

Team Contest Reference

getRandomNumber(){return 4;}

Universität zu Lübeck

22. November 2012

Inhaltsverzeichnis

1	Mathematische Algorithmen	1
1.1	Primzahlen	1
1.1.1	Sieb des Eratosthenes	1
1.1.2	Primzahlentest	1
1.2	Binomial Koeffizient	2
1.3	Modulare Arithmetik	2
1.3.1	Erweiterter Euklidischer Algorithmus	2
2	Datenstrukturen	2
2.1	Fenwick Tree (Binary Indexed Tree)	2
3	Graphenalgorithmen	3
3.1	Topologische Sortierung	3
3.2	Prim (Minimum Spanning Tree)	3
3.3	Maximaler Fluss (Ford-Fulkerson)	4
4	Geometrische Algorithmen	6
4.1	Graham Scan (Convex Hull)	6
4.2	Punkt in Polygon	6
5	Verschiedenes	7
5.1	Potenzmenge	7
5.2	Longest Common Subsequence	7

1 Mathematische Algorithmen

1.1 Primzahlen

Für Primzahlen gilt immer (aber nicht nur für Primzahlen)

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

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 }

```

1.3 Modulare Arithmetik

Bedeutung der größten gemeinsamen Teiler:

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

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

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

1.3.1 Erweiterter Euklidischer Algorithmus

```

1 static int[] eea(int a, int b) {
2     int[] dst = new int[3];
3     if (b == 0) {
4         dst[0] = a;
5         dst[1] = 1;
6         return dst; // a, 1, 0
7     }
8     dst = eea(b, a % b);
9     int tmp = dst[2];
10    dst[2] = dst[1] - ((a / b) * dst[2]);
11    dst[1] = tmp;
12    return dst;
13 }

```

2 Datenstrukturen

2.1 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 }

```

3 Graphenalgorithmen

3.1 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 }

```

3.2 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 }

```

3.3 Maximaler Fluss (Ford-Fulkerson)

```

1  #include <stdio.h>
2  #include <limits.h>
3
4  #define n_MAX 36
5  #define m_MAX 30
6  #define SIZE (m+6+2)
7  #define SIZE_MAX 38
8  #define QUELLE (m+6)
9  #define SENKE (m+7)
10 #define NONE -1
11 #define INF INT_MAX/2
12
13 int n, m;
14 int capacity[SIZE_MAX][SIZE_MAX];
15 int flow[SIZE_MAX][SIZE_MAX];
16 int queue[SIZE_MAX], *head, *tail;
17 int state[SIZE_MAX];
18 int pred[SIZE_MAX];
19
20 enum { XS, S, M, L, XL, XXL };
21 enum { UNVISITED, WAITING, PROCESSED };
22
23 int strToOffset( char *str);
24 int maxFlow( int quelle, int senke);
25
26 int main(){
27
28     int numOfProps;
29     scanf("%d\n", &numOfProps);
30
31     while( numOfProps--){
32         scanf("%d_%d\n", &n, &m);
33
34         int i, j;
35
36         /* Matrix initialisieren */
37         for( i=0; i< SIZE; i++){
38             for( j=0; j< SIZE; j++){
39                 capacity[i][j] = flow[i][j] = 0;
40
41                 if( i == QUELLE && j < m){
42                     capacity[i][j] = 1;
43                     continue;
44                 }
45
46                 if( j == SENKE && i >= m && i < QUELLE){
47                     capacity[i][j] = n/6;
48                     continue;
49                 }
50             }
51         }
52
53         char str[4];
54
55         /* Matrix einlesen */
56         for( i=0; i< m; i++){
57             scanf("%s", str);
58             capacity[i][m+strToOffset(str)] = 1;
59             scanf("%s", str);
60             capacity[i][m+strToOffset(str)] = 1;

```

```
61     }
62
63
64     int foo = maxFlow( QUELLE, SENKE);
65     printf("%s\n", foo >= m ? "YES" : "NO");
66
67 }
68
69 return 0;
70 }
71
72 int strToOffset( char *str){
73     /*snip*/
74 }
75
76 void enqueue( int x){
77     *tail++ = x;
78     state[x] = WAITING;
79 }
80
81 int dequeue(){
82     int x = *head++;
83     state[x] = PROCESSED;
84     return x;
85 }
86
87 int bfs( int start, int target){
88     int u, v;
89     for( u=0; u< SIZE; u++){
90         state[u] = UNVISITED;
91     }
92     head = tail = queue;
93     pred[start] = NONE;
94
95     enqueue(start);
96
97     while( head < tail){
98         u = dequeue();
99
100         for( v= 0; v< SIZE; v++){
101             if( state[v] == UNVISITED &&
102                capacity[u][v] - flow[u][v] > 0){
103
104                 enqueue(v);
105                 pred[v] = u;
106             }
107         }
108     }
109
110     return state[target] == PROCESSED;
111 }
112
113 int maxFlow( int quelle, int senke){
114     int max_flow = 0;
115
116     int u;
117
118     while( bfs( quelle, senke)){
119         int increment = INF, temp;
120
121         for( u= senke; pred[u] != NONE; u = pred[u]){
122             temp = capacity[pred[u]][u] - flow[pred[u]][u];
123             if( temp < increment){
124                 increment = temp;
125             }
126         }
127
128         for( u= senke; pred[u] != NONE; u = pred[u]){
129             flow[pred[u]][u] += increment;
130             flow[u][pred[u]] -= increment;
131         }
132
133         max_flow += increment;
134     }
135
136     return max_flow;
```

137 }

4 Geometrische Algorithmen

4.1 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 }

```

4.2 Punkt in Polygon

```

1 /**
2  * -1: A liegt links von 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)) return 0;
10        else return +1;
11    }
12    if (by > cy) {
13        double tmpx = bx, tmpy = by;
14        bx = cx;
15        by = cy;
16        cx = tmpx;

```

```

17     cy = tmpy;
18 }
19 if (ay == by && ax == bx) return 0;
20 if (ay <= by || ay > cy) return +1;
21 double delta = (bx - ax) * (cy - ay) - (by - ay) * (cx - ax);
22 if (delta > 0) return -1;
23 else if (delta < 0) return +1;
24 else return 0;
25 }
26
27 /**
28  * Input: P[i] (x[i],y[i]); P[0]:=P[n]
29  * -1: Q ausserhalb Polygon
30  * 0: Q auf Polygon
31  * +1: Q innerhalb des Polygons
32  */
33 public static int PunktInPoly(double[] x, double[] y, double qx, double qy) {
34     int t = -1;
35     for (int i = 0; i < x.length - 1; i++)
36         t = t * KreuzProdTest(qx, qy, x[i], y[i], x[i + 1], y[i + 1]);
37     return t;
38 }

```

5 Verschiedenes

5.1 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 }

```

5.2 Longest Common Subsequence

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5
6 int lcs( char *a, char *b){
7     int len = strlen( a);
8     int lenb =strlen(b);
9
10    int *zeile = malloc( (len+1) * sizeof(int)), *temp,
11        *neue = malloc( (len+1) * sizeof(int)), i, j;
12
13    for(i=0; i<len+1; i++){
14        zeile[i] = neue[i] = 0;
15    }
16
17    for(j=0; j<lenb; j++){
18        for(i=0; i<len; i++){
19            if( a[i] == b[j]){
20                neue[i+1] = zeile[i] + 1;
21            } else {
22                neue[i+1] = neue[i] > zeile[i+1] ? neue[i] : zeile[i+1];
23            }
24        }
25        temp = zeile;
26        zeile = neue;
27        neue = temp;
28    }

```

```
29
30     int res = zeile[len];
31     free( zeile);
32     free( neue);
33     return res;
34 }
```