小团是美团外卖的区域配送负责人，众所周知，外卖小哥一般都会同时配送若干单，小团在接单时希望把同一个小区的单子放在一起，然后由一名骑手统一配送。但是由于订单是叠在一起的，所以，他归类订单时只能知道新订单和已有的某个订单的小区是相同的，他觉得这样太麻烦了，所以希望你帮他写一个程序解决这个问题。

即给出若干个形如 a b 的关系，表示 a 号订单和 b 号订单是同一个小区的 ，请你把同一个小区的订单按照编号顺序排序，并分行输出，优先输出最小的订单编号较小的小区订单集合。订单的编号是 1 到 n。(可能存在同时出现 a b 和 b a 这样的关系,也有可能出现 a a 这样的关系)

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.TreeMap;
import java.util.ArrayList;

public class Main {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String[] params = br.readLine().trim().split(" ");
      int n = Integer.parseInt(params[0]);
      int m = Integer.parseInt(params[1]);
      UnionFind uf = new UnionFind(n);
      int a, b;
      for(int i = 0; i < m; i++){
         params = br.readLine().trim().split(" ");
         a = Integer.parseInt(params[0]);
         b = Integer.parseInt(params[1]);
         uf.union(a, b);
      }
      // 先输出小区数
      System.out.println(uf.count);
      // 再输出每个小区的订单号
      TreeMap<Integer, ArrayList<Integer>> region = new TreeMap<>();
      ArrayList<Integer> temp;
      for(int i = 1; i <= n; i++){
         if(region.containsKey(uf.parent[i]))
            temp = region.get(uf.parent[i]);
         else
            temp = new ArrayList<>();
         temp.add(i);
         region.put(uf.parent[i], temp);
      }
      for(int id: region.keySet()){
         temp = region.get(id);
```

```
            for(int i = 0; i < temp.size(); i++)
                System.out.print(temp.get(i) + " ");
            System.out.println();
        }
    }
}

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

    public int find(int x) {
        while(parent[x] != x){
            // 路径压缩
            parent[x] = parent[parent[x]];
            x = parent[x];
        }
        return x;
    }

    public void union(int x, int y) {
        if(x == y) return;
        int rootX = find(x);
        int rootY = find(y);
        if(rootX == rootY) return;
        // 将节点编号大的合并到节点编号小的节点下面
        if(rootX < rootY){
            for(int i = 0; i < parent.length; ++i){
                if(parent[i] == rootY)
                    parent[i] = rootX;
            }
        }else{
            for(int i = 0; i < parent.length; ++i) {
                if(parent[i] == rootX)
                    parent[i] = rootY;
            }
        }
```

```
        count --;
    }
}
```

小团最近对逆序数（将一个数字逐位逆序，例如 1234 的逆序数为 4321，1100 的逆序数为 11）特别感兴趣，但是又觉得普通的逆序数问题有点太乏味了。
于是他想出了一个新的定义：如果一个数的 4 倍恰好是它的逆序数，那么称这两个数是新定义下的逆序对。
接下来给定一正整数 n，问：不超过 n 的正整数中有多少对新定义下的逆序对？

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

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());
        if(n < 2178){
            System.out.println(0);
        }else{
            int num = 2178;
            int count = 0;
            ArrayList<String> res = new ArrayList<>();
            while(num <= n / 4){
                int revNum = reverse(num);
                if(4*num == revNum){
                    count ++;
                    res.add(num + " " + revNum);
                }
                num ++;
            }
            System.out.println(count);
            for(int i = 0; i < res.size(); i++)
                System.out.println(res.get(i));
        }
    }
    // 对数字 num 进行逆序
    private static int reverse(int num){
        int res = 0;
        while(num > 0){
            res = res*10 + num%10;
            num /= 10;
        }
        return res;
```

```
    }
}
```

小团是一个旅游爱好者，快要过春节了，他想统计一下，在过去的一年中他进行过几次旅行，于是他打开了美团 app 的订单记录，记录显示了他的购买车票的记录。记录是按时间顺序给出的，已知一次旅行的线路一定是一个闭环，即起点和终点是同一个地点。因此当每找到一段闭合的行程，即认为完成了一次旅行。数据保证不会出现不在闭环路径中的数据。

请你在小团的购票记录中统计出他全年共进行了多少次旅行？

```
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 count = 0;
        String[] pos;
        String start = "";
        for(int i = 0; i < n; i++){
            pos = br.readLine().trim().split(" ");
            if(start.equals("")){
                // 起点
                start = pos[0];
            }
            if(pos[1].equals(start)){
                // 回到了起点，旅游次数+1
                count ++;
                start = "";
            }
        }
        System.out.println(count);
    }
}
```

小团是美团汽车租赁公司的调度师，某个时刻 A 和 B 两地都向该公司提交了租车的订单，分别需要 a 和 b 辆汽车。此时，公司的所有车辆都在外运营，通过北斗定位，可以得到所有车辆的位置，小团分别计算了每辆车前往 A 地和 B 地完成订单的利润。作为一名精明的调度师，当然是想让公司的利润最大化了。

请你帮他分别选择 a 辆车完成 A 地的任务，选择 b 辆车完成 B 地的任务。使得公司获利最大,每辆车最多只能完成一地的任务。

```
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[] params = br.readLine().trim().split(" ");
        int n = Integer.parseInt(params[0]);
        int a = Integer.parseInt(params[1]);
        int b = Integer.parseInt(params[2]);
        int x, y;
        // 循环到 i 时，dp[j][k]表示前 i 辆车中派出 j 辆到 A 地，派出 k 辆到 B 地可以获得的最大利润
        int[][] dp = new int[a + 1][b + 1];
        for(int i = 1; i <= n; i++) {
            params = br.readLine().trim().split(" ");
            x = Integer.parseInt(params[0]);
            y = Integer.parseInt(params[1]);
            for(int j = Math.min(a, i); j >= Math.max(0, a - n + i); j--) {
                for(int k = Math.min(b, i - j); k >= Math.max(0, a + b - n + i - j); k--) {
                    if(j == 0 && k == 0) continue;
                    if(k == 0){
                        // B 地的车够了，第 i 辆车不派出或者派到 A 地
                        dp[j][k] = Math.max(dp[j][k], dp[j - 1][k] + x);
                    }else if (j == 0){
                        // A 地的车够了，第 i 辆车不派出或者派到 B 地
                        dp[j][k] = Math.max(dp[j][k], dp[j][k - 1] + y);
                    }else{
                        // 此时第 i 辆车可以选择往 A 地派也可以选择往 B 地派或不派
                        dp[j][k] = Math.max(dp[j][k], Math.max(dp[j - 1][k] + x, dp[j][k - 1] + y));
                    }
                }
            }
        }
        System.out.println(dp[a][b]);
    }
}
```