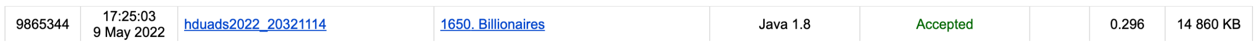
Laboratory work # 6

Student: HU Riqian
Student ID: 20321114
Timus Name: hduads2022_20321114

Mail: jhlxhrq@163.com

Problem # 1650. Millionaires

Screenshot from Timus:



9865344	17:25:03 9 May 2022	hduads2022_20321114	1650. Billionaires	Java 1.8	Accepted		0.296	14 860 KB

Explanation of algorithm:

1. Store the Billionaires and the Cities data in the Map.

2. Create the class of item and event, which are used to describe the millionaires' next movement to the money-dense cities.

3. Every time a new day is given in the input, update the score using the top values in the set as data.

Computational complexity of algorithm:

$$O(\mathrm{N}\log N)$$

Source code:

```
import java.io.*;
import java.util.*;
import java.util.Map.Entry;

import static java.util.Arrays.sort;

public class Timus1650 {
    public static void main(String[] args) throws IOException {
        new Timus1650().run();
    }

    BufferedReader in;
    PrintWriter out;
    StringTokenizer st = new StringTokenizer("");
```

```java
    int billionairesNum = 0;
    int citiesNum = 0;
    long[] fortune;
    Map<String, Integer> billionairesMap = new HashMap<>();
    Map<String, Integer> citiesMap = new HashMap<>();

    String nextToken() throws IOException {
        while (!st.hasMoreTokens())
            st = new StringTokenizer(in.readLine());
        return st.nextToken();
    }

    int nextInt() throws IOException {
        return Integer.parseInt(nextToken());
    }

    long nextLong() throws IOException {
        return Long.parseLong(nextToken());
    }


    void run() throws IOException {
        int[] where;
        int days;
        Event[] events;
        Item[] items;
        long[] init;

        RapidMapping rm;
        in = new BufferedReader(new InputStreamReader(System.in));
        out = new PrintWriter(System.out);

        int billionairesNum = nextInt();
        fortune = new long[billionairesNum];
        where = new int[billionairesNum];

        for (int i = 0; i < billionairesNum; i++) {
            int bil = indexBil(nextToken());
            int dst = indexCity(nextToken());
            long frt = nextLong();
            fortune[bil] = frt;
            where[bil] = dst;
        }

        days = nextInt();

        int mapNum = nextInt();
        events = new Event[mapNum];
        for (int i = 0; i < mapNum; i++)
            events[i] = new Event(nextInt(), indexBil(nextToken()),
indexCity(nextToken()));

        items = new Item[citiesNum];
        for (Entry<String, Integer> e : citiesMap.entrySet()) {
            items[e.getValue()] = new Item(e.getKey());
        }

        init = new long[citiesNum];
        for (int i = 0; i < billionairesNum; i++) {
            init[where[i]] += fortune[i];
        }
        rm = new RapidMapping(init);

        int maxCity;
        int prevDay = 1;
        int curDay;

        for (int i = 0; i < mapNum; ) {
            curDay = events[i].day;
            maxCity = rm.maxIndex();
            if (rm.unique(maxCity))
                items[maxCity].increase(curDay - prevDay + 1);
            prevDay = curDay + 1;
            while (i < mapNum && curDay == events[i].day) {
                int bil = events[i].billionaire;
                int dst = events[i].destination;
                rm.inc(where[bil], -fortune[bil]);
                rm.inc(dst, fortune[bil]);
                where[bil] = dst;
                i++;
            }
        }

        curDay = days;
        maxCity = rm.maxIndex();
        if (rm.unique(maxCity))
            items[maxCity].increase(curDay - prevDay + 1);
        sort(items);

        for (Item item : items)
            if (item.count > 0)
                out.println(item);

        out.close();
    }

    int indexBil(String name) {
        if (!billionairesMap.containsKey(name))
            billionairesMap.put(name, billionairesNum++);
        return billionairesMap.get(name);
    }

    int indexCity(String name) {
        if (!citiesMap.containsKey(name))
            citiesMap.put(name, citiesNum++);
        return citiesMap.get(name);
    }


}

class Event {
    int day;
    int billionaire;
    int destination;

    Event(int day, int billionaire, int destination) {
        this.day = day;
        this.billionaire = billionaire;
        this.destination = destination;
    }
}

class Item implements Comparable<Item> {
    String city;
    int count = 0;

    Item(String city) {
        this.city = city;
    }

    void increase(int add) {
        count += add;
    }

    @Override
    public int compareTo(Item item) {
        return city.compareTo(item.city);
    }

    @Override
    public String toString() {
        return city + " " + count;
    }
}

class RapidMapping {
    int n;
    long[] val;
    int[] index;

    RapidMapping(long[] a) {
        n = a.length;
        val = new long[2 * n];
        index = new int[2 * n];
        for (int i = 0; i < n; i++) {
            val[n + i] = a[i];
            index[n + i] = i;
        }
        build();
    }

    void build() {
        for (int i = n - 1; i > 0; i--) {
            int lt = 2 * i;
            int rt = lt + 1;
            if (val[lt] > val[rt]) {
                val[i] = val[lt];
                index[i] = index[lt];
            } else {
                val[i] = val[rt];
                index[i] = index[rt];
            }
        }
    }

    long get(int i) {
        return val[n + i];
    }

    int maxIndex() {
        return index[1];
    }

    boolean unique(int index) {
        for (int v = (n + index) >> 1; v > 0; v >>= 1) {
            int lt = 2 * v;
            int rt = lt + 1;
            if (val[lt] == val[rt]) {
                return false;
            }
        }
        return true;
    }

    void inc(int ind, long add) {
        set(ind, get(ind) + add);
    }

    void set(int i, long nval) {
        int v = n + i;
        val[v] = nval;
        for (v >>= 1; v > 0; v >>= 1) {
            int lt = 2 * v;
            int rt = lt + 1;
            if (val[lt] > val[rt]) {
                val[v] = val[lt];
                index[v] = index[lt];
            } else {
                val[v] = val[rt];
                index[v] = index[rt];
            }
        }
    }
}
```