小美是美团的一名鲜花快递员，鲜花是一种保质期非常短的商品，所以需要尽快送到客户手中，公司对于骑手的一个要求就是要规划送花的线路，使得骑手送完所有订单走的路程尽可能少。(骑手开始派送时带走了所有需要派送的花，不必每单后返回花店，路程结算是从花店出发，到送完最后一名客户为止，不计算从最后一名客户家回到花店的时间)

公司对于骑手的绩效评价是取决于两个指标，一是从花店到所有客户地址的距离之和，另一个是骑手实际走的路程。

设花店始终位于 1 号位置，客户共有 n-1 个，其编号为 2~n。令 dis(i,j)表示 i 号位置到 j 号位置的距离，即分别计算[illegible], 和骑手实际所走的最短路程。

为了简化问题，我们约束这 n 个位置构成的是一棵树，即只有 n-1 条边在其中互相连接，且保证 n 个点彼此连通。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.HashSet;

public class Main {
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int n = Integer.parseInt(br.readLine().trim());
    UnionFind uf = new UnionFind(n);
    int u, v, w;
    for(int i = 0; i < n - 1; i++){
      String[] params = br.readLine().trim().split(" ");
      u = Integer.parseInt(params[0]);
      v = Integer.parseInt(params[1]);
      w = Integer.parseInt(params[2]);
      uf.union(u, v, w);
    }
    uf.buildLongestPath();    // 构建一下最长路径
    int totalPath = 0, distance = 0;
    for(int i = 0; i < uf.rank.length; i++){
      if(i == 0)
        continue;
      totalPath += uf.rank[i];
      // 不是在最长路径上的边就需要折返
      if(uf.longestPath.contains(i))
        distance += uf.rank[i] - uf.rank[uf.parent[i]];
      else
        distance += 2*(uf.rank[i] - uf.rank[uf.parent[i]]);
    }
    System.out.println(String.format("%d %d", totalPath, distance));
  }
}
```

```
class UnionFind {
    public int[] parent;      // parent[i]用于记录 i 的父节点
    public int[] rank;        // rank[i]表示节点 i（i>2）到花店 1 的距离
    public int maxDisIdx = 1;  // 离花店最远的客户编号
    public int maxDis = 0;    // 离花店最远的距离
    public HashSet<Integer> longestPath;   // 存储最长路径上的节点

    public UnionFind(int n) {
        parent = new int[n + 1];
        rank = new int[n + 1];
        for(int i = 1; i <= n; i++){
            parent[i] = i;
            rank[i] = 0;
        }
    }

    // 合并 u，v 两个节点
    public void union(int u, int v, int w) {
        // v 的父节点为 u
        parent[v] = u;
        // v 到花店的距离为在 u 的基础上加上 w
        rank[v] = rank[u] + w;
        if(rank[v] > maxDis){
            maxDis = rank[v];
            maxDisIdx = v;
        }
    }

    // 构建最长路径
    public void buildLongestPath() {
        longestPath = new HashSet<>();
        int node = maxDisIdx;   // 从距离花店最远的节点开始回溯
        while(node != 1){
            longestPath.add(node);
            node = parent[node];
        }
    }
}
```

美团对于商家的评价体系是 1-5 星评价体系，用户在完成订单之后可以对商家打 1/2/3/4/5 星，而在客户端上，商家的评级却不一定是整数，而是会显示小数点后的一位。很显然这就需要一个计算器了，小美拥有了一些商户的评价数据，希望可以计算出商家在客户端上显示出的评分。

这个评分的计算非常简单，就是对该商家的所有客户的星级评价做求一个平均，然后去尾法显示小数点后的一位即可，例如平均得分是 3.55，则显示的是 3.5。
例如某商家获得了 1-5 星评价各一个，则显示的评分是(1+2+3+4+5)/5=3.0。
如果商家没有获得评价，则显示 0.0。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class Main {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String[] starCounter = br.readLine().trim().split(" ");
      int stars = 0;
      int starNum = 0;
      int count = 0;
      for(int i = 0; i < starCounter.length; i++){
         starNum = Integer.parseInt(starCounter[i]);
         stars += (i + 1)*starNum;
         count += starNum;
      }
      // 通过字符串操作达到去尾法的效果
      String content = stars*1.0 / count + "";
      content = content.substring(0, content.indexOf('.') + 2);
      System.out.println(content);
   }
}
```

2020 年的 618 不再仅仅是购物节啦，同时也是美团外卖节，小美早早就准备好了各种满减代金券，为了最大程度的“省钱”，当然是选择把这些代金券都用光啦！

这些代金券都有一个使用门槛，即满多少元的订单才可以使用。如果使用一个二元组<x,y>表示一张代金券，即需要满 x 元才能优惠 y 元，那么需要注意的是，并不是所有代金券的 x 都是大于等于 y 的，良心美团也会推出一些 x<y 的代金券。如果 x<y,例如 x=1，y=2，则购买 1 元商品的情况下无需付款，不会退款给用户。
请问小美如果想用完这些代金券，在保证总付款金额最小的情况下，她最多购买多少钱的外卖呢？
说明：
1.一个订单只能用一张代金券。
2.同时满足总付款金额最少，且购买的外卖价值最高，例如两个优惠完都是 1 元的外卖，一个原价 3 元另一个原价 4 元，则选四元的。
3.由于美团商户很多，所以对于任何一个价格我们都可以找到至少一种商品购买。
提交的代码

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```

```
public class Main {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int n = Integer.parseInt(br.readLine().trim());
      int value = 0, cost = 0;
      int x, y;
      for(int i = 0; i < n; i++){
         String[] pair = br.readLine().trim().split(" ");
         x = Integer.parseInt(pair[0]);
         y = Integer.parseInt(pair[1]);
         if(x >= y){
            value += x;
            cost += x - y;
         }else
            value += y;
      }
      System.out.println(String.format("%d %d", value, cost));
   }
}
```

外卖节即将过去了，小美还有很代金券没有消费掉，美团面向小美这样的用户推出了一个新的活动，即代金券消消乐活动。系统会把小美的代金券打乱顺序排成一排，小美可以进行任意多次如下操作：

如果存在相邻的两个代金券金额相等,设其面额为 x，小美可以使用这两张代金券换一张面额为 x+1 的代金券，并将其仍放在原来两张券的位置上，每进行一次这样的操作，小美就可以获得 1 元可以无限期使用的奖励金。

小美觉得奖励金可太香了，因此她想获得尽可能多的奖励金，请问她最多可以获得多少奖励金。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.Stack;

public class Main {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int n = Integer.parseInt(br.readLine().trim());
      String[] strArr = br.readLine().trim().split(" ");
      Stack<Integer> stackLeft = new Stack<>();
      Stack<Integer> stackRight = new Stack<>();
      for(int i = 0; i < n; i++)
         stackLeft.push(Integer.parseInt(strArr[i]));
      stackRight.push(stackLeft.pop());
      int money = 0;
```

```
        while(!stackLeft.isEmpty()){
            while(stackLeft.peek() == stackRight.peek()){
                int x = stackLeft.pop() + 1;   // 得到新的代金券值
                stackRight.pop();
                money ++;
                if(!stackRight.isEmpty()){
                    while(!stackRight.isEmpty() && stackRight.peek() == x){
                        x = stackRight.pop();
                        x ++;
                        money ++;
                    }
                    stackRight.push(x);
                }else
                    stackRight.push(x);
            }
            stackRight.push(stackLeft.pop());
        }
        System.out.println(money);
    }
}
```