



Team Contest Reference

Team: Romath

*Roland Haase
Thore Tiemann
Marcel Wienöbst*

Contents

1 DP	2
1.1 LongestIncreasingSubsequence	2
1.2 LongestIncreasingSubsequence	3
2 DataStructures	3
2.1 Fenwick-Tree	3
2.2 Range Maximum Query	3
2.3 Trie	4
2.4 Union-Find	4
2.5 Suffix array	5
3 Graph	5
3.1 2SAT	5
3.2 Breadth First Search	5
3.3 BellmanFord	6
3.4 Bipartite Graph Check	6
3.5 Maximum Bipartite Matching	6
3.6 Bitonic TSP	7
3.7 Single-source shortest paths in dag	7
3.8 Dijkstra	7
3.9 EdmondsKarp	7
3.10 Reference for Edge classes	8
3.11 FloydWarshall	8
3.12 Held Karp	8
3.13 Iterative DFS	9
3.14 Johnsons Algorithm	9
3.15 Kruskal	9
3.16 Min Cut	9
3.17 Prim	10
3.18 Recursive Depth First Search	10
3.19 Strongly Connected Components	11
3.20 Suurballe	11
3.21 Kahns Algorithm for TS	11
3.22 Topological Sort	12
3.23 Tuple	12
3.24 Reference for Vertex classes	12
4 Math	13
4.1 Binomial Coefficient	13
4.2 Binomial Matrix	13
4.3 Divisability	13
4.4 Graham Scan	13
4.5 Iterative EEA	14
4.6 Polynomial Interpolation	14
4.7 Root of permutation	15
4.8 Sieve of Eratosthenes	16

4.9	Greatest Common Divisor	16
4.10	Least Common Multiple	16
4.11	GEV	16
4.12	Fourier transform	18
4.13	Matrix exponentiation	18
4.14	phi function calculator	19
4.15	prints farey seq	19
5	Misc	19
5.1	Binary Search	19
5.2	Next number with n bits set	19
5.3	Next Permutation	19
5.4	Greedy-Scheduling	20
5.5	comparator in C++	20
5.6	Mo's algorithm	20
6	String	21
6.1	Knuth-Morris-Pratt	21
6.2	Levenshtein Distance	21
6.3	Longest Common Subsequence	22
6.4	Longest common substring	22
7	Math	22
7.1	Tree	22
7.2	Divisability Explanation	22
7.3	Combinatorics	22
7.4	Polynomial Interpolation	23
7.4.1	Theory	23
7.5	Fibonacci Sequence	23
7.5.1	Binet's formula	23
7.5.2	Generalization	23
7.5.3	Pisano Period	23
7.6	Reihen	23
7.7	Binomialkoeffizienten	23
7.8	Catalanzahlen	23
7.9	Geometrie	23
7.10	Zahlentheorie	23
7.11	Faltung	23
8	Java Knowhow	23
8.1	System.out.printf() und String.format()	23
8.2	Modulo: Avoiding negative Integers	23
8.3	Speed up IO	24

n	Runtime $100 \cdot 10^6$ in 3s
[10, 11]	$\mathcal{O}(n!)$
< 22	$\mathcal{O}(n2^n)$
≤ 100	$\mathcal{O}(n^4)$
≤ 400	$\mathcal{O}(n^3)$
≤ 2.000	$\mathcal{O}(n^2 \log n)$
≤ 10.000	$\mathcal{O}(n^2)$
$\leq 1.000.000$	$\mathcal{O}(n \log n)$
$\leq 100.000.000$	$\mathcal{O}(n)$

byte (8 Bit, signed): -128 ... 127

short (16 Bit, signed): -32.768 ... 32.767

integer (32 Bit, signed): -2.147.483.648 ... 2.147.483.647

long (64 Bit, signed): $-2^{63} \dots 2^{63} - 1$

MD5: cat <string> | tr -d [:space:] | md5sum

1 DP

1.1 LongestIncreasingSubsequence

Computes the length of the longest increasing subsequence and is easy to be adapted.

Input: array *arr* containing a sequence of length *N*

Output: length of the longest increasing subsequence in *arr*

```

1 // This has not been tested yet
2 // (adapted from tested C++ Murcia Code)
3 public static int LISeasy(int[] arr, int N) {
4     int[] m = new int[N];
5     for (int i = N - 1; i >= 0; i--) {
6         m[i] = 1; //init table
7         for (int j = i + 1; j < N; j++) {
8             // if arr[i] increases the length
9             // of subsequence from array[j]
10            if (arr[j] > arr[i])
11                if (m[i] < m[j] + 1)
12                    // store length of new subseq
13                    m[i] = m[j] + 1;
14        }
15    }
16    // find max in array
17    int longest = 0;
18    for (int i = 0; i < N; i++) {
19        if (m[i] > longest)
20            longest = m[i];
21    }
22    return longest;
23 }
```

MD5: 7561f576d50b1dc6262568c0fc6c42dd | $\mathcal{O}(n^2)$

1.2 LongestIncreasingSubsequence

Computes the longest increasing subsequence using binary search.

Input: array *arr* containing a sequence and empty array *p* of length *arr.length* for storing indices of the LIS (might be usefull to have)

Output: array *s* containing the longest increasing subsequence

```

1 public static int[] LISfast(int[] arr, int[] p) {
2     // p[k] stores index of the predecessor of arr[k]
3     // in the LIS ending at arr[k]
4     // m[j] stores index k of smallest value arr[k]
5     // so there is a LIS of length j ending at arr[k]
6     int[] m = new int[arr.length+1];
7     int l = 0;
8     for (int i = 0; i < arr.length; i++) {
9         // bin search for the largest positive j <= l
10        // with arr[m[j]] < arr[i]
11        int lo = 1;
12        int hi = l;
13        while (lo <= hi) {
14            int mid = (int) (((lo + hi) / 2.0) + 0.6);
15            if (arr[m[mid]] <= arr[i])
16                lo = mid+1;
17            else
18                hi = mid-1;
19        }
20        // lo is 1 greater than length of the
21        // longest prefix of arr[i]
22        int newL = lo;
23        p[i] = m[newL-1];
24        m[newL] = i;
25    }
```

```

25    // if LIS found is longer than the ones
26    // found before, then update l
27    if (newL > l)
28        l = newL;
29    }
30    // reconstruct the LIS
31    int[] s = new int[l];
32    int k = m[l];
33    for (int i = l-1; i >= 0; i--) {
34        s[i] = arr[k];
35        k = p[k];
36    }
37    return s;
38 }
```

MD5: 1d75905f78041d832632cb76af985b8e | $\mathcal{O}(n \log n)$

2 DataStructures

2.1 Fenwick-Tree

Can be used for computing prefix sums.

```

1 //note that 0 can not be used
2 int[] fwktree = new int[m + n + 1];
3 public static int read(int index, int[] fenwickTree) {
4     int sum = 0;
5     while (index > 0) {
6         sum += fenwickTree[index];
7         index -= (index & -index);
8     }
9     return sum;
10 }
11 public static int[] update(int index, int addValue,
12     int[] fenwickTree) {
13     while (index <= fenwickTree.length - 1) {
14         fenwickTree[index] += addValue;
15         index += (index & -index);
16     }
17     return fenwickTree;
18 }
```

MD5: 410185d657a3a5140bde465090ff6fb5 | $\mathcal{O}(\log n)$

2.2 Range Maximum Query

process processes an array *A* of length *N* in $\mathcal{O}(N \log N)$ such that *query* can compute the maximum value of *A* in interval $[i, j]$. Therefore $M[a, b]$ stores the maximum value of interval $[a, a + 2^b - 1]$.

Input: dynamic table *M*, array to search *A*, length *N* of *A*, start index *i* and end index *j*

Output: filled dynamic table *M* or the maximum value of *A* in interval $[i, j]$

```

1 public static void process(int[][] M, int[] A, int N)
2 {
3     for (int i = 0; i < N; i++)
4         M[i][0] = i;
5     // filling table M
6     // M[i][j] = max(M[i][j-1], M[i+(1<<(j-1))][j-1]),
7     // cause interval of length 2^j can be partitioned
8     // into two intervals of length 2^(j-1)
9     for (int j = 1; 1 << j <= N; j++) {
```

```

9   for(int i = 0; i + (1 << j) - 1 < N; i++) {
10       if(A[M[i][j-1]] >= A[M[i+(1 << (j-1))][j-1]])
11           M[i][j] = M[i][j-1];
12       else
13           M[i][j] = M[i + (1 << (j-1))][j-1];
14   }
15 }
16 }
17
18 public static int query(int[][] M, int[] A, int N,
19                         int i, int j) {
20     // k = ⌊ log2(j-i+1) ⌋
21     int k = (int) (Math.log(j - i + 1) / Math.log(2));
22     if(A[M[i][k]] >= A[M[j- (1 << k) + 1][k]])
23         return M[i][k];
24     else
25         return M[j - (1 << k) + 1][k];
26 }

```

MD5: db0999fa40037985ff27dd1a43c53b80 | $\mathcal{O}(N \log N, 1)$

2.3 Trie

```

1   public static boolean insert(TrieNode root, String
2       word){
3       char[] s = word.toCharArray();
4       TrieNode node = root;
5
6       for(int i = 0; i < s.length; ++i){
7           int index = charToIndex(s[i]);
8           if(node.children[index] == null){
9               node.children[index] = new TrieNode(node);
10          }
11          node = node.children[index];
12      }
13      node.isEnd = true;
14
15      return true;
16  }
17
18  public static boolean search(TrieNode root, String
19      word){
20      char[] s = word.toCharArray();
21      TrieNode node = root;
22
23      for(int i = 0; i < s.length; ++i){
24          int index = charToIndex(s[i]);
25          if(node.children[index] == null){
26              return false;
27          }
28          node = node.children[index];
29      }
30
31      return node.isEnd;
32  }
33
34  public static int charToIndex(char c){
35      return ((int) c - (int) a);
36  }
37
38  static class TrieNode{
39
40      boolean isEnd;
41      TrieNode[] children;
42
43      public TrieNode(){
44          isEnd = false;

```

```

43     children = new TrieNode[26];
44 }
45 }

```

MD5: 95ebde7b285a97b8834aedd9c2bf9ff2 | $\mathcal{O}(|w|)$

2.4 Union-Find

Union-Find is a data structure that keeps track of a set of elements partitioned into a number of disjoint subsets. *UnionFind* creates n disjoint sets each containing one element. *union* joins the sets x and y are contained in. *find* returns the representative of the set x is contained in.

Input: number of elements n , element x , element y

Output: the representative of element x or a boolean indicating whether sets got merged.

```

1   class UnionFind {
2       private int[] p = null;
3       private int[] r = null;
4       private int count = 0;
5
6       public int count() {
7           return count;
8       } // number of sets
9
10      public UnionFind(int n) {
11          count = n; // every node is its own set
12          r = new int[n]; // every node is its own tree with
13                          height 0
14          p = new int[n];
15          for (int i = 0; i < n; i++)
16              p[i] = -1; // no parent = -1
17      }
18
19      public int find(int x) {
20          int root = x;
21          while (p[root] >= 0) { // find root
22              root = p[root];
23          }
24          while (p[x] >= 0) { // path compression
25              int tmp = p[x];
26              p[x] = root;
27              x = tmp;
28          }
29          return root;
30      }
31
32      // return true, if sets merged and false, if already
33      // from same set
34      public boolean union(int x, int y) {
35          int px = find(x);
36          int py = find(y);
37          if (px == py)
38              return false; // same set -> reject edge
39          if (r[px] < r[py]) { // swap so that always h[px]
40                          >= h[py]
41          }
42          int tmp = px;
43          px = py;
44          py = tmp;
45          p[py] = px; // hang flatter tree as child of
46                      higher tree
47          r[px] = Math.max(r[px], r[py] + 1); // update (
48                      worst-case) height

```

```

44     count--;
45     return true;
46 }
47 }

```

MD5: 5c507168e1ffd9ead25babf7b3769cfd | $\mathcal{O}(\alpha(n))$

2.5 Suffix array

```

1  #include<vector>
2  #include<string>
3  #include<algorithm>
4
5  using namespace std;
6
7  vector<int> sa, pos, tmp, lcp;
8  string s;
9  int N, gap;
10
11 bool sufCmp(int i, int j) {
12     if(pos[i] != pos[j])
13         return pos[i] < pos[j];
14     i += gap;
15     j += gap;
16     return (i < N && j < N) ? pos[i] < pos[j] : i > j;
17 }
18
19 void buildSA()
20 {
21     N = s.size();
22     for(int i = 0; i < N; ++i) {
23         sa.push_back(i);
24         pos.push_back(s[i]);
25     }
26     tmp.resize(N);
27     for(gap = 1; gap * 2 < N; gap *= 2) {
28         sort(sa.begin(), sa.end(), sufCmp);
29         for(int i = 0; i < N - 1; ++i) {
30             tmp[i+1] = tmp[i] + sufCmp(sa[i], sa[i+1]);
31         }
32         for(int i = 0; i < N; ++i) {
33             pos[sa[i]] = tmp[i];
34         }
35         if(tmp[N-1] == N-1) break;
36     }
37 }
38
39 void buildLCP()
40 {
41     lcp.resize(N);
42     for(int i = 0, k = 0; i < N; ++i) {
43         if(pos[i] != N - 1) {
44             for(int j = sa[pos[i] + 1]; s[i + k] == s[j + k]; ++k);
45             lcp[pos[i]] = k;
46             if(k > 0) --k;
47         }
48     }
49 }
50
51 }
52
53 int main()
54 {
55     string r, t;
56     cin >> r >> t;
57     s = r + "$" + t;

```

```

58 buildSA();
59 buildLCP();
60 for(int i = 0; i < N; ++i) {
61     cout << sa[i] << " " << lcp[i] << endl;
62 }
63 int mx = 0, mxi = -1;
64 for(int i = 0; i+1 < s.size(); ++i) {
65     bool a_in_s = sa[i] < r.size(), b_in_s = sa[i+1] <
66         r.size();
67     if(a_in_s != b_in_s) {
68         int l = lcp[i];
69         if(l > mx) {
70             mx = l;
71             mxi = sa[i];
72         }
73     }
74 }
75 cout << mx << endl;
76 cout << s.substr(mxi, mx) << endl;
77 }

```

MD5: 96e0269748dc2834567a075768eb871a | $\mathcal{O}(?)$

3 Graph

3.1 2SAT

```

1 //We assume that ind(not a) = ind(a) + N, with N being
2 //the number of variables
3 //could however be changed easily
4 public static boolean 2SAT(Vertex[] G) {
5     //call SCC
6     double DFS(G);
7     //check for contradiction
8     boolean poss = true;
9     for(int i = 0; i < S+A; i++) {
10         if(G[i].comp == G[i + (S+A)].comp) {
11             poss = false;
12         }
13     }
14     return poss;
15 }

```

MD5: 6c06a2b59fd3a7df3c31b06c58fdaaf5 | $\mathcal{O}(V + E)$

3.2 Breadth First Search

Iterative BFS. Uses ref Vertex class, no Edge class needed. In this version we look for a shortest path from s to t though we could also find the BFS-tree by leaving out t . *Input:* IDs of start and goal vertex and graph as AdjList *Output:* true if there is a connection between s and g , false otherwise

```

1 public static boolean BFS(Vertex[] G, int s, int t) {
2     //make sure that Vertices vis values are false etc
3     Queue<Vertex> q = new LinkedList<Vertex>();
4     G[s].vis = true;
5     G[s].dist = 0;
6     G[s].pre = -1;
7     q.add(G[s]);
8     //expand frontier between undiscovered and
9     //discovered vertices
10    while(!q.isEmpty()) {
11        Vertex u = q.poll();

```

```

11 //when reaching the goal, return true
12 //if we want to construct a BFS-tree delete this
    line
13 if(u.id == t) return true;
14 //else add adj vertices if not visited
15 for(Vertex v : u.adj) {
16     if(!v.vis) {
17         v.vis = true;
18         v.dist = u.dist + 1;
19         v.pre = u.id;
20         q.add(v);
21     }
22 }
23 }
24 //did not find target
25 return false;
26 }

```

MD5: 71f3fa48b4f1b2abdf3557a27a9a136 | $\mathcal{O}(|V| + |E|)$

3.3 BellmanFord

Finds shortest pathes from a single source. Negative edge weights are allowed. Can be used for finding negative cycles.

```

1 public static boolean bellmanFord(Vertex[] G) {
2     //source is 0
3     G[0].dist = 0;
4     //calc distances
5     //the path has max length |V|-1
6     for(int i = 0; i < G.length-1; i++) {
7         //each iteration relax all edges
8         for(int j = 0; j < G.length; j++) {
9             for(Edge e : G[j].adj) {
10                 if(G[j].dist != Integer.MAX_VALUE
11                     && e.t.dist > G[j].dist + e.w) {
12                     e.t.dist = G[j].dist + e.w;
13                 }
14             }
15         }
16     }
17     //check for negative-length cycle
18     for(int i = 0; i < G.length; i++) {
19         for(Edge e : G[i].adj) {
20             if(G[i].dist != Integer.MAX_VALUE
21                 && e.t.dist > G[i].dist + e.w) {
22                 return true;
23             }
24         }
25     }
26     return false;
27 }

```

MD5: d101e6b6915f012b3f0c02dc79e1fc6f | $\mathcal{O}(|V| \cdot |E|)$

3.4 Bipartite Graph Check

Checks a graph represented as adjList for being bipartite. Needs a little adaption, if the graph is not connected.

Input: graph as adjList, amount of nodes N as int

Output: true if graph is bipartite, false otherwise

```

1 public static boolean bipartiteGraphCheck(Vertex[] G){
2     // use bfs for coloring each node
3     G[0].color = 1;

```

```

4 Queue<Vertex> q = new LinkedList<Vertex>();
5 q.add(G[0]);
6 while(!q.isEmpty()) {
7     Vertex u = q.poll();
8     for(Vertex v : u.adj) {
9         // if node i not yet visited,
10        // give opposite color of parent node u
11        if(v.color == -1) {
12            v.color = 1-u.color;
13            q.add(v);
14            // if node i has same color as parent node u
15            // the graph is not bipartite
16        } else if(u.color == v.color)
17            return false;
18        // if node i has different color
19        // than parent node u keep going
20    }
21 }
22 return true;
23 }

```

MD5: e93d242522e5b4085494c86f0d218dd4 | $\mathcal{O}(|V| + |E|)$

3.5 Maximum Bipartite Matching

Finds the maximum bipartite matching in an unweighted graph using DFS.

Input: An unweighted adjacency matrix boolean[M][N] with M nodes being matched to N nodes.

Output: The maximum matching. (For getting the actual matching, little changes have to be made.)

```

1 // A DFS based recursive function that returns true
2 // if a matching for vertex u is possible
3 boolean bpm(boolean bpGraph[][], int u,
4             boolean seen[], int matchR[]) {
5     // Try every job one by one
6     for (int v = 0; v < N; v++) {
7         // If applicant u is interested in job v and v
8         // is not visited
9         if (bpGraph[u][v] && !seen[v]) {
10             seen[v] = true; // Mark v as visited
11
12             // If job v is not assigned to an applicant OR
13             // previously assigned applicant for job v
14             // (which is matchR[v]) has an alternate job
15             // available. Since v is marked as visited in
16             // the above line, matchR[v] in the following
17             // recursive call will not get job v again
18             if (matchR[v] < 0 ||
19                 bpm(bpGraph, matchR[v], seen, matchR)) {
20                 matchR[v] = u;
21                 return true;
22             }
23         }
24     }
25     return false;
26 }
27
28 // Returns maximum number of matching from M to N
29 int maxBPM(boolean bpGraph[][]) {
30     // An array to keep track of the applicants assigned
31     // to jobs. The value of matchR[i] is the applicant
32     // number assigned to job i, the value -1 indicates
33     // nobody is assigned.
34     int matchR[] = new int[N];

```

```

35 // Initially all jobs are available
36 for(int i = 0; i < N; ++i)
37     matchR[i] = -1;
38 // Count of jobs assigned to applicants
39 int result = 0;
40 for (int u = 0; u < M; u++) {
41     // Mark all jobs as not seen for next applicant.
42     boolean seen[] = new boolean[N];
43     for(int i = 0; i < N; ++i)
44         seen[i] = false;
45     // Find if the applicant u can get a job
46     if (bpm(bpGraph, u, seen, matchR))
47         result++;
48 }
49 return result;
50 }

```

MD5: a4cc90bf91c41309ad7aaa0c2514ff06 | $\mathcal{O}(M \cdot N)$

3.6 Bitonic TSP

Input: Distance matrix d with vertices sorted in x-axis direction.

Output: Shortest bitonic tour length

```

1 public static double bitonic(double[][] d) {
2     int N = d.length;
3     double[][] B = new double[N][N];
4     for (int j = 0; j < N; j++) {
5         for (int i = 0; i <= j; i++) {
6             if (i < j - 1)
7                 B[i][j] = B[i][j - 1] + d[j - 1][j];
8             else {
9                 double min = 0;
10                for (int k = 0; k < j; k++) {
11                    double r = B[k][i] + d[k][j];
12                    if (min > r || k == 0)
13                        min = r;
14                }
15                B[i][j] = min;
16            }
17        }
18    }
19    return B[N-1][N-1];
20 }

```

MD5: 49fca508fb184da171e4c8e18b6ca4c7 | $\mathcal{O}(?)$

3.7 Single-source shortest paths in dag

Not tested but should be working fine Similar approach can be used for longest paths. Simply go through ts and add 1 to the largest longest path value of the incoming neighbors

```

1 public static void dagSSP(Vertex[] G, int s) {
2     //calls topological sort method
3     LinkedList<Integer> sorting = TS(G);
4     G[s].dist = 0;
5     //go through vertices in ts order
6     for(int u : sorting) {
7         for(Edge e : G[u].adj) {
8             Vertex v = e.t;
9             if(v.dist > u.dist + e.w) {
10                v.dist = u.dist + e.w;
11                v.pre = u.id;
12            }
13        }
14    }
15 }

```

```

14 }
15 }

```

MD5: 552172db2968f746c4ac0bd322c665f9 | $\mathcal{O}(|V| + |E|)$

3.8 Dijkstra

Finds the shortest paths from one vertex to every other vertex in the graph (SSSP).

For negative weights, add $|\min|+1$ to each edge, later subtract from result.

To get a different shortest path when edges are ints, add an $\varepsilon = \frac{1}{k+1}$ on each edge of the shortest path of length k , run again.

Input: A source vertex s and an adjacency list G .

Output: Modified adj. list with distances from s and predecessor vertices set.

```

1 public static void dijkstra(Vertex[] G, int s) {
2     G[s].dist = 0;
3     Tuple st = new Tuple(s, 0);
4     PriorityQueue<Tuple> q = new PriorityQueue<Tuple>();
5     q.add(st);
6
7     while(!q.isEmpty()) {
8         Tuple sm = q.poll();
9         Vertex u = G[sm.id];
10        //this checks if the Tuple is still useful, both
11        //checks should be equivalent
12        if(u.vis || sm.dist > u.dist) continue;
13        u.vis = true;
14        for(Edge e : u.adj) {
15            Vertex v = e.t;
16            if(!v.vis && v.dist > u.dist + e.w) {
17                v.pre = u.id;
18                v.dist = u.dist + e.w;
19                Tuple nt = new Tuple(v.id, v.dist);
20                q.add(nt);
21            }
22        }
23    }
24 }

```

MD5: e46eb1b919179dab6a42800376f04d7a | $\mathcal{O}(|E| \log |V|)$

3.9 EdmondsKarp

Finds the greatest flow in a graph. Capacities must be positive.

```

1 public static boolean BFS(Vertex[] G, int s, int t) {
2     int N = G.length;
3     for(int i = 0; i < N; i++) {
4         G[i].vis = false;
5     }
6
7     Queue<Vertex> q = new LinkedList<Vertex>();
8     G[s].vis = true;
9     G[s].pre = -1;
10    q.add(G[s]);
11
12    while(!q.isEmpty()) {
13        Vertex u = q.poll();
14        if(u.id == t) return true;
15        for(int i : u.adj.keySet()) {
16            Edge e = u.adj.get(i);

```

```

17     Vertex v = e.t;
18     if(!v.vis && e.rw > 0) {
19         v.vis = true;
20         v.pre = u.id;
21         q.add(v);
22     }
23 }
24 }
25 return (G[t].vis);
26 }
27 //We store the edges in the graph in a hashmap
28 public static int edKarp(Vertex[] G, int s, int t) {
29     int maxflow = 0;
30     while(BFS(G, s, t)) {
31         int pflow = Integer.MAX_VALUE;
32         for(int v = t; v != s; v = G[v].pre) {
33             int u = G[v].pre;
34             pflow = Math.min(pflow, G[u].adj.get(v).rw);
35         }
36         for(int v = t; v != s; v = G[v].pre) {
37             int u = G[v].pre;
38             G[u].adj.get(v).rw -= pflow;
39             G[v].adj.get(u).rw += pflow;
40         }
41         maxflow += pflow;
42     }
43     return maxflow;
44 }

```

MD5: 6067fa877ff237d82294e7511c79d4bc | $\mathcal{O}(|V|^2 \cdot |E|)$

3.10 Reference for Edge classes

Used for example in Dijkstra algorithm, implements edges with weight. Needs testing.

```

1 //for Kruskal we need to sort edges, use: java.lang.
   Comparable
2 class Edge implements Comparable<Edge> {}
3
4 class Edge {
5     //for Kruskal it is helpful to store the start as
6     //well, moreover we might not need the vertex class
7     int s;
8     int t;
9
10    //for EdKarp we also want to store residual weights
11    int rw;
12
13    Vertex t;
14    int w;
15
16    public Edge(Vertex t, int w) {
17        this.t = t;
18        this.w = w;
19        this.rw = w;
20    }
21
22    public Edge(int s, int t, int w) {...}
23
24    public int compareTo(Edge other) {
25        return Integer.compare(this.w, other.w);
26    }
27 }

```

MD5: aae80ac4bfbfcc0b9ac4c65085f6f123 | $\mathcal{O}(1)$

3.11 FloydWarshall

Finds all shortest paths. Paths in array next, distances in ans.

```

1 public static void floydWarshall(int[][] graph,
2                                 int[][] next, int[][] ans) {
3     for(int i = 0; i < ans.length; i++)
4         for(int j = 0; j < ans.length; j++)
5             ans[i][j] = graph[i][j];
6
7     for (int k = 0; k < ans.length; k++)
8         for (int i = 0; i < ans.length; i++)
9             for (int j = 0; j < ans.length; j++)
10                if (ans[i][k] + ans[k][j] < ans[i][j]
11                    && ans[i][k] < Integer.MAX_VALUE
12                    && ans[k][j] < Integer.MAX_VALUE) {
13                    ans[i][j] = ans[i][k] + ans[k][j];
14                    next[i][j] = next[i][k];
15                }
16 }

```

MD5: a98bbda7e53be8ee0df72dbd8721b306 | $\mathcal{O}(|V|^3)$

3.12 Held Karp

Algorithm for TSP

```

1 public static int[] tsp(int[][] graph) {
2     int n = graph.length;
3     if(n == 1) return new int[]{0};
4     //C stores the shortest distance to node of the
5     //second dimension, first dimension is the
6     //bitstring of included nodes on the way
7     int[][] C = new int[1<<n][n];
8     int[][] p = new int[1<<n][n];
9     //initialize
10    for(int k = 1; k < n; k++) {
11        C[1<<k][k] = graph[0][k];
12    }
13    for(int s = 2; s < n; s++) {
14        for(int S = 1; S < (1<<n); S++) {
15            if(Integer.bitCount(S)!=s || (S&1) == 1)
16                continue;
17            for(int k = 1; k < n; k++) {
18                if((S & (1 << k)) == 0) continue;
19
20                //Smk is the set of nodes without k
21                int Smk = S ^ (1<<k);
22
23                int min = Integer.MAX_VALUE;
24                int minprev = 0;
25                for(int m=1; m<n; m++) {
26                    if((Smk & (1<<m)) == 0) continue;
27                    //distance to m with the nodes in Smk +
28                    //connection from m to k
29                    int tmp = C[Smk][m] + graph[m][k];
30                    if(tmp < min) {
31                        min = tmp;
32                        minprev = m;
33                    }
34                }
35                C[S][k] = min;
36                p[S][k] = minprev;
37            }
38        }
39    }
40    //find shortest tour length

```



```

38 int min = Integer.MAX_VALUE;
39 int minprev = -1;
40 for(int k = 1; k < n; k++) {
41     //Set of all nodes except for the first + cost
42     //from 0 to k
43     int tmp = C[(1<n) - 2][k] + graph[0][k];
44     if(tmp < min) {
45         min = tmp;
46         minprev = k;
47     }
48 }
49 //Note that the tour has not been tested yet, only
50 //the correctness of the min-tour-value backtrack
51 //tour
52 int[] tour = new int[n+1];
53 tour[n] = 0;
54 tour[n-1] = minprev;
55 int bits = (1<n)-2;
56 for(int k = n-2; k>0; k--) {
57     tour[k] = p[bits][tour[k+1]];
58     bits = bits ^ (1<tour[k+1]);
59 }
60 tour[0] = 0;
61 return tour;

```

MD5: f3e9730287dcbf2695bf7372fc4baf0 | $\mathcal{O}(2^n n^2)$

3.13 Iterative DFS

Simple iterative DFS, the recursive variant is a bit fancier. Not tested.

```

1 //if we want to start the DFS for different connected
2 //components, there is such a method in the
3 //recursive variant of DFS
4 public static boolean ItDFS(Vertex[] G, int s, int t){
5     //take care that all the nodes are not visited at
6     //the beginning
7     Stack<Integer> S = new Stack<Integer>();
8     S.push(s);
9     while(!S.isEmpty()) {
10         int u = S.pop();
11         if(u.id == t) return true;
12         if(!G[u].vis) {
13             G[u].vis = true;
14             for(Vertex v : G[u].adj) {
15                 if(!v.vis)
16                     S.push(v.id);
17             }
18         }
19     }
20     return false;
21 }

```

MD5: 80f28ea9b2a04af19b48277e3c6bce9e | $\mathcal{O}(|V| + |E|)$

3.14 Johnsons Algorithm

```

1 public static int[][] johnson(Vertex[] G) {
2     Vertex[] Gd = new Vertex[G.length+1];
3     int s = G.length;
4     for(int i = 0; i < G.length; i++)
5         Gd[i] = G[i];

```

```

6 //init new vertex with zero-weight-edges to each
7 //vertex
8 Vertex S = new Vertex(G.length);
9 for(int i = 0; i < G.length; i++)
10     S.adj.add(new Edge(Gd[i], 0));
11 Gd[G.length] = S;
12
13 //bellman-ford to check for neg-weight-cycles and to
14 //adapt edges to enable running dijkstra
15 if(bellmanFord(Gd, s)) {
16     System.out.println("False");
17     //this should not happen and will cause troubles
18     return null;
19 }
20 //change weights
21 for(int i = 0; i < G.length; i++)
22     for(Edge e : Gd[i].adj)
23         e.w = e.w + Gd[i].dist - e.t.dist;
24 //store distances to invert this step later
25 int[] h = new int[G.length];
26 for(int i = 0; i < G.length; i++)
27     h[i] = Gd[i].dist;
28
29 //create shortest path matrix
30 int[][] apsp = new int[G.length][G.length];
31
32 //now use original graph G
33 //start a dijkstra for each vertex
34 for(int i = 0; i < G.length; i++) {
35     //reset weights
36     for(int j = 0; j < G.length; j++) {
37         G[j].vis = false;
38         G[j].dist = Integer.MAX_VALUE;
39     }
40     dijkstra(G, i);
41     for(int j = 0; j < G.length; j++)
42         apsp[i][j] = G[j].dist + h[j] - h[i];
43 }
44 return apsp;

```

MD5: 0a5c741be64b65c5211fe6056ffc1e02 | $\mathcal{O}(|V|^2 \log V + VE)$

3.15 Kruskal

Computes a minimum spanning tree for a weighted undirected graph.

```

1 public static int kruskal(Edge[] edges, int n) {
2     Arrays.sort(edges);
3     //n is the number of vertices
4     UnionFind uf = new UnionFind(n);
5     //we will only compute the sum of the MST, one could
6     //of course also store the edges
7     int sum = 0;
8     int cnt = 0;
9     for(int i = 0; i < edges.length; i++) {
10         if(cnt == n-1) break;
11         if(uf.union(edges[i].s, edges[i].t)) {
12             sum += edges[i].w;
13             cnt++;
14         }
15     }
16     return sum;

```

MD5: 91a1657706750a76d384d3130d98e5fb | $\mathcal{O}(|E| + \log |V|)$

3.16 Min Cut

Calculates the min cut using Edmonds Karp algorithm.

```

1 public static void bfs(Vertex[] G, int s) {
2     for(int i = 0; i < G.length; i++) {
3         G[i].vis = false;
4     }
5     Queue<Vertex> q = new LinkedList<Vertex>();
6     q.add(G[s]);
7
8     while(!q.isEmpty()) {
9         Vertex u = q.poll();
10        u.vis = true;
11
12        for(int i : u.adj.keySet()) {
13            Edge e = u.adj.get(i);
14            if(e.rw == 0) continue;
15            Vertex v = e.t;
16            if(v.vis) continue;
17            q.add(v);
18        }
19    }
20 }
21
22 public static int minCut(Vertex[] G, int s, int t) {
23     //get residual graph
24     edmondsKarp(G, s, t);
25     //find all vertices reachable from s
26     bfs(G, s);
27     int sum = 0;
28     for(int i = 0; i < G.length; i++) {
29         for(int j : G[i].adj.keySet()) {
30             Edge e = G[i].adj.get(j);
31             Vertex v = e.t;
32             //if i is reachable and j not this is a cut edge
33             if(G[i].vis && !G[j].vis) {
34                 //System.out.println((i+1) + " " + (j+1));
35                 sum += e.w;
36             }
37         }
38     }
39     return sum;
40 }

```

MD5: 3f081f37a378d8dd750bfe8877e50a87 | $\mathcal{O}(?)$

3.17 Prim

```

1 //s is the startpoint of the algorithm, in general not
2 //too important; we assume that graph is connected
3 public static int prim(Vertex[] G, int s) {
4     //make sure dists are maxint
5     G[s].dist = 0;
6     Tuple st = new Tuple(s, 0);
7
8     PriorityQueue<Tuple> q = new PriorityQueue<Tuple>();
9     q.add(st);
10    //we will store the sum and each nodes predecessor
11    int sum = 0;
12
13    while(!q.isEmpty()) {
14        Tuple sm = q.poll();
15        Vertex u = G[sm.id];
16        //u has been visited already
17        if(u.vis) continue;
18        //this is not the latest version of u

```

```

18        if(sm.dist > u.dist) continue;
19        u.vis = true;
20        //u is part of the new tree and u.dist the cost of
21        //adding it
22        sum += u.dist;
23        for(Edge e : u.adj) {
24            Vertex v = e.t;
25            if(!v.vis && v.dist > e.w) {
26                v.pre = u.id;
27                v.dist = e.w;
28                Tuple nt = new Tuple(v.id, e.w);
29                q.add(nt);
30            }
31        }
32    }
33    return sum;
34 }

```

MD5: c82f0bcc19cb735b4ef35dfc7ccfe197 | $\mathcal{O}(?)$

3.18 Recursive Depth First Search

Recursive DFS with different options (storing times, connected/unconnected graph). Needs testing.

Input: A source vertex s , a target vertex t , and adjlist G and the time (0 at the start)

Output: Indicates if there is connection between s and t .

```

1 //if we want to visit the whole graph, even if it is
2 //not connected we might use this
3 public static void DFS(Vertex[] G) {
4     //make sure all vertices vis value is false etc
5     int time = 0;
6     for(int i = 0; i < G.length; i++) {
7         if(!G[i].vis) {
8             //note that we leave out t so this does not work
9             //with the below function
10            //adaption will not be too difficult though
11            //time should not always start at zero, change
12            //if needed
13            recDFS(i, G, 0);
14        }
15    }
16 }
17
18 //first call with time = 0
19 public static boolean recDFS(int s, int t, Vertex[] G,
20     int time){
21     //it might be necessary to store the time of
22     //discovery
23     time = time + 1;
24     G[s].dtime = time;
25
26     G[s].vis = true; //new vertex has been discovered
27     //For cycle check vis should be int and 0 are not
28     //vis nodes
29     //1 are vis nodes which havent been finished and 2
30     //are finished nodes
31     //cycle exists iff edge to node with vis=1
32     //when reaching the target return true
33     //not necessary when calculating the DFS-tree
34     if(s == t) return true;
35     for(Vertex v : G[s].adj) {
36         //exploring a new edge
37         if(!v.vis) {
38             v.pre = s.id;

```

```

32     if(recDFS(v.id, t, G)) return true;
33 }
34 }
35 //storing finishing time
36 time = time + 1;
37 G[s].ftime = time;
38 return false;
39 }

```

MD5: 0829da7a5f49d16eeb886174e5d45213 | $\mathcal{O}(|V| + |E|)$

3.19 Strongly Connected Components

```

1 public static void fDFS(Vertex u, LinkedList<Integer>
   sorting) {
2     //compare with TS
3     u.vis = true;
4     for(Vertex v : u.out)
5         if(!v.vis)
6             fDFS(v, sorting);
7     sorting.addFirst(u.id);
8     return sorting;
9 }
10
11
12 public static void sDFS(Vertex u, int cnt) {
13     //basic DFS, all visited vertices get cnt
14     u.vis = true;
15     u.comp = cnt;
16     for(Vertex v : u.in)
17         if(!v.vis)
18             sDFS(v, cnt);
19 }
20
21 public static void doubleDFS(Vertex[] G) {
22     //first calc a topological sort by first DFS
23     LinkedList<Integer> sorting = new LinkedList<Integer>
24         >();
25     for(int i = 0; i < G.length; i++)
26         if(!G[i].vis)
27             fDFS(G[i], sorting);
28     for(int i = 0; i < G.length; i++)
29         G[i].vis = false;
30     //then go through the sort and do another DFS on G^T
31     //each tree is a component and gets a unique number
32     int cnt = 0;
33     for(int i : sorting)
34         if(!G[i].vis)
35             sDFS(G[i], cnt++);
36 }

```

MD5: 1e023258a9249a1bc0d6898b670139ea | $\mathcal{O}(|V| + |E|)$

3.20 Suurballe

Finds the min cost of two edge disjoint paths in a graph. If vertex disjoint needed, split vertices.

Input: Graph G , Source s , Target t

Output: Min cost as int

```

1 public static int suurballe(Vertex[] G, int s, int t){
2     //this uses the usual dijkstra implementation with
   stored predecessors
3     dijkstra(G, s);
4     //Modifying weights
5     for(int i = 0; i < G.length; i++)

```

```

   for(Edge e : G[i].adj)
6       e.dist = e.dist - e.t.dist + G[i].dist;
7
8   //reversing path and storing used edges
9   int old = t;
10  int pre = G[t].pre;
11  HashMap<Integer, Integer> hm = new HashMap<Integer,
   Integer>();
12  while(pre != -1) {
13      for(int i = 0; i < G[pre].adj.size(); i++) {
14          if(G[pre].adj.get(i).t.id == old) {
15              hm.put(pre * G.length + old, G[pre].adj.get(i)
16                  .tdist);
17              G[pre].adj.remove(i);
18              break;
19          }
20      }
21      boolean found = false;
22      for(int i = 0; i < G[old].adj.size(); i++) {
23          if(G[old].adj.get(i).t.id == pre) {
24              G[old].adj.get(i).dist = 0;
25              found = true;
26              break;
27          }
28      }
29      if(!found)
30          G[old].adj.add(new Edge(G[pre], 0));
31      old = pre;
32      pre = G[pre].pre;
33  }
34  //reset graph
35  for(int i = 0; i < G.length; i++) {
36      G[i].pre = -1;
37      G[i].dist = Integer.MAX_VALUE;
38      G[i].vis = false;
39  }
40
41  dijkstra(G, s);
42  //store edges of second path
43  old = t;
44  pre = G[t].pre;
45  while(pre != -1) {
46      //store edges and remove if reverse
47      for(int i = 0; i < G[pre].adj.size(); i++) {
48          if(G[pre].adj.get(i).t.id == old) {
49              if(!hm.containsKey(pre + old * G.length))
50                  hm.put(pre * G.length + old, G[pre].adj.get(
51                      i).tdist);
52              else
53                  hm.remove(pre + old * G.length);
54              break;
55          }
56      }
57      old = pre;
58      pre = G[pre].pre;
59  }
60  //sum up weights
61  int sum = 0;
62  for(int i : hm.keySet())
63      sum += hm.get(i);
64  return sum;
65 }

```

MD5: 222dac2a859273efbddd0ec0d6285dd7 | $\mathcal{O}(V \log V + E)$

3.21 Kahns Algorithm for TS

Gives the specific TS where Vertices first in G are first in the sorting

```

1 public static LinkedList<Integer> TS(Vertex[] G) {
2     LinkedList<Integer> sorting = new LinkedList<Integer>
3         >();
4     PriorityQueue<Vertex> p = new PriorityQueue<Vertex>
5         >();
6     //inc counts the number of incoming edges, if they
7     //are zero put the vertex in the queue
8     for(int i = 0; i < G.length; i++) {
9         if(G[i].inc == 0) {
10            p.add(G[i]);
11            G[i].vis = true;
12        }
13    }
14    while(!p.isEmpty()) {
15        Vertex u = p.poll();
16        sorting.add(u.id);
17        //update inc
18        for(Vertex v : u.out) {
19            if(v.vis) continue;
20            v.inc--;
21            if(v.inc == 0) {
22                p.add(v);
23                v.vis = true;
24            }
25        }
26    }
27    return sorting;
28 }
```

MD5: e53d13c7467873d1c5d210681f4450d8 | $\mathcal{O}(V + E)$

3.22 Topological Sort

```

1 public static LinkedList<Integer> TS(Vertex[] G) {
2     LinkedList<Integer> sorting = new LinkedList<Integer>
3         >();
4     for(int i = 0; i < G.length; i++)
5         if(!G[i].vis)
6             recTS(G[i], sorting);
7     //check sorting for a -1 if the graph is not
8     //necessarily dag
9     //maybe checking if there are too many values in
10    //sorting is easier?!
11    return sorting;
12 }
13
14 public static LinkedList<Integer> recTS(Vertex u,
15     LinkedList<Integer> sorting) {
16     u.vis = true;
17     for(Vertex v : u.adj)
18         if(v.vis)
19             //the -1 indicates that it will not be possible
20             //to find an TS
21             //there might be a much faster and elegant way (
22             //flag?!)
23             sorting.addFirst(-1);
24     else
25         recTS(v, sorting);
26     sorting.addFirst(u.id);
27     return sorting;
28 }
```

MD5: f6459575bf0d53344ddd9e5daf1dfbb8 | $\mathcal{O}(|V| + |E|)$

3.23 Tuple

Simple tuple class used for priority queue in Dijkstra and Prim

```

1 class Tuple implements Comparable<Tuple> {
2
3     int id;
4     int dist;
5
6     public Tuple(int id, int dist) {
7         this.id = id;
8         this.dist = dist;
9     }
10
11     public int compareTo(Tuple other) {
12         return Integer.compare(this.dist, other.dist);
13     }
14 }
```

MD5: fb1aa32dc32b9a2bac6f44a84e7f82c7 | $\mathcal{O}(1)$

3.24 Reference for Vertex classes

Used in many graph algorithms, implements a vertex with its edges. Needs testing.

```

1 class Vertex {
2
3     int id;
4     boolean vis = false;
5     int pre = -1;
6
7     //for dijkstra and prim
8     int dist = Integer.MAX_VALUE;
9
10    //for SCC store number indicating the dedicated
11    //component
12    int comp = -1;
13
14    //for DFS we could store the start and finishing
15    //times
16    int dtime = -1;
17    int ftime = -1;
18
19    //use an ArrayList of Edges if those information are
20    //needed
21    ArrayList<Edge> adj = new ArrayList<Edge>();
22    //use an ArrayList of Vertices else
23    ArrayList<Vertex> adj = new ArrayList<Vertex>();
24    //use two ArrayLists for SCC
25    ArrayList<Vertex> in = new ArrayList<Vertex>();
26    ArrayList<Vertex> out = new ArrayList<Vertex>();
27
28    //for EdmondsKarp we need a HashMap to store Edges,
29    //Integer is target
30    HashMap<Integer, Edge> adj = new HashMap<Integer,
31        Edge>();
32
33    //for bipartite graph check
34    int color = -1;
35
36    //we store as key the target
37 }
```

```

32 public Vertex(int id) {
33     this.id = id;
34 }
35 }

```

MD5: 90e8120ce9f665b07d4388e30395dd36 | $\mathcal{O}(1)$

4 Math

4.1 Binomial Coefficient

Gives binomial coefficient (n choose k)

```

1 public static long bin(int n, int k) {
2     if (k == 0)
3         return 1;
4     else if (k > n/2)
5         return bin(n, n-k);
6     else
7         return n*bin(n-1, k-1)/k;
8 }

```

MD5: 32414ba5a444038b9184103d28fa1756 | $\mathcal{O}(k)$

4.2 Binomial Matrix

Gives binomial coefficients for all $K \leq N$.

```

1 public static long[][] binomial_matrix(int N, int K) {
2     long[][] B = new long[N+1][K+1];
3     for (int k = 1; k <= K; k++)
4         B[0][k] = 0;
5     for (int m = 1; m <= N; m++)
6         B[m][0] = 1;
7     for (int m = 1; m <= N; m++)
8         for (int k = 1; k <= K; k++)
9             B[m][k] = B[m-1][k-1] + B[m-1][k];
10    return B;
11 }

```

MD5: e6f103bd9852173c02a1ec64264f4448 | $\mathcal{O}(N \cdot K)$

4.3 Divisability

Calculates (alternating) k-digitSum for integer number given by M.

```

1 public static long digit_sum(String M, int k, boolean
2     alt) {
3     long dig_sum = 0;
4     int vz = 1;
5     while (M.length() > k) {
6         if (alt) vz *= -1;
7         dig_sum += vz*Integer.parseInt(M.substring(M.
8             length()-k));
9         M = M.substring(0, M.length()-k);
10    }
11    if (alt)
12        vz *= -1;
13    dig_sum += vz*Integer.parseInt(M);
14    return dig_sum;
15 }

```

// example: divisibility of M by 13

```

16 public static boolean divisible13(String M) {
17     return digit_sum(M, 3, true)%13 == 0;
18 }

```

MD5: 33b3094ebf431e1e71cd8e8db3c9cdd6 | $\mathcal{O}(|M|)$

4.4 Graham Scan

Multiple unresolved issues: multiple points as well as collinearity.
 N denotes the number of points

```

1 public static Point[] grahamScan(Point[] points) {
2     //find leftmost point with lowest y-coordinate
3     int xmin = Integer.MAX_VALUE;
4     int ymin = Integer.MAX_VALUE;
5     int index = -1;
6     for(int i = 0; i < points.length; i++) {
7         if(points[i].y < ymin || (points[i].y == ymin &&
8             points[i].x < xmin)) {
9             xmin = points[i].x;
10            ymin = points[i].y;
11            index = i;
12        }
13    }
14    //get that point to the start of the array
15    Point tmp = new Point(points[index].x, points[index
16        ].y);
17    points[index] = points[0];
18    points[0] = tmp;
19    for(int i = 1; i < points.length; i++)
20        points[i].src = points[0];
21    Arrays.sort(points, 1, points.length);
22    //for collinear points eliminate all but the
23    farthest
24    boolean[] isElem = new boolean[points.length];
25    for(int i = 1; i < points.length-1; i++) {
26        Point a = new Point(points[i].x - points[i].src.x,
27            points[i].y - points[i].src.y);
28        Point b = new Point(points[i+1].x - points[i+1].
29            src.x, points[i+1].y - points[i+1].src.y);
30        if(Calc.crossProd(a, b) == 0)
31            isElem[i] = true;
32    }
33    //works only if there are more than three non-
34    collinear points
35    Stack<Point> s = new Stack<Point>();
36    int i = 0;
37    for(; i < 3; i++) {
38        while(isElem[i++]);
39        s.push(points[i]);
40    }
41    for(i < points.length; i++) {
42        if(isElem[i]) continue;
43        while(true) {
44            Point first = s.pop();
45            Point second = s.pop();
46            s.push(second);
47            Point a = new Point(first.x - second.x, first.y
48                - second.y);
49            Point b = new Point(points[i].x - second.x,
50                points[i].y - second.y);
51            //use >= if straight angles are needed
52            if(Calc.crossProd(a, b) > 0) {
53                s.push(first);
54                s.push(points[i]);
55                break;
56            }
57        }
58    }
59 }

```

```

49     }
50 }
51 Point[] convexHull = new Point[s.size()];
52 for(int j = s.size()-1; j >= 0; j--)
53     convexHull[j] = s.pop();
54 return convexHull;
55 /*Sometimes it might be necessary to also add points
   to the convex hull that form a straight angle.
   The following lines of code achieve this. Only
   at the first and last diagonal we have to add
   those. Of course the previous return-statement
   has to be deleted as well as allowing straight
   angles in the above implementation. */
56 }
57 class Point implements Comparable<Point> {
58     Point src; //set seperately in GrahamScan method
59     int x;
60     int y;
61
62     public Point(int x, int y) {
63         this.x = x;
64         this.y = y;
65     }
66
67     //might crash if one point equals src
68     //major issues with multiple points on same location
69     !
70     public int compareTo(Point cmp) {
71         Point a = new Point(this.x - src.x, this.y - src.y);
72         Point b = new Point(cmp.x - src.x, cmp.y - src.y);
73         //checks if points are identical
74         if(a.x == b.x && a.y == b.y) return 0;
75         //if same angle, sort by dist
76         if(Calc.crossProd(a, b) == 0 && Calc.dotProd(a, b) >
77             0)
78             return Integer.compare(Calc.dotProd(a, a), Calc.
79                 dotProd(b, b));
80         //angle of a is 0, thus b>a
81         if(a.y == 0 && a.x > 0) return -1;
82         //angle of b is 0, thus a>b
83         if(b.y == 0 && b.x > 0) return 1;
84         //a ist between 0 and 180, b between 180 and 360
85         if(a.y > 0 && b.y < 0) return -1;
86         if(a.y < 0 && b.y > 0) return 1;
87         //return negative value if cp larger than zero
88         return Integer.compare(0, Calc.crossProd(a, b));
89     }
90 }
91 class Calc {
92     public static int crossProd(Point p1, Point p2) {
93         return p1.x * p2.y - p2.x * p1.y;
94     }
95     public static int dotProd(Point p1, Point p2) {
96         return p1.x * p2.x + p1.y * p2.y;
97     }
98 }

```

MD5: 2555d858fadcf8cb404a9c52420545d | $\mathcal{O}(N \log N)$

4.5 Iterative EEA

Berechnet den ggT zweier Zahlen a und b und deren modulare Inverse $x = a^{-1} \bmod b$ und $y = b^{-1} \bmod a$.

```

3  if (b > a) {
4      long tmp = a;
5      a = b;
6      b = tmp;
7  }
8  long x = 0, y = 1, u = 1, v = 0;
9  while (a != 0) {
10     long q = b / a, r = b % a;
11     long m = x - u * q, n = y - v * q;
12     b = a; a = r; x = u; y = v; u = m; v = n;
13 }
14 long gcd = b;
15 // x = a^-1 % b, y = b^-1 % a
16 // ax + by = gcd
17 long[] erg = { gcd, x, y };
18 return erg;
19 }

```

MD5: 81fe8cd4adab21329dcbe1ce0499ee75 | $\mathcal{O}(\log a + \log b)$

4.6 Polynomial Interpolation

```

1 public class interpol {
2
3     // divided differences for points given by vectors x
4     and y
5     public static rat[] divDiff(rat[] x, rat[] y) {
6         rat[] temp = y.clone();
7         int n = x.length;
8         rat[] res = new rat[n];
9         res[0] = temp[0];
10        for (int i=1; i < n; i++) {
11            for (int j = 0; j < n-i; j++) {
12                temp[j] = (temp[j+1].sub(temp[j])).div(x[j+i].
13                    sub(x[j]));
14            }
15            res[i] = temp[0];
16        }
17        return res;
18    }
19
20    // evaluates interpolating polynomial p at t for
21    given
22    // x-coordinates and divided differences
23    public static rat p(rat t, rat[] x, rat[] dD) {
24        int n = x.length;
25        rat p = new rat(0);
26        for (int i = n-1; i > 0; i--) {
27            p = (p.add(dD[i])).mult(t.sub(x[i-1]));
28        }
29        p = p.add(dD[0]);
30        return p;
31    }
32 }

```

// implementation of rational numbers

```

32 class rat {
33
34     public long c;
35     public long d;
36
37     public rat (long c, long d) {
38         this.c = c;
39         this.d = d;
40         this.shorten();
41     }
42 }

```

```

1 // Extended Euclidean Algorithm - iterativ
2 public static long[] eea(long a, long b) {

```



```

43 public rat (long c) {
44     this.c = c;
45     this.d = 1;
46 }
47
48 public static long ggT(long a, long b) {
49     while (b != 0) {
50         long h = a%b;
51         a = b;
52         b = h;
53     }
54     return a;
55 }
56
57 public static long kgV(long a, long b) {
58     return a*b/ggT(a,b);
59 }
60
61 public static rat[] commonDenominator(rat[] c) {
62     long kgV = 1;
63     for (int i = 0; i < c.length; i++) {
64         kgV = kgV(kgV, c[i].d);
65     }
66     for (int i = 0; i < c.length; i++) {
67         c[i].c *= kgV/c[i].d;
68         c[i].d *= kgV/c[i].d;
69     }
70     return c;
71 }
72
73 public void shorten() {
74     long ggT = ggT(this.c, this.d);
75     this.c = this.c / ggT;
76     this.d = this.d / ggT;
77     if (d < 0) {
78         this.d *= -1;
79         this.c *= -1;
80     }
81 }
82
83 public String toString() {
84     if (this.d == 1) return ""+c;
85     return ""+c+"/"+d;
86 }
87
88 public rat mult(rat b) {
89     return new rat(this.c*b.c, this.d*b.d);
90 }
91
92 public rat div(rat b) {
93     return new rat(this.c*b.d, this.d*b.c);
94 }
95
96 public rat add(rat b) {
97     long new_d = kgV(this.d, b.d);
98     long new_c = this.c*(new_d/this.d) + b.c*(new_d/b.d);
99     return new rat(new_c, new_d);
100 }
101
102 public rat sub(rat b) {
103     return this.add(new rat(-b.c, b.d));
104 }
105 }

```

MD5: e7b408030f7e051e93a8c55056ba930b | O(?)

4.7 Root of permutation

Calculates the K'th root of permutation of size N. Number at place i indicates where this dancer ended. needs commenting

```

1 public static int[] rop(int[] perm, int N, int K) {
2     boolean[] incyc = new boolean[N];
3     int[] cntcyc = new int[N+1];
4     int[] g = new int[N+1];
5     int[] needed = new int[N+1];
6     for(int i = 1; i < N+1; i++) {
7         int j = i;
8         int k = K;
9         int div;
10        while(k > 1 && (div = gcd(k, i)) > 1) {
11            k /= div;
12            j *= div;
13        }
14        needed[i] = j;
15        g[i] = gcd(K, j);
16    }
17
18    HashMap<Integer, ArrayList<Integer>> hm = new
19        HashMap<Integer, ArrayList<Integer>>();
20    for(int i = 0; i < N; i++) {
21        if(incyc[i]) continue;
22        ArrayList<Integer> cyc = new ArrayList<Integer>();
23        incyc[i] = true;
24        int newelem = perm[i];
25        while(newelem != i) {
26            cyc.add(newelem);
27            incyc[newelem] = true;
28            newelem = perm[newelem];
29        }
30        int len = cyc.size();
31        cntcyc[len]++;
32        if(hm.containsKey(len)) {
33            hm.get(len).addAll(cyc);
34        } else {
35            hm.put(len, cyc);
36        }
37    }
38    boolean end = false;
39    for(int i = 1; i < N+1; i++) {
40        if(cntcyc[i] % g[i] != 0) end = true;
41    }
42    if(end) {
43        //not possible
44        return null;
45    } else {
46        int[] out = new int[N];
47        for(int length = 0; length < N; length++) {
48            if(!hm.containsKey(length)) continue;
49            ArrayList<Integer> p = hm.get(length);
50            int totalsize = p.size();
51            int diffcyc = totalsize / needed[length];
52            for(int i = 0; i < diffcyc; i++) {
53                int[] c = new int[needed[length]];
54                for(int it = 0; it < needed[length]; it++) {
55                    c[it] = p.get(it + i * needed[length]);
56                }
57                int move = K / (needed[length]/length);
58                int[] rewind = new int[needed[length]];
59                for(int set = 0; set < needed[length]/length;
60                    set++) {
61                    int pos = set * length;
62                    for(int it = 0; it < length; it++) {

```

```

62         rewind[pos] = c[it + set * length];
63         pos = ((pos - set * length + move) %
               length) + set * length;
64     }
65 }
66 int[] merge = new int[needed[length]];
67 for(int it = 0; it < needed[length]/length; it
    ++){
68     for(int set = 0; set < length; set++){
69         merge[set * needed[length] / length + it]
            = rewind[it * length + set];
70     }
71 }
72 for(int it = 0; it < needed[length]; it++){
73     out[merge[it]] = merge[(it+1) % needed[
        length]];
74 }
75 }
76 }
77 return out;
78 }
79 }

```

MD5: b446a7c21eddf7d14dbdc71174e8d498 | $\mathcal{O}(?)$

4.8 Sieve of Eratosthenes

Calculates Sieve of Eratosthenes.

Input: A integer N indicating the size of the sieve.

Output: A boolean array, which is true at an index i iff i is prime.

```

1 public static boolean[] sieveOfEratosthenes(int N) {
2     boolean[] isPrime = new boolean[N+1];
3     for (int i=2; i<=N; i++) isPrime[i] = true;
4     for (int i = 2; i*i <= N; i++)
5         if (isPrime[i])
6             for (int j = i*i; j <= N; j+=i)
7                 isPrime[j] = false;
8     return isPrime;
9 }

```

MD5: 95704ae7c1fe03e91adeb8d695b2f5bb | $\mathcal{O}(n)$

4.9 Greatest Common Divisor

Calculates the gcd of two numbers a and b or of an array of numbers *input*.

Input: Numbers a and b or array of numbers *input*

Output: Greatest common divisor of the input

```

1 private static long gcd(long a, long b) {
2     while (b > 0) {
3         long temp = b;
4         b = a % b; // % is remainder
5         a = temp;
6     }
7     return a;
8 }
9
10 private static long gcd(long[] input) {
11     long result = input[0];
12     for(int i = 1; i < input.length; i++)
13         result = gcd(result, input[i]);
14     return result;
15 }

```

MD5: 48058e358a971c3ed33621e3118818c2 | $\mathcal{O}(\log a + \log b)$

4.10 Least Common Multiple

Calculates the lcm of two numbers a and b or of an array of numbers *input*.

Input: Numbers a and b or array of numbers *input*

Output: Least common multiple of the input

```

1 private static long lcm(long a, long b) {
2     return a * (b / gcd(a, b));
3 }
4
5 private static long lcm(long[] input) {
6     long result = input[0];
7     for(int i = 1; i < input.length; i++)
8         result = lcm(result, input[i]);
9     return result;
10 }

```

MD5: 3cfaab4559ea05c8434d6cf364a24546 | $\mathcal{O}(\log a + \log b)$

4.11 GEV

```

1 #include <vector>
2 #include <algorithm>
3 #include <string>
4 #include <cmath>
5 #include <cstdio>
6 #include <cstring>
7
8 using namespace std;
9
10 template<int M> class vec
11 {
12 public:
13     double co[M];
14
15     vec<M>() { memset(co, 0, M * sizeof(double)); }
16
17     double* operator[](int i) { return &co[i]; }
18
19     vec<M> operator+(vec<M> v)
20     {
21         vec<M> r;
22         for(int i = 0; i < M; ++i)
23             *r[i] = co[i] + *v[i];
24         return r;
25     }
26
27     vec<M> operator-(vec<M> v)
28     {
29         vec<M> r;
30         for(int i = 0; i < M; ++i)
31             *r[i] = co[i] - *v[i];
32         return r;
33     }
34
35     vec<M> operator-()
36     {
37         vec<M> r;
38         for(int i = 0; i < M; ++i)
39             *r[i] = -co[i];

```



```

40     return r;
41 }
42
43 vec<M> operator*(double s)
44 {
45     vec<M> r;
46     for(int i = 0; i < M; ++i)
47         *r[i] = s * co[i];
48     return r;
49 }
50
51 // Kreuzprodukt
52 vec<3> cross(vec<3> v)
53 {
54     vec<3> r;
55     *r[0] = co[1] * *v[2] - co[2] * *v[1];
56     *r[1] = co[2] * *v[0] - co[0] * *v[2];
57     *r[2] = co[0] * *v[1] - co[1] * *v[0];
58     return r;
59 }
60 };
61
62 template<int M, int N> class mat
63 {
64 public:
65     double el[M][N];
66
67     mat<M, N>() { memset(el, 0, M * N * sizeof(double)); }
68
69     double* operator[](int i) { return el[i]; } // Gib
70                                             // Zeile i
71
72     // MxN-Matrix mal Nx1-Vektor = Mx1-Vektor
73     vec<M> operator*(vec<N> v)
74     {
75         vec<M> r;
76         for(int i = 0; i < M; ++i)
77             for(int j = 0; j < N; ++j)
78                 *r[i] += el[i][j] * *v[j]; // r ist durch
79                                             // Konstruktor genullt
80         return r;
81     }
82
83     // Gauß-Jordan-Algorithmus-Aufruf für MxN-Matrix und
84     // Mx1-Vektor
85     // Setzt voraus, dass Lösung existiert! => Nur bei
86     // MxM-Matrizen sinnvoll
87     vec<M> solveLGS(vec<M> in)
88     {
89         mat<M, N> inp;
90         for(int i = 0; i < M; ++i)
91             inp[i][0] = *in[i];
92         mat<M, N> re = gaussJordan(inp);
93         vec<M> r;
94         for(int i = 0; i < M; ++i)
95             *r[i] = re[i][0];
96         return r;
97     }
98
99     // Gauß-Jordan-Algorithmus für zwei MxN-Matrizen
100    // Setzt voraus, dass Lösung existiert! => Nur bei
101    // MxM-Matrizen sinnvoll
102    mat<M, N> gaussJordan(mat<M, N> in)
103    {
104        // Erweiterte Matrix erstellen
105        double ext[M][N << 1];
106        for(int i = 0; i < M; ++i)
107            memcpy(ext[i], el[i], N * sizeof(double));
108            memcpy(ext[i] + N, in[i], N * sizeof(double));
109        }
110
111        // Für jede Restmatrix Schritte durchführen
112        for(int LC = 0; LC < M && LC < N; ++LC)
113        {
114            // Finde Spalte mit Zelle != 0
115            int c = LC;
116            int l = LC;
117            for(; c < N; ++c, l = LC)
118                for(; l < M; ++l)
119                    if(!ext[l][c] == 0)
120                        goto br;
121
122            // Zeile mit gewähltem Element nach oben
123            // schieben und alle anderen Elemente durch
124            // dieses teilen
125            br:
126            double tmp[N << 1];
127            double top = ext[l][c];
128            //if(top == 0) // Dies ist erforderlich, wenn
129            //keine Lösung existiert oder das System
130            //überbestimmt ist
131            // break;
132            if(l > LC)
133                memcpy(tmp, ext[LC], (N << 1) * sizeof(double));
134            for(int j = LC; j < (N << 1); ++j)
135                ext[LC][j] = ext[l][j] / top;
136            if(l > LC)
137                memcpy(ext[l], tmp, (N << 1) * sizeof(double));
138
139            // Erstes Element jeder Zeile durch Subtraktion
140            // von Vielfachen der ersten Zeile auf 0
141            // bringen
142            for(int i = LC + 1; i < M; ++i)
143                for(int j = (N << 1) - 1; j >= c; --j)
144                    ext[i][j] -= ext[i][c] * ext[LC][j];
145        }
146
147        // Aus oberer Dreiecksmatrix Einheitsmatrix
148        // erstellen
149        for(int i = M - 1; i > 0; --i)
150            for(int i2 = i - 1; i2 >= 0; --i2)
151                for(int j = (N << 1) - 1; j > i2; --j)
152                    ext[i2][j] -= ext[i2][i] * ext[i][j];
153
154        // Ergebnismatrix erstellen
155        mat<M, N> r;
156        for(int i = 0; i < M; ++i)
157            memcpy(r[i], ext[i] + N, N * sizeof(double));
158        return r;
159    }
160 };
161
162 int main()
163 {
164     int T;
165     cin >> T;
166     while(T --> 0)
167     {
168         mat<7, 7> m;
169         for(int i = 0; i < 7; ++i)
170             for(int j = 0; j < 7; ++j)
171                 cin >> m[i][j];
172     }
173 }

```

```

161
162     mat<7, 7> unit;
163     for(int i = 0; i < 7; ++i)
164         unit[i][i] = 1;
165
166     mat<7, 7> res = m.gaussJordan(unit); // Inverses
167         berechnen
168     for(int i = 0; i < 7; ++i)
169     {
170         for(int j = 0; j < 7; ++j)
171             printf("%.03f", res[i][j]);
172         cout << endl;
173     }
174     cout << endl;
175 }
176
177 mat<3, 3> m2;
178 m2[0][0] = 1;
179 m2[0][1] = 1;
180 m2[0][2] = 1;
181 m2[1][0] = 4;
182 m2[1][1] = 2;
183 m2[1][2] = 1;
184 m2[2][0] = 9;
185 m2[2][1] = 3;
186 m2[2][2] = 1;
187
188 vec<3> v2;
189 *v2[0] = 0;
190 *v2[1] = 1;
191 *v2[2] = 3;
192
193 vec<3> result = m2.solveLGS(v2);
194 cout << *result[0] << " " << *result[1] << " " << *
    result[2] << endl;
195 }

```

MD5: b67cae273b9f6ac04b16ee619c9b351d | $\mathcal{O}(?)$

4.12 Fourier transform

```

1 #include<complex>
2 #include<vector>
3 #include<algorithm>
4 #include<cmath>
5
6 using namespace std;
7
8 void iterativefft(const vector<long long> &pol, vector
    <complex<double>> &fft, int n, bool inv)
9 {
10     //copy pol into fft
11     if(!inv) {
12         for(int i = 0; i < n; ++i) {
13             complex<double> cp (pol[i], 0);
14             fft[i] = cp;
15         }
16     }
17     //swap positions accordingly
18     for(int i = 0, j = 0; i < n; ++i) {
19         if(i < j) swap(fft[i], fft[j]);
20         int m = n >> 1;
21         while(1 <= m && m <= j) j -= m, m >>= 1;
22         j += m;
23     }
24     for(int m = 1; m <= n; m <= 1) { //<= or <
25         double theta = (inv ? -1 : 1) * 2 * M_PI / m;

```

```

26         complex<double> wm(cos(theta), sin(theta));
27         for(int k = 0; k < n; k += m) {
28             complex<double> w = 1;
29             for(int j = 0; j < m/2; ++j) {
30                 complex<double> t = w * fft[k + j + m
31                     /2];
32                 complex<double> u = fft[k + j];
33                 fft[k + j] = u + t;
34                 fft[k + j + m/2] = u - t;
35                 w = w*wm;
36             }
37         }
38         if(inv) {
39             for(int i = 0; i < n; ++i) {
40                 fft[i] /= complex<double> (n);
41             }
42         }
43     }
44 }
45 int main()
46 {
47     int N;
48     cin >> N;
49     vector<long long> pol (262144);
50     int min = 60000;
51     int max = -60000;
52     for(int i = 0; i < N; ++i) {
53         int ind;
54         cin >> ind;
55         if(ind < min) min = ind;
56         if(ind > max) max = ind;
57         ++pol[ind+65536];
58     }
59     vector<complex<double>> fft (262144);
60     iterativefft(pol, fft, 262144, false);
61     for(int i = 0; i < 262144; ++i) {
62         fft[i] *= fft[i];
63     }
64     iterativefft(pol, fft, 262144, true);
65     long long sum = 0;
66     for(int i = 81072; i <= 181072; ++i) {
67         int ind = i - 131072;
68         if(ind < min) continue;
69         if(ind > max) break;
70         long long resi = round(fft[i].real());
71         if(ind % 2 == 0 && ind != 0) {
72             resi -= pol[ind/2 + 65536] * pol[ind/2 +
73                 65536];
74             resi += pol[ind/2 + 65536]*(pol[ind/2 +
75                 65536]-1);
76         }
77         resi *= pol[ind + 65536];
78         if(ind != 0) {
79             resi -= 2*pol[65536] * pol[ind + 65536] *
80                 pol[ind + 65536];
81             resi += 2*pol[65536] * pol[ind + 65536] *
82                 (pol[ind + 65536]-1);
83         }
84         sum += resi;
85     }
86     cout << sum << endl;
87 }

```

MD5: fd9669c4967b6f26c13f464f98bdfb2a | $\mathcal{O}(?)$

4.13 Matrix exponentiation

```

1 void mult(int a[][nos], int b[][nos], int N)
2 {
3     int res[nos][nos] = {0};
4     for(int i = 0; i < N; i++) {
5         for(int j = 0; j < N; j++) {
6             for(int k = 0; k < N; k++) {
7                 res[i][j] = (res[i][j] + a[i][k]*b[k][
8                     j]) % 10000;
9             }
10        }
11    }
12    for(int i = 0; i < N; i++) {
13        for(int j = 0; j < N; j++) {
14            a[i][j] = res[i][j];
15        }
16    }
17    //start with g^L by succ squaring
18    int res[nos][nos] = {0};
19    for(int i = 0; i < N; i++) {
20        for(int j = 0; j < N; j++) {
21            if(i == j) res[i][j] = 1;
22        }
23    }
24    for(int i = 0; (1 << i) <= L; i++) {
25        if(((1 << i) & L) == (1 << i)) {
26            mult(res, g, N);
27        }
28        mult(g, g, N);
29    }

```

MD5: dcabdd3a0beceb4221f4c41071ac9b6d | $\mathcal{O}(?)$

4.14 phi function calculator

takes sqrt(n) time

```

1 int phi(int n)
2 {
3     double result = n;
4     for(int p = 2; p * p <= n; ++p) {
5         if(n % p == 0) {
6             while(n % p == 0) n /= p;
7             result *= (1.0 - (1.0 / (double) p));
8         }
9     }
10    if(n > 1) result *= (1.0 - (1.0 / (double) n));
11    return round(result);
12 }

```

MD5: 2ec930cc10935f1638700bb74e3439d9 | $\mathcal{O}(?)$

4.15 prints farey seq

```

1 def farey( n, asc=True ):
2     """Python function to print the nth Farey sequence
3     , either ascending or descending."""
4     if asc:
5         a, b, c, d = 0, 1, 1, n # (*)
6     else:
7         a, b, c, d = 1, 1, n-1, n # (*)
8     print "%d/%d" % (a,b)
9     while (asc and c <= n) or (not asc and a > 0):

```

```

k = int((n + b)/d)
a, b, c, d = c, d, k*c - a, k*d - b
print "%d/%d" % (a,b)

```

MD5: 5fe50f5717cb7d4e3eb91c8c8f6a1e85 | $\mathcal{O}(?)$

5 Misc

5.1 Binary Search

Binary searches for an element in a sorted array.

Input: sorted array to search in, amount N of elements in array, element to search for a

Output: returns the index of a in array or -1 if array does not contain a

```

1 public static int BinarySearch(int[] array,
2                               int N, int a) {
3     int lo = 0;
4     int hi = N-1;
5     // a might be in interval [lo,hi] while lo <= hi
6     while(lo <= hi) {
7         int mid = (lo + hi) / 2;
8         // if a > elem in mid of interval,
9         // search the right subinterval
10        if(array[mid] < a)
11            lo = mid+1;
12        // else if a < elem in mid of interval,
13        // search the left subinterval
14        else if(array[mid] > a)
15            hi = mid-1;
16        // else a is found
17        else
18            return mid;
19    }
20    // array does not contain a
21    return -1;
22 }

```

MD5: 203da61f7a381564ce3515f674fa82a4 | $\mathcal{O}(\log n)$

5.2 Next number with n bits set

From x the smallest number greater than x with the same amount of bits set is computed. Little changes have to be made, if the calculated number has to have length less than 32 bits.

Input: number x with n bits set ($x = (1 << n) - 1$)

Output: the smallest number greater than x with n bits set

```

1 public static int nextNumber(int x) {
2     //break when larger than limit here
3     if(x == 0) return 0;
4     int smallest = x & -x;
5     int ripple = x + smallest;
6     int new_smallest = ripple & -ripple;
7     int ones = ((new_smallest/smallest) >> 1) - 1;
8     return ripple | ones;
9 }

```

MD5: 2d8a79cb551648e67fc3f2f611a4f63c | $\mathcal{O}(1)$

5.3 Next Permutation

Returns true if there is another permutation. Can also be used to compute the nextPermutation of an array.

Input: String a as char array

Output: true, if there is a next permutation of a , false otherwise

```

1 public static boolean nextPermutation(char[] a) {
2     int i = a.length - 1;
3     while(i > 0 && a[i-1] >= a[i])
4         i--;
5     if(i <= 0)
6         return false;
7     int j = a.length - 1;
8     while (a[j] <= a[i-1])
9         j--;
10    char tmp = a[i - 1];
11    a[i - 1] = a[j];
12    a[j] = tmp;
13
14    j = a.length - 1;
15    while(i < j) {
16        tmp = a[i];
17        a[i] = a[j];
18        a[j] = tmp;
19        i++;
20        j--;
21    }
22    return true;
23 }
```

MD5: 7d1fe65d3e77616dd2986ce6f2af089b | $\mathcal{O}(n)$

5.4 Greedy-Scheduling

```

1 public class ebox {
2
3     public static void main(String[] args) {
4         Scanner s = new Scanner(System.in);
5         int n = s.nextInt();
6         int k = s.nextInt();
7         Show[] S = new Show[n];
8         for(int i = 0; i < n; i++) {
9             Show cur = new Show(s.nextInt(), s.nextInt());
10            S[i] = cur;
11        }
12        Arrays.sort(S);
13        TreeSet<Band> t = new TreeSet<Band>();
14        for(int i = 0; i < k; i++) {
15            t.add(new Band(0, i));
16        }
17        int sum = 0;
18        for(int i = 0; i < n; i++) {
19            Band cmp = new Band(S[i].s, Integer.MAX_VALUE);
20            Band rm = t.floor(cmp);
21            if(rm == null) continue;
22            int id = rm.id;
23            t.remove(rm);
24            t.add(new Band(S[i].f, id));
25            sum++;
26        }
27        System.out.println(sum);
28    }
29 }
30 }
```

```

31 class Show implements Comparable<Show> {
32
33     int s;
34     int f;
35
36     public Show(int s, int f) {
37         this.s = s;
38         this.f = f;
39     }
40
41     public int compareTo(Show o) {
42         if(Integer.valueOf(this.f).compareTo(Integer.valueOf(o.f)) != 0) {
43             return Integer.valueOf(this.f).compareTo(Integer.valueOf(o.f));
44         } else {
45             return Integer.valueOf(this.s).compareTo(Integer.valueOf(o.s));
46         }
47     }
48 }
49
50 class Band implements Comparable<Band> {
51
52     int lt;
53     int id;
54
55     public Band(int lt, int id) {
56         this.lt = lt;
57         this.id = id;
58     }
59
60     public int compareTo(Band o) {
61         if(Integer.valueOf(this.lt).compareTo(Integer.valueOf(o.lt)) != 0) {
62             return Integer.valueOf(this.lt).compareTo(Integer.valueOf(o.lt));
63         } else {
64             return Integer.valueOf(this.id).compareTo(Integer.valueOf(o.id));
65         }
66     }
67 }
```

MD5: 3269c711c682fc93f2c3837d2c755714 | $\mathcal{O}(?)$

5.5 comparator in C++

```

1 bool myfunction (int i, int j) {return (i<j); }
2
3 int main() {
4     vector<int> vec;
5     sort(vec.begin(), vec.end(), myfunction);
6     priority_queue<int, vector<int>, decltype(myfunction)> *> pq(myfunction);
7 }
```

MD5: f4beb6e197be08977fd4f74b2537ae09 | $\mathcal{O}(?)$

5.6 Mo's algorithm

Works for queries on intervals. Sort queries and add, remove on borders in $\mathcal{O}(1)$. Thus only usable when this is possible for the task.

```

1 #include<vector>
2 #include<utility>
3 #include<algorithm>
4
5 using namespace std;
6
7 int BLOCK_SIZE;
8 int cur_answer;
9 vector<int> lmen;
10 vector<int> lwomen;
11 vector<int> cmen;
12 vector<int> cwomen;
13
14 bool cmp(const pair<pair<int, int>, int> &i, const
    pair<pair<int, int>, int> &j) {
15     if(i.first.first / BLOCK_SIZE != j.first.first /
        BLOCK_SIZE) {
16         return i.first.first < j.first.first;
17     }
18     return i.first.second < j.first.second;
19 }
20
21 void add(int i, int j) {
22     //adds values i, j to function
23     cur_answer -= min(cmen[i], cwomen[i]);
24     cur_answer -= min(cmen[j], cwomen[j]);
25     if(i == j) cur_answer += min(cmen[j], cwomen[j]);
26     ++cmen[i];
27     ++cwomen[j];
28     cur_answer += min(cmen[i], cwomen[i]);
29     cur_answer += min(cmen[j], cwomen[j]);
30     if(i == j) cur_answer -= min(cmen[j], cwomen[j]);
31 }
32
33 void remove(int i, int j) {
34     //removes values i, j from function
35     cur_answer -= min(cmen[i], cwomen[i]);
36     cur_answer -= min(cmen[j], cwomen[j]);
37     if(i == j) cur_answer += min(cmen[j], cwomen[j]);
38     --cmen[i];
39     --cwomen[j];
40     cur_answer += min(cmen[i], cwomen[i]);
41     cur_answer += min(cmen[j], cwomen[j]);
42     if(i == j) cur_answer -= min(cmen[j], cwomen[j]);
43 }
44
45 int main()
46 {
47     int N, M, K;
48     cin >> N >> M >> K;
49     lmen.resize(N);
50     lwomen.resize(N);
51     cmen.resize(K);
52     cwomen.resize(K);
53     BLOCK_SIZE = static_cast<int>(sqrt(N));
54     vector<pair<pair<int, int>, int>> queries(M);
55     vector<int> answers(M);
56     for(int i = 0; i < N; ++i) {
57         cin >> lmen[i];
58     }
59     for(int i = 0; i < N; ++i) {
60         cin >> lwomen[i];
61     }
62     for(int i = 0; i < M; ++i) {
63         cin >> queries[i].first.first >> queries[i].
            first.second;
64         queries[i].second = i;

```

```

65     }
66     //sort the queries into buckets
67     sort(queries.begin(), queries.end(), cmp);
68     int mo_left = 0, mo_right = -1;
69     for(int i = 0; i < M; ++i) {
70         int left = queries[i].first.first;
71         int right = queries[i].first.second;
72         while(mo_right < right) {
73             ++mo_right;
74             add(lmen[mo_right], lwomen[mo_right]);
75         }
76         while(mo_right > right) {
77             remove(lmen[mo_right], lwomen[mo_right]);
78             --mo_right;
79         }
80         while(mo_left < left) {
81             remove(lmen[mo_left], lwomen[mo_left]);
82             ++mo_left;
83         }
84         while(mo_left > left) {
85             --mo_left;
86             add(lmen[mo_left], lwomen[mo_left]);
87         }
88         answers[queries[i].second] = cur_answer;
89     }
90     for(int i = 0; i < M; ++i) {
91         cout << answers[i] << endl;
92     }
93 }

```

MD5: a7af72b67f95a76818d1dabadf4f9e5c | $\mathcal{O}(?)$

6 String

6.1 Knuth-Morris-Pratt

Input: String s to be searched, String w to search for.

Output: Array with all starting positions of matches

```

1 public static ArrayList<Integer> kmp(String s, String
    w) {
2     ArrayList<Integer> ret = new ArrayList<>();
3     //Build prefix table
4     int[] N = new int[w.length()+1];
5     int i=0; int j = -1; N[0]=-1;
6     while (i<w.length()) {
7         while (j>=0 && w.charAt(j) != w.charAt(i))
8             j = N[j];
9         i++; j++; N[i]=j;
10    }
11    //Search string
12    i=0; j=0;
13    while (i<s.length()) {
14        while (j>=0 && s.charAt(i) != w.charAt(j))
15            j = N[j];
16        i++; j++;
17        if (j==w.length()) { //match found
18            ret.add(i-w.length()); //add its start index
19            j = N[j];
20        }
21    }
22    return ret;
23 }

```

MD5: 3cb03964744db3b14b9bff265751c84b | $\mathcal{O}(n + m)$

6.2 Levenshtein Distance

Calculates the Levenshtein distance for two strings (minimum number of insertions, deletions, or substitutions).

Input: A string *a* and a string *b*.

Output: An integer holding the distance.

```

1 public static int levenshteinDistance(String a, String
2     b) {
3     a = a.toLowerCase();
4     b = b.toLowerCase();
5     int[] costs = new int[b.length() + 1];
6     for (int j = 0; j < costs.length; j++)
7         costs[j] = j;
8
9     for (int i = 1; i <= a.length(); i++) {
10        costs[0] = i;
11        int nw = i - 1;
12        for (int j = 1; j <= b.length(); j++) {
13            int cj = Math.min(1 + Math.min(costs[j], costs[j
14                - 1]),
15                a.charAt(i - 1) == b.charAt(j - 1) ? nw : nw +
16                1);
17            nw = costs[j];
18            costs[j] = cj;
19        }
20    }
21    return costs[b.length()];
22 }
```

MD5: 79186003b792bc7fd5c1ffbbcfcb2b1c6 | $\mathcal{O}(|a| \cdot |b|)$

6.3 Longest Common Subsequence

Finds the longest common subsequence of two strings.

Input: Two strings *string1* and *string2*.

Output: The LCS as a string.

```

1 public static String longestCommonSubsequence(String
2     string1, String string2) {
3     char[] s1 = string1.toCharArray();
4     char[] s2 = string2.toCharArray();
5     int[][] num = new int[s1.length + 1][s2.length + 1];
6     // Actual algorithm
7     for (int i = 1; i <= s1.length; i++)
8         for (int j = 1; j <= s2.length; j++)
9             if (s1[i - 1] == s2[j - 1])
10                num[i][j] = 1 + num[i - 1][j - 1];
11            else
12                num[i][j] = Math.max(num[i - 1][j], num[i][j -
13                    1]);
14    // System.out.println("length of LCS = " + num[s1.
15        length][s2.length]);
16    int s1position = s1.length, s2position = s2.length;
17    List<Character> result = new LinkedList<Character>()
18        ;
19    while (s1position != 0 && s2position != 0) {
20        if (s1[s1position - 1] == s2[s2position - 1]) {
21            result.add(s1[s1position - 1]);
22            s1position--;
23            s2position--;
24        } else if (num[s1position][s2position - 1] >= num[
25            s1position][s2position])
26            s2position--;
27        else
28            s1position--;
29    }
```

```

23     s1position--;
24 }
25 Collections.reverse(result);
26 char[] resultString = new char[result.size()];
27 int i = 0;
28 for (Character c : result) {
29     resultString[i] = c;
30     i++;
31 }
32 return new String(resultString);
33 }
```

MD5: 4dc4ee3af14306bea5724ba8a859d5d4 | $\mathcal{O}(n \cdot m)$

6.4 Longest common substring

gets two String and finds all LCSs and returns them in a set

```

1 public static TreeSet<String> LCS(String a, String b)
2 {
3     int[][] t = new int[a.length()+1][b.length()+1];
4     for(int i = 0; i <= b.length(); i++)
5         t[0][i] = 0;
6
7     for(int i = 0; i <= a.length(); i++)
8         t[i][0] = 0;
9
10    for(int i = 1; i <= a.length(); i++)
11        for(int j = 1; j <= b.length(); j++)
12            if(a.charAt(i-1) == b.charAt(j-1))
13                t[i][j] = t[i-1][j-1] + 1;
14            else
15                t[i][j] = 0;
16    int max = -1;
17    for(int i = 0; i <= a.length(); i++)
18        for(int j = 0; j <= b.length(); j++)
19            if(max < t[i][j])
20                max = t[i][j];
21    if(max == 0 || max == -1)
22        return new TreeSet<String>();
23    TreeSet<String> res = new TreeSet<String>();
24    for(int i = 0; i <= a.length(); i++)
25        for(int j = 0; j <= b.length(); j++)
26            if(max == t[i][j])
27                res.add(a.substring(i-max, i));
28    return res;
29 }
```

MD5: 9de393461e1faebe99af3ff8db380bde | $\mathcal{O}(|a| * |b|)$

7 Math

7.1 Tree

Diameter: BFS from any node, then BFS from last visited node.

Max dist is then the diameter. Center: Middle vertex in second step from above.

7.2 Divisability Explanation

$D \mid M \Leftrightarrow D \mid \text{digit_sum}(M, k, \text{alt})$, refer to table for values of D, k, alt .

7.3 Combinatorics

- Variations (ordered): k out of n objects (permutations for $k = n$)
 - without repetition:
 $M = \{(x_1, \dots, x_k) : 1 \leq x_i \leq n, x_i \neq x_j \text{ if } i \neq j\}$,
 $|M| = \frac{n!}{(n-k)!}$
 - with repetition:
 $M = \{(x_1, \dots, x_k) : 1 \leq x_i \leq n\}$, $|M| = n^k$
- Combinations (unordered): k out of n objects
 - without repetition: $M = \{(x_1, \dots, x_n) : x_i \in \{0, 1\}, x_1 + \dots + x_n = k\}$, $|M| = \binom{n}{k}$
 - with repetition: $M = \{(x_1, \dots, x_n) : x_i \in \{0, 1, \dots, k\}, x_1 + \dots + x_n = k\}$, $|M| = \binom{n+k-1}{k}$
- Ordered partition of numbers: $x_1 + \dots + x_k = n$ (i.e. $1+3 = 3+1 = 4$ are counted as 2 solutions)
 - #Solutions for $x_i \in \mathbb{N}_0$: $\binom{n+k-1}{k-1}$
 - #Solutions for $x_i \in \mathbb{N}$: $\binom{n-1}{k-1}$
- Unordered partition of numbers: $x_1 + \dots + x_k = n$ (i.e. $1+3 = 3+1 = 4$ are counted as 1 solution)
 - #Solutions for $x_i \in \mathbb{N}$: $P_{n,k} = P_{n-k,k} + P_{n-1,k-1}$
 where $P_{n,1} = P_{n,n} = 1$
- Derangements (permutations without fixed points): $!n = n! \sum_{k=0}^n \frac{(-1)^k}{k!} = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$

7.4 Polynomial Interpolation

7.4.1 Theory

Problem: for $\{(x_0, y_0), \dots, (x_n, y_n)\}$ find $p \in \Pi_n$ with $p(x_i) = y_i$ for all $i = 0, \dots, n$.

Solution: $p(x) = \sum_{i=0}^n \gamma_{0,i} \prod_{j=0}^{i-1} (x - x_j)$ where $\gamma_{j,k} = y_j$ for $k = 0$

and $\gamma_{j,k} = \frac{\gamma_{j+1,k-1} - \gamma_{j,k-1}}{x_{j+k} - x_j}$ otherwise.

Efficient evaluation of $p(x)$: $b_n = \gamma_{0,n}$, $b_i = b_{i+1}(x - x_i) + \gamma_{0,i}$ for $i = n-1, \dots, 0$ with $b_0 = p(x)$.

7.5 Fibonacci Sequence

7.5.1 Binet's formula

$$\begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow f_n = \frac{1}{\sqrt{5}}(\phi^n - \tilde{\phi}^n) \text{ where } \phi = \frac{1+\sqrt{5}}{2} \text{ and } \tilde{\phi} = \frac{1-\sqrt{5}}{2}.$$

7.5.2 Generalization

$$g_n = \frac{1}{\sqrt{5}}(g_0(\phi^{n-1} - \tilde{\phi}^{n-1}) + g_1(\phi^n - \tilde{\phi}^n)) = g_0 f_{n-1} + g_1 f_n$$

for all $g_0, g_1 \in \mathbb{N}_0$

7.5.3 Pisano Period

Both $(f_n \bmod k)_{n \in \mathbb{N}_0}$ and $(g_n \bmod k)_{n \in \mathbb{N}_0}$ are periodic.

7.6 Reihen

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{i=0}^n c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1, \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^n c^i = \frac{c}{1-c}, |c| < 1$$

$$\sum_{i=0}^n i c^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, c \neq 1, \sum_{i=0}^{\infty} i c^i = \frac{c}{(1-c)^2}, |c| < 1$$

7.7 Binomialkoeffizienten

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}, \binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}$$

7.8 Catalanzahlen

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, C_{n+1} = \sum_{k=0}^n C_k C_{n-k}, C_{n+1} = \frac{4n+2}{n+2} C_n$$

7.9 Geometrie

Polygonfläche: $A = \frac{1}{2}(x_1 y_2 - x_2 y_1 + x_2 y_3 - x_3 y_2 + \dots + x_{n-1} y_n - x_n y_{n-1} + x_n y_1 - x_1 y_n)$

7.10 Zahlentheorie

Chinese Remainder Theorem: Es existiert eine Zahl C , sodass:
 $C \equiv a_1 \pmod{n_1}, \dots, C \equiv a_k \pmod{n_k}, \text{ggT}(n_i, n_j) = 1, i \neq j$
 Fall $k = 2$: $m_1 n_1 + m_2 n_2 = 1$ mit EEA finden.

Lösung ist $x = a_1 m_2 n_2 + a_2 m_1 n_1$.

Allgemeiner Fall: iterative Anwendung von $k = 2$

Eulersche φ -Funktion: $\varphi(n) = n \prod_{p|n} (1 - \frac{1}{p})$, p prim

$\varphi(p) = p - 1, \varphi(pq) = \varphi(p)\varphi(q), p, q$ prim

$\varphi(p^k) = p^k - p^{k-1}, p, q$ prim, $k \geq 1$

Eulers Theorem: $a^{\varphi(n)} \equiv 1 \pmod{n}$

Fermats Theorem: $a^p \equiv a \pmod{p}, p$ prim

7.11 Faltung

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) = \sum_{m=-\infty}^{\infty} f(n-m)g(m)$$

8 Java Knowhow

8.1 System.out.printf() und String.format()

Syntax: %[flags][width][.precision][conv]

flags:

- left-justify (default: right)
- + always output number sign
- 0 zero-pad numbers
- (space) space instead of minus for pos. numbers
- , group triplets of digits with ,

width specifies output width

precision is for floating point precision

conv:
d byte, short, int, long
f float, double
c char (use C for uppercase)
s String (use S for all uppercase)

8.2 Modulo: Avoiding negative Integers

```
1 int mod = (((nums[j] % D) + D) % D);
```

8.3 Speed up IO

Use

1 BufferedReader br = new BufferedReader(new
2 InputStreamReader(System.in));

Use

1 Double.parseDouble(Scanner.next());