Team Contest Reference

Universität zu Lübeck

17. November 2012

1 Mathematische Algorithmen

1.1 Primzahlen

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
}
```

2 Mathematisch Formeln und Gesetze

2.1 Catalan

$C_n = \frac{1}{n+1}\binom{2n}{n} = \prod_{k=2}^{n}(n+k)/k$

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
$P_g(\vec{x}) = \vec{r_0} + \frac{(\vec{x}-\vec{r_0})\cdot\vec{u}}{\vec{u}\cdot\vec{u}}\,\vec{u}$
$P_g(\vec{x}) = \vec{x} - \frac{(\vec{x}-\vec{r_0})\cdot\vec{n}}{\vec{n}\cdot\vec{n}}\,\vec{n}$(nur 2D bzw. 3D auf Ebene)

2.4.1 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;
}
```

2.5 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;
}
```

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;
}
```

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); }
}
```

6 Verschiedenes

6.1 Potenzmenge

```
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

```
#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;

```

```
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

6.4 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        }
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  }
```