and lists when implementing sorting algorithms. You cannot use standard sorting.

Input

The input starts with a line, containing two numbers N ($1 \leq N \leq 10^7$) and D ($1 \leq D \leq 10^3$). D defines the number of trailing days that the Bank is tracking. The following N lines describe spendings of the client. Each spending contains a date and the amount of spendings. For each date, spendings are presented in chronological order. However, spendings are not necessarily sorted by date. Example (corresponding to the example above):

```
8 3
2022-01-14 $30.00
2022-01-15 $25.00
2022-01-17 $20.00
2022-01-18 $20.00
2022-01-17 $35.00
2022-01-19 $40.00
2022-01-16 $5.00
2022-01-16 $10.00
```

Output

The output should contain a single number — total number of alerts the Bank sent to the client.

2

2.2 Range queries

When analysing data, it is common to ask questions that involve intervals of some sort:

- How much did a client spend from January 2012 to December 2018?
- Which points/objects lie in a given rectangular area?
- How many students have a grade in range from 60%–90%?

Search trees are well suited to answer these questions efficiently, since in-order traversal of the tree (starting from the first node, satisfying the query) will yield answers sequentially. That is, if the answer to a range query contains k items, then instead of k independent lookups, we can perform a single lookup, followed by a partial in-order traversal ($k-1$ calls to `node.successor`). B-tree is particularly well-suited for range queries, as consequent items are often located in the same array, leveraging high-performance cache.

In this task, you will interpret a series of simple commands operating with a history of client's operations with a bank account. You are required to implement a B-tree and use it to store the history of operations, using date of the operation as key. The implemented B-tree must support efficient range query implementation. You should base your implementation on the following interface:

```
interface RangeMap<K,V> {
    int size();
    boolean isEmpty();
    void add(K key, V value);    // insert new item into the map
    boolean contains(K key);     // check if a key is present
    V lookup(K key);             // lookup a value by the key
    List<V> lookupRange(K from, K to);  // lookup values for a range of keys

    // optional: remove an item from a map (+1% extra credit)
    Object remove(K key);
}
```

Input format. First line of input contains a single number N ($1 \leq N \leq 10^7$). The next N lines contain commands that operate on the bank account. Each command is one of the following:

1. `<YYYY-MM-DD> DEPOSIT <AMOUNT>` — deposit `<AMOUNT>` units of cur-

rency to the client's bank account;

2. `<YYYY-MM-DD> WITHDRAW <AMOUNT>` — withdraw `<AMOUNT>` units of currency from the client's bank account;
3. `REPORT FROM <YYYY-MM-DD> TO <YYYY-MM-DD>` — display the total amount changed during the given period (dates are inclusive); it is guaranteed that `FROM` date is always less than or equal to the `TO` date, constituting a valid date range.

Sample input:

```
8
2022-03-01 DEPOSIT 100
2022-03-02 WITHDRAW 33
2022-03-03 DEPOSIT 45
REPORT FROM 2022-03-01 TO 2022-03-03
2022-03-03 WITHDRAW 72
2022-03-04 WITHDRAW 4
2022-03-05 DEPOSIT 10
REPORT FROM 2022-03-02 TO 2022-03-04
```

Output format. Each line of the output should correspond to a `REPORT` command in the input and contain a single integer — the total amount of balance change in the given period. Note that the amount of change can be negative.

Sample output (corresponding to the sample input above):

```
112
-64
```

2.3 Car rental company

A car rental company wants to place its branches in different cities, with logistics figured out in a way that each branch is accessible from at leas one other branch, and the overall cost of connections is minimized. Each branch (point) has a name and a penalty number which is based on the income the branch makes per year. Branches should receive the products in order of lowest distance penalty ratio. Thus, the cost of a connection is computed as the ratio between the distance and the penalty of the target branch.

In this exercise, your task is to implement

1. A Priority Queue, using binary heap or Fibonacci heap;
2. A Dynamic Graph based on adjacency matrix;
3. Prim's algorithm for the Dynamic Graph, using your heap for the priority queue.

2.3.1 Priority Queue

In this task, you should implement a priority queue satisfying the following interface using a binary heap or Fibonacci Heap implementation. For the Fibonacci heap, you can find detailed description of each method in Section 19 of Cormen et al. *Note that implementing Fibonacci heap will reward you with extra points.*

```
public interface IPriorityQueue<K,V> {
    void insert(Object item);
    Object findMin();
    Object extractMin();
    void decreaseKey(Object item, K newKey);
    void delete(Object item);
    void union(Object anotherQueue);
}
```

Test your heap implementation Write a small program that takes a sequence of instructions to manipulate a set of unordered branches represented by their names and penalty value. The program should output the name of the branch with the minimum penalty every time it is required. Use your implementation of heap.

Input format. The first line of the input contains a single number N ($0 \leq N \leq 10^6$) representing the number operations. Possible operations are:

1. `ADD <BRANCH_NAME> <PENALTY>` — to add a branch in the list;
2. `PRINT_MIN` — to print the branch with minimum penalty (and remove it from the queue);

Output format. Each line of output should correspond to a `PRINT_MIN` line of input and contain the corresponding name of a branch.

Sample input:

```
7
ADD Branch_A 40
ADD Point_B 11
ADD Branch_C 10
PRINT_MIN
ADD Branch_C 32
ADD Branch_D 8
PRINT_MIN
```

Sample output:

```
Branch_C
Branch_D
```

2.3.2 Building the graph and Minimum Spanning Forest

Implement the following interface for a graph based on adjacency matrix.

```
public interface IGraph<V,E> {
    Object insertVertex(V v);
    Object insertEdge(Object from, Object to, E w);
    void removeVertex(Object v);
    void removeEdge(Object e);
    boolean areAdjacent(Object v, Object u);
    int degree(Object v);
    boolean areAdjacent(Object v, Object u);
}
```

Since the company wants to build its network of branches in stages, you

need to compute the minimum cost of building the connections between the branches at different points.

You have to use the methods `insertVertex` and `insertEdge` to build the graph and, to interpret `PRINT_MIN` command, you must use your implementation of Prim's algorithm that relies on your heap priority queue implementation (Task 2.3.1).

Input format. The first line of input contains a single number N ($0 \leq N \leq 10^6$), representing the nmber of commands. Each of the next N lines contains one of the following commands:

1. `ADD <BRANCH_NAME> <PENALTY>` — to add a branch in the list;
2. `CONNECT <BRANCH_NAME_1> <BRANCH_NAME_2> <DISTANCE>` — add possible connection between two existing branches; Note that the weight of the connection is the ratio between `<DISTANCE>` and the `<PENALTY>` of `<BRANCH_NAME_2>` (Cf. figure 1).
3. `PRINT_MIN` — to print the branch with minimum penalty (and remove it from the queue).

Output format. Each line of output should correspond to a `PRINT_MIN` line of input and contain the corresponding minimum spanning forest. Each spanning forest with k edges should be printed as `<EDGE_1> <EDGE_2> ...<EDGE_k>`, where each edge is denoted as `<BRANCH_FROM>:<BRANCH_TO>`.

Sample input:

```
10
ADD Point_A 30000
ADD Point_B 40000
ADD Point_C 50000
ADD Point_D 5000
CONNECT Point_A Point_B 20
CONNECT Point_B Point_D 20
CONNECT Point_D Point_C 30
PRINT_MIN
CONNECT Point_A Point_D 10
PRINT_MIN
```

Sample output:

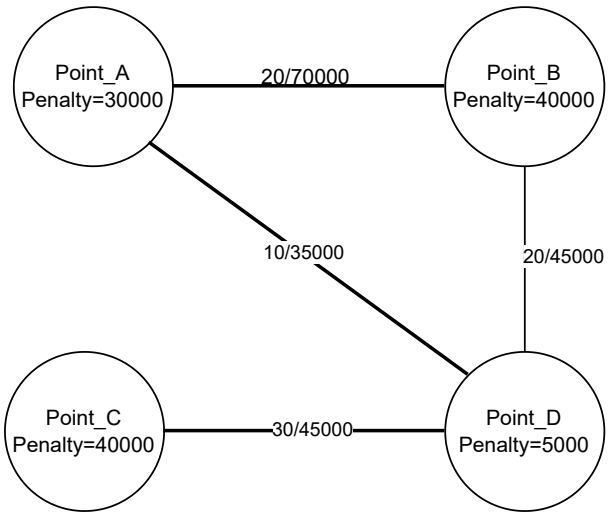

Figure 1: Example of graph representing branches and connection between them. The weight of an edge e connecting branches a and b is $\frac{d_e}{p_a+p_b}$ where d_e is the distance between the two branches, and p_a, p_b the penalties of the branch a and b respectively.

```
Point_A:Point_B Point_B:Point_D Point_D:Point_C
Point_A:Point_B Point_A:Point_D Point_D:Point_C
```

3 Theoretical part (40 points)

3.1 Quicksort fused with insertion sort

We can improve the running time of quicksort in practice by taking advantage of the fast running time of insertion sort when its input is "nearly" sorted. Upon calling quicksort on a subarray with fewer than k elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process.

1. Argue that this sorting algorithm runs in $O(nk + n \log \frac{n}{k})$ expected time.
2. Explain, how should we pick k, both in theory and in practice?

3.2 Binary Search Trees with Equal Keys

Equal keys pose a problem for the implementation of Binary Search Trees. We propose to improve insertion into a Binary Search Tree by testing for each node whether inserted key is equal to the key stored in the inspected nodes. If equality holds, then we employ one of the following strategies.

For each strategy, find the asymptotic time complexity of inserting n items with identical keys into an initially empty binary search tree. Justify your answer for each strategy.

1. No special strategy, use regular insertion algorithm.
2. Keep a boolean flag at every node. This flag determines whether to go left or right when inserting an item with the key equal to the one in the node. Every time we visit the node with the key equal to node's key, we switch the flag.
3. Keep a list of values with equal keys in each node. Insert the item into that list.
4. Randomly choose either left of right. For this strategy, show both the worst-case time complexity, and informally derive the expected time complexity.

3.3 d-ary heaps

A d-ary heap is similar to a binary heap, except non-leaf nodes have d children instead of 2 children (except the last non-leaf node, which is allowed to have fewer children).

1. How should a d-ary heap be represented in an array?
2. What is the height of a d-ary heap with n elements in terms of n and d?
3. Give an efficient implementation of `extractMax` procedure in a d-ary max heap. Analyze the worst-case running rime in terms of n and d.
4. Give an efficient implementation of `insert` procedure in a d-ary max heap. Analyze the worst-case running rime in terms of n and d.
5. **(+1% extra credit)**
   Give an efficient implementation of `increaseKey` procedure in a d-ary max heap. Analyze the worst-case running rime in terms of n and d.

3.4 Bottleneck spanning tree

A bottleneck spanning tree T of an undirected graph G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G. We say that the value of the bottleneck spanning tree is the weight of its maximum-weight edge.

1. Argue that a minimum spanning tree is a bottleneck spanning tree.
2. **(+0.5% extra credit)**
   Give a linear-time algorithm that given a graph G and an integer m, determines whether the value of the bottleneck spanning tree is less than or equal to m.
3. **(+1% extra credit)**
   Use this algorithm as a subroutine in a linear-time algorithm for the bottleneck-spanning-tree problem. *Hint: you may want to implement a procedure to contract (see Section B.4 of Cormen et al.) some edges of a graph.*

3.5 A DP problem (+5% extra credit)

The National Health Council wants to improve health care in three of the most underdeveloped regions. Currently it has five medical teams available to allocate among such regions to improve their medical care, health education, and training programs. Therefore, the council needs to determine how many teams (if any) to allocate to each of these regions to maximize the total effectiveness of the five teams. The teams must be kept intact, so the number allocated to each region must be an integer.

The measure of performance being used is additional person-years of life. (For a particular region, this measure equals the increased life expectancy in years times the region's population.) The following table gives the estimated additional person-years of life (in multiples of 1,000) for each region for each possible allocation of medical teams.

Medical Teams	Region A	Region B	Region C
no teams sent	0	0	0
1 team sent	45	50	20
2 teams sent	70	70	45
3 teams sent	90	80	75
4 teams sent	105	100	110
5 teams sent	110	120	140

1. Describe a general algorithm for any number M of available medical teams, any number R of regions and table `E` with estimates:
    1. Summarize the idea for a recursive algorithm.
    2. Identify overlapping subproblems.
    3. Write down pseudocode for the dynamic programming algorithm that solves the problem (top-down or bottom-up). It is enough to compute maximum performance without specifying team allocation.
    4. Provide asymptotic worst-case time complexity of the algorithm.
2. Apply the algorithm to an instance of the problem above.