Team Contest Reference
Team: Romath

Roland Haase
Thore Tiemann
Marcel Wienöbst

Contents

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 …23.767
`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 containig a sequence of length N
Output: lenght of the longest increasing subsequence in arr

```
// This has not been tested yet
// (adapted from tested C++ Murcia Code)
public static int LISeasy(int[] arr, int N) {
  int[] m = new int[N];
  for (int i = N - 1; i >= 0; i--) {
    m[i] = 1; //init table
    for (int j = i + 1; j < N; j++) {
      // if arr[i] increases the length
      // of subsequence from array[j]
      if (arr[j] > arr[i])
        if (m[i] < m[j] + 1)
          // store lenght of new subseq
          m[i] = m[j] + 1;
    }
  }
  // find max in array
```

```
  int longest = 0;
  for (int i = 0; i < N; i++) {
    if (m[i] > longest)
      longest = m[i];
  }
  return longest;
}
```

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

```
public static int[] LISfast(int[] arr, int[] p) {
  // p[k] stores index of the predecessor of arr[k]
  // in the LIS ending at arr[k]
  // m[j] stores index k of smallest value arr[k]
  // so there is a LIS of length j ending at arr[k]
  int[] m = new int[arr.length+1];
  int l = 0;
  for(int i = 0; i < arr.length; i++) {
    // bin search for the largest positive j <= l
    // with arr[m[j]] < arr[i]
    int lo = 1;
    int hi = l;
    while(lo <= hi) {
      int mid = (int) (((lo + hi) / 2.0) + 0.6);
      if(arr[m[mid]] <= arr[i])
        lo = mid+1;
      else
        hi = mid-1;
    }
    // lo is 1 greater than length of the
    // longest prefix of arr[i]
    int newL = lo;
    p[i] = m[newL-1];
    m[newL] = i;
    // if LIS found is longer than the ones
    // found before, then update l
    if(newL > l)
      l = newL;
  }
  // reconstruct the LIS
  int[] s = new int[l];
  int k = m[l];
  for(int i= l-1; i>= 0; i--) {
    s[i] = arr[k];
    k = p[k];
  }
  return s;
}
```

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

2 DataStructures

2.1 Fenwick-Tree

Can be used for computing prefix sums.

```
//note that 0 can not be used
int[] fwktree = new int[m + n + 1];
public static int read(int index, int[] fenwickTree) {
   int sum = 0;
   while (index > 0) {
      sum += fenwickTree[index];
      index -= (index & -index);
   }
   return sum;
}
public static int[] update(int index, int addValue,
     int[] fenwickTree) {
   while (index <= fenwickTree.length - 1) {
      fenwickTree[index] += addValue;
      index += (index & -index);
   }
   return fenwickTree;
}
```

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

2.2 Range Maximum Query

$process$ processes an array A of length N in 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 $[i, j]$

```
public static void process(int[][] M, int[] A, int N)
    {
  for(int i = 0; i < N; i++)
    M[i][0] = i;
  // filling table M
  // M[i][j] = max(M[i][j-1], M[i+(1<<(j-1))][j-1]),
  // cause interval of length 2^j can be partitioned
  // into two intervals of length 2^(j-1)
  for(int j = 1; 1 << j <= N; j++) {
    for(int i = 0; i + (1 << j) - 1 < N; i++) {
      if(A[M[i][j-1]] >= A[M[i+(1 << (j-1))][j-1]])
        M[i][j] = M[i][j-1];
      else
        M[i][j] = M[i + (1 << (j-1))][j-1];
    }
  }
}

public static int query(int[][] M, int[] A, int N,
                                      int i, int j) {
  // k = |_ log_2(j-i+1) _|
  int k = (int) (Math.log(j - i + 1) / Math.log(2));
  if(A[M[i][k]] >= A[M[j- (1 << k) + 1][k]])
    return M[i][k];
  else
    return M[j - (1 << k) + 1][k];
}
```

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

2.3 Trie

```java
public static boolean insert(TrieNode root, String
    word){
  char[] s = word.toCharArray();
  TrieNode node = root;

  for(int i = 0; i < s.length; ++i){
    int index = charToIndex(s[i]);
    if(node.children[index] == null){
      node.children[index] = new TrieNode(node);
    }
    node = node.children[index];
  }
  node.isEnd = true;

  return true;
}

public static boolean search(TrieNode root, String
    word){
  char[] s = word.toCharArray();
  TrieNode node = root;

  for(int i = 0; i < s.length; ++i){
    int index = charToIndex(s[i]);
    if(node.children[index] == null){
      return false;
    }
    node = node.children[index];
  }

  return node.isEnd;
}

public static int charToIndex(char c){
  return ((int) c - (int) a);
}

static class TrieNode{

  boolean isEnd;
  TrieNode[] children;

  public TrieNode(){
    isEnd = false;
    children = new TrieNode[26];
  }
}
```

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.

```java
class UnionFind {
  private int[] p = null;
  private int[] r = null;
  private int count = 0;

  public int count() {
    return count;
  } // number of sets

  public UnionFind(int n) {
    count = n; // every node is its own set
    r = new int[n]; // every node is its own tree with
        height 0
    p = new int[n];
    for (int i = 0; i < n; i++)
      p[i] = -1; // no parent = -1
  }

  public int find(int x) {
    int root = x;
    while (p[root] >= 0) { // find root
      root = p[root];
    }
    while (p[x] >= 0) { // path compression
      int tmp = p[x];
      p[x] = root;
      x = tmp;
    }
    return root;
  }

  // return true, if sets merged and false, if already
       from same set
  public boolean union(int x, int y) {
    int px = find(x);
    int py = find(y);
    if (px == py)
      return false; // same set -> reject edge
    if (r[px] < r[py]) { // swap so that always h[px
        ]>=h[py]
      int tmp = px;
      px = py;
      py = tmp;
    }
    p[py] = px; // hang flatter tree as child of
        higher tree
    r[px] = Math.max(r[px], r[py] + 1); // update (
        worst-case) height
    count--;
    return true;
  }
}
```

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

2.5 Suffix array

```cpp
#include<vector>
#include<string>
#include<algorithm>

using namespace std;

vector<int> sa, pos, tmp, lcp;
string s;
int N, gap;

bool sufCmp(int i, int j) {
  if(pos[i] != pos[j])
    return pos[i] < pos[j];
  i += gap;
```

```
  j += gap;
  return (i < N && j < N) ? pos[i] < pos[j] : i > j;
}

void buildSA()
{
  N = s.size();
  for(int i = 0; i < N; ++i) {
    sa.push_back(i);
    pos.push_back(s[i]);
  }
  tmp.resize(N);
  for(gap = 1;;gap *= 2) {
    sort(sa.begin(), sa.end(), sufCmp);
    for(int i = 0; i < N - 1; ++i) {
      tmp[i+1] = tmp[i] + sufCmp(sa[i], sa[i+1]);
    }
    for(int i = 0; i < N; ++i) {
      pos[sa[i]] = tmp[i];
    }
    if(tmp[N-1] == N-1) break;
  }
}

void buildLCP()
{
  lcp.resize(N);
  for(int i = 0, k = 0; i < N; ++i) {
    if(pos[i] != N - 1) {
      for(int j = sa[pos[i] + 1]; s[i + k] == s[j + k];) {
        ++k;
      }
      lcp[pos[i]] = k;
      if (k) --k;
    }
  }
}

int main()
{
  string r, t;
  cin >> r >> t;
  s = r + "§" + t;
  buildSA();
  buildLCP();
  for(int i = 0; i < N; ++i) {
    cout << sa[i] << "␣" << lcp[i] << endl;
  }
  int mx = 0, mxi = -1;
  for(int i = 0; i+1 < s.size(); ++i) {
    bool a_in_s = sa[i] < r.size(), b_in_s = sa[i+1] < r.size();
    if(a_in_s != b_in_s) {
      int l = lcp[i];
      if(l > mx) {
        mx = l;
        mxi = sa[i];
      }
    }
  }
  cout << mx << endl;
  cout << s.substr(mxi, mx) << endl;
}
```

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

3 Graph

3.1 2SAT

```
//We assume that ind(not a) = ind(a) + N, with N being the number of variables
//could however be changed easily
public static boolean 2SAT(Vertex[] G) {
  //call SCC
  double DFS(G);
  //check for contradiction
  boolean poss = true;
  for(int i = 0; i < S+A; i++) {
    if(G[i].comp == G[i + (S+A)].comp) {
      poss = false;
    }
  }
  return poss;
}
```

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

```
public static boolean BFS(Vertex[] G, int s, int t) {
  //make sure that Vertices vis values are false etc
  Queue<Vertex> q = new LinkedList<Vertex>();
  G[s].vis = true;
  G[s].dist = 0;
  G[s].pre = -1;
  q.add(G[s]);
  //expand frontier between undiscovered and discovered vertices
  while(!q.isEmpty()) {
    Vertex u = q.poll();
    //when reaching the goal, return true
    //if we want to construct a BFS-tree delete this line
    if(u.id = t) return true;
    //else add adj vertices if not visited
    for(Vertex v : u.adj) {
      if(!v.vis) {
        v.vis = true;
        v.dist = u.dist + 1;
        v.pre = u.id;
        q.add(v);
      }
    }
  }
  //did not find target
  return false;
}
```

MD5: `71f3fa48b4f1b2abdff3557a27a9a136` | $\mathcal{O}(|V| + |E|)$

3.3 BellmanFord

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

```java
public static boolean bellmanFord(Vertex[] G) {
  //source is 0
  G[0].dist = 0;
  //calc distances
  //the path has max length |V|-1
  for(int i = 0; i < G.length-1; i++) {
    //each iteration relax all edges
    for(int j = 0; j < G.length; j++) {
      for(Edge e : G[j].adj) {
        if(G[j].dist != Integer.MAX_VALUE
        && e.t.dist > G[j].dist + e.w) {
          e.t.dist = G[j].dist + e.w;
        }
      }
    }
  }
  //check for negative-length cycle
  for(int i = 0; i < G.length; i++) {
    for(Edge e : G[i].adj) {
      if(G[i].dist != Integer.MAX_VALUE
          && e.t.dist > G[i].dist + e.w) {
        return true;
      }
    }
  }
  return false;
}
```

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

```java
public static boolean bipartiteGraphCheck(Vertex[] G){
  // use bfs for coloring each node
  G[0].color = 1;
  Queue<Vertex> q = new LinkedList<Vertex>();
  q.add(G[0]);
  while(!q.isEmpty()) {
    Vertex u = q.poll();
    for(Vertex v : u.adj) {
      // if node i not yet visited,
      // give opposite color of parent node u
      if(v.color == -1) {
        v.color = 1-u.color;
        q.add(v);
      // if node i has same color as parent node u
      // the graph is not bipartite
      } else if(u.color == v.color)
        return false;
      // if node i has different color
      // than parent node u keep going
    }
  }
  return true;
}
```

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.)

```java
// A DFS based recursive function that returns true
// if a matching for vertex u is possible
boolean bpm(boolean bpGraph[][], int u,
            boolean seen[], int matchR[]) {
  // Try every job one by one
  for (int v = 0; v < N; v++) {
    // If applicant u is interested in job v and v
    // is not visited
    if (bpGraph[u][v] && !seen[v]) {
      seen[v] = true; // Mark v as visited

      // If job v is not assigned to an applicant OR
      // previously assigned applicant for job v
      // (which is matchR[v]) has an alternate job
      // available. Since v is marked as visited in
      // the above line, matchR[v] in the following
      // recursive call will not get job v again
      if (matchR[v] < 0 ||
      bpm(bpGraph, matchR[v], seen, matchR)) {
        matchR[v] = u;
        return true;
      }
    }
  }
  return false;
}

// Returns maximum number of matching from M to N
int maxBPM(boolean bpGraph[][]) {
  // An array to keep track of the applicants assigned
  // to jobs. The value of matchR[i] is the applicant
  // number assigned to job i, the value -1 indicates
  // nobody is assigned.
  int matchR[] = new int[N];
  // Initially all jobs are available
  for(int i = 0; i < N; ++i)
    matchR[i] = -1;
  // Count of jobs assigned to applicants
  int result = 0;
  for (int u = 0; u < M; u++) {
    // Mark all jobs as not seen for next applicant.
    boolean seen[] = new boolean[N];
    for(int i = 0; i < N; ++i)
      seen[i] = false;
    // Find if the applicant u can get a job
    if (bpm(bpGraph, u, seen, matchR))
      result++;
  }
  return result;
}
```

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

```
public static double bitonic(double[][] d) {
  int N = d.length;
  double[][] B = new double[N][N];
  for (int j = 0; j < N; j++) {
    for (int i = 0; i <= j; i++) {
      if (i < j - 1)
        B[i][j] = B[i][j - 1] + d[j - 1][j];
      else {
        double min = 0;
        for (int k = 0; k < j; k++) {
          double r = B[k][i] + d[k][j];
          if (min > r || k == 0)
            min = r;
        }
        B[i][j] = min;
      }
    }
  }
  return B[N-1][N-1];
}
```

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

```
public static void dagSSP(Vertex[] G, int s) {
  //calls topological sort method
  LinkedList<Integer> sorting = TS(G);
  G[s].dist = 0;
  //go through vertices in ts order
  for(int u : sorting) {
    for(Edge e : G[u].adj) {
      Vertex v = e.t;
      if(v.dist > u.dist + e.w) {
        v.dist = u.dist + e.w;
        v.pre = u.id;
      }
    }
  }
}
```

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 predcessor vertices set.

```
public static void dijkstra(Vertex[] G, int s) {
  G[s].dist = 0;
  Tuple st = new Tuple(s, 0);
  PriorityQueue<Tuple> q = new PriorityQueue<Tuple>();
  q.add(st);

  while(!q.isEmpty()) {
    Tuple sm = q.poll();
    Vertex u = G[sm.id];
    //this checks if the Tuple is still useful, both
        checks should be equivalent
    if(u.vis || sm.dist > u.dist) continue;
    u.vis = true;
    for(Edge e : u.adj) {
      Vertex v = e.t;
      if(!v.vis && v.dist > u.dist + e.w) {
        v.pre = u.id;
        v.dist = u.dist + e.w;
        Tuple nt = new Tuple(v.id, v.dist);
        q.add(nt);
      }
    }
  }
}
```

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

3.9 EdmondsKarp

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

```
public static boolean BFS(Vertex[] G, int s, int t) {
  int N = G.length;
  for(int i = 0; i < N; i++) {
    G[i].vis = false;
  }

  Queue<Vertex> q = new LinkedList<Vertex>();
  G[s].vis = true;
  G[s].pre = -1;
  q.add(G[s]);

  while(!q.isEmpty()) {
    Vertex u = q.poll();
    if(u.id == t) return true;
    for(int i : u.adj.keySet()) {
      Edge e = u.adj.get(i);
      Vertex v = e.t;
      if(!v.vis && e.rw > 0) {
        v.vis = true;
        v.pre = u.id;
        q.add(v);
      }
    }
  }
  return (G[t].vis);
}
//We store the edges in the graph in a hashmap
public static int edKarp(Vertex[] G, int s, int t) {
  int maxflow = 0;
  while(BFS(G, s, t)) {
    int pflow = Integer.MAX_VALUE;
    for(int v = t; v!= s; v = G[v].pre) {
      int u = G[v].pre;
      pflow = Math.min(pflow, G[u].adj.get(v).rw);
    }
    for(int v = t; v != s; v = G[v].pre) {
      int u = G[v].pre;
      G[u].adj.get(v).rw -= pflow;
      G[v].adj.get(u).rw += pflow;
    }
```

```
    maxflow += pflow;
  }
  return maxflow;
}
```

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.

```
//for Kruskal we need to sort edges, use: java.lang.
    Comparable
class Edge implements Comparable<Edge> {}

class Edge {
  //for Kruskal it is helpful to store the start as
  //well, moreover we might not need the vertex class
  int s;
  int t;

  //for EdKarp we also want to store residual weights
  int rw;

  Vertex t;
  int w;

  public Edge(Vertex t, int w) {
    this.t = t;
    this.w = w;
    this.rw = w;
  }

  public Edge(int s, int t, int w) {...}

  public int compareTo(Edge other) {
    return Integer.compare(this.w, other.w);
  }
}
```

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

3.11 FloydWarshall

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

```
public static void floydWarshall(int[][] graph,
                                 int[][] next, int[][] ans) {
  for(int i = 0; i < ans.length; i++)
    for(int j = 0; j < ans.length; j++)
      ans[i][j] = graph[i][j];

  for (int k = 0; k < ans.length; k++)
    for (int i = 0; i < ans.length; i++)
      for (int j = 0; j < ans.length; j++)
        if (ans[i][k] + ans[k][j] < ans[i][j]
                 && ans[i][k] < Integer.MAX_VALUE
                 && ans[k][j] < Integer.MAX_VALUE) {
          ans[i][j] = ans[i][k] + ans[k][j];
          next[i][j] = next[i][k];
        }
}
```

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

3.12 Held Karp

Algorithm for TSP

```
public static int[] tsp(int[][] graph) {
  int n = graph.length;
  if(n == 1) return new int[]{0};
  //C stores the shortest distance to node of the
      second dimension, first dimension is the
      bitstring of included nodes on the way
  int[][] C = new int[1<<n][n];
  int[][] p = new int[1<<n][n];
  //initialize
  for(int k = 1; k < n; k++) {
    C[1<<k][k] = graph[0][k];
  }
  for(int s = 2; s < n; s++) {
    for(int S = 1; S < (1<<n); S++) {
      if(Integer.bitCount(S)!=s || (S&1) == 1)
        continue;
      for(int k = 1; k < n; k++) {
        if((S & (1 << k)) == 0) continue;

        //Smk is the set of nodes without k
        int Smk = S ^ (1<<k);

        int min = Integer.MAX_VALUE;
        int minprev = 0;
        for(int  m=1; m<n; m++) {
          if((Smk & (1<<m)) == 0) continue;
          //distance to m with the nodes in Smk +
              connection from m to k
          int tmp = C[Smk][m] +graph[m][k];
          if(tmp < min) {
            min = tmp;
            minprev = m;
          }
        }
        C[S][k] = min;
        p[S][k] = minprev;
      }
    }
  }

  //find shortest tour length
  int min = Integer.MAX_VALUE;
  int minprev = -1;
  for(int k = 1; k < n; k++) {
    //Set of all nodes except for the first + cost
        from 0 to k
    int tmp = C[(1<<n) - 2][k] + graph[0][k];
    if(tmp < min) {
      min = tmp;
      minprev = k;
    }
  }

  //Note that the tour has not been tested yet, only
      the correctness of the min-tour-value backtrack
      tour
  int[] tour = new int[n+1];
  tour[n] = 0;
  tour[n-1] = minprev;
  int bits = (1<<n)-2;
  for(int k = n-2; k>0; k--) {
    tour[k] = p[bits][tour[k+1]];
    bits = bits ^ (1<<tour[k+1]);
  }
  tour[0] = 0;
```

```
  return tour;
}
```

MD5: `f3e9730287dcbf2695bf7372fc4bafe0` | $\mathcal{O}(2^n n^2)$

3.13 Iterative DFS

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

```
//if we want to start the DFS for different connected
    components, there is such a method in the
    recursive variant of DFS
public static boolean ItDFS(Vertex[] G, int s, int t){
  //take care that all the nodes are not visited at
      the beginning
  Stack<Integer> S = new Stack<Integer>();
  s.push(s);
  while(!S.isEmpty()) {
    int u = S.pop();
    if(u.id == t) return true;
    if(!G[u].vis) {
      G[u].vis = true;
      for(Vertex v : G[u].adj) {
        if(!v.vis)
          S.push(v.id);
      }
    }
  }
  return false;
}
```

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

3.14 Johnsons Algorithm

```
public static int[][] johnson(Vertex[] G) {
  Vertex[] Gd = new Vertex[G.length+1];
  int s = G.length;
  for(int i = 0; i < G.length; i++)
    Gd[i] = G[i];
  //init new vertex with zero-weight-edges to each
      vertex
  Vertex S = new Vertex(G.length);
  for(int i = 0; i < G.length; i++)
    S.adj.add(new Edge(Gd[i], 0));
  Gd[G.length] = S;

  //bellman-ford to check for neg-weight-cycles and to
      adapt edges to enable running dijkstra
  if(bellmanFord(Gd, s)) {
    System.out.println("False");
    //this should not happen and will cause troubles
    return null;
  }
  //change weights
  for(int i = 0; i < G.length; i++)
    for(Edge e : Gd[i].adj)
      e.w = e.w + Gd[i].dist - e.t.dist;
  //store distances to invert this step later
  int[] h = new int[G.length];
  for(int i = 0; i < G.length; i++)
    h[i] = G[i].dist;

  //create shortest path matrix
  int[][] apsp = new int[G.length][G.length];

  //now use original graph G
  //start a dijkstra for each vertex
  for(int i = 0; i < G.length; i++) {
    //reset weights
    for(int j = 0; j < G.length; j++) {
      G[j].vis = false;
      G[j].dist = Integer.MAX_VALUE;
    }
    dijkstra(G, i);
    for(int j = 0; j < G.length; j++)
      apsp[i][j] = G[j].dist + h[j] - h[i];
  }
  return apsp;
}
```

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

3.15 Kruskal

Computes a minimum spanning tree for a weighted undirected graph.

```
public static int kruskal(Edge[] edges, int n) {
  Arrays.sort(edges);
  //n is the number of vertices
  UnionFind uf = new UnionFind(n);
  //we will only compute the sum of the MST, one could
      of course also store the edges
  int sum = 0;
  int cnt = 0;
  for(int i = 0; i < edges.length; i++) {
    if(cnt == n-1) break;
    if(uf.union(edges[i].s, edges[i].t)) {
      sum += edges[i].w;
      cnt++;
    }
  }
  return sum;
}
```

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

3.16 Min Cut

Calculates the min cut using Edmonds Karp algorithm.

MD5: `d41d8cd98f00b204e9800998ecf8427e` | $\mathcal{O}(?)$

3.17 Prim

```
//s is the startpoint of the algorithm, in general not
    too important; we assume that graph is connected
public static int prim(Vertex[] G, int s) {
  //make sure dists are maxint
  G[s].dist = 0;
  Tuple st = new Tuple(s, 0);

  PriorityQueue<Tuple> q = new PriorityQueue<Tuple>();
  q.add(st);
  //we will store the sum and each nodes predecessor
  int sum = 0;
```

```
  
  while(!q.isEmpty()) {
    Tuple sm = q.poll();
    Vertex u = G[sm.id];
    //u has been visited already
    if(u.vis) continue;
    //this is not the latest version of u
    if(sm.dist > u.dist) continue;
    u.vis = true;
    //u is part of the new tree and u.dist the cost of
         adding it
    sum += u.dist;
    for(Edge e : u.adj) {
      Vertex v = e.t;
      if(!v.vis && v.dist > e.w) {
        v.pre = u.id;
        v.dist = e.w;
        Tuple nt = new Tuple(v.id, e.w);
        q.add(nt);
      }
    }
  }
  return sum;
}
```

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.

```
//if we want to visit the whole graph, even if it is
    not connected we might use this
public static void DFS(Vertex[] G) {
  //make sure all vertices vis value is false etc
  int time = 0;
  for(int i = 0; i < G.length; i++) {
    if(!G[i].vis) {
      //note that we leave out t so this does not work
           with the below function
      //adaption will not be too difficult though
      //time should not always start at zero, change
          if needed
      recDFS(i, G, 0);
    }
  }
}

//first call with time = 0
public static boolean recDFS(int s, int t, Vertex[] G,
     int time){
  //it might be necessary to store the time of
      discovery
  time = time + 1;
  G[s].dtime = time;

  G[s].vis = true; //new vertex has been discovered
  //when reaching the target return true
  //not necessary when calculating the DFS-tree
  if(s == t) return true;
  for(Vertex v : G[s].adj) {
    //exploring a new edge
    if(!v.vis) {
      v.pre = u.id;
      if(recDFS(v.id, t, G)) return true;
    }
  }
  //storing finishing time
  time = time + 1;
  G[s].ftime = time;
  return false;
}
```

MD5: `3cef44fd916e1aecfb0e3eacc355e2e3` | $\mathcal{O}(|V|+|E|)$

3.19 Strongly Connected Components

```
public static void fDFS(Vertex u, LinkedList<Integer>
    sorting) {
  //compare with TS
  u.vis = true;
  for(Vertex v : u.out)
    if(!v.vis)
      fDFS(v, sorting);
  sorting.addFirst(u.id);
  return sorting;
}

public static void sDFS(Vertex u, int cnt) {
  //basic DFS, all visited vertices get cnt
  u.vis = true;
  u.comp = cnt;
  for(Vertex v : u.in)
    if(!v.vis)
      sDFS(v, cnt);
}

public static void doubleDFS(Vertex[] G) {
  //first calc a topological sort by first DFS
  LinkedList<Integer> sorting = new LinkedList<Integer
    >();
  for(int i = 0; i < G.length; i++)
    if(!G[i].vis)
      fDFS(G[i], sorting);
  for(int i = 0; i < G.length; i++)
    G[i].vis = false;
  //then go through the sort and do another DFS on G^T
  //each tree is a component and gets a unique number
  int cnt = 0;
  for(int i : sorting)
    if(!G[i].vis)
      sDFS(G[i], cnt++);
}
```

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

```
public static int suurballe(Vertex[] G, int s, int t){
  //this uses the usual dijkstra implementation with
      stored predecessors
  dijkstra(G, s);
```

```java
  //Modifying weights
  for(int i = 0; i < G.length; i++)
    for(Edge e : G[i].adj)
      e.dist = e.dist - e.t.dist + G[i].dist;
  //reversing path and storing used edges
  int old = t;
  int pre = G[t].pre;
  HashMap<Integer, Integer> hm = new HashMap<Integer,
      Integer>();
  while(pre != -1) {
    for(int i = 0; i < G[pre].adj.size(); i++) {
      if(G[pre].adj.get(i).t.id == old) {
        hm.put(pre * G.length + old, G[pre].adj.get(i)
            .tdist);
        G[pre].adj.remove(i);
        break;
      }
    }
    boolean found = false;
    for(int i = 0; i < G[old].adj.size(); i++) {
      if(G[old].adj.get(i).t.id == pre) {
        G[old].adj.get(i).dist = 0;
        found = true;
        break;
      }
    }
    if(!found)
      G[old].adj.add(new Edge(G[pre], 0));
    old = pre;
    pre = G[pre].pre;
  }
  //reset graph
  for(int i = 0; i < G.length; i++) {
    G[i].pre = -1;
    G[i].dist = Integer.MAX_VALUE;
    G[i].vis = false;
  }

  dijkstra(G, s);
  //store edges of second path
  old = t;
  pre = G[t].pre;
  while(pre != -1) {
    //store edges and remove if reverse
    for(int i = 0; i < G[pre].adj.size(); i++) {
      if(G[pre].adj.get(i).t.id == old) {
        if(!hm.containsKey(pre + old * G.length))
          hm.put(pre * G.length + old, G[pre].adj.get(
              i).tdist);
        else
          hm.remove(pre + old * G.length);
        break;
      }
    }
    old = pre;
    pre = G[pre].pre;
  }
  //sum up weights
  int sum = 0;
  for(int i : hm.keySet())
    sum += hm.get(i);
  return sum;
}
```

MD5: 222dac2a859273efbbdd0ec0d6285dd7 | $\mathcal{O}(V logV + E)$

3.21 Kahns Algorithm for TS

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

```java
public static LinkedList<Integer> TS(Vertex[] G) {
  LinkedList<Integer> sorting = new LinkedList<Integer
      >();
  PriorityQueue<Vertex> p = new PriorityQueue<Vertex
      >();
  //inc counts the number of incoming edges, if they
      are zero put the vertex in the queue
  for(int i = 0; i < G.length; i++) {
    if(G[i].inc == 0) {
      p.add(G[i]);
      G[i].vis = true;
    }
  }
  while(!p.isEmpty()) {
    Vertex u = p.poll();
    sorting.add(u.id);
    //update inc
    for(Vertex v : u.out) {
      if(v.vis) continue;
      v.inc--;
      if(v.inc == 0) {
        p.add(v);
        v.vis = true;
      }
    }
  }
  return sorting;
}
```

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

3.22 Topological Sort

```java
public static LinkedList<Integer> TS(Vertex[] G) {
  LinkedList<Integer> sorting = new LinkedList<Integer
      >();
  for(int i = 0; i < G.length; i++)
    if(!G[i].vis)
      recTS(G[i], sorting);
    //check sorting for a -1 if the graph is not
        necessarily dag
    //maybe checking if there are too many values in
        sorting is easier?!
    return sorting;
}

public static LinkedList<Integer> recTS(Vertex u,
    LinkedList<Integer> sorting) {
  u.vis = true;
  for(Vertex v : u.adj)
    if(v.vis)
      //the -1 indicates that it will not be possible
          to find an TS
      //there might be a much faster and elegant way (
          flag?!)
      sorting.addFirst(-1);
    else
      recTS(v, sorting);
  sorting.addFirst(u.id);
  return sorting;
}
```

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

3.23 Tuple

Simple tuple class used for priority queue in Dijkstra and Prim

```
class Tuple implements Comparable<Tuple> {

  int id;
  int dist;

  public Tuple(int id, int dist) {
    this.id = id;
    this.dist = dist;
  }

  public int compareTo(Tuple other) {
    return Integer.compare(this.dist, other.dist);
  }
}
```

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

3.24 Reference for Vertex classes

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

```
class Vertex {

  int id;
  boolean vis = false;
  int pre = -1;

  //for dijkstra and prim
  int dist = Integer.MAX_VALUE;

  //for SCC store number indicating the dedicated
      component
  int comp = -1;

  //for DFS we could store the start and finishing
      times
  int dtime = -1;
  int ftime = -1;

  //use an ArrayList of Edges if those information are
       needed
  ArrayList<Edge> adj = new ArrayList<Edge>();
  //use an ArrayList of Vertices else
  ArrayList<Vertex> adj = new ArrayList<Vertex>();
  //use two ArrayLists for SCC
  ArrayList<Vertex> in = new ArrayList<Vertex>();
  ArrayList<Vertex> out = new ArrayList<Vertex>();

  //for EdmondsKarp we need a HashMap to store Edges,
      Integer is target
  HashMap<Integer, Edge> adj = new HashMap<Integer,
      Edge>();

  //for bipartite graph check
  int color = -1;

  //we store as key the target
  public Vertex(int id) {
    this.id = id;
  }
}
```

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

4 Math

4.1 Binomial Coefficient

Gives binomial coefficient (n choose k)

```
public static long bin(int n, int k) {
  if (k == 0)
    return 1;
  else if (k > n/2)
    return bin(n, n-k);
  else
    return n*bin(n-1, k-1)/k;
}
```

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

4.2 Binomial Matrix

Gives binomial coefficients for all K <= N.

```
public static long[][] binomial_matrix(int N, int K) {
  long[][] B = new long[N+1][K+1];
  for (int k = 1; k <= K; k++)
    B[0][k] = 0;
  for (int m = 0; m <= N; m++)
    B[m][0] = 1;
  for (int m = 1; m <= N; m++)
    for (int k = 1; k <= K; k++)
      B[m][k] = B[m-1][k-1] + B[m-1][k];
  return B;
}
```

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

4.3 Divisability

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

```
public static long digit_sum(String M, int k, boolean
     alt) {
  long dig_sum = 0;
  int vz = 1;
  while (M.length() > k) {
    if (alt) vz *= -1;
    dig_sum += vz*Integer.parseInt(M.substring(M.
        length()-k));
    M = M.substring(0, M.length()-k);
  }
  if (alt)
    vz *= -1;
  dig_sum += vz*Integer.parseInt(M);
  return dig_sum;
}

// example: divisibility of M by 13
```

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

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

4.4 Graham Scan

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

```
public static Point[] grahamScan(Point[] points) {
  //find leftmost point with lowest y-coordinate
  int xmin = Integer.MAX_VALUE;
  int ymin = Integer.MAX_VALUE;
  int index = -1;
  for(int i = 0; i < points.length; i++) {
    if(points[i].y < ymin || (points[i].y == ymin &&
        points[i].x < xmin)) {
      xmin = points[i].x;
      ymin = points[i].y;
      index = i;
    }
  }
  //get that point to the start of the array
  Point tmp = new Point(points[index].x, points[index
      ].y);
  points[index] = points[0];
  points[0] = tmp;
  for(int i = 1; i < points.length; i++)
    points[i].src = points[0];
  Arrays.sort(points, 1, points.length);
  //for collinear points eliminate all but the
      farthest
  boolean[] isElem = new boolean[points.length];
  for(int i = 1; i < points.length-1; i++) {
    Point a = new Point(points[i].x - points[i].src.x,
         points[i].y - points[i].src.y);
    Point b = new Point(points[i+1].x - points[i+1].
        src.x, points[i+1].y - points[i+1].src.y);
    if(Calc.crossProd(a, b) == 0)
      isElem[i] = true;
  }
  //works only if there are more than three non-
      collinear points
  Stack<Point> s = new Stack<Point>();
  int i = 0;
  for(; i < 3; i++) {
    while(isElem[i++]);
    s.push(points[i]);
  }
  for(; i < points.length; i++) {
    if(isElem[i]) continue;
    while(true) {
      Point first = s.pop();
      Point second = s.pop();
      s.push(second);
      Point a = new Point(first.x - second.x, first.y
        - second.y);
      Point b = new Point(points[i].x - second.x,
        points[i].y - second.y);
      //use >= if straight angles are needed
      if(Calc.crossProd(a, b) > 0) {
        s.push(first);
        s.push(points[i]);
        break;
      }
    }
  }
  Point[] convexHull = new Point[s.size()];
  for(int j = s.size()-1; j >= 0; j--)
    convexHull[j] = s.pop();
  return convexHull;
  /*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. */
}
class Point implements Comparable<Point> {
  Point src; //set seperately in GrahamScan method
  int x;
  int y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  //might crash if one point equals src
  //major issues with multiple points on same location
      !
  public int compareTo(Point cmp) {
  Point a = new Point(this.x - src.x, this.y - src.y);
  Point b = new Point(cmp.x - src.x, cmp.y - src.y);
  //checks if points are identical
  if(a.x == b.x && a.y == b.y) return 0;
  //if same angle, sort by dist
  if(Calc.crossProd(a, b) == 0 && Calc.dotProd(a, b) >
      0)
    return Integer.compare(Calc.dotProd(a, a), Calc.
        dotProd(b, b));
  //angle of a is 0, thus b>a
  if(a.y == 0 && a.x > 0) return -1;
  //angle of b is 0, thus a>b
  if(b.y == 0 && b.x > 0) return 1;
  //a ist between 0 and 180, b between 180 and 360
  if(a.y > 0 && b.y < 0) return -1;
  if(a.y < 0 && b.y > 0) return 1;
  //return negative value if cp larger than zero
  return Integer.compare(0, Calc.crossProd(a, b));
  }
}

class Calc {
  public static int crossProd(Point p1, Point p2) {
    return p1.x * p2.y  - p2.x * p1.y;
  }
  public static int dotProd(Point p1, Point p2) {
    return p1.x * p2.x + p1.y * p2.y;
  }
}
```

MD5: 2555d858fadcfe8cb404a9c52420545d | $\mathcal{O}(N \log N)$

4.5 Iterative EEA

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

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

```java
  if (b > a) {
    long tmp = a;
    a = b;
    b = tmp;
  }
  long x = 0, y = 1, u = 1, v = 0;
  while (a != 0) {
    long q = b / a, r = b % a;
    long m = x - u * q, n = y - v * q;
    b = a; a = r; x = u; y = v; u = m; v = n;
  }
  long gcd = b;
  // x = a^-1 % b, y = b^-1 % a
  // ax + by = gcd
  long[] erg = { gcd, x, y };
  return erg;
}
```

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

4.6 Polynomial Interpolation

```java
public class interpol {

  // divided differences for points given by vectors x and y
  public static rat[] divDiff(rat[] x, rat[] y) {
    rat[] temp = y.clone();
    int n = x.length;
    rat[] res = new rat[n];
    res[0] = temp[0];
    for (int i=1; i < n; i++) {
      for (int j = 0; j < n-i; j++) {
        temp[j] = (temp[j+1].sub(temp[j])).div(x[j+i].sub(x[j]));
      }
      res[i] = temp[0];
    }
    return res;
  }

  // evaluates interpolating polynomial p at t for given
  // x-coordinates and divided differences
  public static rat p(rat t, rat[] x, rat[] dD) {
    int n = x.length;
    rat p = new rat(0);
    for (int i = n-1; i > 0; i--) {
      p = (p.add(dD[i])).mult(t.sub(x[i-1]));
    }
    p = p.add(dD[0]);
    return p;
  }
}

// implementation of rational numbers
class rat {

  public long c;
  public long d;

  public rat (long c, long d) {
    this.c = c;
    this.d = d;
    this.shorten();
  }

  public rat (long c) {
    this.c = c;
    this.d = 1;
  }

  public static long ggT(long a, long b) {
    while (b != 0) {
      long h = a%b;
      a = b;
      b = h;
    }
    return a;
  }

  public static long kgV(long a, long b) {
    return a*b/ggT(a,b);
  }

  public static rat[] commonDenominator(rat[] c) {
    long kgV = 1;
    for (int i = 0; i < c.length; i++) {
      kgV = kgV(kgV, c[i].d);
    }
    for (int i = 0; i < c.length; i++) {
      c[i].c *= kgV/c[i].d;
      c[i].d *= kgV/c[i].d;
    }
    return c;
  }

  public void shorten() {
    long ggT = ggT(this.c, this.d);
    this.c = this.c / ggT;
    this.d = this.d / ggT;
    if (d < 0) {
      this.d *= -1;
      this.c *= -1;
    }
  }

  public String toString() {
    if (this.d == 1) return ""+c;
    return ""+c+"/"+d;
  }

  public rat mult(rat b) {
    return new rat(this.c*b.c, this.d*b.d);
  }

  public rat div(rat b) {
    return new rat(this.c*b.d, this.d*b.c);
  }

  public rat add(rat b) {
    long new_d = kgV(this.d, b.d);
    long new_c = this.c*(new_d/this.d) + b.c*(new_d/b.d);
    return new rat(new_c, new_d);
  }

  public rat sub(rat b) {
    return this.add(new rat(-b.c, b.d));
  }
}
```

MD5: e7b408030f7e051e93a8c55056ba930b | $\mathcal{O}(?)$

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

```
public static int[] rop(int[] perm, int N, int K) {
  boolean[] incyc = new boolean[N];
  int[] cntcyc = new int[N+1];
  int[] g = new int[N+1];
  int[] needed = new int[N+1];
  for(int i = 1; i < N+1; i++) {
    int j = i;
    int k = K;
    int div;
    while(k > 1 && (div = gcd(k, i)) > 1) {
      k /= div;
      j *= div;
    }
    needed[i] = j;
    g[i] = gcd(K, j);
  }

  HashMap<Integer, ArrayList<Integer>> hm = new
      HashMap<Integer, ArrayList<Integer>>();
  for(int i = 0; i < N; i++) {
    if(incyc[i]) continue;
    ArrayList<Integer> cyc = new ArrayList<Integer>();
    cyc.add(i);
    incyc[i] = true;
    int newelem = perm[i];
    while(newelem != i) {
      cyc.add(newelem);
      incyc[newelem] = true;
      newelem = perm[newelem];
    }
    int len = cyc.size();
    cntcyc[len]++;
    if(hm.containsKey(len)) {
      hm.get(len).addAll(cyc);
    } else {
      hm.put(len, cyc);
    }
  }
  boolean end = false;
  for(int i = 1; i < N+1; i++) {
    if(cntcyc[i] % g[i] != 0) end = true;
  }
  if(end) {
    //not possible
    return null;
  } else {
    int[] out = new int[N];
    for(int length = 0; length < N; length++) {
      if(!hm.containsKey(length)) continue;
      ArrayList<Integer> p = hm.get(length);
      int totalsize = p.size();
      int diffcyc = totalsize / needed[length];
      for(int i = 0; i < diffcyc; i++) {
        int[] c = new int[needed[length]];
        for(int it = 0; it < needed[length]; it++) {
          c[it] = p.get(it + i * needed[length]);
        }
        int move = K / (needed[length]/length);
        int[] rewind = new int[needed[length]];
        for(int set = 0; set < needed[length]/length;
            set++) {
          int pos = set * length;
          for(int it = 0; it < length; it++) {
            rewind[pos] = c[it + set * length];
            pos = ((pos - set * length + move) %
                length)+ set * length;
          }
        }
        int[] merge = new int[needed[length]];
        for(int it = 0; it < needed[length]/length; it
            ++) {
          for(int set = 0; set < length; set++) {
            merge[set * needed[length] / length + it]
                = rewind[it * length + set];
          }
        }
        for(int it = 0; it < needed[length]; it++) {
          out[merge[it]] = merge[(it+1) % needed[
              length]];
        }
      }
    }
    return out;
  }
}
```

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.

```
public static boolean[] sieveOfEratosthenes(int N) {
  boolean[] isPrime = new boolean[N+1];
  for (int i=2; i<=N; i++) isPrime[i] = true;
  for (int i = 2; i*i <= N; i++)
    if (isPrime[i])
      for (int j = i*i; j <= N; j+=i)
        isPrime[j] = false;
  return isPrime;
}
```

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

4.9 Greatest Common Devisor

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 devisor of the input

```
private static long gcd(long a, long b) {
    while (b > 0) {
        long temp = b;
        b = a % b; // % is remainder
        a = temp;
    }
    return a;
}

private static long gcd(long[] input) {
    long result = input[0];
    for(int i = 1; i < input.length; i++)
    result = gcd(result, input[i]);
    return result;
}
```

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

```java
private static long lcm(long a, long b) {
    return a * (b / gcd(a, b));
}

private static long lcm(long[] input) {
  long result = input[0];
  for(int i = 1; i < input.length; i++)
    result = lcm(result, input[i]);
  return result;
}
```

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

5 Misc

5.1 Binary Search

Binary searchs 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

```java
public static int BinarySearch(int[] array,
                                  int N, int a) {
  int lo = 0;
  int hi = N-1;
  // a might be in interval [lo,hi] while lo <= hi
  while(lo <= hi) {
    int mid = (lo + hi) / 2;
    // if a > elem in mid of interval,
    // search the right subinterval
    if(array[mid] < a)
      lo = mid+1;
    // else if a < elem in mid of interval,
    // search the left subinterval
    else if(array[mid] > a)
      hi = mid-1;
    // else a is found
    else
      return mid;
  }
  // array does not contain a
  return -1;
}
```

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

```java
public static int nextNumber(int x) {
  //break when larger than limit here
  if(x == 0) return 0;
  int smallest = x & -x;
  int ripple = x + smallest;
  int new_smallest = ripple & -ripple;
  int ones  = ((new_smallest/smallest) >> 1) - 1;
  return ripple | ones;
}
```

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

```java
public static boolean nextPermutation(char[] a) {
  int i = a.length - 1;
  while(i > 0 && a[i-1] >= a[i])
    i--;
  if(i <= 0)
    return false;
  int j = a.length - 1;
  while (a[j] <=  a[i-1])
    j--;
  char tmp = a[i - 1];
  a[i - 1] = a[j];
  a[j] = tmp;

  j = a.length - 1;
  while(i < j) {
    tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
    i++;
    j--;
  }
  return true;
}
```

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

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

```java
public static ArrayList<Integer> kmp(String s, String
    w) {
  ArrayList<Integer> ret = new ArrayList<>();
  //Build prefix table
  int[] N = new int[w.length()+1];
  int i=0; int j =-1; N[0]=-1;
  while (i<w.length()) {
```

```
    while (j>=0 && w.charAt(j) != w.charAt(i))
      j = N[j];
    i++; j++; N[i]=j;
  }
  //Search string
  i=0; j=0;
  while (i<s.length()) {
    while (j>=0 && s.charAt(i) != w.charAt(j))
      j = N[j];
      i++; j++;
      if (j==w.length()) { //match found
      ret.add(i-w.length()); //add its start index
      j = N[j];
    }
  }
  return ret;
}
```

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.

```
public static int levenshteinDistance(String a, String
    b) {
  a = a.toLowerCase();
  b = b.toLowerCase();
  int[] costs = new int[b.length() + 1];

  for (int j = 0; j < costs.length; j++)
    costs[j] = j;

  for (int i = 1; i <= a.length(); i++) {
    costs[0] = i;
    int nw = i - 1;
    for (int j = 1; j <= b.length(); j++) {
      int cj = Math.min(1 + Math.min(costs[j], costs[j
          - 1]),
        a.charAt(i - 1) == b.charAt(j - 1) ? nw : nw +
            1);
      nw = costs[j];
      costs[j] = cj;
    }
  }
  return costs[b.length()];
}
```

MD5: 79186003b792bc7fd5c1ffbbcfc2b1c6 | $\mathcal{O}(|a| \cdot |b|)$

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.

```
public static String longestCommonSubsequence(String
    string1, String string2) {
  char[] s1 = string1.toCharArray();
  char[] s2 = string2.toCharArray();
  int[][] num = new int[s1.length + 1][s2.length + 1];
  // Actual algorithm
  for (int i = 1; i <= s1.length; i++)
    for (int j = 1; j <= s2.length; j++)
      if (s1[i - 1] == s2[j - 1])
        num[i][j] = 1 + num[i - 1][j - 1];
      else
        num[i][j] = Math.max(num[i - 1][j], num[i][j -
            1]);
  // System.out.println("length of LCS = " + num[s1.
      length][s2.length]);
  int s1position = s1.length, s2position = s2.length;
  List<Character> result = new LinkedList<Character>()
      ;
  while (s1position != 0 && s2position != 0) {
    if (s1[s1position - 1] == s2[s2position - 1]) {
      result.add(s1[s1position - 1]);
      s1position--;
      s2position--;
    } else if (num[s1position][s2position - 1] >= num[
        s1position][s2position])
      s2position--;
    else
      s1position--;
  }
  Collections.reverse(result);
  char[] resultString = new char[result.size()];
  int i = 0;
  for (Character c : result) {
    resultString[i] = c;
    i++;
  }
  return new String(resultString);
}
```

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

```
public static TreeSet<String> LCS(String a, String b)
    {
  int[][] t = new int[a.length()+1][b.length()+1];
  for(int i = 0; i <= b.length(); i++)
    t[0][i] = 0;

  for(int i = 0; i <= a.length(); i++)
    t[i][0] = 0;

  for(int i = 1; i <= a.length(); i++)
    for(int j = 1; j <= b.length(); j++)
      if(a.charAt(i-1) == b.charAt(j-1))
        t[i][j] = t[i-1][j-1] + 1;
      else
        t[i][j] = 0;
  int max = -1;
  for(int i = 0; i <= a.length(); i++)
    for(int j = 0; j <= b.length(); j++)
      if(max < t[i][j])
        max = t[i][j];
  if(max == 0 || max == -1)
    return new TreeSet<String>();
  TreeSet<String> res = new TreeSet<String>();
  for(int i = 0; i <= a.length(); i++)
    for(int j = 0; j <= b.length(); j++)
      if(max == t[i][j])
        res.add(a.substring(i-max, i));
  return res;
```

```
}
```

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

7 Math Roland

7.1 Divisability Explanation

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

7.2 Combinatorics

- Variations (ordered): k out of n objects (permutations for $k = n$)
  - without repetition: $M = \{(x_1, \ldots, x_k) : 1 \leq x_i \leq n,\ x_i \neq x_j \text{ if } i \neq j\}$, $|M| = \frac{n!}{(n-k)!}$
  - with repetition: $M = \{(x_1, \ldots, x_k) : 1 \leq x_i \leq n\}, |M| = n^k$
- Combinations (unordered): k out of n objects
  - without repetition: $M = \{(x_1, \ldots, x_n) : x_i \in \{0, 1\},\ x_1 + \ldots + x_n = k\}, |M| = \binom{n}{k}$
  - with repetition: $M = \{(x_1, \ldots, x_n) : x_i \in \{0, 1, \ldots, k\},\ x_1 + \ldots + x_n = k\}, |M| = \binom{n+k-1}{k}$
- Ordered partition of numbers: $x_1 + \ldots + x_k = n$ (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: $x_1 + \ldots + x_k = n$ (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): $!n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$

7.3 Polynomial Interpolation

7.3.1 Theory

Problem: for $\{(x_0, y_0), \ldots, (x_n, y_n)\}$ find $p \in \Pi_n$ with $p(x_i) = y_i$ for all $i = 0, \ldots, n$.
Solution: $p(x) = \sum\limits_{i=0}^{n} \gamma_{0,i} \prod\limits_{j=0}^{i-1} (x - x_i)$ 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 $i = n-1, \ldots, 0$ with $b_0 = p(x)$.

7.4 Fibonacci Sequence

7.4.1 Binet's formula

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

7.4.2 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 $g_0, g_1 \in \mathbb{N}_0$

7.4.3 Pisano Period

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

7.5 Reihen

$\sum\limits_{i=1}^{n} i = \frac{n(n+1)}{2}, \sum\limits_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \sum\limits_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$
$\sum\limits_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1, \sum\limits_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum\limits_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$
$\sum\limits_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, c \neq 1, \sum\limits_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, |c| < 1$

7.6 Binomialkoeffizienten

7.7 Catalanzahlen

$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$
$C_0 = 1, C_{n+1} = \sum\limits_{k=0}^{n} C_k C_{n-k}, C_{n+1} = \frac{4n+2}{n+2} C_n$

7.8 Geometrie

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

7.9 Zahlentheorie

Chinese Remainder Theorem: Es existiert eine Zahl C, sodass: $C \equiv a_1 \mod n_1, \cdots, C \equiv a_k \mod n_k, \mathrm{ggt}(n_i, n_j) = 1, i \neq j$
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: $a^{\varphi(n)} \equiv 1 \mod n$
Fermats Theorem: $a^p \equiv a \mod p, p$ prim

7.10 Faltung

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

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

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

8.3 Speed up IO

Use

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

Use

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