Team Contest Reference

Universität zu Lübeck

22. November 2012

Inhaltsverzeichnis

1 Mathematische Algorithmen

1.1 Primzahlen

Für Primzahlen gilt immer (aber nicht nur für Primzahlen)

$$a^p \equiv a \mod p \quad \text{bzw.} \quad a^{p-1} \equiv 1 \mod p.$$

1.1.1 Sieb des Eratosthenes

```
static boolean[] sieve(int until) {
  boolean[] a = new boolean[until + 1];
  Arrays.fill(a, true);
  for (int i = 2; i < Math.sqrt(a.length); i++) {
    if (a[i]) {
      for (int j = i * i; j < a.length; j += i) a[j] = false;
    }
  }
  return a; // a[i] == true, iff. i is prime. a[0] is ignored
}
```

1.1.2 Primzahlentest

```
static boolean isPrim(int p) {
  if (p < 2 || p > 2 && p % 2 == 0) return false;
  for (int i = 3; i <= Math.sqrt(p); i += 2)
    if (p % i == 0) return false;
  return true;
}
```

1.2 Binomial Koeffizient

```
static int[][] mem = new int[MAX_N][(MAX_N + 1) / 2];
static int binoCo(int n, int k) {
  if (k < 0 || k > n) return 0;
  if (2 * k > n) binoCo(n, n - k);
  if (mem[n][k] > 0) return mem[n][k];
  int ret = 1;
  for (int i = 1; i <= k; i++) {
    ret *= n - k + i;
    ret /= i;
    mem[n][i] = ret;
  }
  return ret;
}
```

1.3 Eulersche φ-Funktion

$\varphi(n \in \mathbb{N}) := |\{a \in \mathbb{N} | 1 \leq a \leq n \wedge \mathrm{ggT}(a, n) = 1\}|$
$\varphi(n \cdot m) = \varphi(n) \cdot \varphi(m)$

```
#include <iostream>
#include <cmath>
using namespace std;
int phi(int);
int main(){
  int n;
  while((cin>>n)!=0) cout << phi(n) << endl;
  return 0;
}

int phi(int n){
  int coprime = 1;
  int primes[] = {2,3,5,7,11,13};//...
```

```
  int primessizes  = 6; //anpassen !
  //zusaetzlich Primfaktorzerlegung v. n
  for(int i =0; i<primessizes; i++){
    int anz = 0;
    while(n % primes[i] == 0){
      n = n / primes[i];
      anz ++;
      cout<<" p: "<<primes[i]<<endl;
    }
    if(anz>0)
      coprime *= ((int) pow((double) primes[i],
        (double)(anz-1))*(primes[i] -
1));
    if(n==1) break;
  }
  if(n != 1){
    coprime *= (n - 1);
  }
  return coprime;
}
```

2 Mathematisch Formeln und Gesetze

2.1 Catalan

$C_n = \frac{1}{n+1}\binom{2n}{n} = \prod_{k=2}^{n}(n+k)/k$
$C_{n+1} = \frac{4n+2}{n+2}C_n = \sum_{k=0}^{n} C_k C_{n-k}$

2.2 kgV und ggT

$ggT(n,m) \cdot kgV(m,n) = |m \cdot n|$

2.3 modulare Exponentiation

$b^e \equiv c(\mod)m$
$b^e = b^{\left(\sum_{i=0}^{n-1} a_i 2^i\right)} = \prod_{i=0}^{n-1}\left(b^{2^i}\right)^{a_i}$

```
function modular_pow(base, exponent, modulus)
    result := 1
    while exponent > 0
        if (exponent mod 2 == 1):
           result := (result * base) mod modulus
        exponent := exponent >> 1
        base = (base * base) mod modulus
    return result
```

2.4 Kreuzprodukt

$$\vec{a} \times \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

2.5 Orthogonale Projektion

r_0 : Ortsvektor; u : Richtungsvektor; n : Normalenvektor
$P_g(\vec{x}) = \vec{r}_0 + \frac{(\vec{x}-\vec{r}_0)\cdot\vec{u}}{\vec{u}\cdot\vec{u}}\,\vec{u}$
$P_g(\vec{x}) = \vec{x} - \frac{(\vec{x}-\vec{r}_0)\cdot\vec{n}}{\vec{n}\cdot\vec{n}}\,\vec{n}$(nur 2D bzw. 3D auf Ebene)

2.6 Geradenschnittpunkt

$g_1 : ax + by = c;\ g_2 : px + qx = r;\ \Rightarrow \vec{p} = \frac{1}{aq-bp} \begin{pmatrix} x = cq - br \\ y = ar - cp \end{pmatrix}$

$g_1 : \vec{p} = \begin{pmatrix} r_x \\ r_y \end{pmatrix} + s \begin{pmatrix} s_x \\ s_y \end{pmatrix}\ g_2 : \vec{p} = \begin{pmatrix} q_x \\ q_y \end{pmatrix} + t \begin{pmatrix} t_x \\ t_y \end{pmatrix}\ w_x = (r_x - q_x), w_y = (r_y - q_y)$

$\Rightarrow D = (s_x t_y - t_x s_y)\ D_s = (t_x w_y - t_y w_x)\ D_t = (s_y w_x - s_x w_y)\ s = D_s/D, t = D_t/D$

2.7 Dreicksfläche

$F = \sqrt{s(s-a)(s-b)(s-c)};\ s = \frac{a+b+c}{2}$

2.8 Kombinatorik

	mit ZL	ohne ZL
Variationen	n^k	$\frac{n!}{(n-k)!}$
Kombinationen	$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$	$\binom{n+k-1}{k} = \binom{n+k-1}{n-1}$

2.9 Modulare Arithmetik

Bedeutung der größten gemeinsamen Teiler:

$$d = \mathrm{ggT}(a, b) = as + bt$$

Verwendung zu Berechnung des inversen Elements b zu a bezüglich einer Restklassengruppe n (a und n müssen teilerfremd sein):

$$ab \equiv 1 \mod n \quad \Leftrightarrow \quad s \equiv b \mod n \quad \text{für } 1 = \mathrm{ggT}(a, n)$$

2.9.1 Erweiterter Euklidischer Algorithmus

```
static int[] eea(int a, int b) {
  int[] dst = new int[3];
  if (b == 0) {
    dst[0] = a;
    dst[1] = 1;
    return dst; // a, 1, 0
  }
  dst = eea(b, a % b);
  int tmp = dst[2];
  dst[2] = dst[1] - ((a / b) * dst[2]);
  dst[1] = tmp;
  return dst;
}
```

3 Datenstukturen

3.1 Fenwick Tree (Binary Indexed Tree)

```
class FenwickTree {
  private int[] values;
  private int n;
  public FenwickTree(int n) {
    this.n = n;
    values = new int[n];
  }
  public int get(int i) { //get value of i
    int x = values[0];
    while (i > 0) {
      x += values[i];
      i -= i & -i; }
    return x;
  }
  public void add(int i, int x) { // add x to interval [i,n]
```

```
    if (i == 0) values[0] += x;
    else {
      while (i < n) {
        values[i] += x;
        i += i & -i; }
    }
  }
}
```

4 Graphenalgorithmen

4.1 Topologische Sortierung

```
static List<Integer> topoSort(Map<Integer, List<Integer>> edges,
    Map<Integer, List<Integer>> revedges) {
  Queue<Integer> q = new LinkedList<Integer>();
  List<Integer> ret = new LinkedList<Integer>();
  Map<Integer, Integer> indeg = new HashMap<Integer, Integer>();
  for (int v : revedges.keySet()) {
    indeg.put(v, revedges.get(v).size());
    if (revedges.get(v).size() == 0)
      q.add(v);
  }
  while (!q.isEmpty()) {
    int tmp = q.poll();
    ret.add(tmp);
    for (int dest : edges.get(tmp)) {
      indeg.put(dest, indeg.get(dest) - 1);
      if (indeg.get(dest) == 0)
        q.add(dest);
    }
  }
  return ret;
}
```

4.2 Prim (Minimum Spanning Tree)

```
#define WHITE 0
#define BLACK 1
#define INF INT_MAX

int baum( int **matrix, int N){
  int i, sum = 0;

  int color[N];
  int dist[N];

    // markiere alle Knoten ausser 0 als unbesucht
  color[0] = BLACK;
  for( i=1; i<N; i++){
    color[i] = WHITE;
    dist[i] = INF;
  }

    // berechne den Rand
  for( i=1; i<N; i++){
        if( dist[i] > matrix[i][nextIndex]){
            dist[i] = matrix[i][nextIndex];
        }
    }

  while( 1){
    int nextDist = INF, nextIndex = -1;

    /* Den naechsten Knoten waehlen */
```

```
    for(i=0; i<N; i++){
      if( color[i] != WHITE) continue;

      if( dist[i] < nextDist){
        nextDist = dist[i];
        nextIndex = i;
      }
    }

    /* Abbruchbedingung*/
    if( nextIndex == -1) break;

    /* Knoten in MST aufnehmen */
    color[nextIndex] = RED;
    sum += nextDist;

    /* naechste kuerzeste Distanzen berechnen */
    for( i=0; i<N; i++){
            if( i == nextIndex || color[i] == BLACK ) continue;

            if( dist[i] > matrix[i][nextIndex]){
                dist[i] = matrix[i][nextIndex];
            }
    }
  }

  return sum;
}
```

4.3 Kruskal

```
public static LinkedList<Edge> kruskal(LinkedList<Edge> adjList, int root, int nodeCount) {
  LinkedList<SortedSet<Integer>> branches = new LinkedList<SortedSet<Integer>>();
  for (int i = 0; i < nodeCount; i++) {
    branches.add(new TreeSet<Integer>());
    branches.get(branches.size() - 1).add(i);
  }

  PriorityQueue<Edge> edges = new PriorityQueue<Edge>(1, new Comparator<Edge>() {
    @Override
    public int compare(Edge e1, Edge e2) {
      if (e1.weight <= e2.weight) {
        return -1;
      } else {
        return 1;
      }
    }
  });
  edges.addAll(adjList);
  LinkedList<Edge> result = new LinkedList<Edge>();

  while (branches.size() > 1) {
    Edge min = edges.remove();

    SortedSet<Integer> from = null;
    for (SortedSet<Integer> branchFrom : branches) {
      if (branchFrom.contains(min.from)) {
        if (!branchFrom.contains(min.to)) {
          from = branchFrom;
          break;
        }
      }
    }

    if (from != null) {
      for (SortedSet<Integer> branchTo : branches) {
        if (!(from.equals(branchTo))) {
```

```
          if (branchTo.contains(min.to)) {
            from.addAll(branchTo);
            branches.remove(branchTo);
            result.add(min);
            break;
          }
        }
      }
    }
  }

  return result;
}
```

4.4 Dijkstra

- alle kürzesten Wege von einem Knoten aus in $\mathcal{O}(\#Kanten + \#Knoten)$
- negative Kanten:
  - auf alle Kantengewichte $|min| + 1$ (damit 0 nicht entsteht)
  - Kantenanzahl zum Ziel mitspeichern

$$\frac{Wegl\ddot{a}nge}{Kantenanzahl \cdot (|min| + 1)}$$

```
// look for shortest distance from a to b in adjacency matrix
// visited nodes for breadth first search
bool nodeVisited[26];
for (int k=0; k<26; k++) {
	nodeVisited[k]=false;
}
queue<int> searchQueue;
queue<string> outputQueue;
searchQueue.push(aNumber); // start search from a
string start="";
start += a[0];
outputQueue.push(start);
string outputString;
while (searchQueue.empty()==false && nodeVisited[bNumber]==false) {
	int node=searchQueue.front();
	searchQueue.pop();
	string nodeString=outputQueue.front();
	outputQueue.pop();
	for (int k=0; k<26; k++) {
		if (cities[node][k]==true && nodeVisited[k]==false) {
			searchQueue.push(k);
			nodeVisited[k]=true;
			char addToOutput=k+'A';
			string s=nodeString;
			s += addToOutput;
			outputQueue.push(s);
			if (k==bNumber) {
				outputString=s;
			}
		}
	}
}
cout << outputString << "\n";
```

4.5 Belman-Ford

```
procedure BellmanFord(list vertices, list edges, vertex source)
   // This implementation takes in a graph, represented as lists of vertices
   // and edges, and modifies the vertices so that their distance and
   // predecessor attributes store the shortest paths.
```

```

  // Step 1: initialize graph
  for each vertex v in vertices:
      if v is source tn v.distance := 0
      else v.distance := infinity
      v.predecessor := null

  // Step 2: relax edges repeatedly
  for i from 1 to size(vertices)-1:
      for each edge uv in edges: // uv is the edge from u to v
          u := uv.source
          v := uv.destination
          if u.distance + uv.weight < v.distance:
              v.distance := u.distance + uv.weight
              v.predecessor := u

  // Step 3: check for negative-weight cycles
  for each edge uv in edges:
      u := uv.source
      v := uv.destination
      if u.distance + uv.weight < v.distance:
          error "Graph contains a negative-weight cycle"
```

4.6 FordFulkerson

```java
import java.util.HashMap;
import java.util.LinkedList;
import java.util.ArrayList;

public class MaximumFlow {
  public static void main(String[] args) {
    int source = 1;
    int sink = 4;
    DirectedGraph g = new DirectedGraph();
    g.addEdge(1, 2, 4);
    g.addEdge(1, 3, 2);
    g.addEdge(2, 4, 1);
    g.addEdge(2, 3, 3);
    g.addEdge(3, 4, 6);
    HashMap<Edge, Integer> flow = getMaxFlow(g, source, sink);
    System.out.println(getFlowSize(flow, g, source));
  }

  static HashMap<Edge, Integer> getMaxFlow(DirectedGraph g, Object source,
      Object sink) {
    LinkedList<Edge> path;
    HashMap<Edge, Integer> flow = new HashMap<Edge, Integer>();
    for (Edge e : g.getEdges()) {
      flow.put(e, 0);
    }

    while ((path = bfs(g, source, sink, flow)) != null) {
      int minCapacity = Integer.MAX_VALUE;
      Object lastNode = source;
      for (Edge edge : path) {
        int c;
        if (edge.getStart().equals(lastNode)) {
          c = edge.getCapacity() - flow.get(edge);
          lastNode = edge.getTarget();
        } else {
          c = flow.get(edge);
          lastNode = edge.getStart();
        }
        if (c < minCapacity) {
          minCapacity = c;
        }
      }
```

```

      lastNode = source;
      for (Edge edge : path) {
        if (edge.getStart().equals(lastNode)) {
          flow.put(edge, flow.get(edge) + minCapacity);
          lastNode = edge.getTarget();
        } else {
          flow.put(edge, flow.get(edge) - minCapacity);
          lastNode = edge.getStart();
        }
      }
    }
    return flow;
  }

  static int getFlowSize(HashMap<Edge, Integer> flow, DirectedGraph g,
      Object source) {
    int maximumFlow = 0;
    Node sourceNode = g.getNode(source);
    for (int i = 0; i < sourceNode.getOutLeadingOrder(); i++) {
      maximumFlow += flow.get(sourceNode.getEdge(i));
    }
    return maximumFlow;
  }

  static LinkedList<Edge> bfs(DirectedGraph g, Object start, Object target,
      HashMap<Edge, Integer> flow) {
    HashMap<Object, Edge> parent = new HashMap<Object, Edge>();
    LinkedList<Object> fringe = new LinkedList<Object>();
    parent.put(start, null);
    fringe.add(start);
    all: while (!fringe.isEmpty()) {
      LinkedList<Object> newFringe = new LinkedList<Object>();
      for (Object nodeID : fringe) {
        Node node = g.getNode(nodeID);
        for (int i = 0; i < node.getOutLeadingOrder(); i++) {
          Edge e = node.getEdge(i);
          if (e.getStart().equals(nodeID)
              && !parent.containsKey(e.getTarget())
              && flow.get(e) < e.getCapacity()) {
            parent.put(e.getTarget(), e);
            if (e.getTarget().equals(target)) {
              break all;
            }
            newFringe.add(e.getTarget());
          } else if (e.getTarget().equals(nodeID)
              && !parent.containsKey(e.getStart())
              && flow.get(e) > 0) {
            parent.put(e.getStart(), e);
            if (e.getStart().equals(target)) {
              break all;
            }
            newFringe.add(e.getStart());
          }
        }
      }
      fringe = newFringe;
    }

    if (fringe.isEmpty()) {
      return null;
    }
    Object node = target;
    LinkedList<Edge> path = new LinkedList<Edge>();
    while (!node.equals(start)) {
      Edge e = parent.get(node);
      path.addFirst(e);
      if (e.getStart().equals(node)) {
```

```
        node = e.getTarget();
      } else {
        node = e.getStart();
      }
    }

    return path;
  }

  public static class DirectedGraph {
    private HashMap<Object, Node> nodes = new HashMap<Object, Node>();
    private LinkedList<Edge> edges = new LinkedList<Edge>();

    void addEdge(Object startNodeID, Object endNodeID, int capacity) {
      Node startNode;
      Node endNode;
      if (!this.nodes.containsKey(startNodeID)) {
        startNode = new Node();
        this.nodes.put(startNodeID, startNode);
      } else {
        startNode = this.nodes.get(startNodeID);
      }
      if (!this.nodes.containsKey(endNodeID)) {
        endNode = new Node();
        this.nodes.put(endNodeID, endNode);
      } else {
        endNode = this.nodes.get(endNodeID);
      }
      Edge edge = new Edge(startNodeID, endNodeID, capacity);
      startNode.addEdge(edge);
      endNode.addEdge(edge);
      this.edges.add(edge);
    }

    Node getNode(Object nodeID) {
      return this.nodes.get(nodeID);
    }

    LinkedList<Edge> getEdges() {
      return this.edges;
    }
  }

  public static class Edge {

    private final Object target;
    private final Object start;
    private final int capacity;

    Edge(Object start, Object target, int capacity) {
      this.capacity = capacity;
      this.target = target;
      this.start = start;
    }

    Object getTarget() {
      return target;
    }

    Object getStart() {
      return start;
    }

    int getCapacity() {
      return capacity;
    }

    @Override
```

```java
    public String toString() {
      return this.start + "->" + this.target + "(" + this.capacity + ")";
    }
  }

  public class Node {

    private ArrayList<Edge> edges = new ArrayList<Edge>();

    void addEdge(Edge edge) {
      this.edges.add(edge);
    }

    Edge getEdge(int number) {
      if (this.edges.size() <= number) {
        return null;
      } else {
        return this.edges.get(number);
      }
    }

    int getOutLeadingOrder() {
      return this.edges.size();
    }
  }
}
```

4.7 Bipartite Matching

4.7.1 JAVA

```java
import java.util.*;

public class BipartiteMatching {
  //Vertex, own class for possible additional properties like names
  static class Vertex {
    List<Edge> links = new ArrayList<Edge>();
  }

  //Edge, saves capacity and saves flow, can compute residual
  static class Edge {
    int capacity;
    int flow = 0;
    Vertex source;
    Vertex dest;

    Edge(int c, Vertex s, Vertex d) {
      capacity = c;
      source = s;
      dest = d;
    }
    //For the on the fly residual graph
    int residualFrom(Vertex v) {
      if (v == dest) return flow;
      else return capacity - flow;
    }
  }

  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    int cases = in.nextInt();

    while (cases-- > 0) {
      int nLeft = in.nextInt();
      int nRight = in.nextInt();
      Vertex source = new Vertex();
      Vertex sink = new Vertex();
```

```java

    // read and add vertices to leftBi (left part of bipartite graph) and connect to source
    List<Vertex> leftBi = new ArrayList<Vertex>();
    for (int i = 0; i < nLeft; i++) {
      Vertex v = new Vertex();
      leftBi.add(capacity=1, source, v);
    }
    // read and add vertices to rightBi (right part of bipartite graph) and connect to source
    List<Vertex> rightBi = new ArrayList<Vertex>();
    for (int i = 0; i < nRight; i++) {
      Vertex v = new Vertex();
      rightBi.add(capacity=1, v, sink);
    }
    // add edges inbetween to both vertices, so that during the BFS
    // the residual flow can be found easily -- Vertex.links.add(Edge) - TODO

    // add all vertices to the flow Network
    List<Vertex> flowNet = new ArrayList<Vertex>();
    flowNet.add(source); flowNet.addAll(leftBi);
    flowNet.addAll(rightBi); flowNet.add(sink);

    //do Ford-Fulkerson
    ford_fulkerson: while (true) {
      // 1 - Find Augmenting Path in Residual Flow Network per BFS

      //HashMap for reconstructing the augmenting path
      HashMap<Vertex, Edge> edgeToParent = new HashMap<Vertex, Edge>();
      List<Vertex> fringe = new ArrayList<Vertex>();
      fringe.add(source);
      edgeToParent.put(source, null);
      int minResidual = Integer.MAX_VALUE;
      boolean foundResPath = false;

      bfs: while (!fringe.isEmpty()) {
        List<Vertex> newFringe = new ArrayList<Vertex>();
        for (Vertex v : fringe) {
          for (Edge e : v.links) {
            //determine the child node, since edges can be in both directions
            Vertex child = (e.dest == v) ? e.source : e.dest;
            //only handle, if this vertex has not been visited
            //and still has residual capacity
            if (!edgeToParent.containsKey(child) && e.residualFrom(v) > 0) {
              edgeToParent.put(child, e);
              newFringe.add(child);
              minResidual = Math.min(minResidual, e.residualFrom(v));
              if (child == sink) {
                foundResPath = true;
                break bfs;
              }
            }
          }
        }
        fringe = newFringe;
      }
      if (!foundResPath) break ford_fulkerson;

      // 2 - alter graph according to augmenting path
      Vertex nextVertex = sink;
      while (nextVertex != source) {
        Vertex prevVertex = nextVertex;
        Edge edge = edgeToParent.get(prevVertex);
        if (edge.source == prevVertex) {
          edge.flow = edge.flow - minResidual;
          nextVertex = edge.dest;
        } else {
          edge.flow = edge.flow + minResidual;
          nextVertex = edge.source;
        }
```

```
        }
      }
      // check condition and print answer
    }
  }
}
```

4.7.2 fast implementaion

```
 int m, n;
    boolean[][] graph;
    boolean seen[];
    int matchL[]; //What left vertex i is matched to (or -1 if unmatched)
    int matchR[]; //What right vertex j is matched to (or -1 if unmatched)

    int maximumMatching() {
        //Read input and populate graph[][]
        //Set m to be the size of L, n to be the size of R
        Arrays.fill(matchL, -1);
        Arrays.fill(matchR, -1);

        int count = 0;
        for (int i = 0; i < m; i++) {
            Arrays.fill(seen, false);
            if (bpm(i)) count++;
        }
        return count;
    }

    boolean bpm(int u) {
        //try to match with all vertices on right side
        for (int v = 0; v < n; v++) {
            if (!graph[u][v] || seen[v]) continue;
            seen[v] = true;
            //match u and v, if v is unassigned, or if v's match on the left side can be reassigned to another right ve
            if (matchR[v] == -1 || bpm(matchR[v])) {
                matchL[u] = v;
                matchR[v] = u;
                return true;
            }
        }
        return false;
    }
```

5 Geometrische Algorithmen

5.1 Graham Scan (Convex Hull)

```
static List<P> graham(List<P> l) {
  if (l.size() < 3)
    return l;
  P temp = l.get(0);
  for (P p : l)
    if (temp.y > p.y || temp.y == p.y && temp.x > p.x)
      temp = p;
  final P start = temp; // min y (then leftmost)

  Collections.sort(l, new Comparator<P>() {
    public int compare(P o1, P o2) {
      if (new Double(Math.atan2(o1.y - start.y, o1.x - start.x)) // same angle
          .compareTo(Math.atan2(o2.y - start.y, o2.x - start.x)) == 0)
        return new Double(Math.sqrt((o1.x - start.x)
            * (o1.x - start.x) + (o1.y - start.y)
            * (o1.y - start.y))).compareTo((o2.x - start.x)
            * (o2.x - start.x) + (o2.y - start.y)
```

```
            * (o2.y - start.y)); // use distance
      return new Double(Math.atan2(o1.y - start.y, o1.x - start.x))
          .compareTo(Math.atan2(o2.y - start.y, o2.x - start.x));
    }
  });
  Stack<P> s = new Stack<P>();
  s.add(start);
  s.add(l.get(1));
  for (int i = 2; i < l.size(); i++) {
    while (s.size() >= 2
        && ccw(s.get(s.size() - 2), s.get(s.size() - 1), l.get(i)) <= 0)
      s.pop();
    s.push(l.get(i));
  }
  return s;
}

// turn is counter-clockwise if > 0; collinear if = 0; clockwise else
static double ccw(P p1, P p2, P p3) {
  return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
}

public static class P {
  double x, y;

  P(double x, double y) {
    this.x = x;
    this.y = y;
  }
  // polar coordinates (not used)
  // double r() { return Math.sqrt(x * x + y * y); }
  // double d() { return Math.atan2(y, x); }
}
```

5.2 Line Intersection

- **Mehr als 2 Linien:**
- findet nicht alle Intersection Points, aber immer wenn einer existiert, dann angegeben
- $O(n \log n + l \log n)$

---

initialize the structure Q (sorted by y-coordinates) for the event points and T for the adjacency of line segments.
insert all end points of lines into Q (they will get sorted). Upper end points are stored with their line segment.
while event point in Q **do**
    find all line segments in T that contain p
    if this are more than one, store p as intersection point
    sort the line segments in T so that they are in the order that exists directly below p
    check the both outer line segments that passed p for intersection with their neighbours which have nnot passed p
    if there us ab intersection, store it as an event point in Q
    remove p from Q
end while

- **2 Linien:**
- line intersection (test if possible!)
- Achtung: beide Reihenfolgen testen: if ((checkLines(readLines[j],newLine) == true) && (checkLines(newLine,readLines[j]) == true))

```
struct line {
   int x0;
   int y0;
```

```
    int x1;
    int y1;
};

// prueft, ob sich die Linien schneiden koennen
bool checkLines(line a, line b) {
    // Vektor Linie a
    int x0 = a.x1 - a.x0;
    int y0 = a.y1 - a.y0;
    // Vektor zu Startpunkt b
    int x1 = b.x0 - a.x0;
    int y1 = b.y0 - a.y0;
    // Vektor zu Endpunkt b
    int x2 = b.x1 - a.x0;
    int y2 = b.y1 - a.y0;
    // Kreuzprodukte berechnen
    int crossProduct1 = x0 * y1 + x1 * y0;
    int crossProduct2 = x0 * y2 + x2 * y0;
    // Wenn ein Produkt negativ, das andere positiv ist, koennen sich die Linien schneiden
    if (crossProduct1 * crossProduct2 < 0) {
        return true;
    }
    return false;
}
```

5.3 Punkt in Polygon

```
  /**
   * -1: A->R schneidet BC (ausser unterer Endpunkt)
   *  0: A auf BC
   * +1: sonst
   */
  public static int KreuzProdTest(double ax, double ay, double bx, double by,
      double cx, double cy) {
    if (ay == by && by == cy) {
      if ((bx <= ax && ax <= cx) || (cx <= ax && ax <= bx))
        return 0;
      else
        return +1;
    }
    if(by>cy){double tmpx=bx;double tmpy=by; bx=cx;by=cy;cx=tmpx;cy=tmpy;}
    if(ay==by && ax==bx) return 0;
    if(ay<=by || ay>cy) return +1;
    double delta = (bx-ax)*(cy-ay)-(by-ay)*(cx-ax);
    if(delta>0)return -1; else if(delta<0)return +1;else return 0;
  }
  /**
   * Input: P[i] (x[i],y[i]); P[0]:=P[n]
   * -1: Q ausserhalb Polygon
   *  0: Q auf Polygon
   * +1: Q innerhalt des Polygons
   */
  public static int PunktInPoly(double[] x,double[] y, double qx,double qy){
    int n = x.length - 1;
    int t = -1;
    for (int i = 0; i <= n - 1; i++) {
      t = t * KreuzProdTest(qx, qy, x[i], y[i], x[i + 1], y[i + 1]);
    }
    return t;
  }
```

6 Verschiedenes

6.1 Potenzmenge

```java
static <T> Iterator<List<T>> powerSet(final List<T> l) {
  return new Iterator<List<T>>() {
    int i; // careful: i becomes 2^l.size()
    public boolean hasNext() {
      return i < (1 << l.size());
    }
    public List<T> next() {
      Vector<T> temp = new Vector<T>();
      for (int j = 0; j < l.size(); j++)
        if (((i >>> j) & 1) == 1)
          temp.add(l.get(j));
      i++;
      return temp;
    }
    public void remove() {}
    };
  }
```

6.2 LongestCommonSubsequence

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>
using namespace std;

#define MAX(a,b) (a > b) ? a : b

string X,Y;
vector< vector<int> > c(101, vector<int>(101,0));
int m,n,ctr;

int LCS()
{
     m = X.length(),n=Y.length();

    c.resize(m+1);
  for(int i = 0; i<n+1; i++) {
    c[i].resize(n+1);
    c[i][0] = 0;
  }

     int i,j;

     for (i=0;i<=m;i++)
         for (j=0;j<=n;j++)
             c[i][j]=0;

     for (i=1;i<=m;i++)
         for (j=1;j<=n;j++)
         {
             if (X[i-1]==Y[j-1])
                c[i][j]=c[i-1][j-1]+1;
             else
                 c[i][j]=max(c[i][j-1],c[i-1][j]);
         }
     return c[m][n];
}
/** Print a songle LCS */
void printLCS(int i,int j)
{
    if (i==0 || j==0)
       return;
    if (X[i-1]==Y[j-1])
```

```
    {
       printLCS(i-1,j-1);
       cout<<X[i-1];
    }
    else if (c[i][j]==c[i-1][j])
         printLCS(i-1,j);
    else
        printLCS(i,j-1);
}

int main()
{
    while(cin>>X>>Y)
    {
  cout << "Length:␣" << LCS() << endl;
        printLCS(m,n);
        cout<<endl ;
    }
}
```

6.3 LongestCommonSubstring

```
  private static List<String> longestCommonSubstring(String S1, String S2)
  {
    List<String> ret = new ArrayList<String>();
    List<Integer> idx  =new ArrayList<Integer>();
      int Start = 0;
      int Max = 0;
      for (int i = 0; i < S1.length(); i++)
      {
          for (int j = 0; j < S2.length(); j++)
          {
              int x = 0;
              while (S1.charAt(i + x) == S2.charAt(j + x))
              {
                  x++;
                  if (((i + x) >= S1.length()) || ((j + x) >= S2.length())) break;
              }
              if (x > Max)
              {
                  Max = x;
                Start = i;
                idx.clear();
                idx.add(Start);
              } else if(x==Max){
                Start = i;
                idx.add(Start);
              }
           }
      }
      HashSet<String> set = new HashSet<String>(idx.size(),1f);
      for(Integer start : idx){
        String substr = S1.substring(start,start+Max);
        if(!set.contains(substr)){
          ret.add(substr);
          set.add(substr);
        }
      }
      Collections.sort(ret);
      //return S1.substring(Start, (Start + Max));
      return ret;
  }
```

6.4 LongestIncreasingSubsequence

```cpp
#include <vector>
using namespace std;

/** finde LIS in O(n log k)
 *a: Sequenz (in)
 *b: LIS (out)
 */
void find_lis(vector<int> &a, vector<int> &b)
{
  vector<int> p(a.size());
  int u, v;
  if (a.empty()) return;
  b.push_back(0);

  for (size_t i = 1; i < a.size(); i++)
        {
        // ist naechstes Element a[i] groesser als letztes der aktuelle LIS
    // a[b.back()], fuege es (Index) an "b" an.
    if (a[b.back()] < a[i]) {
      p[i] = b.back();
      b.push_back(i);
      continue;
    }

        // finde kleinstes El. in LIS (index in b) welches gerade groesser als a[i] ist
        // binaere suche |b|<=k  =>  O(log k)
    for (u = 0, v = b.size()-1; u < v;)
        {
      int c = (u + v) / 2;
      if (a[b[c]] < a[i]) u=c+1; else v=c;
    }

        // aktualisiere b falls neuer Wert kleiner als vorheriger kleinerer Wert
    if (a[i] < a[b[u]])
                {
      if (u > 0) p[i] = b[u-1];
      b[u] = i;
    }
  }

  for (u = b.size(), v = b.back(); u--; v = p[v]) b[u] = v;
}

#include <cstdio>
int main()
{
  int a[] = { 1, 9, 3, 8, 11, 4, 5, 6, 4, 19, 7, 1, 7 };
  vector<int> seq(a, a+sizeof(a)/sizeof(a[0])); // seq : Eingabesequent
  vector<int> lis;                              // lis : Index Vektor fuer LIS
    find_lis(seq, lis);
     //Sequenz ausgeben:
  for (size_t i = 0; i < lis.size(); i++)
    printf("%d ", seq[lis[i]]);
        printf("\n");

  return 0;
}
```

6.5 Permutation & Sequenzen

```java
import java.util.Scanner;

public class PermsAndSequ {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n;
```

```java
    while ((n = sc.nextInt()) != 0) {
      int k = sc.nextInt();
      Sequences(n, k);
      Permutations(n);
    }

  }

  public static void Sequences(int n, int k) {
    int[] x = new int[k];
    for (int i = 0; i < k; i++)
      x[i] = 1;
    Print(x);
    while (true) {
      boolean lastX = true;
      for (int i = 0; i < k; i++)
        if (x[i] != n) {
          lastX = false;
          break;
        }
      if (lastX)
        break;
      int p = k - 1;
      while (!(x[p] < n))
        p--;
      x[p] = x[p] + 1;
      for (int i = p + 1; i < k; i++)
        x[i] = 1;
      Print(x);
    }
  }

  public static void Permutations(int n) {
    int[] x = new int[n];
    for (int i = 0; i < n; i++)
      x[i] = i + 1;
    Print(x);
    while (true) {
      boolean lastX = true;
      for (int i = 0; i < n - 1; i++)
        if (x[i] < x[i + 1]) {
          lastX = false;
          break;
        }
      if (lastX)
        break;
      int k = n - 1 - 1;

      while (x[k] > x[k + 1])
        k--;
      int t = k + 1;

      while (t < (n - 1) && x[t + 1] > x[k])
        t++;

      int tmp = x[k];
      x[k] = x[t];
      x[t] = tmp;
      // reverse x[k+1] ... x[n-1]

      for (int i = 0; i <= ((n - 1) - (k + 1)) / 2; i++) {
        tmp = x[k + 1 + i];
        x[k + 1 + i] = x[n - 1 - i];
        x[n - 1 - i] = tmp;

      }
      Print(x);
    }
```

```
  }

  public static void Print(int[] x) {
    for (int i = 0; i < x.length; i++)
      System.out.print(x[i] + " ");
    System.out.println("");
  }

}
```

7 Formatierung & Sonstiges

7.1 Ausgabeformatierung mit JAVA - `DecimalFormat`

Symbol	Bedeutung
0	(Ziffer) – unbelegt wird eine Null angezeigt. (0.234=(00.00)=>00.23)
#	(Ziffer) – unbelegt bleibt leer, (keine unnötigen nullen).
.	Dezimaltrenner.
,	Gruppiert die Ziffern (eine Gruppe ist so groß wie der Abstand von ",ßu ".").
;	Trennzeichen. Links Muster für pos., rechts für neg. Zahlen
-	Das Standardzeichen für Negativpräfix
%	Prozentwert.
%%	Promille.
X	Alle anderen Zeichen X können ganz normal benutzt werden.
’	Ausmarkieren von speziellen Symbolen im Präfix oder Suffix

7.2 Ausgabeformatierung mit `printf`

```
%d %i Decimal signed integer.
%o Octal int.
%x %X Hex int.
%u Unsigned int.
%c Character.
%s String. siehe unten.
%f double
%e %E double.
%g %G double.

-      linksbündig.
0     Felder mit 0 ausfüllen
      (an Stelle von Leerzeichen).


+   Vorzeichen immer ausgeben.
blank  pos. Zahlen mit Leerzeichen beg.
#    verschiedene Bedeutung:
 %#o (Oktal) 0 Präfix wird eingefügt.
 %#x (Hex)   0x Präfix bei !=0
 %#X (Hex)   0X Präfix bei !=0
 %#e  Dezimalpunkt immer anzeigen.
 %#E  Dezimalpunkt immer anzeigen.
 %#f  Dezimalpunkt immer anzeigen.
 %#g
 %#G  Dezimalpunkt immer anzeigen.
      Nullen nach Dzmpkt. bleiben

int i = 123;
```

```
printf( "|%d|   |%d|\n" ,     i, -i);    // |123|   |-123|
printf( "|%5d| |%5d|\n" ,     i, -i);    // |  123| | -123|
printf( "|%-5d| |%-5d|\n" ,   i, -i);    // |123  | |-123 |
printf( "|%+-5d| |%+-5d|\n" , i, -i);    // |+123 | |-123 |
printf( "|%05d| |%05d|\n\n",  i, -i);    // |00123| |-0123|
printf( "|%X| |%x|\n", 0xabc, 0xabc );   // |ABC| |abc|
printf( "|%08x| |%#x|\n\n", 0xabc, 0xabc ); // |00000abc| |0xabc|
double d = 1234.5678;
printf( "|%f| |%f|\n" ,         d, -d);  // |1234,567800| |-1234,567800|
printf( "|%.2f| |%.2f|\n" ,     d, -d);  // |1234,57| |-1234,57|
printf( "|%10f| |%10f|\n" ,     d, -d);  // |1234,567800| |-1234,567800|
printf( "|%10.2f| |%10.2f|\n" , d, -d);  // |   1234,57| |  -1234,57|
printf( "|%010.2f| |%010.2f|\n",d, -d);  // |0001234,57| |-001234,57|
String s = "Monsterbacke";
printf( "\n|%s|\n", s );                 // |Monsterbacke|
printf( "|%20s|\n", s );                 // |        Monsterbacke|
printf( "|%-20s|\n", s );                // |Monsterbacke        |
printf( "|%7s|\n", s );                  // |Monsterbacke|
printf( "|%.7s|\n", s );                 // |Monster|
printf( "|%20.7s|\n", s );               // |             Monster|
```

7.3 C++ Eingabe ohne bekannt Länge

```
#include <iostream>
#include <sstream>
#include <istream>
#include <string>
#include <vector>
#include <cstdlib>

using namespace std;
int main(){
  string s;
  do{
    getline(cin,s);
    istringstream* ss;
    ss = new istringstream( s );
    while (!ss->eof())
    {
      string xs;
      getline( *ss, xs, ' ' );  // try to read the next field into it

      int x = atoi(xs.c_str());
      cout<<" "<<xs;
    }
    cout<<endl;
  } while(!cin.eof());
}
```