Java (1)

Template.java

45 lines

```
//Praise our lord and saviour qlf9
//DecimalFormat f = new DecimalFormat("##.00");

import java.util.*;
import java.io.*;
import java.math.*;
import java.text.*;
public class Template{
  public static void main(String[] omkar) throws Exception
  {
    BufferedReader in = new BufferedReader(new
        InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer(in.readLine());
    StringBuilder sb = new StringBuilder();
    int cases = Integer.parseInt(st.nextToken());
    for(int i = 0; i < cases; i++)
    {
      solve(in, st, sb);
    }
    System.out.println(sb);
  }

  public static void sort(int[] arr)
  {
    ArrayList<Integer> ls = new ArrayList<Integer>();
    for(int x: arr){
      ls.add(x);}
    Collections.sort(ls);
    for(int i=0; i < arr.length; i++){
      arr[i] = ls.get(i);}
  }
  public static void solve(BufferedReader in, StringTokenizer
      st, StringBuilder sb) throws Exception
  {
    st = new StringTokenizer(in.readLine());
    int n = Integer.parseInt(st.nextToken());
    st = new StringTokenizer(in.readLine());
  }
  public static int[] readArr(int N, BufferedReader in,
      StringTokenizer st) throws Exception
  {
    int[] arr = new int[N];
    st = new StringTokenizer(in.readLine());
    for(int i=0; i < N; i++)
      arr[i] = Integer.parseInt(st.nextToken());
    return arr;
  }
}
```

Exp.java

26 lines

```
public static int exp(int a, int exp, int mod)
{
  int e = exp;
  if(e==0)
  {
    return 1;
  }
  long b = (long)(a);
  long ret = 1;
  while(e > 1)
  {
    if(e % 2 == 0)
    {
      e = e/2;
      b = (b*b)%mod;
    }
    else
    {
      ret = (ret*b)%mod;
      e = e/2;
      b = (b*b)%mod;
    }
  }
  b = (b*ret)%mod;
  return (int)(b);
}
```

Pair.java

16 lines

```
static class Pair implements Comparable<Pair> {
  int a;
  int b;
  Pair(int a, int b) {
    this.a = a;
    this.b = b;
  }
  @Override
  public int compareTo(Pair other) {
    if (a != other.a) {
      return other.a-a;
    } else {
      return other.b-b;
    }
  }
}
```

Node.java

19 lines

```
static class Node
{
  public int par = -1;
  public ArrayList<Integer> children;
  public int depth = 0;
  public Node()
  {
    children = new ArrayList<Integer>();
  }
  public void add(int i)
  {
    children.add(i);
  }
  public void setPar(int i)
  {
    par = i;
    children.remove(Integer.valueOf(i));
  }
}
```

DisjointUnionSets.java

41 lines

```
class DisjointUnionSets {
  int[] rank, parent;
  int n;
  public DisjointUnionSets(int n)
  {
    rank = new int[n];
    parent = new int[n];
    this.n = n;
    makeSet();
  }
  void makeSet()
  {
    for (int i = 0; i < n; i++) {
      parent[i] = i;
    }
  }
  int find(int x)
  {
    if (parent[x] != x) {
      parent[x] = find(parent[x]);
    }

    return parent[x];
  }
  void union(int x, int y)
  {
    int xRoot = find(x), yRoot = find(y);
    if (xRoot == yRoot)
      return;
    if (rank[xRoot] < rank[yRoot])
      parent[xRoot] = yRoot;
    else if (rank[yRoot] < rank[xRoot])
      parent[yRoot] = xRoot;

    else
    {
      parent[yRoot] = xRoot;
      rank[xRoot] = rank[xRoot] + 1;
    }
  }
}
```

SegmentTree.java

40 lines

```
class SegmentTree
{
  final int[] val;
  final int treeFrom;
  final int length;

  public SegmentTree(int treeFrom, int treeTo)
  {
    this.treeFrom = treeFrom;
    int length = treeTo - treeFrom + 1;
    int l;
    for (l = 0; (1 << l) < length; l++);
    val = new int[1 << (l + 1)];
    this.length = 1 << l;
  }
  public void update(int index, int delta)
  {
    int node = index - treeFrom + length;
    val[node] = delta;
    for (node >>= 1; node > 0; node >>= 1)
      val[node] = comb(val[node << 1], val[(node << 1) + 1]);
  }
  public int query(int from, int to)
  {
    if (to < from)
      return 0;
    from += length - treeFrom;
    to += length - treeFrom + 1;
    int res = 0;
    for (; from + (from & -from) <= to; from += from & -from)
      res = comb(res, val[from / (from & -from)]);
    for (; to - (to & -to) >= from; to -= to & -to)
      res = comb(res, val[(to - (to & -to)) / (to & -to)]);
    return res;
  }
  public int comb(int a, int b)
  {
    return Math.max(a,b);
```

```
  }
}
```

Totient.java

15 lines

```
public static long totient(long n)
{
  long result = n;
  for (int p = 2; p*p <= n; ++p)
    if (n % p == 0)
    {
      while(n%p == 0)
        n /= p;
      result -= result/p;
    }
  if (n > 1)
    result -= result/n;
  return result;

}
```

RangeTotient.java

9 lines

```
// find phi(i) from 1 to N fast
// O(N*loglogN)
long[] arr = new long[N+1];
for(int i=1; i <= N; i++)
  arr[i] = i;
for(int v=2; v <= N; v++)
  if(arr[v] == v)
    for(int a=v; a <= N; a+=v)
      arr[a] -= arr[a]/v;
```

isPrime.java

11 lines

```
public static boolean isPrime(long n)
{
  if(n < 2) return false;
  if(n == 2 || n == 3) return true;
  if(n%2 == 0 || n%3 == 0) return false;
  long sqrtN = (long)Math.sqrt(n)+1;
  for(long i = 6L; i <= sqrtN; i += 6) {
    if(n%(i-1) == 0 || n%(i+1) == 0) return false;
  }
  return true;
}
```

ConvexHull.java

153 lines

```
//make sure to make new file!
import java.io.*;
import java.util.*;
//standard convex hull
public class ConvexHull{

  public static void main(String[] args)throws IOException{
    BufferedReader f = new BufferedReader(new InputStreamReader
        (System.in));
    PrintWriter out = new PrintWriter(System.out);

    while(true){
      int n = Integer.parseInt(f.readLine());

      if(n==0) break;

      ArrayList<Point> points = new ArrayList<Point>();
      HashSet<Point> seen = new HashSet<Point>();
      for(int k = 0; k < n; k++){
        StringTokenizer st = new StringTokenizer(f.readLine());
        int a = Integer.parseInt(st.nextToken());
        int b = Integer.parseInt(st.nextToken());

        Point p = new Point(a,b);
        if(!seen.contains(p)){
          points.add(p);
          seen.add(p);
        }
      }

      Collections.sort(points);

      ArrayList<Point> answer = convexhull(points);

      out.println(answer.size());
      for(Point p : answer){
        out.println(p);
      }
    }

    out.close();
  }

  //takes in array of points sorted by x coordinate from left
      to right
  //outputs convex hull in counterclockwise order, starting
      from leftmost point.
  public static ArrayList<Point> convexhull(ArrayList<Point>
      points){

    ArrayList<Point> hull = new ArrayList<Point>();

    //lower
    Stack<Point> lower = new Stack<Point>();
    lower.add(points.get(0));

    for(int k = 1; k < points.size(); k++){
      //while the are at least 2 elements currently in stack
          and the the last two points + this point make a
          right turn (or collinear), pop
      while(lower.size() >= 2){
        Point last = lower.pop();
        Point last2 = lower.pop();

        if(crossproduct(last2,last,points.get(k)) <= 0){
          lower.push(last2);
        } else {
          lower.push(last2);
          lower.push(last);
          break;
        }
      }
      lower.push(points.get(k));
    }

    //upper hull
    Stack<Point> upper = new Stack<Point>();
    upper.add(points.get(points.size()-1));

    for(int k = points.size()-2; k >= 0; k--){
      //while the are at least 2 elements currently in stack
          and the the last two points + this point make a
          right turn (or collinear), pop
      while(upper.size() >= 2){
        Point last = upper.pop();
        Point last2 = upper.pop();

        if(crossproduct(last2,last,points.get(k)) <= 0){
          upper.push(last2);
        } else {
          upper.push(last2);
          upper.push(last);
          break;
        }
      }
      upper.add(points.get(k));
    }

    //add upper then lower then reverse
    ArrayList<Point> hullupper = new ArrayList<Point>();
    while(!upper.isEmpty()){
      hullupper.add(upper.pop());
    }

    ArrayList<Point> hulllower = new ArrayList<Point>();
    while(!lower.isEmpty()){
      hulllower.add(lower.pop());
    }

    Collections.reverse(hullupper);
    Collections.reverse(hulllower);

    for(Point p : hulllower) hull.add(p);
    for(int k = 1; k < hullupper.size()-1; k++) hull.add(
        hullupper.get(k));          //don't add first and last
          points

    return hull;

  }

  //positive if a->b->c is left turn (counter-clockwise)
  //negative if right turn (clockwise)
  //0 is a,b,c are collinear
  public static int crossproduct(Point a, Point b, Point c){
    return (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
  }

  public static class Point implements Comparable<Point>{
    int x;
    int y;
    public Point(int a, int b){
      x = a;
      y = b;
    }
    //sort by x coordinate from left to right
    //if same x coordinate, sort by y coordinate from down to
        up
    public int compareTo(Point p){
      if(x != p.x)
        return x-p.x;
      return y-p.y;
    }

    public int hashCode(){
      return toString().hashCode();
    }

    public boolean equals(Object o){
      Point p = (Point)o;
      return x==p.x && y==p.y;
    }

    public String toString(){
      return "" + x + " " + y;
    }
  }
```

```
}
```

MaxFlow.java

102 lines

```
class MaxFlow
{
  //Dinic with optimizations (see magic array in dfs function)
  public int N, source, sink;
  public ArrayList<Edge>[] edges;
  private int[] depth;

  public MaxFlow(int n, int x, int y)
  {
    N = n;
    source = x;
    sink = y;
    edges = new ArrayList[N+1];
    for(int i=0; i <= N; i++)
      edges[i] = new ArrayList<Edge>();
    depth = new int[N+1];
  }
  public void addEdge(int from, int to, long cap)
  {
    Edge forward = new Edge(from, to, cap);
    Edge backward = new Edge(to, from, 0L);
    forward.residual = backward;
    backward.residual = forward;
    edges[from].add(forward);
    edges[to].add(backward);
  }
  public long mfmc()
  {
    long res = 0L;
    int[] magic = new int[N+1];
    while(assignDepths())
    {
      long flow = dfs(source, Long.MAX_VALUE/2, magic);
      while(flow > 0)
      {
        res += flow;
        flow = dfs(source, Long.MAX_VALUE/2, magic);
      }
      magic = new int[N+1];
    }
    return res;
  }
  private boolean assignDepths()
  {
    Arrays.fill(depth, -69);
    ArrayDeque<Integer> q = new ArrayDeque<Integer>();
    q.add(source);
    depth[source] = 0;
    while(q.size() > 0)
    {
      int curr = q.poll();
      for(Edge e: edges[curr])
        if(e.capacityLeft() > 0 && depth[e.to] == -69)
        {
          depth[e.to] = depth[curr]+1;
          q.add(e.to);
        }
    }
    return depth[sink] != -69;
  }
  private long dfs(int curr, long bottleneck, int[] magic)
  {
    if(curr == sink)
      return bottleneck;
    for(; magic[curr] < edges[curr].size(); magic[curr]++)
    {
      Edge e = edges[curr].get(magic[curr]);
      if(e.capacityLeft() > 0 && depth[e.to]-depth[curr] == 1)
      {
        long val = dfs(e.to, Math.min(bottleneck, e.
            capacityLeft()), magic);
        if(val > 0)
        {
          e.augment(val);
          return val;
        }
      }
    }
    return 0L;  //no flow
  }
  private class Edge
  {
    public int from, to;
    public long flow, capacity;
    public Edge residual;

    public Edge(int f, int t, long cap)
    {
      from = f;
      to = t;
      capacity = cap;
    }
    public long capacityLeft()
    {
      return capacity-flow;
    }
    public void augment(long val)
    {
      flow += val;
      residual.flow -= val;
    }
  }
}
```