Team Contest Reference

Universität zu Lübeck

22. November 2012

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

```java
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

```java
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

```java
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

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

```cpp
#include <iostream>
#include <cmath>
using namespace std;
```

```cpp
4  int phi(int);
5  int main(){
6    int n;
7    while((cin>>n)!=0) cout << phi(n) << endl;
8    return 0;
9  }
10
11 int phi(int n){
12   int coprime = 1;
13   int primes[] = {2,3,5,7,11,13};//...
14   int primessizes  = 6; //anpassen !
15   //zusaetzlich Primfaktorzerlegung v. n
16   for(int i =0; i<primessizes; i++){
17     int anz = 0;
18     while(n % primes[i] == 0){
19       n = n / primes[i];
20       anz ++;
21       cout<<" p: "<<primes[i]<<endl;
22     }
23     if(anz>0)
24       coprime *= ((int) pow((double) primes[i],
25         (double)(anz-1))*(primes[i] -
26 1));
27     if(n==1) break;
28   }
29   if(n != 1){
30     coprime *= (n - 1);
31   }
32   return coprime;
33 }
```

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

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

```
1  function modular_pow(base, exponent, modulus)
2      result := 1
3      while exponent > 0
4          if (exponent mod 2 == 1):
5              result := (result * base) mod modulus
6          exponent := exponent >> 1
7          base = (base * base) mod modulus
8      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   Dreiecksfläche

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

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
```

```
 9      int x = values[0];
10      while (i > 0) {
11        x += values[i];
12        i -= i & -i; }
13      return x;
14    }
15    public void add(int i, int x) { // add x to interval [i,n]
16      if (i == 0) values[0] += x;
17      else {
18        while (i < n) {
19          values[i] += x;
20          i += i & -i; }
21      }
22    }
23 }
```

4 Graphenalgorithmen

4.1 Topologische Sortierung

```
 1 static List<Integer> topoSort(Map<Integer, List<Integer>> edges,
 2     Map<Integer, List<Integer>> revedges) {
 3   Queue<Integer> q = new LinkedList<Integer>();
 4   List<Integer> ret = new LinkedList<Integer>();
 5   Map<Integer, Integer> indeg = new HashMap<Integer, Integer>();
 6   for (int v : revedges.keySet()) {
 7     indeg.put(v, revedges.get(v).size());
 8     if (revedges.get(v).size() == 0)
 9       q.add(v);
10   }
11   while (!q.isEmpty()) {
12     int tmp = q.poll();
13     ret.add(tmp);
14     for (int dest : edges.get(tmp)) {
15       indeg.put(dest, indeg.get(dest) - 1);
16       if (indeg.get(dest) == 0)
17         q.add(dest);
18     }
19   }
20   return ret;
21 }
```

4.2 Prim (Minimum Spanning Tree)

```
 1 #define WHITE 0
 2 #define BLACK 1
 3 #define INF INT_MAX
 4
 5 int baum( int **matrix, int N){
 6   int i, sum = 0;
 7
 8   int color[N];
 9   int dist[N];
10
11   // markiere alle Knoten ausser 0 als unbesucht
12   color[0] = BLACK;
13   for( i=1; i<N; i++){
14     color[i] = WHITE;
15     dist[i] = INF;
16   }
17
18   // berechne den Rand
19   for( i=1; i<N; i++){
20       if( dist[i] > matrix[i][nextIndex]){
21           dist[i] = matrix[i][nextIndex];
22       }
23   }
```

```
24
25    while( 1){
26      int nextDist = INF, nextIndex = -1;
27
28      /* Den naechsten Knoten waehlen */
29      for(i=0; i<N; i++){
30        if( color[i] != WHITE) continue;
31
32        if( dist[i] < nextDist){
33          nextDist = dist[i];
34          nextIndex = i;
35        }
36      }
37
38      /* Abbruchbedingung*/
39      if( nextIndex == -1) break;
40
41      /* Knoten in MST aufnehmen */
42      color[nextIndex] = RED;
43      sum += nextDist;
44
45      /* naechste kuerzeste Distanzen berechnen */
46      for( i=0; i<N; i++){
47            if( i == nextIndex || color[i] == BLACK ) continue;
48
49            if( dist[i] > matrix[i][nextIndex]){
50                dist[i] = matrix[i][nextIndex];
51            }
52      }
53    }
54
55    return sum;
56 }
```

4.3 Kruskal

```
1  public static LinkedList<Edge> kruskal(LinkedList<Edge> adjList, int root, int nodeCount) {
2    LinkedList<SortedSet<Integer>> branches = new LinkedList<SortedSet<Integer>>();
3    for (int i = 0; i < nodeCount; i++) {
4      branches.add(new TreeSet<Integer>());
5      branches.get(branches.size() - 1).add(i);
6    }
7
8    PriorityQueue<Edge> edges = new PriorityQueue<Edge>(1, new Comparator<Edge>() {
9      @Override
10     public int compare(Edge e1, Edge e2) {
11       if (e1.weight <= e2.weight) {
12         return -1;
13       } else {
14         return 1;
15       }
16     }
17   });
18   edges.addAll(adjList);
19   LinkedList<Edge> result = new LinkedList<Edge>();
20
21   while (branches.size() > 1) {
22     Edge min = edges.remove();
23
24     SortedSet<Integer> from = null;
25     for (SortedSet<Integer> branchFrom : branches) {
26       if (branchFrom.contains(min.from)) {
27         if (!branchFrom.contains(min.to)) {
28           from = branchFrom;
29           break;
30         }
31       }
32     }
```

```
33
34     if (from != null) {
35       for (SortedSet<Integer> branchTo : branches) {
36         if (!(from.equals(branchTo))) {
37           if (branchTo.contains(min.to)) {
38             from.addAll(branchTo);
39             branches.remove(branchTo);
40             result.add(min);
41             break;
42           }
43         }
44       }
45     }
46   }
47
48   return result;
49 }
```

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

$$\frac{Weglänge}{Kantenanzahl \cdot (|min| + 1)}$$

```
1  // look for shortest distance from a to b in adjacency matrix
2  // visited nodes for breadth first search
3  bool nodeVisited[26];
4  for (int k=0; k<26; k++) {
5         nodeVisited[k]=false;
6  }
7  queue<int> searchQueue;
8  queue<string> outputQueue;
9  searchQueue.push(aNumber); // start search from a
10 string start="";
11 start += a[0];
12 outputQueue.push(start);
13 string outputString;
14 while (searchQueue.empty()==false && nodeVisited[bNumber]==false) {
15        int node=searchQueue.front();
16        searchQueue.pop();
17        string nodeString=outputQueue.front();
18        outputQueue.pop();
19        for (int k=0; k<26; k++) {
20                if (cities[node][k]==true && nodeVisited[k]==false) {
21                        searchQueue.push(k);
22                        nodeVisited[k]=true;
23                        char addToOutput=k+'A';
24                        string s=nodeString;
25                        s += addToOutput;
26                        outputQueue.push(s);
27                        if (k==bNumber) {
28                                outputString=s;
29                        }
30                }
31        }
32 }
33 cout << outputString << "\n";
```

4.5   Belman-Ford

```
1  procedure BellmanFord(list vertices, list edges, vertex source)
2      // This implementation takes in a graph, represented as lists of vertices
3      // and edges, and modifies the vertices so that their distance and
4      // predecessor attributes store the shortest paths.
5
6      // Step 1: initialize graph
7      for each vertex v in vertices:
8          if v is source tn v.distance := 0
9          else v.distance := infinity
10         v.predecessor := null
11
12     // Step 2: relax edges repeatedly
13     for i from 1 to size(vertices)-1:
14         for each edge uv in edges: // uv is the edge from u to v
15             u := uv.source
16             v := uv.destination
17             if u.distance + uv.weight < v.distance:
18                 v.distance := u.distance + uv.weight
19                 v.predecessor := u
20
21     // Step 3: check for negative-weight cycles
22     for each edge uv in edges:
23         u := uv.source
24         v := uv.destination
25         if u.distance + uv.weight < v.distance:
26             error "Graph contains a negative-weight cycle"
```

4.6   FordFulkerson

```java
1  import java.util.HashMap;
2  import java.util.LinkedList;
3  import java.util.ArrayList;
4
5  public class MaximumFlow {
6    public static void main(String[] args) {
7      int source = 1;
8      int sink = 4;
9      DirectedGraph g = new DirectedGraph();
10     g.addEdge(1, 2, 4);
11     g.addEdge(1, 3, 2);
12     g.addEdge(2, 4, 1);
13     g.addEdge(2, 3, 3);
14     g.addEdge(3, 4, 6);
15     HashMap<Edge, Integer> flow = getMaxFlow(g, source, sink);
16     System.out.println(getFlowSize(flow, g, source));
17   }
18
19   static HashMap<Edge, Integer> getMaxFlow(DirectedGraph g, Object source,
20       Object sink) {
21     LinkedList<Edge> path;
22     HashMap<Edge, Integer> flow = new HashMap<Edge, Integer>();
23     for (Edge e : g.getEdges()) {
24       flow.put(e, 0);
25     }
26
27     while ((path = bfs(g, source, sink, flow)) != null) {
28       int minCapacity = Integer.MAX_VALUE;
29       Object lastNode = source;
30       for (Edge edge : path) {
31         int c;
32         if (edge.getStart().equals(lastNode)) {
33           c = edge.getCapacity() - flow.get(edge);
34           lastNode = edge.getTarget();
35         } else {
36           c = flow.get(edge);
37           lastNode = edge.getStart();
38         }
39         if (c < minCapacity) {
```

```
40          minCapacity = c;
41        }
42      }
43
44      lastNode = source;
45      for (Edge edge : path) {
46        if (edge.getStart().equals(lastNode)) {
47          flow.put(edge, flow.get(edge) + minCapacity);
48          lastNode = edge.getTarget();
49        } else {
50          flow.put(edge, flow.get(edge) - minCapacity);
51          lastNode = edge.getStart();
52        }
53      }
54    }
55    return flow;
56  }
57
58  static int getFlowSize(HashMap<Edge, Integer> flow, DirectedGraph g,
59      Object source) {
60    int maximumFlow = 0;
61    Node sourceNode = g.getNode(source);
62    for (int i = 0; i < sourceNode.getOutLeadingOrder(); i++) {
63      maximumFlow += flow.get(sourceNode.getEdge(i));
64    }
65    return maximumFlow;
66  }
67
68  static LinkedList<Edge> bfs(DirectedGraph g, Object start, Object target,
69      HashMap<Edge, Integer> flow) {
70    HashMap<Object, Edge> parent = new HashMap<Object, Edge>();
71    LinkedList<Object> fringe = new LinkedList<Object>();
72    parent.put(start, null);
73    fringe.add(start);
74    all: while (!fringe.isEmpty()) {
75      LinkedList<Object> newFringe = new LinkedList<Object>();
76      for (Object nodeID : fringe) {
77        Node node = g.getNode(nodeID);
78        for (int i = 0; i < node.getOutLeadingOrder(); i++) {
79          Edge e = node.getEdge(i);
80          if (e.getStart().equals(nodeID)
81              && !parent.containsKey(e.getTarget())
82              && flow.get(e) < e.getCapacity()) {
83            parent.put(e.getTarget(), e);
84            if (e.getTarget().equals(target)) {
85              break all;
86            }
87            newFringe.add(e.getTarget());
88          } else if (e.getTarget().equals(nodeID)
89              && !parent.containsKey(e.getStart())
90              && flow.get(e) > 0) {
91            parent.put(e.getStart(), e);
92            if (e.getStart().equals(target)) {
93              break all;
94            }
95            newFringe.add(e.getStart());
96          }
97        }
98      }
99      fringe = newFringe;
100   }
101
102   if (fringe.isEmpty()) {
103     return null;
104   }
105   Object node = target;
106   LinkedList<Edge> path = new LinkedList<Edge>();
107   while (!node.equals(start)) {
```

```
108       Edge e = parent.get(node);
109       path.addFirst(e);
110       if (e.getStart().equals(node)) {
111         node = e.getTarget();
112       } else {
113         node = e.getStart();
114       }
115     }
116
117     return path;
118   }
119
120   public static class DirectedGraph {
121     private HashMap<Object, Node> nodes = new HashMap<Object, Node>();
122     private LinkedList<Edge> edges = new LinkedList<Edge>();
123
124     void addEdge(Object startNodeID, Object endNodeID, int capacity) {
125       Node startNode;
126       Node endNode;
127       if (!this.nodes.containsKey(startNodeID)) {
128         startNode = new Node();
129         this.nodes.put(startNodeID, startNode);
130       } else {
131         startNode = this.nodes.get(startNodeID);
132       }
133       if (!this.nodes.containsKey(endNodeID)) {
134         endNode = new Node();
135         this.nodes.put(endNodeID, endNode);
136       } else {
137         endNode = this.nodes.get(endNodeID);
138       }
139       Edge edge = new Edge(startNodeID, endNodeID, capacity);
140       startNode.addEdge(edge);
141       endNode.addEdge(edge);
142       this.edges.add(edge);
143     }
144
145     Node getNode(Object nodeID) {
146       return this.nodes.get(nodeID);
147     }
148
149     LinkedList<Edge> getEdges() {
150       return this.edges;
151     }
152   }
153
154   public static class Edge {
155
156     private final Object target;
157     private final Object start;
158     private final int capacity;
159
160     Edge(Object start, Object target, int capacity) {
161       this.capacity = capacity;
162       this.target = target;
163       this.start = start;
164     }
165
166     Object getTarget() {
167       return target;
168     }
169
170     Object getStart() {
171       return start;
172     }
173
174     int getCapacity() {
175       return capacity;
```

```
176      }
177
178      @Override
179      public String toString() {
180        return this.start + "->" + this.target + "(" + this.capacity + ")";
181      }
182    }
183
184    public class Node {
185
186      private ArrayList<Edge> edges = new ArrayList<Edge>();
187
188      void addEdge(Edge edge) {
189        this.edges.add(edge);
190      }
191
192      Edge getEdge(int number) {
193        if (this.edges.size() <= number) {
194          return null;
195        } else {
196          return this.edges.get(number);
197        }
198      }
199
200      int getOutLeadingOrder() {
201        return this.edges.size();
202      }
203    }
204 }
```

5   Geometrische Algorithmen

5.1   Graham Scan (Convex Hull)

```
1  static List<P> graham(List<P> l) {
2    if (l.size() < 3)
3      return l;
4    P temp = l.get(0);
5    for (P p : l)
6      if (temp.y > p.y || temp.y == p.y && temp.x > p.x)
7        temp = p;
8    final P start = temp; // min y (then leftmost)
9
10   Collections.sort(l, new Comparator<P>() {
11     public int compare(P o1, P o2) {
12       if (new Double(Math.atan2(o1.y - start.y, o1.x - start.x)) // same angle
13           .compareTo(Math.atan2(o2.y - start.y, o2.x - start.x)) == 0)
14         return new Double(Math.sqrt((o1.x - start.x)
15             * (o1.x - start.x) + (o1.y - start.y)
16             * (o1.y - start.y))).compareTo((o2.x - start.x)
17             * (o2.x - start.x) + (o2.y - start.y)
18             * (o2.y - start.y)); // use distance
19       return new Double(Math.atan2(o1.y - start.y, o1.x - start.x))
20           .compareTo(Math.atan2(o2.y - start.y, o2.x - start.x));
21     }
22   });
23   Stack<P> s = new Stack<P>();
24   s.add(start);
25   s.add(l.get(1));
26   for (int i = 2; i < l.size(); i++) {
27     while (s.size() >= 2
28         && ccw(s.get(s.size() - 2), s.get(s.size() - 1), l.get(i)) <= 0)
29       s.pop();
30     s.push(l.get(i));
31   }
32   return s;
33 }
```

```
34
35   // turn is counter-clockwise if > 0; collinear if = 0; clockwise else
36   static double ccw(P p1, P p2, P p3) {
37     return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
38   }
39
40   public static class P {
41     double x, y;
42
43     P(double x, double y) {
44       this.x = x;
45       this.y = y;
46     }
47     // polar coordinates (not used)
48     // double r() { return Math.sqrt(x * x + y * y); }
49     // double d() { return Math.atan2(y, x); }
50   }
```

5.2 Punkt in Polygon

```
1    /**
2     * -1: A->R schneidet BC (ausser unterer Endpunkt)
3     *  0: A auf BC
4     * +1: sonst
5     */
6    public static int KreuzProdTest(double ax, double ay, double bx, double by,
7        double cx, double cy) {
8      if (ay == by && by == cy) {
9        if ((bx <= ax && ax <= cx) || (cx <= ax && ax <= bx))
10         return 0;
11       else
12         return +1;
13     }
14     if(by>cy){double tmpx=bx;double tmpy=by; bx=cx;by=cy;cx=tmpx;cy=tmpy;}
15     if(ay==by && ax==bx) return 0;
16     if(ay<=by || ay>cy) return +1;
17     double delta = (bx-ax)*(cy-ay)-(by-ay)*(cx-ax);
18     if(delta>0)return -1; else if(delta<0)return +1;else return 0;
19   }
20   /**
21    * Input: P[i] (x[i],y[i]); P[0]:=P[n]
22    * -1: Q ausserhalb Polygon
23    *  0: Q auf Polygon
24    * +1: Q innerhalt des Polygons
25    */
26   public static int PunktInPoly(double[] x,double[] y, double qx,double qy){
27     int n = x.length - 1;
28     int t = -1;
29     for (int i = 0; i <= n - 1; i++) {
30       t = t * KreuzProdTest(qx, qy, x[i], y[i], x[i + 1], y[i + 1]);
31     }
32     return t;
33   }
```

6 Verschiedenes

6.1 Potenzmenge

```
1  static <T> Iterator<List<T>> powerSet(final List<T> l) {
2    return new Iterator<List<T>>() {
3      int i; // careful: i becomes 2^l.size()
4      public boolean hasNext() {
5        return i < (1 << l.size());
6      }
7      public List<T> next() {
8        Vector<T> temp = new Vector<T>();
9        for (int j = 0; j < l.size(); j++)
```

```
10        if (((i >>> j) & 1) == 1)
11            temp.add(l.get(j));
12        i++;
13        return temp;
14      }
15      public void remove() {}
16      };
17    }
```

6.2   LongestCommonSubsequence

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <sstream>
5  #include <algorithm>
6  #include <iterator>
7  using namespace std;
8
9  #define MAX(a,b) (a > b) ? a : b
10
11 string X,Y;
12 vector< vector<int> > c(101, vector<int>(101,0));
13 int m,n,ctr;
14
15 int LCS()
16 {
17     m = X.length(),n=Y.length();
18
19     c.resize(m+1);
20   for(int i = 0; i<n+1; i++) {
21     c[i].resize(n+1);
22     c[i][0] = 0;
23   }
24
25     int i,j;
26
27     for (i=0;i<=m;i++)
28         for (j=0;j<=n;j++)
29             c[i][j]=0;
30
31     for (i=1;i<=m;i++)
32         for (j=1;j<=n;j++)
33         {
34             if (X[i-1]==Y[j-1])
35                 c[i][j]=c[i-1][j-1]+1;
36             else
37                 c[i][j]=max(c[i][j-1],c[i-1][j]);
38         }
39     return c[m][n];
40 }
41 /** Print a songle LCS */
42 void printLCS(int i,int j)
43 {
44     if (i==0 || j==0)
45         return;
46     if (X[i-1]==Y[j-1])
47     {
48         printLCS(i-1,j-1);
49         cout<<X[i-1];
50     }
51     else if (c[i][j]==c[i-1][j])
52         printLCS(i-1,j);
53     else
54         printLCS(i,j-1);
55 }
56
57 int main()
```

```
58  {
59      while(cin>>X>>Y)
60      {
61   cout << "Length:␣" << LCS() << endl;
62         printLCS(m,n);
63         cout<<endl ;
64      }
65  }
```

6.3 LongestCommonSubstring

```
1   private static List<String> longestCommonSubstring(String S1, String S2)
2   {
3     List<String> ret = new ArrayList<String>();
4     List<Integer> idx  =new ArrayList<Integer>();
5       int Start = 0;
6       int Max = 0;
7       for (int i = 0; i < S1.length(); i++)
8       {
9           for (int j = 0; j < S2.length(); j++)
10          {
11              int x = 0;
12              while (S1.charAt(i + x) == S2.charAt(j + x))
13              {
14                  x++;
15                  if (((i + x) >= S1.length()) || ((j + x) >= S2.length())) break;
16              }
17              if (x > Max)
18              {
19                  Max = x;
20                Start = i;
21                idx.clear();
22                idx.add(Start);
23              } else if(x==Max){
24                Start = i;
25                idx.add(Start);
26              }
27          }
28      }
29      HashSet<String> set = new HashSet<String>(idx.size(),1f);
30      for(Integer start : idx){
31        String substr = S1.substring(start,start+Max);
32        if(!set.contains(substr)){
33          ret.add(substr);
34          set.add(substr);
35        }
36      }
37      Collections.sort(ret);
38      //return S1.substring(Start, (Start + Max));
39      return ret;
40    }
```

6.4 LongestIncreasingSubsequence

```
1   #include <vector>
2   using namespace std;
3
4   /** finde LIS in O(n log k)
5    *a: Sequenz (in)
6    *b: LIS (out)
7    */
8   void find_lis(vector<int> &a, vector<int> &b)
9   {
10    vector<int> p(a.size());
11    int u, v;
12    if (a.empty()) return;
13    b.push_back(0);
14
```

```
15    for (size_t i = 1; i < a.size(); i++)
16        {
17        // ist naechstes Element a[i] groesser als letztes der aktuelle LIS
18    // a[b.back()], fuege es (Index) an "b" an.
19    if (a[b.back()] < a[i]) {
20      p[i] = b.back();
21      b.push_back(i);
22      continue;
23    }
24
25        // finde kleinstes El. in LIS (index in b) welches gerade groesser als a[i] ist
26        // binaere suche |b|<=k  =>  O(log k)
27    for (u = 0, v = b.size()-1; u < v;)
28        {
29      int c = (u + v) / 2;
30      if (a[b[c]] < a[i]) u=c+1; else v=c;
31    }
32
33        // aktualisiere b falls neuer Wert kleiner als vorheriger kleinerer Wert
34    if (a[i] < a[b[u]])
35                {
36      if (u > 0) p[i] = b[u-1];
37      b[u] = i;
38    }
39  }
40
41    for (u = b.size(), v = b.back(); u--; v = p[v]) b[u] = v;
42 }
43
44 #include <cstdio>
45 int main()
46 {
47    int a[] = { 1, 9, 3, 8, 11, 4, 5, 6, 4, 19, 7, 1, 7 };
48    vector<int> seq(a, a+sizeof(a)/sizeof(a[0])); // seq : Eingabesequent
49    vector<int> lis;                              // lis : Index Vektor fuer LIS
50      find_lis(seq, lis);
51      //Sequenz ausgeben:
52    for (size_t i = 0; i < lis.size(); i++)
53      printf("%d␣", seq[lis[i]]);
54          printf("\n");
55
56    return 0;
57 }
```

6.5 Permutation & Sequenzen

```
1 import java.util.Scanner;
2
3 public class PermsAndSequ {
4    public static void main(String[] args) {
5      Scanner sc = new Scanner(System.in);
6      int n;
7      while ((n = sc.nextInt()) != 0) {
8        int k = sc.nextInt();
9        Sequences(n, k);
10       Permutations(n);
11     }
12
13   }
14
15   public static void Sequences(int n, int k) {
16     int[] x = new int[k];
17     for (int i = 0; i < k; i++)
18       x[i] = 1;
19     Print(x);
20     while (true) {
21       boolean lastX = true;
22       for (int i = 0; i < k; i++)
```

```java
23        if (x[i] != n) {
24          lastX = false;
25          break;
26        }
27      if (lastX)
28        break;
29      int p = k - 1;
30      while (!(x[p] < n))
31        p--;
32      x[p] = x[p] + 1;
33      for (int i = p + 1; i < k; i++)
34        x[i] = 1;
35      Print(x);
36    }
37  }
38
39  public static void Permutations(int n) {
40    int[] x = new int[n];
41    for (int i = 0; i < n; i++)
42      x[i] = i + 1;
43    Print(x);
44    while (true) {
45      boolean lastX = true;
46      for (int i = 0; i < n - 1; i++)
47        if (x[i] < x[i + 1]) {
48          lastX = false;
49          break;
50        }
51      if (lastX)
52        break;
53      int k = n - 1 - 1;
54
55      while (x[k] > x[k + 1])
56        k--;
57      int t = k + 1;
58
59      while (t < (n - 1) && x[t + 1] > x[k])
60        t++;
61
62      int tmp = x[k];
63      x[k] = x[t];
64      x[t] = tmp;
65      // reverse x[k+1] ... x[n-1]
66
67      for (int i = 0; i <= ((n - 1) - (k + 1)) / 2; i++) {
68        tmp = x[k + 1 + i];
69        x[k + 1 + i] = x[n - 1 - i];
70        x[n - 1 - i] = tmp;
71
72      }
73      Print(x);
74    }
75  }
76
77  public static void Print(int[] x) {
78    for (int i = 0; i < x.length; i++)
79      System.out.print(x[i] + "␣");
80    System.out.println("");
81  }
82
83 }
```

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

```cpp
#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());
}
```