Contents

This pdf was generated on September 21, 2017

1 Basic

1.1 Template

```
import java.io.*;
import java.util.*;

public class _stub {
    public static void main(String[] args) throws IOException {
        IO io = new IO(System.in);
        io.close();
    }

    static class IO extends PrintWriter {
        static BufferedReader r;
        static StringTokenizer t;

        public IO(InputStream i) {
            super(new BufferedOutputStream(System.out));
            r = new BufferedReader(new InputStreamReader(i));
            t = new StringTokenizer("");
        }

        public String next() throws IOException {
            while (!t.hasMoreTokens()) {
                t = new StringTokenizer(r.readLine());
            }
            return t.nextToken();
        }

        public int nextInt() throws IOException{
            return Integer.parseInt(next());
        }

        public long nextLong() throws IOException {
            return Long.parseLong(next());
        }

        public double nextDouble() throws IOException {
            return Double.parseDouble(next());
        }
    }
}
```

2 Data Structures

2.1 Binary Indexed Tree

```
class BIT {
    private long[] s;

    public BIT(int n) {
        s = new long[n];
    }

    public void add(int pos, long dif) {
        for (int i = pos; i < s.length; i |= i + 1)
            s[i] += dif;
    }

    public long sum(int from, int to) {
        if (from > 0) return sum(0, to) - sum(0, from - 1);
        long res = 0;
        for (int i = to; i >= 0; i = (i & (i + 1)) - 1)
            res += s[i];
        return res;
    }
}
```

2.2 Binary Indexed Tree - 2D

```
class BIT2D {
    private long[][] t;

    public BIT2D(int r, int c) {
        t = new long[r][c];
    }

    public void add(int r, int c, long dif) {
        for (int i = r; i < t.length; i |= i + 1) {
            for (int j = c; j < t[0].length; j |= j + 1) {
                t[i][j] += dif;
    }}}

    public long sum(int r1, int c1, int r2, int c2) {
        if (r1 != 0 || c1 != 0) {
            return sum(0,0, r2, c2) - sum(0,0, r1 - 1, c2)
```

```
                    - sum(0,0, r2, c1 - 1) + sum(0,0, r1 - 1, c1 - 1);
        }
        long res = 0;
        for (int i = r2; i >= 0; i = (i & (i + 1)) - 1) {
            for (int j = c2; j >= 0; j = (j & (j + 1)) - 1) {
                res += t[i][j];
        }}
        return res;
    }
}
```

2.3 UnionFind

```
public class UnionFind {
    private int[] parent;
    private int[] rank;

    public UnionFind(int size) {
        parent = new int[size];
        rank = new int[size];
        for (int i = 0; i < parent.length; i++)
            parent[i] = i;
    }

    public void union(int x, int y) {
        int xRoot = find(x);
        int yRoot = find(y);

        if (xRoot == yRoot) return;

        if (rank[xRoot] < rank[yRoot]) {
            parent[xRoot] = yRoot;
        } else if (rank[xRoot] > rank[yRoot]) {
            parent[yRoot] = xRoot;
        } else {
            parent[yRoot] = xRoot;
            rank[xRoot]++;
    }}

    public int find(int x) {
        if (parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
}}
```

2.4 Prefix Tree

```
class Prefix_Tree {
    private V root;

    public Prefix_Tree() {
        root = new V();
    }

    public void insert(String word) {
        V p = root;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            int index = c - 'a';
            if (p.arr[index] == null) {
                V temp = new V();
                p.arr[index] = temp;
                p = temp;
            } else {
                p = p.arr[index];
        }}
        p.isEnd = true;
    }

    public V searchNode(String s) {
        V p = root;
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            int index = c - 'a';
            if (p.arr[index] != null) {
                p = p.arr[index];
            } else {
                return null;
        }}
        return (p == root) ? null : p;
    }

    public static class V {
        V[] arr;
        boolean isEnd;
        public V() {
            this.arr = new V[26];
}}}
```

2.5 Segment Tree

```
class Segtree {
    private long[] arr;
    private long[] tree;
    private long[] lazy;

    public Segtree(long[] a) {
        arr = a.clone();
        tree = new long[4*a.length];
        lazy = new long[4*a.length];
        build(1,0,arr.length-1);
    }

    private void u(int node, int a, int b) {
        if (lazy[node] != 0) {
            tree[node] += lazy[node];
            if (a != b) {
                lazy[node * 2] += lazy[node];
                lazy[node * 2 + 1] += lazy[node];
            }
            lazy[node] = 0;
        }
    }

    private void build(int node, int a, int b) {
        if (a > b) return;

        if (a == b)  {
            tree[node] = arr[a];
            return;
        }
        build(node * 2, a, (a + b) / 2);
        build(node * 2 + 1, 1 + (a + b) / 2, b);
        tree[node] = Math.max(tree[node * 2],
            tree[node * 2 + 1]); // operation
    }

    private void update(int node, int a, int b, int i, int j, long val) {
        u(node,a,b);

        if (a > b || a > j || b < i) return;

        if (a >= i && b <= j) {
            tree[node] += val;

            if (a != b) {
                lazy[node * 2] += val;
                lazy[node * 2 + 1] += val;
            }
            return;
        }

        update(node * 2, a, (a + b) / 2, i, j, val);
        update(node * 2 + 1, 1 + (a + b) / 2, b, i, j, val);
        tree[node] = Math.max(tree[node * 2],
            tree[node * 2 + 1]); // operation
    }

    private long query(int node, int a, int b, int i, int j) {
        if (a > b || a > j || b < i)
            return Long.MIN_VALUE; // operation

        u(node,a,b);

        if (a>=i && b <= j) {
            return tree[node];
        }

        long q1 = query(node*2,a,(a+b)/2,i,j);
        long q2 = query(node*2+1,1+(a+b)/2,b,i,j);
        long res = Math.max(q1,q2); // operation
        return res;
    }

    public void update(int i, int j, int val) {
        update(1,0,arr.length-1,i,j,val);
    }

    public long query(int i, int j) {
        return query(1,0,arr.length-1,i,j);
    }
}
```

3 Dynamic Programming

3.1 Knapsack

```
// Solve the Knapsack problem with O(n*c) time and O(c) space.
public static void main(String[] args) throws IOException {
    // int[] v of values, int[] c of costs
    int[] dp = new int[cap + 1];
    for (int j = 0; j < nItems ; j++) {
        for (int j2 = cap; j2 >= c[j]; j2--) {
            dp[j2] = Math.max(dp[j2], dp[j2 - c[j]] + v[j]);
    }}
    System.out.println(dp[cap]); // shows the max possible value
}
```

3.2 Longest Increasing Subsequence

```
// Return longest increasing subsequence (not consecutive) O(n log n )
static int[] LIS(int[] x) {
    int n = x.length;
    int[] p = new int[n];
    int[] m = new int[n + 1];

    int l = 0;
    for (int i = 0; i < n; i++) {
        double lo = 1;
        double hi = l;
        while (lo <= hi) {
            int mid = (int) Math.ceil((lo + hi) / 2);
            if (x[m[mid]] < x[i]) {
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }

        int newL = (int) lo;
        p[i] = m[newL - 1];
        m[newL] = i;
        if (newL > l) {
            l = newL;
        }
    }
    int[] s = new int[l];
    int k = m[l];
    for (int j = l-1; j >= 0; j--) {
        s[j] = x[k];
        k = p[k];
    }
    return s;
}
```

3.3 Longest Common Substring

```
// Given two strings, returns a longest common substring (consecutive).
// O(nm).
static String LCS(String s, String t) {
    int m = s.length();
    int n = t.length();
    int[][] L = new int[m][n];
    int z = 0;
    String res = "";

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (s.charAt(i) == t.charAt(j)) {
                if (i == 0 || j == 0) {
                    L[i][j] = 1;
                } else {
                    L[i][j] = L[i - 1][j - 1] + 1;
                }
                if (L[i][j] > z) {
                    z = L[i][j];
                    res = s.substring(i - z + 1, i + 1);
                }
            }
        }
    }
    return res;
}
```

3.4 DP over subsets

```
int bits = 30;

// Top down
for (int item = 0; item < bits; item++) {
    for (int mask = (1 << (bits + 1) - 1); mask >= 0; mask--) {
        if ((mask & (1 << item)) == 0) {
            int superset = mask ^ (1 << item);
            // Do things
}}}

// Bottom up
for (int item = 0; item < bits; item++) {
    for (int mask = 0; mask < (1 << bits); mask++) {
        if ((mask & (1 << item)) != 0) {
            int subset = mask ^ (1 << item);
            // Do things
}}}
```

4 Graphs

4.1 Dijkstra (Single-Source Shortest Path)

```
static class V implements Comparable<V> {
    public final int name;
    public List<E> adj;
    public double dist = Double.POSITIVE_INFINITY;
    public V prev;

    public V(int _n) {
        name = _n;
        adj = new ArrayList<E>();
    }

    public int compareTo(V o) {
        return Double.compare(dist, o.dist);
    }
}

static class E {
    public final V end;
    public final double w;

    public E(V _e, double _w) {
        end = _e;
        w = _w;
    }
}

static void compute(V source) {
    source.dist = 0.;
    PriorityQueue<V> q = new PriorityQueue<>();
    q.add(source);

    while (!q.isEmpty()) {
        V v = q.poll();
        for (E e : v.adj) {
            V u = e.end;
            double uDist = v.dist + e.w;
            if (uDist < u.dist) {
                q.remove(u);
                u.dist = uDist;
                u.prev = v;
                q.add(u);
}}}}
```

4.2 Floyd-Warshall (All-Pairs Shortest Path)

```
// All pairs shortest path. O(V^3).
// Negative weights allowed, but no negative cycles!
static int[][] floydWarshall(int[][] graph) {
    int nV = graph.length;
    int dist[][] = graph.clone();

    for (int k = 0; k < nV; k++) {
        for (int i = 0; i < nV; i++) {
            for (int j = 0; j < nV; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
    }}}
    return dist; }

public static void main(String[] args) {
    int nV = 4;
    int[][] graph = new int[nV][nV];
```

```
    for (int i = 0; i < nV; i++) {
        Arrays.fill(graph[i], 100000);
    }
    for (int i = 0; i < nV; i++) {
        graph[i][i] = 0;
    }

    graph[0][1] = 1;
    graph[1][2] = 1;
    graph[2][3] = 1;

    int[][] res = floydWarshall(graph);
}
```

4.3 Hopcroft-Karp (Maximum Bipartite Matching)

```
public static class bipartiteGraph {
    public ArrayList<ArrayList<Integer>> adj;
    public final int xV;
    public final int yV;
    private int[] pair;
    private int[] dist;

    bipartiteGraph(int _xV, int _yV) {
        xV = _xV;
        yV = _yV;
        adj = new ArrayList<ArrayList<Integer>>();
        for (int i = 0; i < xV + yV + 1; i++) {
            adj.add(new ArrayList<Integer>());
        }
        pair = new int[xV + yV + 1];
        dist = new int[xV + yV + 1];
    }

    public void addEdge(int x, int y) {
        adj.get(x + 1).add(xV + y + 1);
        adj.get(xV + y + 1).add(x + 1);
    }

    private boolean BFS() {
        Queue<Integer> q = new LinkedList<Integer>();
        for (int v = 1; v <= xV; v++) {
            if (pair[v] == 0) {
                dist[v] = 0;
                q.add(v);
            } else {
                dist[v] = Integer.MAX_VALUE;
            }
        }
        dist[0] = Integer.MAX_VALUE;

        while (!q.isEmpty()) {
            int v = q.poll();
            if (dist[v] < dist[0]) {
                for (int u : adj.get(v)) {
                    if (dist[pair[u]] == Integer.MAX_VALUE) {
                        dist[pair[u]] = dist[v] + 1;
                        q.add(pair[u]);
                    }
                }
            }
        }
        return dist[0] != Integer.MAX_VALUE;
    }

    private boolean DFS(int v) {
        if (v != 0) {
            for (int u : adj.get(v)) {
                if (dist[pair[u]] == dist[v] + 1) {
                    if (DFS(pair[u])) {
                        pair[u] = v;
                        pair[v] = u;
                        return true;
                    }
                }
            }
            dist[v] = Integer.MAX_VALUE;
            return false;
        }
        return true;
    }

    public int hc() {
        int matching = 0;
        while (BFS()) {
```

```
            for (int v = 1; v <= xV; v++) {
                if (pair[v] == 0 && DFS(v)) {
                    matching = matching + 1;
                }
            }
        }
        return matching;
    }
}

// Main class for example
public static void main(String[] args) throws IOException {
    IO io = new IO(System.in);

    int xV = in.nextInt();
    int yV = in.nextInt();

    bipartiteGraph g = new bipartiteGraph(xV, yV);

    int nE = in.nextInt();
    for (int i = 0; i < nE; i++)
        g.addEdge(in.nextInt(), in.nextInt());

    int matches = g.hc();
    io.println(matches);
}
```

4.4 Topological Sort

```
//Sort vertices such that if (u,v) is an edge, u comes before v.
//Only works on acyclic graph. Gives wrong output otherwise! O(E + V)
static void dfs(List<Integer>[] g, boolean[] used,
    List<Integer> res, int u) {
    used[u] = true;
    for (int v : g[u])
        if (!used[v]) dfs(g, used, res, v);
    res.add(u);
}

static List<Integer> topSort(List<Integer>[] g) {
    int n = g.length;
    boolean[] used = new boolean[n];
    List<Integer> res = new ArrayList<>();
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs(g, used, res, i);
    Collections.reverse(res);
    return res;
}
```

4.5 2-Color

```
// Given a graph, returns true if it is 2-colorable. O(E).
static boolean twoColor(List<Integer>[] graph) {
    int size = graph.length;
    boolean[] visited = new boolean[size];
    int[] colors = new int[size];
    int nVisited = 0;

    for (int i = 0; i < size && nVisited < size; i++) {
        if (!visited[i]) {
            visited[i] = true;
            Queue<Integer> q = new LinkedList<Integer>();
            q.add(i);
            colors[i] = 1;
            nVisited++;

            while (!q.isEmpty()) {
                int u = q.poll();
                for (int v : graph[u]) {
                    if (!visited[v]) {
                        nVisited++;
                        visited[v] = true;
                        q.add(v);
                    }
                    if (colors[v] == 0) {
                        colors[v] = 3 - colors[u];
                    } else if (colors[v] == colors[u]) {
                        return false;
        }}}}
    return true;
}
```

4.6 Bellman-Ford (Single-Source Shortest Path, Neg. Weights)

```
// Single source shortest path, negative weights allowed. O(V*E).
// Return true if there is no negative weight cycle, false otherwise.
static void BF (V source, int n, ArrayList<E> edges) {
    source.dist=0;
    for (int i = 0; i < n-1; i++) {
        for (E e : edges) {
            if (e.start.dist!= Long.MAX_VALUE &&
                    e.start.dist + e.w < e.end.dist) {
                e.end.dist = e.start.dist + e.w;
                e.end.prev = e.start;
    }}}

    for (E e : edges) {
        if (e.start.dist!= Long.MAX_VALUE &&
                e.start.dist + e.w < e.end.dist) {
            dfs(e.end);
}}}
// dfs to mark vertices reachable from negative cycles
static void dfs(V start) {
    start.neg_inf = true;
    for (E e : start.adj) {
        if (!e.end.neg_inf) dfs(e.end);
}}

static class V {
    public ArrayList<E> adj = new ArrayList<>();
    public long dist = Long.MAX_VALUE;
    public V prev = null;
    public boolean neg_inf = false;
}

static class E {
    public final V start;
    public final V end;
    public final long w;

    public E(V _s, V _e, long _w) {
        start = _s;
        end = _e;
        w = _w;
}}
```

4.7 Coloring

```
// Color graph by backtracking, only for small graph or few colors.
public static void main(String[] args) throws IOException {
    int nV = io.nextInt();
    TreeSet<Integer>[] g = new TreeSet[nV];
    // Make graph here, dont forget to initialize all g[i]
    int maxCol = nV;
    for (int nCol = 1; nCol < maxCol; nCol++) {
        TreeSet<Integer> colors = new TreeSet<>();
        for (int i = 1; i <= nCol; i++)
            colors.add(i);
        boolean[] flag = new boolean[nV];
        int[] assign = new int[nV];
        if (solve(0, nV, flag, assign, g, colors))
            io.println(nCol);
    }
}

static boolean solve(int index, int n, boolean[] flag,
    int[] assign, TreeSet[] g, TreeSet<Integer> colors) {
    if (index == n) return true;
    TreeSet<Integer> avail = (TreeSet<Integer>) colors.clone();
    if (index == 0) {
        avail.clear();
        avail.add(1);
    } else {
        TreeSet<Integer> adj = g[index];
        for (int a : adj) {
            if (flag[a]) avail.remove(assign[a]);
            if (avail.isEmpty()) return false;
        }
    }
    for (int col : avail) {
        assign[index] = col;
        flag[index] = true;
        boolean outp = solve(index + 1, n, flag, assign, g, colors);
        if (outp) return true;
    }
    assign[index] = 0;
    flag[index] = false;
    return false;
}
```

4.8 Kruskal (Minimum Spanning Tree)

```
// O(E log E).
static ArrayList<E> compute(int nV,E[] edges) {
    ArrayList<E> res = new ArrayList<E>();
    UnionFind uni = new UnionFind(nV);
    PriorityQueue<E> q = new PriorityQueue<E>();
    for (int i = 0; i < edges.length; i++)
        q.add(edges[i]);

    while (!q.isEmpty()) {
        E e = q.poll();
        int start = e.start.name;
        int end = e.end.name;
        if (uni.find(start) != uni.find(end)) {
            uni.union(start, end);
            res.add(e);
    }}
    return res;
}

static class V {
    public final int name;
    public List<E> adj;

    public V(int nm) {
        name = nm;
        adj = new ArrayList<E>();
    }
}

static class E implements Comparable<E>{
    public final int w;
    public final V start, end;
    public E(V s, V e, int _w) {
        start = s;
        end = e;
        w = _w;
    }
    public int compareTo(E other) {
        return Integer.compare(this.w, other.w);
    }
}
```

4.9 Tarjan (Strongly Connected Components)

```
//Find strongly connected components of a graph. O(E + V).
static List<List<Integer>> scc(List<Integer>[] g) {
    int n = g.length;
    boolean [] visited = new boolean[n];
    Stack<Integer> st = new Stack<>();
    int t = 0;
    int[] link = new int[n];
    List<List<Integer>> comp = new ArrayList<>();
    for (int u = 0; u < n; u++)
        if (!visited[u])
            dfs(u, link, t, visited, st, g, comp);
    return comp;
}

static void dfs(int u, int[] link, int t, boolean[] visited,
    Stack<Integer> st, List<Integer>[] g, List<List<Integer>> comp) {
    link[u] = t++;
    visited[u] = true;
    st.add(u);
    boolean isComponentRoot = true;

    for (int v : g[u]) {
        if (!visited[v])
            dfs(v, link, t, visited, st, g, comp);
        if (link[u] > link[v]) {
            link[u] = link[v];
            isComponentRoot = false;
        }
    }

    if (isComponentRoot) {
        List<Integer> component = new ArrayList<>();
        while (true) {
            int x = st.pop();
            component.add(x);
            link[x] = Integer.MAX_VALUE;
            if (x == u) break;
        }
        comp.add(component);
    }
}
```

4.10 Dinic (Maximum Flow)

```
// Dinic's max flow O(V^2 * E)
static class Edge {
    int end, rev, cap, flow;

    public Edge(int t, int rev, int cap) {
        this.end = t;
        this.rev = rev;
        this.cap = cap;
    }
}

public static ArrayList<ArrayList<Edge>> createGraph(int nodes) {
    ArrayList<ArrayList<Edge>> graph =
        new ArrayList<ArrayList<Edge>>(nodes);
    for (int i = 0; i < nodes; i++)
        graph.add(new ArrayList<Edge>());
    return graph;
}

public static void addEdge(ArrayList<ArrayList<Edge>> graph, int s,
    int t, int cap) {
    graph.get(s).add(new Edge(t, graph.get(t).size(), cap));
    graph.get(t).add(new Edge(s, graph.get(s).size() - 1, 0));
}

static boolean dinicBfs(ArrayList<ArrayList<Edge>> graph, int src,
    int dest, int[] dist) {
    Arrays.fill(dist, -1);
    dist[src] = 0;
    int[] Q = new int[graph.size()];
    int sizeQ = 0;
    Q[sizeQ++] = src;
    for (int i = 0; i < sizeQ; i++) {
        int u = Q[i];
        for (Edge e : graph.get(u)) {
            if (dist[e.end] < 0 && e.flow < e.cap) {
                dist[e.end] = dist[u] + 1;
                Q[sizeQ++] = e.end;
    }}}
    return dist[dest] >= 0;
}

static int dinicDfs(ArrayList<ArrayList<Edge>> graph, int[] ptr, int[] dist,
    int dest, int u, int f) {
    if (u == dest)
        return f;
    for (; ptr[u] < graph.get(u).size(); ++ptr[u]) {
        Edge e = graph.get(u).get(ptr[u]);
        if (dist[e.end] == dist[u] + 1 && e.flow < e.cap) {
            int df = dinicDfs(graph, ptr, dist, dest, e.end,
                Math.min(f, e.cap - e.flow));
            if (df > 0) {
                e.flow += df;
                graph.get(e.end).get(e.rev).flow -= df;
                return df;
            }
        }
    }
    return 0;
}

public static int maxFlow(ArrayList<ArrayList<Edge>> graph, int src,
    int dest) {
    int flow = 0;
    int[] dist = new int[graph.size()];
    while (dinicBfs(graph, src, dest, dist)) {
        int[] ptr = new int[graph.size()];
        while (true) {
            int df = dinicDfs(graph, ptr, dist, dest, src,
                Integer.MAX_VALUE);
            if (df == 0) break;
            flow += df;
        }
    }
    return flow;
}

public static void main(String[] args) {
    ArrayList<ArrayList<Edge>> graph = createGraph(4);
    addEdge(graph, 2, 3, 4);
    System.out.println(maxFlow(graph, 0, 3));
}
```

4.11 Min Cost Max Flow

```
// Find flow of value at least k for minimum cost. O(V^3 * E).
// k = MAX_VALUE for overall max flow.
static long[] MCMF(ArrayList<E>[] g, long k, int s, int t) {
    long flow = 0, cost = 0, INF = Long.MAX_VALUE/1000;
    int n = g.length;

    while (flow < k) {
        int[] id = new int[n];
        long[] d = new long[n];
        Arrays.fill(d, INF);
        int[] q = new int[n];
        int[] p = new int[n];
        int[] p_rib = new int[n];

        int qh = 0, qt = 0;
        q[qt++] = s;
        d[s] = 0;
        while (qh != qt) {
            int v = q[qh++];
            id[v] = 2;
            if (qh == n) qh = 0;
            for (int i = 0; i < g[v].size(); i++) {
                E r = g[v].get(i);
                if (r.f < r.cap && d[v] + r.cost < d[r.b]) {
                    d[r.b] = d[v] + r.cost;
                    if(id[r.b] == 0) {
                        q[qt++] = r.b;
                        if (qt == n) qt = 0;
                    } else if (id[r.b] == 2) {
                        if (--qh == -1) qh = n-1;
                        q[qh] = r.b;
                    }
                    id[r.b] = 1;
                    p[r.b] = v;
                    p_rib[r.b] = i;
                }
            }
        }

        if (d[t] == INF) break;
        long addflow = k - flow;
        for (int v = t; v != s; v = p[v]) {
            int pv = p[v];
            int pr = p_rib[v];
            addflow = Math.min(addflow,
                g[pv].get(pr).cap - g[pv].get(pr).f);
        }
        for (int v = t; v != s; v = p[v]) {
            int pv = p[v];
            int pr = p_rib[v], r = g[pv].get(pr).back;
            g[pv].get(pr).f += addflow;
            g[v].get(r).f -= addflow;
            cost += g[pv].get(pr).cost * addflow;
        }
        flow += addflow;
    }
    return new long[] {flow, cost};
}

static class E {
    int b;
    long cap, cost, f; //end, capacity, cost, flow
    int back;

    E (int _b, long _cap, long _cost, int _back) {
        b = _b;
        cap = _cap;
        cost = _cost;
        f = 0;
        back = _back;
    }
}

static void addEdge(ArrayList<E>[] g, int a, int b,
        long cap, long cost) {
    E e1 = new E(b, cap, cost, g[b].size());
    E e2 = new E(a, 0, -cost, g[a].size());
    g[a].add(e1);
    g[b].add(e2);
}
```

4.12 Bron-Kerbosch (All Maximal Cliques)

```
static TreeSet<Integer> r;
static TreeSet<Integer> p;
static TreeSet<Integer> x;
static HashSet<TreeSet<Integer>> maxCliques;

// Find all maximal cliques in g.
// WARNING exponential complexity, this is NP-complete! n <= 40 roughly.
public static void bk(TreeSet<Integer> r, TreeSet<Integer> p,
        TreeSet<Integer> x, ArrayList<Integer>[] g,
        HashSet<TreeSet<Integer>> max) {
    if (p.isEmpty() && x.isEmpty()) {
        maxCliques.add((TreeSet<Integer>) r.clone());
        return;
    }
    int u = p.isEmpty() ? x.first() : p.first();
    for (Iterator<Integer> i = p.iterator(); i.hasNext();) {
        int v = i.next();
        if (g[u].contains(v)) {
            continue;
        }
        TreeSet<Integer> r2 = (TreeSet<Integer>) r.clone();
        r2.add(v);
        TreeSet<Integer> p2 = new TreeSet<>();
        TreeSet<Integer> x2 = new TreeSet<>();
        for (int a : g[v]) {
            if (p.contains(a)) {
                p2.add(a);
            }
            if (x.contains(a)) {
                x2.add(a);
            }
        }

        bk(r2, p2, x2, g, max);
        x.add(v);
        i.remove();
    }
}
```

5 Geometry

5.1 Basics

```
static final double EPS = 0.000000001;
```

5.1.1 Point

```
static class P {
    double x, y;

    P(double _x, double _y) {
        x = _x;
        y = _y;
    }

    public P add(P o) {
        return new P(x + o.x, y + o.y);
    }

    public P sub(P o) {
        return new P(x - o.x, y - o.y);
    }

    public double dist(P o) {
        return Math.sqrt((x - o.x) * (x - o.x) + (y - o.y) * (y - o.y));
    }

    public double abs() {
        return Math.sqrt(x * x + y * y);
    }

    public P sc(double t) {
        return new P(t * x, t * y);
    }
}
```

5.1.2 Line

```
static class L {
    P a, b;
    boolean seg;
```

```
    L(P _a, P _b, boolean s) {
        a = _a;
        b = _b;
        seg = s;
    }

    public boolean degen() {
        return Math.abs(a.x - b.x) < EPS && Math.abs(a.y - b.y) < EPS;
    }
}
```

5.1.3 Angle

```
static double angle(P a, P b, P c) {
return Math.acos(dot(a.sub(b), c.sub(b))
/ a.sub(b).abs() / c.sub(b).abs());
}
```

5.1.4 Dot product

```
static double dot(P a, P b) {
    return a.x * b.x + a.y * b.y;
}
```

5.1.5 Cross product

```
static double cross(P a, P b) {
    return a.x * b.y - a.y * b.x;
}
```

5.1.6 Projection

```
static P proj(P p, L l) {
    if (l.degen()) {
        return l.a;
    }
    if (l.seg) {
        if (dot(l.b.sub(l.a), p.sub(l.b)) > 0)
        return l.b;
        if (dot(l.a.sub(l.b), p.sub(l.a)) > 0)
        return l.a;
    }
    double t = dot(p.sub(l.a), l.b.sub(l.a)) / l.b.sub(l.a).abs();
    P dir = l.b.sub(l.a).sc(1 / l.b.sub(l.a).abs());
    return l.a.add(dir.sc(t));
}
```

5.1.7 Distance from line (or segment) to point

```
static double distLinePoint(P p, L l) {
    P q = proj(p, l);
    return q.dist(p);
}
```

5.1.8 Orientation

```
static double ccw(P a, P b, P c) {
    return cross(b.sub(a), b.sub(c));
}
```

5.1.9 Collinear

```
static boolean collinear(P a, P b, P c) {
    return Math.abs(ccw(a, b, c)) < EPS;
}
```

5.1.10 Intersection of lines

```
// Parallel lines give null results!
static P intersect(L l, L m) {
    double A0 = l.b.y - l.a.y;
    double B0 = l.a.x - l.b.x;
    double C0 = A0 * l.a.x + B0 * l.a.y;
    double A1 = m.b.y - m.a.y;
    double B1 = m.a.x - m.b.x;
    double C1 = A1 * m.a.x + B1 * m.a.y;
    double D = A0 * B1 - A1 * B0;
    if (D == 0) return null;

    double x = (B1 * C0 - B0 * C1) / D;
    double y = (A0 * C1 - A1 * C0) / D;

    if (!l.seg && !m.seg) {
        return new P(x, y);
    } else {
        P p = new P(x, y);
        if (l.seg && distLinePoint(p, l) > EPS) {
            return null;
```

```
        }
        if (m.seg && distLinePoint(p, m) > EPS) {
            return null;
        }
        return p;
    }
}
```

5.1.11 Full intersection of segments

```
static L segment_intersect(L l, L m) {
    if (!collinear(l.a, l.b, m.a) || !collinear(l.a, l.b, m.b)) {
        P p = intersect(l, m);
        return p == null ? null : new L(p, p, true);
    } else {
        P[] pt = new P[] { l.a, l.b, m.a, m.b };
        double[] d = new double[] { distLinePoint(l.a, m),
            distLinePoint(l.b, m), distLinePoint(m.a, l),
            distLinePoint(m.b, l)};
        if (d[0] < EPS && d[1] < EPS)
            return new L(pt[0],pt[1],true);
        if (d[2] < EPS && d[3] < EPS)
            return new L(pt[2],pt[3],true);
        if (d[0] > EPS && d[1] > EPS && d[2] > EPS && d[3] > EPS)
            return null;
        if (d[0] < EPS) {
            if (d[2] < EPS) {
                return new L(pt[0],pt[2],true);
            } else {
                return new L(pt[0],pt[3],true);
            }
        } else {
            if (d[2] < EPS) {
                return new L(pt[1],pt[2],true);
            } else {
                return new L(pt[1],pt[3],true);
            }
        }
    }
}
```

5.1.12 Check if points are on same side of line

```
static boolean sameSide(L l, P p, P q) {
    P u = l.b.sub(l.a);
    P v = p.sub(l.a);
    P w = q.sub(l.a);
    return cross(u, v) * cross(u, w) > -EPS;
}
```

5.2 Polygon area

```
//Find the area of any polygon in 2d. Points must be in cw or ccw order.
static double area(double[] xcoord, double[] ycoord) {
    double res = 0;
    for (int i = 0; i < xcoord.length - 1; i++) {
        res += xcoord[i] * ycoord[i + 1] - xcoord[i + 1] * ycoord[i];
    }
    res += xcoord[xcoord.length-1] * ycoord[0] - xcoord[0] * ycoord[xcoord.length-1];
    return Math.abs(res/2);
}
```

5.3 Circle tangents

```
// Input two circles by [x, y, r]. Returns [phi, delta] such
// that the angels of the outer tangents are phi +/- delta.
static double[] circleTan(double[] c1, double[] c2) {
    double dx = (c1[0]-c2[0]);
    double dy = (c1[1]-c2[1]);
    double d = Math.sqrt(dx*dx+dy*dy);
    return new double[] {
    Math.atan(dy/dx),
    Math.asin((c1[2]+c2[2])/d) };
}
```

5.4 Graham Scan (Convex Hull)

```
//Given List<P> of points, return List<P> with all points on convex hull
//in ccw order. O(V log V).
private static boolean leftTurn(Point p1, Point p2, Point p3) {
        return (p2.x - p1.x) * (p3.y - p1.y) -
            (p2.y - p1.y) * (p3.x - p1.x) >= 0;
}

static ArrayList<Point> hull(ArrayList<Point> points) {
    int n = points.size();
    ArrayList<Point> pointsByX = (ArrayList<Point>) points.clone();
    Collections.sort(pointsByX, new Comparator<Point>() {
        public int compare(Point o1, Point o2) {
            int r = new Integer(o1.x).compareTo(new Integer(o2.x));
            return r == 0 ?
                new Integer(o1.y).compareTo(new Integer(o2.y)) : r;
    }});

    Point[] up = new Point[pointsByX.size()];
    up[0] = pointsByX.get(0);
    up[1] = pointsByX.get(1);
    int upInd = 2;
    for (int i = 2; i < n; i++) {
        up[upInd] = pointsByX.get(i);
        upInd++;

        while (upInd > 2 && leftTurn(up[upInd - 3], up[upInd - 2],
            up[upInd - 1])) {
                up[upInd - 2] = up[upInd - 1];
                up[upInd - 1] = null;
                upInd--;
        }
    }

    Point[] low = new Point[n];
    low[0] = pointsByX.get(n - 1);
    low[1] = pointsByX.get(n - 2);
    int lowInd = 2;
    for (int i = 3; i <= n; i++) {
        low[lowInd] = pointsByX.get(n - i);
        lowInd++;

        while (lowInd > 2 && leftTurn(low[lowInd - 3], low[lowInd - 2],
            low[lowInd - 1])) {
                low[lowInd - 2] = low[lowInd - 1];
                low[lowInd - 1] = null;
                lowInd--;
        }
    }

    ArrayList<Point> hull = new ArrayList<Point>(upInd + lowInd);
    for (int i = 0; i < upInd; i++)
        hull.add(up[i]);

    for (int i = 1; i < lowInd - 1; i++)
        hull.add(low[i]);

    return hull;
}
```

5.5 Points to plane

```
// Find plane equation ax + by + cz + d = 0. {a,b,c,d} is returned.
// Points should be in a double[3] {x,y,z}.
static double[] pointsToPlane(double[] p1, double[] p2, double[] p3) {
    double[] v1 = { p1[0] - p2[0], p1[1] - p2[1], p1[2] - p2[2] };
    double[] v2 = { p1[0] - p3[0], p1[1] - p3[1], p1[2] - p3[2] };
    double[] n = crossProduct(v1, v2);
    double d = -n[0] * p1[0] - n[1] * p1[1] - n[2] * p1[2];
    return new double[] { n[0], n[1], n[2], d };
}

static double[] crossProduct(double[] u, double[] v) {
    double[] res = new double[3];
    res[0] = u[1] * v[2] - u[2] * v[1];
    res[1] = u[2] * v[0] - u[0] * v[2];
    res[2] = u[0] * v[1] - u[1] * v[0];
    return res;
}
```

5.6 Intersect line and plane

```
// Line given by p0 + t * p1, l = {x0, y0, z0, x1, y1, z1}.
// Plane given by ax + by + cz + d = 0, p = {a, b, c, d}.
static double[] intersectLinePlane(double[] l, double[] p) {
    double p1 = p[0] * l[0] + p[1] * l[1] + p[2] * l[2] + p[3];
    double p2 = p[0] * l[3] + p[1] * l[4] + p[2] * l[5];
    double t = -p1 / p2;
    return new double[]{l[0] + t * l[3], l[1] + t * l[4], l[2] + t * l[5]};
}
```

5.7 Precise Point on Line

```
static class P {
    long x, y;
    P(long _x, long _y) {
        x = _x;
        y = _y;
    }
}

static class L {
    P a, b;
    boolean seg;
    L(P _a, P _b, boolean s) {
        a = _a;
        b = _b;
        seg = s;
    }
}

static boolean PointOnLine(P p, L l) {
    if (l.seg && (p.x < Math.min(l.a.x, l.b.x) ||
        p.x > Math.max(l.a.x, l.b.x))) {
        return false;
    }
    if (l.seg && (p.y < Math.min(l.a.y, l.b.y) || p.y >
        Math.max(l.a.y, l.b.y))) {
        return false;
    }
    long dx = l.b.x - l.a.x;
    long dy = l.b.y - l.a.y;

    long g = gcd(dx, dy);
    dx = dx / g;
    dy = dy / g;

    long x = p.x - l.b.x;
    long y = p.y - l.b.y;

    if (dx == 0) return p.x == l.a.x;
    if (x % dx != 0) return false;
    return (x / dx) * dy == y;
}
```

6 Number Theory

6.1 Matrix Exponentiation

```java
// Fast matrix exponentiation (must be square) O(l^2 * log(n))
public static double[][] fme(double[][] a, long n, long mod) {
    int l = a.length;
    double[][] res = new double[l][l];
    for (int i = 0; i < l; i++) {
        res[i][i] = 1;
    }

    String bin = Long.toBinaryString(n);
    int exp = 0;
    while (Math.pow(2, exp) <= n) {
        if (bin.charAt(bin.length()-1-exp) == '1') {
            res = mult(res, a, mod);
        }
        a = mult(a,a,mod);
        exp++;
    }

    return res;
}

// Multiply 2 matrices
public static double[][] mult(double[][] a, double[][] b, long mod) {
    int k = a.length;
    int n = a[0].length;
    int m = b[0].length;
    double[][] res = new double[k][m];
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < m; j++) {
            for (int j2 = 0; j2 < n; j2++) {
                res[i][j] = (res[i][j] + (a[i][j2] * b[j2][j])) % mod;
            }
        }
    }
    return res;
}
```

6.2 Simultaneous Congruences

```java
// Input is modular equations x = r[i] mod m[i].
// The m[i] must be rel. prime. Returns x.
static long SC(long[] r, long[] mods) {
    int n = r.length;
    long p = 1;
    for (int i = 0; i < n; i++)
        p *= mods[i];
    long[] m = new long[n];
    for (int i = 0; i < n; i++)
        m[i] = p / mods[i];

    long[] N = new long[n];
    for (int i = 0; i < n; i++)
        N[i] = (r[i] * modInv(m[i], mods[i])) % mods[i];

    long x = 0;
    for (int i = 0; i < n; i++)
        x += m[i] * N[i];

    return ((x % p) + p) % p;
}
```

6.3 BigInteger methods

```java
// Inverse of a mod m. Throws error if a and m not rel. prime!
BigInteger inv = a.modInverse(m);

// Fast modular exponentiation, find a^exp % m.
BigInteger res = a.modPow(exp, m);

// Check if a is prime. Prob of error is < 1/2^cert.
a.isProbablePrime(int cert = 40);
```

6.4 Catalan numbers

```java
// Calculate the n-th Catalan number
static BigInteger cat(int n) {
    return ncr(2*n,n).divide(BigInteger.valueOf(n+1));
}
```

6.5 Combinations

```
// Calculate n choose k, also knows as n nCr k, or n above k
// Works for n <= 10 000.
static BigInteger ncr(int n, int k) {
    k = k > n / 2 ? n - k : k;
    BigInteger a = BigInteger.ONE;
    for (int i = 1; i < k + 1; i++) {
        a = a.multiply(BigInteger.valueOf
        (n - i + 1)).divide(BigInteger.valueOf(i));
    }
    return a;
}
```

6.6 GCD

```
// Euclid's algorithm to find gcd.
static int gcd(int p, int q) {
    return q == 0 ? Math.abs(p) : gcd(q, p % q);
}

// Returns array [d, a, b] such that d = gcd(p, q), ap + bq = d.
static int[] extgcd(int p, int q) {
    if (q == 0) {
        return new int[] { Math.abs(p), Integer.signum(p), 0 };
    }
    int[] vals = extgcd(q, p % q);
    int b = vals[1] - (p / q) * vals[2];
    return new int[] { vals[0], vals[2], b };
}
```

6.7 Euler phi

```
// phi(n) is the amount of numbers less than n rel. prime to n
static long phi(long n) {
    double res = n;
    // Use Set instead of List!
    HashSet<Long> factors = primeFactors(n);
    for (long f : factors)
        res *= (1.-1./f);
    return Math.round(res);
}
```

6.8 Fast Exponentiation

```
static long modPow(long a, long b, long mod) {
    long res = 1 % mod;
    while (b > 0) {
        if ((b & 1) == 1) {
            res = res * a % mod;
        }
        a = a * a % mod;
        b >>>= 1;
    }
    return res;
}
```

6.9 Modular Inverse

```
// Inverse of a in Z/pZ, with p prime: x such that ax = 1 (mod p).
static long modInvPrime(long a, long p) {
    return (long) Math.pow(a, p - 2) % p;
}

// Compute inverse of a in Z/rZ, with a and r co-prime.
// If they are not co-prime, inverse does not exist.
static long modInv(long a, long r) {
    if (extgcd(a, r)[0] != 1) {
        return -1;
    }
    return (extgcd(a, r)[1] + r) % r;
}
```

6.10 Factorial

```
// Calculate n! Works for n <= 5 000.
static BigInteger fac(int n) {
    BigInteger a = BigInteger.ONE;
    for (int i = 2; i < n + 1; i++)
        a = a.multiply(BigInteger.valueOf(i));
    return a;
}
```

6.11 Primality Test

```
//Efficient primality test.
static boolean isPrime(long n) {
    if (n < 2) return false;
    if (n == 2 || n == 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    long sqrtN = (long) Math.sqrt(n) + 1;
    for (long i = 6L; i <= sqrtN; i += 6) {
        if (n % (i - 1) == 0 || n % (i + 1) == 0)
            return false;
    }
    return true;
}
```

6.12 Prime Decomposition

```
static HashMap<Long, Integer> primeFactors(long a) {
    HashMap<Long, Integer> res = new HashMap<Long, Integer>();
    while (a % 2 == 0) {
        res.put(2L, res.get(2L) == null ? 1 : res.get(2L) + 1);
        a = a / 2;
    }
    // Careful! Loop variable is a long
    for (long i = 3; i * i <= a; i = i + 2) {
        while (a % i == 0) {
        res.put(i, res.get(i) == null ? 1 : res.get(i) + 1);
        a = a / i;
        }
    }
    if (a > 2) {
        res.put(a, res.get(a) == null ? 1 : res.get(a) + 1);
    }
    return res;
}
```

6.13 Theorems and Formulas

Fermat's Little Theorem: $a^p \equiv a \pmod{p}$
Euler's Theorem: if a, n rel. prime: $a^{\phi(n)} \equiv 1 \pmod{n}$
Least Common Multiple: $\text{LCM}(a,b) = a * \frac{b}{\text{GCD}(a,b)}$

6.13.1 Combinatorics

$\sum_{i=0}^{n} \binom{n}{i} = 2^n$ $\quad$ $\binom{n}{i} = \frac{n}{i}\binom{n-1}{i-1}$

$\binom{n}{h}\binom{n-h}{i} = \binom{n}{i}\binom{n-i}{h}$ $\quad$ $\binom{n}{i} = \frac{n-i+1}{i}\binom{n}{i-1}$

$\sum_{i=0}^{n} i\binom{n}{i} = n2^{n-1}$ $\quad$ $\sum_{i=0}^{n} i^2\binom{n}{i} = (n+n^2)2^{n-2}$

6.13.2 Series

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

$\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$ $\quad$ $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$

$\sum_{i=0}^{n} z^i = \frac{1-z^{n+1}}{1-z}$ $\quad$ $\sum_{i=0}^{\infty} \frac{z^i}{i!} = e^z$

6.13.3 Sine and Cosine Rule

$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c}$

$c^2 = a^2 + b^2 - 2bc\cos\gamma$

7 Misc

7.1 Comparable

```
static class test implements Comparable<test> {
    public int x;

    test(int _x) {
        x = _x;
    }

    public int compareTo(test o) {
        return Integer.compare(o.x, x);
    }
}
```

7.2 Order

```
static class Order implements Comparator<Integer> {
    public int compare(Integer x, Integer y) {
        return -1 * Integer.compare(x, y);
    }
}
```

7.3 Knuth-Morris-Pratt (String Matching)

```
// Find all occurences of p within s
static ArrayList<Integer> KMP(String p, String s) {
    int n = p.length();
    int h = s.length();
    ArrayList<Integer> res = new ArrayList<>();

    int[] prefix = new int[n + 1];
    prefix[0] = -1;
    int q = -1;
    for (int i = 0; i < n; i++) {
        while (q >= 0 && p.charAt(q) != p.charAt(i)) {
            q = prefix[q];
        }
        q++;
        prefix[i + 1] = q;
    }
    q = 0;
    for (int i = 0; i < h; i++) {
        while (q >= 0 && p.charAt(q) != s.charAt(i)) {
            q = prefix[q];
        }
        q++;
        if (q == n) {
            res.add(i - n + 1);
            q = prefix[q];
        }
    }
    return res;
}
```

7.4 Binary distance (number of differing bits)

```
public static int difference(int num1, int num2) {
    int count = 0;
    int xor = num1 ^ num2;
    while (xor != 0) {
        count++;
        xor &= xor - 1;
    }
    return count;
}
```

7.5 Cycle Finding

```
static int[] TH(int x0) {
    int power = 1, len = 1;
    int t = x0;
    int h = f(x0);
    while (t!= h) {
        if (power==len) {
            t = h;
            power *= 2;
            len = 0;
        }
        h = f(h);
        len++;
    }

    int start = 0;
```

```
    t = h = x0;
    for (int i = 0; i < len; i++) {
        h = f(h);
    }
    while (t!=h) {
        t = f(t);
        h = f(h);
        start++;
    }

    return new int[] {len, start};
}
```

7.6 Permutations of a set

Generate all permutations of a set (of integers). Call with (list, 0).

```
static HashSet<Integer[]> permute(List<Integer> arr, int k) {
    HashSet<Integer[]> set = new HashSet<>();
    for (int i = k; i < arr.size(); i++) {
        Collections.swap(arr, i, k);
        set.addAll(permute(arr, k + 1));
        Collections.swap(arr, k, i);
    }
    if (k == arr.size() - 1) {
        set.add((Integer[]) arr.toArray().clone());
    }
    return set;
}
```

7.7 Linear Equations

```
static double[] gaussElim(double[][] A, double[] b) {
    int N = b.length;
    for (int p = 0; p < N; p++) {
        int max = p;
        for (int i = p + 1; i < N; i++) {
            if (Math.abs(A[i][p]) > Math.abs(A[max][p])) {
                max = i;
        }}
        swap(A, p, max);
        swap(b, p, max);
        if (Math.abs(A[p][p]) <= EPS) {
            return null;
        }
        for (int i = p + 1; i < N; i++) {
            double alpha = A[i][p] / A[p][p];
            b[i] -= alpha * b[p];
            for (int j = p; j < N; j++) {
                A[i][j] -= alpha * A[p][j];
    }}}

    double[] x = new double[N];
    for (int i = N - 1; i >= 0; i--) {
        double sum = 0.0;
        for (int j = i + 1; j < N; j++) {
            sum += A[i][j] * x[j];
        }
        x[i] = (b[i] - sum) / A[i][i];
    }
    return x;
}

static void swap(double[][] A, int x, int y) {
    double[] tmp = A[x];
    A[x] = A[y];
    A[y] = tmp;
}
static void swap(double[] b, int x, int y) {
    double tmp = b[x];
    b[x] = b[y];
    b[y] = tmp;
}
```

7.8 2-SAT

```
static class TwoSAT {
    private List<Integer>[] g;

    public TwoSAT(int n) {
        g = new List[2 * n];
        for (int i = 0; i < g.length; i++)
            g[i] = new ArrayList<>();
    }

    public boolean[] solve() {
        int n = g.length;
        List<List<Integer>> comps = scc(g);
        int[] comp = new int[n];
        for (int i = 0; i < comps.size(); i++)
            for (int x : comps.get(i))
                comp[x] = i;
        for (int i = 0; i < n; ++i)
            if (comp[i] == comp[i ^ 1])
                return null;
        boolean[] res = new boolean[n / 2];
        for (int i = 0; i < n; i += 2)
            res[i / 2] = comp[i] < comp[i ^ 1];
        return res;
    }

    private static int b(boolean b) {
        return b ? 1 : 0;
    }

    private void ae(int i, int j) {
        g[i + 1].add(j + 1);
    }
    public void force(int i, boolean v) {
        if (v) ae(i * 2, i * 2 - 1);
        else ae(i * 2 - 1, i * 2);
    }
    public void not(int i, boolean bi, int j, boolean bj) {
        ae(i * 2 - b(bi), j * 2 - b(!bj));
        ae(j * 2 - b(bj), i * 2 - b(!bi));
    }
}
```

7.9 Nim and Combinatorial Game Theory

7.9.1 Normal play - player who cannot move loses

Nim Given a number n of heaps, 2 players take turns removing any number of beads from any heap. A position is winning if and only if the xor of the heap sizes is nonzero.

Grundy number Suppose only certain amounts of beads are allowed to be removed. Then use the Grundy number: G(0) = 0 and G(pos) is the minimum excluded number (mex) among Grundy numbers of positions reachable from pos. A position is winning if and only the xor of the Grundy numbers of the heaps is nonzero.

Sprague-Grundy Theorem Any two-player impartial (both players have the same available moves) sequential (players take turns) game with perfect information is equivalent to a Grundy number.

7.9.2 Misère play - player who cannot move wins

Nim A position is winning if and only if the xor of the heap sizes is nonzero – unless all heaps have size one, then it is the opposite. (So in the second case, a position is winning if and only if the number of heaps is even).

Other misère games If there is only one heap, but limited move options, just do dynamic programming: dp[0] = true (ending on 0 wins) and dp[pos] = true iff you can reach a losing position from pos.

There is no Sprague-Grundy Theorem version for misère.