

# Team Contest Reference

Universität zu Lübeck

19. November 2012

## 1 Mathematische Algorithmen

### 1.1 Primzahlen

#### 1.1.1 Sieb des Eratosthenes

```
1 static boolean[] sieve(int until) {
2     boolean[] a = new boolean[until + 1];
3     Arrays.fill(a, true);
4     for (int i = 2; i < Math.sqrt(a.length); i++) {
5         if (a[i]) {
6             for (int j = i * i; j < a.length; j += i) a[j] = false;
7         }
8     }
9     return a; // a[i] == true, iff. i is prime. a[0] is ignored
10 }
```

#### 1.1.2 Primzahlentest

```
1 static boolean isPrim(int p) {
2     if (p < 2 || p > 2 && p % 2 == 0) return false;
3     for (int i = 3; i <= Math.sqrt(p); i += 2)
4         if (p % i == 0) return false;
5     return true;
6 }
```

### 1.2 Binomial Koeffizient

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

```
8         ret *= n - k + i;
9         ret /= i;
10        mem[n][i] = ret;
11    }
12    return ret;
13 }
```

## 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 Kreuzprodukt

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

### 2.4 Orthogonale Projektion

$r_0$  : Ortsvektor;  $u$  : Richtungsvektor;  $n$  : Normalenvektor

$$P_g(\vec{x}) = \vec{r}_0 + \frac{(\vec{x}-\vec{r}_0) \cdot \vec{u}}{\vec{u} \cdot \vec{u}} \vec{u}$$
$$P_g(\vec{x}) = \vec{x} - \frac{(\vec{x}-\vec{r}_0) \cdot \vec{n}}{\vec{n} \cdot \vec{n}} \vec{n} \text{ (nur 2D bzw. 3D auf Ebene)}$$

2.5 Geradenschnittpunkt

$$g_1 : ax + by = c; \; g_2 : px + qx = r; \Rightarrow \vec{p} = \frac{1}{aq-bp} \begin{pmatrix} x = cq - br \\ y = ar - cp \end{pmatrix}$$
$$g_1 : \vec{p} = \begin{pmatrix} r_x \\ r_y \end{pmatrix} + s \begin{pmatrix} s_x \\ s_y \end{pmatrix} \; g_2 : \vec{p} = \begin{pmatrix} q_x \\ q_y \end{pmatrix} + t \begin{pmatrix} t_x \\ t_y \end{pmatrix} \; w_x = (r_x - q_x), w_y = (r_y - q_y)$$
$$\Rightarrow D = (s_x t_y - t_x s_y) \; D_s = (t_x w_y - t_y w_x) \; D_t = (s_y w_x - s_x w_y) \; s = D_s / D, t = D_t / D$$

2.6 Dreicksfläche

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

2.7 Kombinatorik

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

3 Datenstukturen

3.1 Fenwick Tree (Binary Indexed Tree)

```
1 class FenwickTree {
2     private int[] values;
3     private int n;
4     public FenwickTree(int n) {
5         this.n = n;
6         values = new int[n];
7     }
8     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 }

```

## 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 innerhalb 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 single 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++)
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         Print(x);
73     }
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 }
```