Team Contest Reference

Universität zu Lübeck

20. 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

```
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 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):

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

1.3.1 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;
}
```

2 Datenstukturen

2.1 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; }
    }
  }
}
```

3 Graphenalgorithmen

3.1 Topologische Sortierung

```java
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;
}
```

3.2 Prim (Minimum Spanning Tree)

```java
#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*/
```

```
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
```

```
                enqueue(v);
                pred[v] = u;
            }
        }
    }

    return state[target] == PROCESSED;
}

int maxFlow( int quelle, int senke){
    int max_flow = 0;

    int u;

    while( bfs( quelle, senke)){
        int increment = INF, temp;

        for( u= senke; pred[u] != NONE; u = pred[u]){
            temp = capacity[pred[u]][u] - flow[pred[u]][u];
            if( temp < increment){
                increment = temp;
            }
        }

        for( u= senke; pred[u] != NONE; u = pred[u]){
            flow[pred[u]][u] += increment;
            flow[u][pred[u]] -= increment;
        }

        max_flow += increment;
    }

    return max_flow;
}
```

4 Geometrische Algorithmen

4.1 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 Verschiedenes

5.1 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() {}
    };
  }
```

5.2 Longest Common Subsequence

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


int lcs( char *a, char *b){
    int len = strlen( a);
    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  }
```