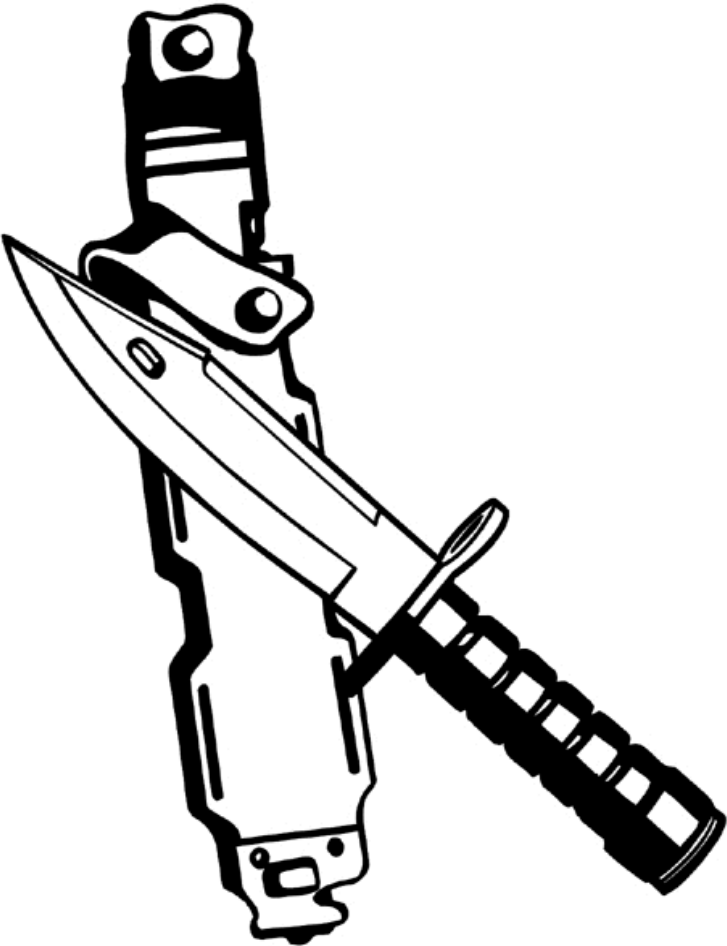


Contents

<b>1</b>	<b>Data Structures</b>	<b>2</b>
1.1	Union Find Disjoint Set - Kruskal	2
1.2	Segment Tree, RSQ, RMQ	2
1.3	Segment Tree, 2D-RMQ	3
1.4	Static RMQ, Lookup Table	4
1.5	Fenwick Tree, Inversions	4
1.6	Fenwick Tree, 2D	5
1.7	Fenwick Tree, Extended	5
1.8	KDTree	5
<b>2</b>	<b>Dynamic Programming</b>	<b>6</b>
2.1	Classic DP problems	6
2.2	Edit Distance	7
<b>3</b>	<b>Graph Algorithms</b>	<b>7</b>
3.1	Articulation Points, Bridges	7
3.2	Strongly Connected Component, Kosaraju	7
3.3	2-Satisfiability (2SAT)	8
3.4	Shortest Path, Dijkstra	9
3.5	LCA Tree Distance	9
3.6	Graphic Sequence	10
3.7	Floyd Warshal, Print Path	10
3.8	Max Flow, Edmonds Karp	10
3.9	Max Flow, Dinic	11
3.10	Min Cut	11
3.11	MaxCardinalityBipartiteMatching, Alternating path	11
3.12	Min Cost Max Flow	12
3.13	Min Cost Matching	12
<b>4</b>	<b>Mathematics and Geometry</b>	<b>13</b>
4.1	Prime Numbers, Factoring	13
4.2	Extended Euclid	14
4.3	Number Theory General	14
4.4	Gauss-Jordan Elimination	14
4.5	BigInteger Square in java	15
4.6	Geometry 1	15
4.7	Geometry 2	17
4.8	Convex Hull	18
4.9	Convex Hull Diameter	19
4.10	Great Circle Distance	19
<b>5</b>	<b>String Algorithms</b>	<b>19</b>
5.1	Suffix Array 1	19
5.2	Suffix Array 2	20
5.3	Prefix Function	20
5.4	Infix to Postfix	21
5.5	Knuth-Morris-Pratt (KMP)	21
5.6	Simple Parser	21
5.7	Longest Palindrome	22
<b>6</b>	<b>Miscellaneous</b>	<b>22</b>
6.1	Notes	22



# 1 Data Structures

## 1.1 Union Find Disjoint Set - Kruskal

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <string>

using namespace std;

typedef vector<int> vi;

struct DisJointSet {
    vi par, rnk, cnt; int numOfSets;

    DisJointSet(int n = 0) {
        par.assign(n, -1); rnk.assign(n, 0); cnt.assign(n, 1); //par==parent
        numOfSets=n; // if we wanna count number of disjoint sets
    }

    int find(int a) {
        int i=a, j=a, tmp;
        while (par[i] != -1) { i=par[i]; }
        while (par[j] != -1) { tmp=par[j]; par[j]=i; j=tmp; } //path compression
        return i;
    }

    int uni(int a, int b) {
        int A=find(a), B=find(b);
        if (A!=B) {
            if (rnk[A]<rnk[B]) swap(A,B); // union using rank
            if (rnk[A]==rnk[B]) rnk[A]++;
            par[B]=A;
            cnt[A]+=cnt[B]; // if we wanna count each set size
            numOfSets--; // if we wanna count number of disjoint sets
        }
        return cnt[A]; // if we wanna count each set size
    }
};

struct Edge { int u, v, w;
    Edge(int u=0, int v=0, int w=0):u(u), v(v), w(w){}
    bool operator<(const Edge& b) const { return w < b.w; }
    string toString() {
        stringstream sstr;
        sstr << u << ", " << v << ", " << w;
        string str; sstr >> str;
        return str;
    } //remove
};

typedef vector<Edge> ve;

struct Kruskal {
    ve edges; vi marked; DisJointSet st;
    Kruskal(int n, ve& edges):edges(edges) { st = DisJointSet(n); }
    int run() { int result; sort(edges.begin(), edges.end());
        for (int i=0; i<edges.size(); i++) { Edge e = edges[i];
            if (st.find(e.u) != st.find(e.v)) {
                st.uni(e.u, e.v); result += e.w; marked.push_back(i);
            }
        }
        return result;
    }

    void printSelectedEdges() {
        cout << "MST edges:" << endl;
        for (int i=0; i<marked.size(); i++) {
            Edge e = edges[marked[i]];
            cout << e.toString() << endl;
        } // remove
    }
};

int main() {
    int n, m;
    cin >> n >> m;

    ve edges;
    for (int i=0; i<m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        edges.push_back(Edge(u, v, w));
    }

    for (int i=0; i<edges.size(); i++) {
        cout << edges[i].toString() << endl;
    }
}
```

```
Kruskal kruskal(n, edges);
cout << kruskal.run() << endl;
kruskal.printSelectedEdges();

}

/*
IN:
5 6
1 3 5
4 5 0
2 1 3
3 2 1
4 3 4
4 2 2

OUT:
4,5,0
3,2,1
4,2,2
2,1,3

*/
```

## 1.2 Segment Tree, RSQ, RMQ

```
/******
 * Compilation:  javac SegmentTree.java
 * Execution:    java SegmentTree
 * <p/>
 * A segment tree data structure.
 *****/

import java.util.Arrays;

class SegmentTree {

    private Node[] heap;
    private int[] array;
    private int size;

    public SegmentTree(int[] array) {
        this.array = Arrays.copyOf(array, array.length);
        //The max size of this array is about 2 * 2 log2(n) + 1
        size = (int) (2 * Math.pow(2.0, Math.floor((Math.log((double) array.length) / Math.log(2.0)) +
            1)));
        heap = new Node[size];
        build(1, 0, array.length);
    }

    public int size() {
        return array.length;
    }

    //Initialize the Nodes of the Segment tree
    private void build(int v, int from, int size) {
        heap[v] = new Node();
        heap[v].from = from;
        heap[v].to = from + size - 1;

        if (size == 1) {
            heap[v].sum = array[from];
            heap[v].min = array[from];
            heap[v].minId = from;
        } else {
            //Build childs
            build(2 * v, from, size / 2);
            build(2 * v + 1, from + size / 2, size - size / 2);

            heap[v].sum = heap[2 * v].sum + heap[2 * v + 1].sum;
            heap[v].min = Math.min(heap[2 * v].min, heap[2 * v + 1].min); //min = min of the children
            heap[v].minId = (heap[2 * v].min < heap[2 * v + 1].min ? heap[2 * v].minId : heap[2 * v +
                1].minId);
        }
    }

    public int rsq(int from, int to) {
        return rsq(1, from, to);
    }

    private int rsq(int v, int from, int to) {
        Node n = heap[v];

        //If you did a range update that contained this node, you can infer the Sum without going down
        the tree
    }
}
```

```

    if (n.pendingVal != null && contains(n.from, n.to, from, to)) {
        return (to - from + 1) * n.pendingVal;
    }

    if (contains(from, to, n.from, n.to)) {
        return heap[v].sum;
    }

    if (intersects(from, to, n.from, n.to)) {
        propagate(v);
        int leftSum = rsq(2 * v, from, to);
        int rightSum = rsq(2 * v + 1, from, to);

        return leftSum + rightSum;
    }

    return 0;
}

public int rMinQ(int from, int to) {
    return rMinQ(1, from, to);
}

private int rMinQ(int v, int from, int to) {
    Node n = heap[v];

    //If you did a range update that contained this node, you can infer the Min value without
    //going down the tree
    if (n.pendingVal != null && contains(n.from, n.to, from, to)) {
        return n.pendingVal;
    }

    if (contains(from, to, n.from, n.to)) {
        return heap[v].min;
    }

    if (intersects(from, to, n.from, n.to)) {
        propagate(v);
        int leftMin = rMinQ(2 * v, from, to);
        int rightMin = rMinQ(2 * v + 1, from, to);

        return Math.min(leftMin, rightMin);
    }

    return Integer.MAX_VALUE;
}

public int rMinIdQ(int from, int to) {
    return rMinIdQ(1, from, to);
}

private int rMinIdQ(int v, int from, int to) {
    Node n = heap[v];

    //If you did a range update that contained this node, you can infer the Min value without
    //going down the tree
    if (n.pendingVal != null && contains(n.from, n.to, from, to)) {
        return n.pendingVal;
    }

    if (contains(from, to, n.from, n.to)) {
        return heap[v].minId;
    }

    if (intersects(from, to, n.from, n.to)) {
        propagate(v);
        int leftMinId = rMinIdQ(2 * v, from, to);
        int rightMinId = rMinIdQ(2 * v + 1, from, to);
        if (leftMinId == Integer.MAX_VALUE) return rightMinId;
        if (rightMinId == Integer.MAX_VALUE) return leftMinId;

        return (array[leftMinId] < array[rightMinId] ? leftMinId : rightMinId);
    }

    return Integer.MAX_VALUE;
}

public void update(int from, int to, int value) {
    update(1, from, to, value);
}

private void update(int v, int from, int to, int value) {
    //The Node of the heap tree represents a range of the array with bounds: [n.from, n.to]
    Node n = heap[v];

    if (contains(from, to, n.from, n.to)) {
        change(n, value);
    }

    if (n.size() == 1) return;

```

```

    if (intersects(from, to, n.from, n.to)) {
        propagate(v);

        update(2 * v, from, to, value);
        update(2 * v + 1, from, to, value);

        n.sum = heap[2 * v].sum + heap[2 * v + 1].sum;
        n.min = Math.min(heap[2 * v].min, heap[2 * v + 1].min);
        n.minId = (heap[2 * v].min < heap[2 * v + 1].min ? heap[2 * v].minId : heap[2 * v + 1].minId);
    }
}

//Propagate temporal values to children
private void propagate(int v) {
    Node n = heap[v];

    if (n.pendingVal != null) {
        change(heap[2 * v], n.pendingVal);
        change(heap[2 * v + 1], n.pendingVal);
        n.pendingVal = null; //unset the pending propagation value
    }
}

//Save the temporal values that will be propagated lazily
private void change(Node n, int value) {
    n.pendingVal = value;
    n.sum = n.size() * value;
    n.min = value;
    n.minId = n.from;
    array[n.from] = value;
}

//Test if the range1 contains range2
private boolean contains(int from1, int to1, int from2, int to2) {
    return from2 >= from1 && to2 <= to1;
}

//check inclusive intersection, test if range1[from1, to1] intersects range2[from2, to2]
private boolean intersects(int from1, int to1, int from2, int to2) {
    return from1 <= from2 && to1 >= from2 // (. [...] or (. [...] ..)
    || from1 >= from2 && from1 <= to2; // [. [...] or [. [...] ..)
}

//The Node class represents a partition range of the array.
static class Node {
    int sum;
    int min;
    int minId;
    //Here We store the value that will be propagated lazily
    Integer pendingVal = null;
    int from;
    int to;

    int size() {
        return to - from + 1;
    }
}

public class Main {
    public static void main(String[] args) {
        int[] a = new int[]{2, 3, 5, 1, 8, 4, 10};

        SegmentTree segmentTree = new SegmentTree(a);
        System.out.println(segmentTree.rsq(1, 4));
        System.out.println(segmentTree.rMinQ(1, 4));
        System.out.println(segmentTree.rMinIdQ(1, 4));

        segmentTree.update(3, 3, 4);
        System.out.println(segmentTree.rsq(1, 4));
        System.out.println(segmentTree.rMinQ(1, 4));
        System.out.println(segmentTree.rMinIdQ(1, 4));
    }
}

```

## 1.3 Segment Tree, 2D-RMQ

```

import java.util.*;

public class SegmentTree2D {
    public static int max(int[][] t, int x1, int y1, int x2, int y2) {
        int n = t.length >> 1;
        x1 += n;
        x2 += n;

```

```

int m = t[0].length >> 1;
y1 += m;
y2 += m;
int res = Integer.MIN_VALUE;
for (int lx = x1, rx = x2; lx <= rx; lx = (lx + 1) >> 1, rx = (rx - 1) >> 1)
    for (int ly = y1, ry = y2; ly <= ry; ly = (ly + 1) >> 1, ry = (ry - 1) >> 1) {
        if ((lx & 1) != 0 && (ly & 1) != 0) res = Math.max(res, t[lx][ly]);
        if ((lx & 1) != 0 && (ry & 1) == 0) res = Math.max(res, t[lx][ry]);
        if ((rx & 1) == 0 && (ly & 1) != 0) res = Math.max(res, t[rx][ly]);
        if ((rx & 1) == 0 && (ry & 1) == 0) res = Math.max(res, t[rx][ry]);
    }
return res;
}

public static void add(int[][] t, int x, int y, int value) {
    x += t.length >> 1;
    y += t[0].length >> 1;
    t[x][y] += value;
    for (int tx = x; tx > 0; tx >>= 1)
        for (int ty = y; ty > 0; ty >>= 1) {
            if (tx > 1) t[tx >> 1][ty] = Math.max(t[tx][ty], t[tx >> 1][ty]);
            if (ty > 1) t[tx][ty >> 1] = Math.max(t[tx][ty], t[tx][ty >> 1]);
        }
}

public static void main(String[] args) {
    int[][] t = new int[sx * 2][sy * 2];
    add(t, x, y, v); // tree-x-y-value
    int res1 = max(t, x1, y1, x2, y2); // t-[x1,y1]*[x2,y2]
}

```

## 1.4 Static RMQ, Lookup Table

```

// keep code simple.
int lookup[MAX][LOGMAX];

struct Query
{
    int L, R;
};

void preprocess(int arr[], int n)
{
    // Initialize M for the intervals with length 1
    for (int i = 0; i < n; i++)
        lookup[i][0] = i;

    for (int j = 1; (1 << j) <= n; j++)
    {
        for (int i = 0; (i + (1 << j) - 1) < n; i++)
        {
            if (arr[lookup[i][j - 1]] < arr[lookup[i + (1 << (j - 1))][j - 1]])
                lookup[i][j] = lookup[i][j - 1];
            else
                lookup[i][j] = lookup[i + (1 << (j - 1))][j - 1];
        }
    }

    // Returns minimum of arr[L..R]
    int query(int arr[], int L, int R)
    {
        int j = (int)log2(R - L + 1);

        if (arr[lookup[L][j]] <= arr[lookup[R - (int)pow(2, j) + 1][j]])
            return arr[lookup[L][j]];

        else return arr[lookup[R - (int)pow(2, j) + 1][j]];
    }
}

void RMQ(int arr[], int n, Query q[], int m)
{
    // Fills table lookup[n][Log n]
    preprocess(arr, n);

    for (int i = 0; i < m; i++)
    {
        // Left and right boundaries of current range
        int L = q[i].L, R = q[i].R;
        // Print sum of current query range
        cout << "Minimum of [" << L << ", "
              << R << "] is " << query(arr, L, R) << endl;
    }
}

```

```

}

int main()
{
    int a[] = { 7, 2, 3, 0, 5, 10, 3, 12, 18 };
    int n = sizeof(a) / sizeof(a[0]);
    Query q[] = { { 0, 4 }, { 4, 7 }, { 7, 8 } };
    int m = sizeof(q) / sizeof(q[0]);
    RMQ(a, n, q, m);
    return 0;
}

```

## 1.5 Fenwick Tree, Inversions

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <cstdio>

using namespace std;

typedef long long int64;
typedef vector<int64> vi;

// vector (vi), iostream, algo,
#define LSONe(i) (i & (-i))
struct FenwickTree {
    vi ft; FenwickTree() {}
    FenwickTree(int n) { ft.assign(n + 1, 0); } // init n + 1 zeroes
    int rsq(int b) { // returns RSQ(1, b), pass b >= 1
        int sum = 0; for (; b; b -= LSONe(b)) sum += ft[b];
        return sum;
    }
    int rsq(int a, int b) { // returns RSQ(a, b)
        return rsq(b) - (a == 1 ? 0 : rsq(a - 1));
    }
    // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
    void update(int k, int v) { // note: n = ft.size() - 1
        for (; k < (int)ft.size(); k += LSONe(k)) ft[k] += v;
    }
};

int main() {
    int f[] = { 2, 4, 5, 5, 6, 6, 6, 7, 7, 8, 9 }; // m = 11 scores
    FenwickTree ft(10); // declare a Fenwick Tree for range [1..10]
    // insert these scores manually one by one into an empty Fenwick Tree
    for (int i = 0; i < 11; i++) ft.update(f[i], 1); // this is O(k log n)
    printf("%d\n", ft.rsq(1, 1)); // 0 => ft[1] = 0
    printf("%d\n", ft.rsq(1, 2)); // 1 => ft[2] = 1
    printf("%d\n", ft.rsq(1, 6)); // 7 => ft[6] + ft[4] = 5 + 2 = 7
    printf("%d\n", ft.rsq(1, 10)); // 11 => ft[10] + ft[8] = 1 + 10 = 11
    printf("%d\n", ft.rsq(3, 6)); // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1
    ft.update(5, 2); // update demo
    printf("%d\n", ft.rsq(1, 10)); // now 13
} // return 0;

/* extra

// get largest value with cumulative sum less than or equal to x;
// for smallest, pass x-1 and add 1 to result
int getind(int x) { // ***Change Needed***
    int idx = 0, mask = TREE_SIZE; // (must be a power of 2)
    while (mask && idx < TREE_SIZE) {
        int t = idx + mask;
        if (x >= tree[t]) { idx = t; x -= tree[t]; }
        mask >>= 1;
    }
    return idx;
}

// how to count inversions
int main() {
    vi a;
    while (cin >> n) { // count inversions, (Change Needed)
        a.assign(n, 0); b.assign(n, 0); tree.assign(n, 0);
        for (int i = 0; i < n; i++) {
            cin >> a[i]; b[i] = a[i];
        }
        sort(b.begin(), b.end());
        for (int i = 0; i < n; i++) {
            int rank = (int)(lower_bound(b.begin(), b.end(), a[i]) - b.begin());
            a[i] = rank + 1;
        }
        int64 invs = 0; // num of inversions
        for (int i = n - 1; i >= 0; i--) {
            invs += read(a[i] - 1);
        }
    }
}

```

```

        update(a[i],1);
    }
    cout << invs << endl;
}
*/

```

## 1.6 Fenwick Tree, 2D

```

public class FenwickTree2D {

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

    // sum[ (0, 0), (r, c) ]
    public static int sum(int[][] t, int r, int c) {
        int res = 0;
        for (int i = r; i >= 0; i = (i & (i + 1)) - 1)
            for (int j = c; j >= 0; j = (j & (j + 1)) - 1)
                res += t[i][j];
        return res;
    }

    // sum[ (r1, c1), (r2, c2) ]
    public static int sum(int[][] t, int r1, int c1, int r2, int c2) {
        return sum(t, r2, c2) - sum(t, r1 - 1, c2) - sum(t, r2, c1 - 1) + sum(t, r1 - 1, c1 - 1);
    }

    public static int get(int[][] t, int r, int c) {
        return sum(t, r, c, r, c);
    }

    public static void set(int[][] t, int r, int c, int value) {
        add(t, r, c, -get(t, r, c) + value);
    }

    // Usage example
    public static void main(String[] args) {
        int[][] t = new int[10][20];
        add(t, 0, 0, 1);
        add(t, 9, 19, -2);
        System.out.println(-1 == sum(t, 0, 0, 9, 19));
    }
}

```

## 1.7 Fenwick Tree, Extended

```

public class FenwickTreeExtended {

    // T[i] += value
    public static void add(int[] t, int i, int value) {
        for (; i < t.length; i |= i + 1)
            t[i] += value;
    }

    // sum[0..i]
    public static int sum(int[] t, int i) {
        int res = 0;
        for (; i >= 0; i = (i & (i + 1)) - 1)
            res += t[i];
        return res;
    }

    public static int[] createTreeFromArray(int[] a) {
        int[] res = new int[a.length];
        for (int i = 0; i < a.length; i++) {
            res[i] += a[i];
            int j = i | (i + 1);
            if (j < a.length)
                res[j] += res[i];
        }
        return res;
    }

    // sum[a..b]
    public static int sum(int[] t, int a, int b) {
        return sum(t, b) - sum(t, a - 1);
    }
}

```

```

public static int get(int[] t, int i) {
    int res = t[i];
    if (i > 0) {
        int lca = (i & (i + 1)) - 1;
        for (--i; i != lca; i = (i & (i + 1)) - 1)
            res -= t[i];
    }
    return res;
}

public static void set(int[] t, int i, int value) {
    add(t, i, -get(t, i) + value);
}

// interval add
public static void add(int[] t, int a, int b, int value) {
    add(t, a, value);
    add(t, b + 1, -value);
}

// point query
public static int get1(int[] t, int i) {
    return sum(t, i);
}

// interval add
public static void add(int[] t1, int[] t2, int a, int b, int value) {
    add(t1, a, value);
    add(t1, b, -value);
    add(t2, a, -value * (a - 1));
    add(t2, b, value * b);
}

// interval query
public static int sum(int[] t1, int[] t2, int i) {
    return sum(t1, i) * i + sum(t2, i);
}

// Returns min(p|sum[0,p]>=sum)
public static int lower_bound(int[] t, int sum) {
    -sum;
    int pos = -1;
    for (int blockSize = Integer.highestOneBit(t.length); blockSize != 0; blockSize >>= 1) {
        int nextPos = pos + blockSize;
        if (nextPos < t.length && sum >= t[nextPos]) {
            sum -= t[nextPos];
            pos = nextPos;
        }
    }
    return pos + 1;
}

// Usage example
public static void main(String[] args) {
    int[] t = new int[10];
    set(t, 0, 1);
    add(t, 9, -2);
    System.out.println(-1 == sum(t, 0, 9));

    t = createTreeFromArray(new int[] {1, 2, 3, 4, 5, 6});
    for (int i = 0; i < t.length; i++)
        System.out.print(get(t, i) + " ");
    System.out.println();
    t = createTreeFromArray(new int[] {0, 0, 1, 0, 0, 1, 0, 0});
    System.out.println(5 == lower_bound(t, 2));

    int[] t1 = new int[10];
    int[] t2 = new int[10];
    add(t1, t2, 0, 9, 1);
    add(t1, t2, 0, 0, -2);
    System.out.println(sum(t1, t2, 9));
}
}

```

## 1.8 KDTree

```

import java.util.*;

public class KdTreePointQuery {

    public static class Point {
        int x, y;

        public Point(int x, int y) {

```

```

        this.x = x;
        this.y = y;
    }
}

int[] tx;
int[] ty;

public KdTreePointQuery(Point[] points) {
    int n = points.length;
    tx = new int[n];
    ty = new int[n];
    build(0, n, true, points);
}

void build(int low, int high, boolean divX, Point[] points) {
    if (low >= high)
        return;
    int mid = (low + high) >> 1;
    nth_element(points, low, high, mid, divX);

    tx[mid] = points[mid].x;
    ty[mid] = points[mid].y;

    build(low, mid, !divX, points);
    build(mid + 1, high, !divX, points);
}

static void nth_element(Point[] a, int low, int high, int n, boolean divX) {
    while (true) {
        int k = randomizedPartition(a, low, high, divX);
        if (n < k)
            high = k;
        else if (n > k)
            low = k + 1;
        else
            return;
    }
}

static final Random rnd = new Random();

static int randomizedPartition(Point[] a, int low, int high, boolean divX) {
    swap(a, low + rnd.nextInt(high - low), high - 1);
    int v = divX ? a[high - 1].x : a[high - 1].y;
    int i = low - 1;
    for (int j = low; j < high; j++)
        if (divX ? a[j].x <= v : a[j].y <= v)
            swap(a, ++i, j);
    return i;
}

static void swap(Point[] a, int i, int j) {
    Point t = a[i];
    a[i] = a[j];
    a[j] = t;
}

long bestDist;
int bestNode;

public int findNearestNeighbour(int x, int y) {
    bestDist = Long.MAX_VALUE;
    findNearestNeighbour(0, tx.length, x, y, true);
    return bestNode;
}

void findNearestNeighbour(int low, int high, int x, int y, boolean divX) {
    if (low >= high)
        return;
    int mid = (low + high) >> 1;
    long dx = x - tx[mid];
    long dy = y - ty[mid];
    long dist = dx * dx + dy * dy;
    if (bestDist > dist) {
        bestDist = dist;
        bestNode = mid;
    }
    long delta = divX ? dx : dy;
    long delta2 = delta * delta;

    if (delta <= 0) {
        findNearestNeighbour(low, mid, x, y, !divX);
        if (delta2 < bestDist)
            findNearestNeighbour(mid + 1, high, x, y, !divX);
    }
    else {
        findNearestNeighbour(mid + 1, high, x, y, !divX);
        if (delta2 < bestDist)
            findNearestNeighbour(low, mid, x, y, !divX);
    }
}

```

```

public static void main(String[] args) {
    Point[] points = new Point[n];
    //fill points
    //build tree
    KdTreePointQuery kdTree = new KdTreePointQuery(points);

    int index = kdTree.findNearestNeighbour(qx, qy);
    Point p = points[index];
}

// ----- Maximum Subrectangle Sum
int main() {
    for (int i=1; i<n; i++) //preprocess
        for (int j=0; j<n; j++)
            a[i][j] += a[i-1][j];

    int Max=0, ans=0;
    for (int k=0; k<n; k++) { //calc
        for (int i=0; i<n-k; i++) { Max=0;
            for (int j=0; j<n; j++) {
                if (Max<0) Max=a[i+k][j]-a[i][j];
                else Max+=a[i+k][j]-a[i][j];
                if (Max>ans) ans=Max;
            }
        }
    }

    //sub array, finsh and start point p=(val, startidx, finishidx)
    p ans=p(-1,0,0); int sum=0, id=1;
    for (int i=1; i<n; i++) {
        if (sum<0) {sum=0; id=i;}
        sum+=a[i];
        p tmp=p(sum,id,i+1); ans=Max(ans,tmp);
    }
}

// ----- Optimal Array Multiplication Sequence (Print Path)
int n, a[10+5], p[10+5][10+5], dp[10+5][10+5];

int solve(int L, int R) {
    if (L==R) return 0;
    if (dp[L][R] != -1) return dp[L][R];
    int Min=INF;
    for (int i=L; i<R; i++) {
        int slv=solve(L,i)+solve(i+1,R)+a[(L-1)]*a[i]*a[R];
        if (Min>slv) Min=slv; p[L][R]=i;
    }
    return dp[L][R]=Min;
}

//prints like this => (A1 x (A2 x A3))
void print(int L, int R) {
    if (L==R) { cout << "A" <<L; return; }
    cout << "("; print(L,p[L][R]);
    cout << " x ";
    print(p[L][R]+1,R); cout << ")";
}

int main() {
    int t=1;
    while (cin >> n && n) {
        for (int i=1; i<=n; i++) cin >> a[i-1] >> a[i];
        memset(dp, -1, sizeof dp);
        solve(1,n); //cout << solve(1,n) << endl;
        printf("Case %d: ", t++); print(1,n); printf("\n");
    }
    return 0;
}

// ----- LIS
int main() {
    vector<int> v;
    v.push_back(1);
    for (int i=0; i<n; i++) {
        int x = dolls[i].w; // array element
        int id = lower_bound(v.begin(), v.end(), x + 1) - v.begin();

        if (id == v.size() - 1) v.push_back(x); v[id] = x;
    }
    cout << v.size() - 1 << endl;
}

```

## 2 Dynamic Programming

### 2.1 Classic DP problems

```
// ----- LCS
int main() {
    dp[MAX][MAX] = {0};
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i-1] == b[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
            else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }
    cout << dp[n][m] << endl;

// ----- TSP
p a[15]; int n, dp[15][1<<15];

int solve(int pos, int bitset) {
    int& dpp = dp[pos][bitset]; //dpp = dp pointer
    if (bitset == (1<<n)-1) return dist(a[pos], a[0]);
    if (dpp != -1) return dpp;
    dpp = INF;
    for (int i = 0; i < n; i++) {
        if (!(bitset & (1<<i))) dpp = min(dpp, solve(i, bitset | (1<<i)) + dist(a[pos], a[i]));
    }
    return dpp;
}

int main() {
    int tc; cin >> tc;
    while (tc--) {
        cin >> a[0].X >> a[0].Y; cin >> n; n++;
        for (int i = 1; i < n; i++) cin >> a[i].X >> a[i].Y;
        memset(dp, -1, sizeof dp);
        cout << solve(0, 1) << endl;
    }
    return 0;
}
```

## 2.2 Edit Distance

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <string>
#define MAX (100+10)

using namespace std;

string str1, str2; int d[MAX][MAX]; char path[MAX][MAX];

int Step, DelCnt, InsCnt;
void PrintPath(int i, int j) {
    //cout << path[i][j] << endl;
    if (path[i][j] == 0) PrintPath(i-1, j-1);
    else if (path[i][j] == 'd') {
        PrintPath(i-1, j);
        printf("%d Delete %d\n", Step++, i + InsCnt - DelCnt);
        DelCnt++;
    }
    else if (path[i][j] == 'i') {
        PrintPath(i, j-1);
        printf("%d Insert %d,%c\n", Step++, j, str2[j]);
        InsCnt++;
    }
    else if (path[i][j] == 'r') {
        PrintPath(i-1, j-1);
        printf("%d Replace %d,%c\n", Step++, i + InsCnt - DelCnt, str2[j]);
    }
}

int solve() {
    memset(d, 0, sizeof d); memset(path, 0, sizeof path);
    for (int i = 1; i <= str1.length(); i++) {
        d[i][0] = i; path[i][0] = 'd';
    }
    for (int j = 1; j <= str2.length(); j++) {
        d[0][j] = j; path[0][j] = 'i';
    }
    int del, ins, rplc;
    for (int i = 1; i <= str1.length(); i++) {
        for (int j = 1; j <= str2.length(); j++) {
            del = d[i-1][j] + 1;
            ins = d[i][j-1] + 1;
            rplc = d[i-1][j-1] + (str1[i] != str2[j]);
            d[i][j] = min(min(del, ins), rplc);

            if (d[i][j] == del) path[i][j] = 'd';
            else if (d[i][j] == ins) path[i][j] = 'i';
            else if (str1[i] != str2[j]) path[i][j] = 'r';
        }
    }
}
```

```

    }
    return d[str1.length()][str2.length()];
}

int main() {
    int t = 1;
    while (getline(cin, str1) && getline(cin, str2)) {
        if (t++ != 1) cout << endl; str1 = ' ' + str1; str2 = ' ' + str2;
        cout << solve() << endl; Step = 1; InsCnt = DelCnt = 0;
        PrintPath(str1.length(), str2.length());
    }
}
```

## 3 Graph Algorithms

### 3.1 Articulation Points, Bridges

```
int n, lev, dfsRoot, rootChilds;
int dfsLow[MAX], dfsNum[MAX], parent[MAX];
vvi adj; set<pii> bridges; set<int> artPoints;

void dfs(int u) {
    dfsLow[u] = dfsNum[u] = lev++;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (dfsNum[v] == 0) {
            if (u == dfsRoot) rootChilds++;
            parent[v] = u; dfs(v);

            if (dfsLow[v] >= dfsNum[u] && u != dfsRoot) //u is articulation point
                artPoints.insert(u);

            if (dfsLow[v] > dfsNum[u]) {
                bridges.insert(pii(v, u));
                bridges.insert(pii(u, v));
            }

            dfsLow[u] = min(dfsLow[u], dfsLow[v]);
        }
        else if (parent[u] != v)
            dfsLow[u] = min(dfsLow[u], dfsNum[v]);
    }
}

int main() {
    while (cin >> n) {
        adj.assign(n, vi()); //initialization
        memset(dfsLow, 0, sizeof dfsLow);
        memset(dfsNum, 0, sizeof dfsNum);
        memset(parent, 0, sizeof parent);
        bridges.clear(); artPoints.clear();
        lev = 1; int tmp, u, m;
        for (int i = 0; i < n; i++) { //construct the graph
            scanf("%d %d", &u, &m); cin.ignore();
            for (int i = 0; i < m; i++) {
                cin >> tmp; adj[u].push_back(tmp);
            }
        }
        for (int i = 0; i < n; i++) {
            if (dfsNum[i] == 0) {
                dfsRoot = i; rootChilds = 0; dfs(i);
                if (rootChilds >= 2) artPoints.insert(dfsRoot);
            }
        }
        printf("%d critical links\n", bridges.size());
        set<pii>::iterator itr; //print answer
        for (itr = bridges.begin(); itr != bridges.end(); itr++)
            printf("%d - %d\n", itr->first, itr->second);
        cout << endl;
    }
    return 0;
}
```

### 3.2 Strongly Connected Component, Kosaraju

```
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
```

```

#include <map>

using namespace std;

typedef vector<int> vi;
typedef vector<vi > vvi;

map<string, int> mpsi;
map<int, string> mpis;

int mapId;
int ID(string name){
    if(mpsi.count(name)) return mpsi[name];
    mpsi[name] = mapId;
    mpis[mapId] = name;
    return mapId++;
}

vvi adjOrg, adjRev; vi vis, ord, col; int n, m;

void dfsOrg(int u){
    if (vis[u]) return; vis[u] = true;
    for (int i = 0; i < adjOrg[u].size(); i++){
        dfsOrg(adjOrg[u][i]);
    }
    ord.push_back(u);
}

int dfsRev(int u, int color){
    if (col[u]) return 0; col[u] = color;

    int ret = 1;
    for (int i = 0; i < adjRev[u].size(); i++){
        ret += dfsRev(adjRev[u][i], color);
    }
    return ret;
}

int main(){
    int cs = 1;
    while (cin >> n >> m && (n+m)) {
        mpsi.clear(); mpis.clear(); mapId = 0;

        adjOrg.assign(n, vi());
        adjRev.assign(n, vi());

        for (int i=0; i<m; i++){
            string uname, vname;
            cin >> uname >> vname;
            int u = ID(uname);
            int v = ID(vname);
            adjOrg[u].push_back(v);
            adjRev[v].push_back(u);
        }

        ord.clear();
        vis.assign(n, 0);
        for (int u = 0; u < n; u++){
            if (!vis[u]) dfsOrg(u);
        }

        int color = 1;
        col.assign(n, 0);
        while (!ord.empty()){
            int u = ord.back();
            if (!col[u]){
                int size = dfsRev(u, color); // SCC Size
                color++;
            }
            ord.pop_back();
        }
        if (cs != 1) cout << endl;
        cout << "Calling circles for data set " << cs++ << ":" << endl;
        for (int c = 1; c<color; c++) {
            string ws = "";
            for (int u=0; u<n; u++) {
                if (col[u] == c) cout << ws << mpis[u], ws=" ",
            }
            cout << endl;
        }
    }
}

```

```

#include <cstdio>
#include <vector>
#include <algorithm>

using namespace std;

#define FOR(i,x,y) for(int i = (x) ; i <= (y) ; ++i)
#define SZ(x) ((int) (x).size())
#define ALL(x) (x).begin(), (x).end()
#define UNIQUE(V) (V).erase(unique((V).begin(), (V).end()), (V).end())

typedef vector<int> vi;
typedef vector<vi > vvi;

struct Episode {
    int s, e;
    Episode(int s=0, int e=0): s(s), e(e) {}
};

#define SCCNODE 1000
struct SCC{
    int num[SCCNODE], low[SCCNODE], col[SCCNODE], cycle[SCCNODE], st[SCCNODE];
    int tail, cnt, cc;
    vi adj[SCCNODE];

    SCC():tail(0),cnt(0),cc(0) {}
    void clear ( int n ) {
        cc += 3;
        FOR(i,0,n) adj[i].clear();
        tail = 0;
    }

    void tarjan ( int s ) {
        num[s] = low[s] = cnt++;
        col[s] = cc + 1;
        st[tail++] = s;
        FOR(i,0,SZ(adj[s])-1) {
            int t = adj[s][i];
            if ( col[t] <= cc ) {
                tarjan ( t );
                low[s]=min(low[s],low[t]);
            }
            /*Back edge*/
            else if (col[t]==cc+1)
                low[s]=min(low[s],low[t]);
        }
        if ( low[s] == num[s] ) {
            while ( 1 ) {
                int temp=st[tail-1];
                tail--;
                col[temp] = cc + 2;
                cycle[temp] = s;
                if ( s == temp ) break;
            }
        }
    }

    void shrink( int n ) {
        FOR(i,0,n){
            FOR(j,0,SZ(adj[i])-1){
                adj[i][j] = cycle[adj[i][j]]; ///Careful. This will create self-loop
            }
        }
        FOR(i,0,n){
            if ( cycle[i] == i ) continue;
            int u = cycle[i];
            FOR(j,0,SZ(adj[i])-1){
                int v = adj[i][j];
                adj[u].push_back( v );
            }
            adj[i].clear();
        }
        FOR(i,0,n){ ///Not always necessary
            sort ( ALL(adj[i]) );
            UNIQUE(adj[i]);
        }
    }

    void findSCC( int n ) {
        FOR(i,0,n) {
            if ( col[i] <= cc ) {
                tarjan ( i );
            }
        }
    }
};

#define SAT2NODE 2000
/*
1. The nodes need to be split. So change convert() accordingly.
2. Using clauses, populate scc edges.
3. Call possible, to find if a valid solution is possible or not.
4. Dont forget to keep space for !A variables
*/
struct SAT2 {

```

### 3.3 2-Satisfiability (2SAT)

```
#include <iostream>
```



```

SCC scc;

SAT2(): bfscc(1) {}
void clear( int n ) { scc.clear( n ); }
int convert ( int n ) { return n * 2; } ///Change here. Depends on how input is provided
void mustTrue ( int a ) { scc.adj[a^1].push_back( a ); } ///A is True
void andClause ( int a, int b ) { scc.adj[a].push_back( b ); scc.adj[b].push_back( a ); }
void orClause ( int a, int b ) { scc.adj[a^1].push_back( b ); scc.adj[b^1].push_back( a ); } /// A
|| B clause ///!a->b !b->a
void xorClause ( int a, int b ) { orClause( a, b ); orClause( a^1, b^1 ); }
void notAndClause ( int a, int b ) { scc.adj[a].push_back( b ^ 1 ); scc.adj[b].push_back( a ^ 1 ); }
void atMostOneClause ( int a[], int n, int flag ) { /// Out of all possible option, only one is
    true
    if ( flag == 0 ) { /// At most one can be false
        FOR(i,0,n){
            a[i] = a[i] ^ 1;
        }
        FOR(i,0,n) {
            FOR(j,i+1,n) {
                orClause( a[i] ^ 1, a[j] ^ 1 ); /// !a || !b both being true not allowed
            }
        }
    }

    ///Send n, total number of nodes, after expansion
    bool possible( int n ) {
        scc.findSCC( n );

        FOR(i,0,n) {
            int a = i, b = i^1;
            ///Falls on same cycle a and !a.
            if ( scc.cycle[a] == scc.cycle[b] ) return false;
        }

        ///Valid solution exists
        return true;
    }

    ///To determine if A can be true. It cannot be true, if a path exists from A to !A.
    int vis[SAT2NODE], qqq[SAT2NODE], bfscc;
    void bfs( int s ) {
        bfscc++;
        int qs = 0, qt = 0;
        vis[s] = bfscc;
        qqq[qt++] = s;
        while ( qs < qt ) {
            s = qqq[qs++];
            FOR(i,0,SZ(scc.adj[s])-1) {
                int t = scc.adj[s][i];
                if ( vis[t] != bfscc ) {
                    vis[t] = bfscc;
                    qqq[qt++] = t;
                }
            }
        }
    }
} sat2;

vector< Episode > ep;
void build_graph(int l, int r) {
    for(int i=1; i<=r; i+=2) {
        sat2.xorClause(sat2.convert(i), sat2.convert(i+1));
    }

    for(int i=1; i<=r; i++) {
        for(int j=i+1; j<=r; j++) {
            if(ep[i].e < ep[j].s || ep[j].e < ep[i].s) continue;
            sat2.notAndClause(sat2.convert(i), sat2.convert(j));
        }
    }
}

int main () {
    int tc;
    cin >> tc;
    while(tc--) {
        int n;
        cin >> n;
        n = n * 2;

        ep.assign(n, Episode());
        for(int i=0; i<n; i++) cin >> ep[i].s >> ep[i].e;

        int ansL = -1, ansR = -1, maxDist = -1;

        int l = 0;
        for (int r = 1; r<n; r+=2) {

```

```

        sat2.clear(n * 2);
        build_graph(l, r);

        while(sat2.possible(n * 2) == false) {
            l+=2;
            sat2.clear(n * 2);
            build_graph(l, r);
        }

        if (r - l > maxDist) {
            ansR = r/2;
            ansL = l/2;
            maxDist = r - l;
        }

        cout << ansL + 1 << " " << ansR + 1 << endl;
    }
}

```

## 3.4 Shortest Path, Dijkstra

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const int INF = 1e8;

struct ToNode{
    int v, w;
    ToNode(int v, int w)
        :v(v), w(w){}
};

struct QEntry{
    int node, cost;
    QEntry(int node, int cost)
        :node(node), cost(cost){}
    bool operator<(const QEntry& op) const {
        return cost < op.cost;
    }
};

typedef vector<int> vi;
typedef vector<ToNode> vtn;
typedef vector<vtn > vvtvn;

int n, m; vvtvn adj;

int dijkstra(int s, int t, vi& dist){
    dist.assign(n, INF);
    priority_queue<QEntry> q;
    q.push(QEntry(s, 0)); dist[s] = 0;

    while (!q.empty()){
        QEntry u = q.top(); q.pop();
        if (u.node == t) return u.cost;
        if (u.cost > dist[u.node]) continue;
        for (int i = 0; i < adj[u.node].size(); i++){
            QEntry v(adj[u.node][i].v, u.cost + adj[u.node][i].w);
            if (dist[v.node] > v.cost){
                dist[v.node] = v.cost; q.push(v);
            }
        }
    }
    return INF;
}

```

## 3.5 LCA Tree Distance

```

const int max_nodes, log_max_nodes;
int num_nodes, log_num_nodes, root;

vector<int> children[max_nodes]; /// children[i] contains the children of node i
int A[max_nodes][log_max_nodes + 1]; /// A[i][j] is the 2^j-th ancestor of node i, or -1 if that
ancestor does not exist
int L[max_nodes]; /// L[i] is the distance between node i and the root

/// floor of the binary logarithm of n
int lb(unsigned int n)
{

```

```

if (n == 0)
    return -1;
int p = 0;
if (n >= 1 << 16) { n >>= 16; p += 16; }
if (n >= 1 << 8) { n >>= 8; p += 8; }
if (n >= 1 << 4) { n >>= 4; p += 4; }
if (n >= 1 << 2) { n >>= 2; p += 2; }
if (n >= 1 << 1) { p += 1; }
return p;
}

void DFS(int i, int l)
{
    L[i] = l;
    for (int j = 0; j < children[i].size(); j++)
        DFS(children[i][j], l + 1);
}

int LCA(int p, int q)
{
    // ensure node p is at least as deep as node q
    if (L[p] < L[q])
        swap(p, q);

    // "binary search" for the ancestor of node p situated on the same level as q
    for (int i = log_num_nodes; i >= 0; i--)
        if (L[p] - (1 << i) >= L[q])
            p = A[p][i];

    if (p == q)
        return p;

    // "binary search" for the LCA
    for (int i = log_num_nodes; i >= 0; i--)
        if (A[p][i] != -1 && A[p][i] != A[q][i])
        {
            p = A[p][i];
            q = A[q][i];
        }

    return A[p][0];
}

int main(int argc, char* argv[])
{
    // read num_nodes, the total number of nodes
    log_num_nodes = lb(num_nodes);

    for (int i = 0; i < num_nodes; i++)
    {
        int p;
        // read p, the parent of node i or -1 if node i is the root

        A[i][0] = p;
        if (p != -1)
            children[p].push_back(i);
        else
            root = i;
    }

    // precompute A using dynamic programming
    for (int j = 1; j <= log_num_nodes; j++)
        for (int i = 0; i < num_nodes; i++)
            if (A[i][j - 1] != -1)
                A[i][j] = A[A[i][j - 1]][j - 1];
            else
                A[i][j] = -1;

    // precompute L
    DFS(root, 0);

    return 0;
}

```

## 3.6 Graphic Sequence

```

// given a sequence of integers see if it s a sequence of degrees of graph or not.
int a[10010]; long long sum, Min;

int main() {
    int n;
    while (cin >> n && n) {
        while (cin >> i < n; i++) scanf("%d", &a[i]);
        sort(a, a+n, greater<int>());
        bool possible = true; sum = 0;
        for (int i = 0; i < n; i++) {

```

```

            sum += a[i]; Min = 0;
            for (int j = i + 1; j < n; j++) Min += min(a[j], i + 1);
            if (sum > i * (i + 1) + Min) {
                possible = false;
                break;
            }
        }
        if (!possible || sum % 2) cout << "Not possible" << endl;
        else cout << "Possible" << endl;
    }
    return 0;
}

```

## 3.7 Floyd Warshal, Print Path

```

#define MAX (100+10)

int adj[MAX][MAX], path[MAX][MAX]; int n;

void print(int i, int j) {
    if (i != j) {
        printf(" %d", i);
        print(path[i][j], j);
    }
}

int main() {
    int tc; cin >> tc;
    while (tc--) {
        cin >> n;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                adj[i][j] = 1e9; if (i == j) adj[i][j] = 0;
                path[i][j] = j; // initial parent
            }
        }

        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (adj[i][j] > adj[i][k] + adj[k][j]) {
                        adj[i][j] = adj[i][k] + adj[k][j];
                        path[i][j] = path[i][k]; // set parent
                    }
                }
            }
        }

        int s, d;
        cin >> s >> d;
        printf("%d euros\n", adj[s][d]);

        // this prints the path even if source and destination are same
        printf("%d", s); print(path[s][d], d); printf(" %d\n", d);

    }
    return 0;
}

```

## 3.8 Max Flow, Edmonds Karp

```

// UVa 820 - Internet Bandwidth
#define INF (int) 1e9
#define MAX 100+10

using namespace std;

int res[MAX][MAX], mf, f, s, t, n, m, par[MAX]; vector<int> dist, adj[MAX];

void augment(int v, int minEdge) {
    if (v == s) f = minEdge;
    else if (par[v] != -1) {
        augment(par[v], min(minEdge, res[par[v]][v]));
        res[par[v]][v] -= f; res[v][par[v]] += f;
    }
}

int main() {
    int tc = 1;
    while (cin >> n && n) {
        mf = 0; memset(res, 0, sizeof res); for (int i = 0; i < n; i++) adj[i].clear();
        cin >> s >> t >> m; s--; t--;
        int u, v, c;
        while (m--) {
            cin >> u >> v >> c; u--; v--;

```

```

    res[u][v] += c; res[v][u] += c;
    adj[u].push_back(v); adj[v].push_back(u);
}
while(1){
    f=0; memset(par,-1,sizeof par); dist.assign(n,INF);
    dist[s]=0; queue<int> q; q.push(s);
    while(!q.empty()){
        int u=q.front(); q.pop();
        if(u==t) break;
        for(int i=0; i<adj[u].size(); i++){
            int v=adj[u][i];
            if(res[u][v]>0 && dist[v]==INF){
                dist[v]=dist[u]+1; q.push(v); par[v]=u;
            }
        }
    }
    agument(t,INF);
    if(f==0) break;
    mf+=f;
}
printf("Network %d\n", tc++);
printf("The bandwidth is %d.\n\n", mf);
}
return 0;
}

```

## 3.9 Max Flow, Dinic

```

// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
// Running time:  $O(|V|^2 |E|)$ 
// INPUT:
// - graph, constructed using AddEdge() - source - sink
// OUTPUT:
// - maximum flow value
// - To obtain the actual flow values, look at all edges with
//   capacity > 0 (zero capacity edges are residual edges).
using namespace std;
const int INF = 2000000000;
struct Edge {
    int from, to, cap, flow, index;
    Edge(int from, int to, int cap, int flow, int index) :
        from(from), to(to), cap(cap), flow(flow), index(index) {}
};
struct Dinic {
    int N; vector<vector<Edge>> > G;
    vector<Edge> * dad; vector<int> Q;
    Dinic(int N) : N(N), G(N), dad(N), Q(N) {}
    void AddEdge(int from, int to, int cap) {
        G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
        if (from == to) G[from].back().index++;
        G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
    }
    long long BlockingFlow(int s, int t) {
        fill(dad.begin(), dad.end(), (Edge *)NULL);
        dad[s] = &G[0][0] - 1;
        int head = 0, tail = 0;
        Q[tail++] = s;
        while (head < tail) {
            int x = Q[head++];
            for (int i = 0; i < G[x].size(); i++) {
                Edge &e = G[x][i];
                if (!dad[e.to] && e.cap - e.flow > 0) {
                    dad[e.to] = &G[x][i];
                    Q[tail++] = e.to;
                }
            }
        }
        if (!dad[t]) return 0;
        long long totflow = 0;
        for (int i = 0; i < G[t].size(); i++) {
            Edge *start = &G[G[t][i].to][G[t][i].index];
            int amt = INF;
            for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
                if (!e) { amt = 0; break; }
                amt = min(amt, e->cap - e->flow);
            }
            if (amt == 0) continue;
            for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
                e->flow += amt;
                G[e->to][e->index].flow -= amt;
            }
            totflow += amt;
        }
        return totflow;
    }
    long long GetMaxFlow(int s, int t) {
        long long totflow = 0;
        while (long long flow = BlockingFlow(s, t))
            totflow += flow;
    }
}

```

```

    return totflow;
};

```

## 3.10 Min Cut

```

// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:  $O(|V|^3)$ 
// INPUT: graph, constructed using AddEdge()
// OUTPUT: (min cut value, nodes in half of min cut)

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
    int N = weights.size();
    VI used(N), cut, best_cut;
    int best_weight = -1;

    for (int phase = N-1; phase >= 0; phase--) {
        VI w = weights[0];
        VI added = used;
        int prev, last = 0;
        for (int i = 0; i < phase; i++) {
            prev = last;
            last = -1;
            for (int j = 1; j < N; j++)
                if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
            if (i == phase-1) {
                for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];
                for (int j = 0; j < N; j++) weights[j][prev] = weights[j][j];
                used[last] = true;
                cut.push_back(last);
                if (best_weight == -1 || w[last] < best_weight) {
                    best_cut = cut;
                    best_weight = w[last];
                }
            } else {
                for (int j = 0; j < N; j++)
                    w[j] += weights[last][j];
                added[last] = true;
            }
        }
        return make_pair(best_weight, best_cut);
    }
}

// BEGIN CUT
// The following code solves UVA problem #10989: Bomb, Divide and Conquer
int main() {
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        int n, m;
        cin >> n >> m;
        VVI weights(n, VI(n));
        for (int j = 0; j < m; j++) {
            int a, b, c;
            cin >> a >> b >> c;
            weights[a-1][b-1] = weights[b-1][a-1] = c;
        }
        pair<int, VI> res = GetMinCut(weights);
        cout << "Case #" << i+1 << ": " << res.first << endl;
    }
}
// END CUT

```

## 3.11 MaxCardinalityBipartiteMatching, Alternating path

```

#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

```

```

typedef vector<int> vi;
typedef vector<vi> vvi;

vvi adj; vi owner, vis; int n,b;

int altpath(int u){
    if(vis[u]) return 0; vis[u]=1;
    for(int i=0; i<adj[u].size(); i++){
        int v=adj[u][i];
        if(owner[v]==-1 || altpath(owner[v])){
            owner[v]=u; return 1;
        }
    }
    return 0;
}

int main(){
    int tmp,tc,t=1; cin >> tc;
    while(tc--){
        cin >> n >> b; adj.assign(n+b,vi());
        for(int i=0; i<n; i++){
            for(int j=0; j<b; j++){
                // if there is an edge from n group to b group
                cin >> tmp; if(tmp==1) adj[i].push_back(j+n);
            }
        }
        int ans=0; owner.assign(n+b,-1);
        for(int u=0; u<n; u++){
            vis.assign(n,0); ans+=altpath(u);
        }
        printf("Case %d: a maximum of %d matched\n", t++, ans);
    }
    return 0;
}

```

## 3.12 Min Cost Max Flow

```

// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972). This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]). For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
// max flow: O(|V|^3) augmentations
// min cost max flow: O(|V|^4 * MAX_EDGE_COST) augmentations
// INPUT:
// - graph, constructed using AddEdge(), source, sink
// OUTPUT:
// - (maximum flow value, minimum cost value)
// - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
    int N;
    VVL cap, flow, cost;
    VI found;
    VL dist, pi, width;
    VPII dad;

    MinCostMaxFlow(int N) :
        N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
        found(N), dist(N), pi(N), width(N), dad(N) {}

    void AddEdge(int from, int to, L cap, L cost) {
        this->cap[from][to] = cap;
        this->cost[from][to] = cost;
    }

    void Relax(int s, int k, L cap, L cost, int dir) {
        L val = dist[s] + pi[s] - pi[k] + cost;
        if (cap && val < dist[k]) {
            dist[k] = val;

```

```

        dad[k] = make_pair(s, dir);
        width[k] = min(cap, width[s]);
    }
}

L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
        int best = -1;
        found[s] = true;
        for (int k = 0; k < N; k++) {
            if (found[k]) continue;
            Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
            Relax(s, k, flow[k][s], -cost[k][s], -1);
            if (best == -1 || dist[k] < dist[best]) best = k;
        }
        s = best;
    }

    for (int k = 0; k < N; k++)
        pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
}

pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
        totflow += amt;
        for (int x = t; x != s; x = dad[x].first) {
            if (dad[x].second == 1) {
                flow[dad[x].first][x] += amt;
                totcost += amt * cost[dad[x].first][x];
            } else {
                flow[x][dad[x].first] -= amt;
                totcost -= amt * cost[x][dad[x].first];
            }
        }
    }
    return make_pair(totflow, totcost);
}

// BEGIN CUT
// The following code solves UVA problem #10594: Data Flow

int main() {
    int N, M;

    while (scanf("%d%d", &N, &M) == 2) {
        VVL v(M, VL(3));
        for (int i = 0; i < M; i++)
            scanf("%Ld%Ld%Ld", &v[i][0], &v[i][1], &v[i][2]);
        L D, K;
        scanf("%Ld%Ld", &D, &K);

        MinCostMaxFlow mcmf(N+1);
        for (int i = 0; i < M; i++) {
            mcmf.AddEdge(int(v[i][0]), int(v[i][1]), K, v[i][2]);
            mcmf.AddEdge(int(v[i][1]), int(v[i][0]), K, v[i][2]);
        }
        mcmf.AddEdge(0, 1, D, 0);

        pair<L, L> res = mcmf.GetMaxFlow(0, N);

        if (res.first == D) {
            printf("%Ld\n", res.second);
        } else {
            printf("Impossible.\n");
        }
    }

    return 0;
}

// END CUT

```

## 3.13 Min Cost Matching

```

////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path

```

```
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in around 1
// second.
//
// cost[i][j] = cost for pairing left node i with right node j
// Lmate[i] = index of right node that left node i pairs with
// Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[i][j] matrix.
//
//
#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VVD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
    int n = int(cost.size());

    // construct dual feasible solution
    VVD u(n);
    VVD v(n);
    for (int i = 0; i < n; i++) {
        u[i] = cost[i][0];
        for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
    }
    for (int j = 0; j < n; j++) {
        v[j] = cost[0][j] - u[0];
        for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
    }

    // construct primal solution satisfying complementary slackness
    Lmate = VI(n, -1);
    Rmate = VI(n, -1);
    int mated = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (Rmate[j] != -1) continue;
            if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
                Lmate[i] = j;
                Rmate[j] = i;
                mated++;
                break;
            }
        }
    }

    VD dist(n);
    VI dad(n);
    VI seen(n);

    // repeat until primal solution is feasible
    while (mated < n) {

        // find an unmatched left node
        int s = 0;
        while (Lmate[s] != -1) s++;

        // initialize Dijkstra
        fill(dad.begin(), dad.end(), -1);
        fill(seen.begin(), seen.end(), 0);
        for (int k = 0; k < n; k++)
            dist[k] = cost[s][k] - u[s] - v[k];

        int j = 0;
        while (true) {

            // find closest
            j = -1;
            for (int k = 0; k < n; k++) {
                if (seen[k]) continue;
                if (j == -1 || dist[k] < dist[j]) j = k;
            }
            seen[j] = 1;

            // termination condition
            if (Rmate[j] == -1) break;

            // relax neighbors
            const int i = Rmate[j];
            for (int k = 0; k < n; k++) {
                if (seen[k]) continue;
                const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
                if (dist[k] > new_dist) {
                    dist[k] = new_dist;

```

```
                dad[k] = j;
            }
        }

        // update dual variables
        for (int k = 0; k < n; k++) {
            if (k == j || !seen[k]) continue;
            const int i = Rmate[k];
            v[k] += dist[k] - dist[j];
            u[i] -= dist[k] - dist[j];
        }
        u[s] += dist[j];

        // augment along path
        while (dad[j] >= 0) {
            const int d = dad[j];
            Rmate[j] = Rmate[d];
            Lmate[Rmate[j]] = j;
            j = d;
        }
        Rmate[j] = s;
        Lmate[s] = j;

        mated++;
    }

    double value = 0;
    for (int i = 0; i < n; i++)
        value += cost[i][Lmate[i]];

    return value;
}
```

## 4 Mathematics and Geometry

### 4.1 Prime Numbers, Factoring

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <bitset>

using namespace std;

typedef long long int64;
const int64 MAX = 1e6 + 100;

bitset<MAX> isp; // isprime
vector<int64> primes, pfs, pws; // pfs = prime factors, pws = prime powers

void genprime() {
    isp.set(); isp[0]=isp[1]=0;
    for (int64 i=2; i<MAX; i++) {
        if (isp[i]) { primes.push_back(i);
            for (int64 j=i*i; j<MAX; j+=i) isp[j]=0;
        }
    }
}

bool isprime(int n) {
    if (n<MAX) return isp[n];
    for (int i=0; i<primes.size() && primes[i]*primes[i]<=n; i++) {
        if (n%primes[i]==0) return 0;
    }
    return 1;
}

// generation prime factors of a number
int main() {
    genprime();
    while (cin >> n) {
        // cop = euler Phi function
        // cop = coprimes = all m (m<n && gcd(m,n)=1)
        // div = divisors = all m (m<n && gcd(m,n)=m)
        for (int i=0, pf=2; pf*pf<=n; i++, pf=primes[i]) {
            int pow=0;
            while (tmp%pf==0) {
                tmp/=pf; pow++;
            }
            if (pow) {
                pfs.push_back(pf); pws.push_back(pow);
                cop -= cop/pf; div *= (pow+1);
            }
        }
        if (tmp>1) cop -= cop/tmp; div *= (1+1);
        cout << cop + div + 1 << endl;
    }
}
```

## 4.2 Extended Euclid

```
import java.util.Scanner;

public class Main {
    public static class ExtendedEuclid {
        public static int x0;
        public static int y0;
        public static int c;
        public static int d;

        public static int gcd(int a, int b) {
            if (b == 0) return a;
            return gcd(b, a % b);
        }

        public static void calculate(int a, int b) {
            if (b == 0) {
                x0 = 1;
                y0 = 0;
                d = a;
                return;
            }
            calculate(b, a % b);

            int x1 = y0;
            int y1 = x0 - (a / b) * y0;

            x0 = x1;
            y0 = y1;
        }

        public static int howManyPositiveSolutions(int a, int b, int c) {
            if (c % gcd(a, b) != 0) return 0; // no solution even negatives

            x0 *= c / gcd(a, b);
            y0 *= c / gcd(a, b); // x = x0 + (b/d)n, y = y0 - (a/d)n
            double lowerBoundForN = (double) (-x0 + 0.5) / (b / d); // for x>0
            double upperBoundForN = (double) (y0 - 0.5) / (a / d); // for y>0
            return (int) Math.max(0, Math.ceil(upperBoundForN) - Math.floor(lowerBoundForN) - 1); //
                how many int between
        }
    }

    public static void main(String[] args) {
        ExtendedEuclid.calculate(25, 18); // Copetitive Programming 2 Example
    }
}
```

## 4.3 Number Theory General

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int, int> PII;

// return a % b (positive value)
int mod(int a, int b) {
    return ((a%b) + b) % b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
    int tmp;
    while (b) { a %= b; tmp = a; a = b; b = tmp; }
    return a;
}

// computes lcm(a,b)
int lcm(int a, int b) {
    return a / gcd(a, b) * b;
}
```

```
// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a / b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x - q*xx; x = t;
        t = yy; yy = y - q*yy; y = t;
    }
    return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
    int x, y;
    VI solutions;
    int d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod(x*(b / d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n / d), n));
    }
    return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
    int x, y;
    int d = extended_euclid(a, n, x, y);
    if (d > 1) return -1;
    return mod(x, n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b. Here, z is unique modulo M = lcm(x,y).
// Return (z,M). On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid(x, y, s, t);
    if (a%d != b%d) return make_pair(0, -1);
    return make_pair(mod(s*b*x + t*a*y, x*y) / d, x*y / d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i. Note that the solution is
// unique modulo M = lcm_i (x[i]). Return (z,M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
    PII ret = make_pair(a[0], x[0]);
    for (int i = 1; i < x.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y = -1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
    int d = gcd(a, b);
    if (c%d) {
        x = y = -1;
    }
    else {
        x = c / d + mod_inverse(a / d, b / d);
        y = (c - a*x) / b;
    }
}

long conquer_fibonacci_lgN(long n) {
    long i, h, j, k, t;
    i = h = 1;
    j = k = 0;
    while (n > 0) {
        if (n % 2 == 1) {
            t = j + h;
            j = i + h + j + k + t;
            i = i + k + t;
        }
        t = h + h;
        h = 2 + k + h + t;
        k = k + k + t;
        n = (long)n / 2;
    }
    return j;
}
```

## 4.4 Gauss-Jordan Elimination

```

// Gauss-Jordan elimination with full pivoting.
// Uses:
// (1) solving systems of linear equations (AX=B)
// (2) inverting matrices (AX=I)
// (3) computing determinants of square matrices
// Running time: O(n^3)
// INPUT:  a[][] = an nxn matrix
//         b[][] = an nxm matrix
// OUTPUT: X      = an nxm matrix (stored in b[][])
//         A^-1    = an nxn matrix (stored in a[][])
//         returns determinant of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
    const int n = a.size();
    const int m = b[0].size();
    VI irow(n), icol(n), ipiv(n);
    T det = 1;

    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
        if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
        ipiv[pj]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        if (pj != pk) det *= -1;
        irow[i] = pj;
        icol[i] = pk;

        T c = 1.0 / a[pk][pk];
        det *= a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }

        for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
            for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
        }

        return det;
    }

int main() {
    const int n = 4;
    const int m = 2;
    double A[n][n] = { {1,2,3,4}, {1,0,1,0}, {5,3,2,4}, {6,1,4,6} };
    double B[n][m] = { {1,2}, {4,3}, {5,6}, {8,7} };
    VVT a(n), b(m);
    for (int i = 0; i < n; i++) {
        a[i] = VT(A[i], A[i] + n);
        b[i] = VT(B[i], B[i] + m);
    }

    double det = GaussJordan(a, b);

    // expected: 60
    cout << "Determinant: " << det << endl;

    // expected: -0.233333 0.166667 0.133333 0.066667
    //              0.166667 0.166667 0.333333 -0.333333
    //              0.233333 0.833333 -0.133333 -0.066667
    //              0.05 -0.75 -0.1 0.2
    cout << "Inverse: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }

    // expected: 1.63333 1.3

```

```

//              -0.166667 0.5
//              2.36667 1.7
//              -1.85 -1.35
cout << "Solution: " << endl;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
        cout << b[i][j] << ' ';
    cout << endl;
}

```

## 4.5 BigInteger Square in java

```

import java.math.BigInteger;
import java.util.Scanner;
//import java.util.

public class Main {
    // https://en.wikipedia.org/wiki/Integer_square_root
    public static BigInteger sqrt(BigInteger n) {
        BigInteger cur = null; // X(k)
        BigInteger nxt = n; // X(k+1)
        while(true) {
            cur = nxt;
            nxt = cur.add(n.divide(cur)).divide(BigInteger.valueOf(2));
            if(nxt.equals(cur)) break;
        }
        if(cur.multiply(cur).equals(n)) return cur;
        else return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int tc = Integer.parseInt(sc.nextLine());
        while(tc-- > 0) {
            sc.nextLine();
            BigInteger y = new BigInteger(sc.nextLine());
            if(y.equals(BigInteger.ZERO)) System.out.println(0);
            else System.out.println(sqrt(y));
            if(tc>0) System.out.println();
        }
    }
}

```

## 4.6 Geometry 1

```

// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
    double x, y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y) {}
    PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
    PT operator + (double c) const { return PT(x+c, y+c); }
    PT operator / (double c) const { return PT(x/c, y/c); }
};

double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
    os << "(" << p.x << ", " << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
    return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b

```

```

// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
    return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
    double r = dot(b-a, b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
    double a, double b, double c, double d)
{
    return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
    return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
    return LinesParallel(a, b, c, d)
        && fabs(cross(a-b, a-c)) < EPS
        && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
    if (LinesCollinear(a, b, c, d)) {
        if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
            dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
        if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
            return false;
        return true;
    }
    if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
    if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
    return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
    b=b-a; d=d-c; c=c-a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
    b=(a+b)/2;
    c=(a+c)/2;
    return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an "exact" test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
    bool c = 0;
    for (int i = 0; i < p.size(); i++){
        int j = (i+1)%p.size();
        if ((p[i].y <= q.y && q.y < p[j].y ||
            p[j].y <= q.y && q.y < p[i].y) &&
            q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
            c = !c;
    }
    return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
    for (int i = 0; i < p.size(); i++)

```

```

        if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
            return true;
        return false;
    }

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
    vector<PT> ret;
    b = b-a;
    a = a-c;
    double A = dot(b, b);
    double B = dot(a, b);
    double C = dot(a, a) - r*r;
    double D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
    if (D > EPS)
        ret.push_back(c+a+b*(-B-sqrt(D))/A);
    return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
    vector<PT> ret;
    double d = sqrt(dist2(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    double x = (d*d-R*R+r*r)/(2*d);
    double y = sqrt(r*r-x*x);
    PT v = (b-a)/d;
    ret.push_back(a+v*x + RotateCCW90(v)*y);
    if (y > 0)
        ret.push_back(a+v*x - RotateCCW90(v)*y);
    return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion. Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
    return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
    PT c(0,0);
    double scale = 6.0 * ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i].x*p[j].y - p[j].x*p[i].y) * (p[i].x + p[j].x) / scale;
    }
    return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
    for (int i = 0; i < p.size(); i++) {
        for (int k = i+1; k < p.size(); k++) {
            int j = (i+1) % p.size();
            int l = (k+1) % p.size();
            if (i == l || j == k) continue;
            if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
                return false;
        }
    }
    return true;
}

int main() {
    // expected: (-5,2)
    cerr << RotateCCW90(PT(2,5)) << endl;

    // expected: (5,-2)
    cerr << RotateCW90(PT(2,5)) << endl;

    // expected: (-5,2)
    cerr << RotateCCW(PT(2,5), M_PI/2) << endl;

    // expected: (5,2)
    cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;
}

```



```
// expected: (5,2) (7.5,3) (2.5,1)
cerr << ProjectPointSegment (PT(-5,-2), PT(10,4), PT(3,7)) << " "
    << ProjectPointSegment (PT(7.5,3), PT(10,4), PT(3,7)) << " "
    << ProjectPointSegment (PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

// expected: 6.78903
cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

// expected: 1 0 1
cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 0 0 1
cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 1 1 1 0
cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

// expected: (1,2)
cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;

// expected: (1,1)
cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

vector<PT> v;
v.push_back(PT(0,0));
v.push_back(PT(5,0));
v.push_back(PT(5,5));
v.push_back(PT(0,5));

// expected: 1 1 1 0 0
cerr << PointInPolygon(v, PT(2,2)) << " "
    << PointInPolygon(v, PT(2,0)) << " "
    << PointInPolygon(v, PT(0,2)) << " "
    << PointInPolygon(v, PT(5,2)) << " "
    << PointInPolygon(v, PT(2,5)) << endl;

// expected: 0 1 1 1 1
cerr << PointOnPolygon(v, PT(2,2)) << " "
    << PointOnPolygon(v, PT(2,0)) << " "
    << PointOnPolygon(v, PT(0,2)) << " "
    << PointOnPolygon(v, PT(5,2)) << " "
    << PointOnPolygon(v, PT(2,5)) << endl;

// expected: (1,6)
// (5,4) (4,5)
// blank line
// (4,5) (5,4)
// blank line
// (4,5) (5,4)
vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

// area should be 5.0
// centroid should be (1.1666666, 1.1666666)
PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
vector<PT> p(pa, pa+4);
PT c = ComputeCentroid(p);
cerr << "Area: " << ComputeArea(p) << endl;
cerr << "Centroid: " << c << endl;

return 0;
}
```

## 4.7 Geometry 2

```
double x, y;
Point (double x = 0, double y = 0) : x(x), y(y) { }
bool operator < (const Point& a) const
{
    if (a.x != x) return x < a.x;
    return y < a.y;
};

typedef Point Vector;

struct Line
{
    Point P;
    Vector v;
    double ang;
    Line() {}
    Line(Point P, Vector v) : P(P), v(v) { ang = atan2(v.y, v.x); }
    bool operator < (const Line& L) const
    {
        return ang < L.ang;
    }
};

Vector operator + (Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator - (Point A, Point B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator * (Vector A, double p) { return Vector(A.x*p, A.y*p); }
Vector operator / (Vector A, double p) { return Vector(A.x / p, A.y / p); }

int dcmp(double x)
{
    if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;
}

bool operator == (const Point& a, const Point& b)
{
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
}

double Dot(Vector A, Vector B) { return A.x*B.x + A.y*B.y; }
double Length(Vector A) { return sqrt(Dot(A, A)); }
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }
double Cross(Vector A, Vector B) { return A.x*B.y - A.y*B.x; }
double Area2(Point A, Point B, Point C) { return fabs(Cross(B - A, C - A)) / 2; }

Vector Rotate(Vector A, double rad)
{
    return Vector(A.x*cos(rad) - A.y*sin(rad), A.x*sin(rad) + A.y*cos(rad));
}

Point GetLineIntersection(Point P, Vector v, Point Q, Vector w)
{
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v*t;
}

bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)
{
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1);
    double c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}

bool OnSegment(Point p, Point a1, Point a2)
{
    return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
}

double PolygonArea(Point* p, int n)
{
    double area = 0;
    for (int i = 1; i < n - 1; i++)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}

double PointDistanceToLine(Point P, Point A, Point B)
{
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2)) / Length(v1);
}

double PointDistanceToSegment(Point P, Point A, Point B)
{
    if (A == B) return Length(P - A);
}
```

```
const double eps = 1e-8;
const double PI = acos(-1.0);
```

```
struct Point
{
}
```

```

Vector v1 = B - A, v2 = P - A, v3 = P - B;
if (dcmp(Dot(v1, v2) < 0)) return Length(v2);
else if (dcmp(Dot(v1, v3) > 0)) return Length(v3);
else return fabs(Cross(v1, v2)) / Length(v1);
}

int isPointInPolygon(Point p, Point *poly, int n)
{
    int wn = 0;
    for (int i = 0; i < n; i++)
    {
        const Point& p1 = poly[i], p2 = poly[(i + 1) % n];
        if (p == p1 || p == p2 || OnSegment(p, p1, p2)) return -1;
        int k = dcmp(Cross(p2 - p1, p - p1));
        int d1 = dcmp(p1.y - p.y);
        int d2 = dcmp(p2.y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if (wn != 0) return 1;
    return 0;
}

Vector Normal(Vector A)
{
    double L = Length(A);
    return Vector(-A.y / L, A.x / L);
}

double Dist2(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y);
}

double RotatingCalipers(Point *P, int n)
{
    if (n == 1) return 0;
    if (n == 2) return Dist2(P[0], P[1]);
    P[n] = P[0];
    double ans = 0;
    for (int u = 0, v = 1; u < n; u++)
    {
        for (;;)
        {
            double diff = Cross(P[u + 1] - P[u], P[v + 1] - P[v]);
            if (diff <= 0)
            {
                ans = max(ans, Dist2(P[u], P[v]));
                if (diff == 0) ans = max(ans, Dist2(P[u], P[v + 1]));
                break;
            }
            v = (v + 1) % n;
        }
    }
    return ans;
}

bool OnLeft(Line L, Point p)
{
    return Cross(L.v, p - L.p) > 0;
}

Point GetLineIntersection(const Line &a, const Line &b)
{
    Vector u = a.p - b.p;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.p + a.v*t;
}

int HalfPlaneIntersection(Line* L, int n, Point* poly)
{
    sort(L, L + n);
    int first, last;
    Point *p = new Point[n];
    Line* q = new Line[n];
    q[first = last = 0] = L[0];
    for (int i = 1; i < n; i++)
    {
        while (first < last && !OnLeft(L[i], p[last - 1])) last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
        q[++last] = L[i];
        if (fabs(Cross(q[last].v, q[last - 1].v)) < eps)
        {
            last--;
            if (OnLeft(q[last], L[i].p)) q[last] = L[i];
        }
        if (first < last) p[last - 1] = GetLineIntersection(q[last - 1], q[last]);
    }
    while (first < last && !OnLeft(q[first], p[last - 1])) last--;
    if (last - first <= 1) return 0;
    p[last] = GetLineIntersection(q[last], q[first]);
}

```

```

int m = 0;
for (int i = first; i <= last; i++) poly[m++] = p[i];
return m;
}

vector<Point> CutPolygon(const vector<Point> &poly, Point A, Point B)
{
    vector<Point> newpoly;
    int n = poly.size();
    for (int i = 0; i < n; i++)
    {
        Point C = poly[i], D = poly[(i + 1) % n];
        if (dcmp(Cross(B - A, C - A)) >= 0) newpoly.push_back(C);
        if (dcmp(Cross(B - A, C - D)) != 0)
        {
            Point ip = GetLineIntersection(A, B - A, C, D - C);
            if (OnSegment(ip, C, D)) newpoly.push_back(ip);
        }
    }
    return newpoly;
}

```

## 4.8 Convex Hull

```

// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm. Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
// Running time: O(n log n)
// INPUT: a vector of input points, unordered.
// OUTPUT: a vector of points in the convex hull, counterclockwise, starting
// with bottommost/leftmost point

#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>
#include <map>

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
    T x, y;
    PT() {}
    PT(T x, T y) : x(x), y(y) {}
    bool operator<(const PT &rhs) const { return make_pair(y,x) < make_pair(rhs.y,rhs.x); }
    bool operator==(const PT &rhs) const { return make_pair(y,x) == make_pair(rhs.y,rhs.x); }
};

T cross(PT p, PT q) { return p.x*q.y - p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }

#ifndef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<PT> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifndef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
}

```

```

    pts = dn;
#endif
}

// The following code solves SPOJ problem #26: Build the Fence (BSHEEP)
int main() {
    int t;
    scanf("%d", &t);
    for (int caseno = 0; caseno < t; caseno++) {
        int n;
        scanf("%d", &n);
        vector<PT> v(n);
        for (int i = 0; i < n; i++) scanf("%lf%lf", &v[i].x, &v[i].y);
        vector<PT> h(v);
        map<PT,int> index;
        for (int i = n-1; i >= 0; i--) index[v[i]] = i+1;
        ConvexHull(h);

        double len = 0;
        for (int i = 0; i < h.size(); i++) {
            double dx = h[i].x - h[(i+1)%h.size()].x;
            double dy = h[i].y - h[(i+1)%h.size()].y;
            len += sqrt(dx*dx+dy*dy);
        }

        if (caseno > 0) printf("\n");
        printf("%.2f\n", len);
        for (int i = 0; i < h.size(); i++) {
            if (i > 0) printf(" ");
            printf("%d", index[h[i]]);
        }
        printf("\n");
    }
}

```

## 4.9 Convex Hull Diameter

```

typedef pair<double, double> point;

bool cw(const point &a, const point &b, const point &c) {
    return (b.first - a.first) * (c.second - a.second) - (b.second - a.second) * (c.first - a.first) <
        0;
}

vector<point> convexHull(vector<point> p) {
    int n = p.size();
    if (n <= 1)
        return p;
    int k = 0;
    sort(p.begin(), p.end());
    vector<point> q(n * 2);
    for (int i = 0; i < n; q[k++] = p[i++])
        for (; k >= 2 && !cw(q[k-2], q[k-1], p[i]); --k)
            ;
    for (int i = n-2, t = k; i >= 0; q[k++] = p[i--])
        for (; k > t && !cw(q[k-2], q[k-1], p[i]); --k)
            ;
    q.resize(k-1 - (q[0] == q[1]));
    return q;
}

double area(const point &a, const point &b, const point &c) {
    return abs((b.first - a.first) * (c.second - a.second) - (b.second - a.second) * (c.first - a.first));
}

double dist(const point &a, const point &b) {
    return hypot(a.first - b.first, a.second - b.second);
}

double diameter(const vector<point> &p) {
    vector<point> h = convexHull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return dist(h[0], h[1]);
    int k = 1;
    while (area(h[m-1], h[0], h[(k+1)%m]) > area(h[m-1], h[0], h[k]))
        ++k;
    double res = 0;
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = max(res, dist(h[i], h[j]));
        while (j < m && area(h[i], h[(i+1)%m], h[(j+1)%m]) > area(h[i], h[(i+1)%m], h[j])) {
            res = max(res, dist(h[i], h[(j+1)%m]));
            ++j;
        }
    }
}

```

```

    return res;
}

int main() {
    vector<point> points(4);
    points[0] = point(0, 0);
    points[1] = point(3, 0);
    points[2] = point(0, 3);
    points[3] = point(1, 1);
    double d = diameter(points);
    cout << d << endl;
}

```

## 4.10 Great Circle Distance

```

struct PT{double lat, lon; PT() {}
    PT(double lat, double lon) : lat(lat), lon(lon) {}
    PT operator * (double c) const { return PT(lat*c, lon *c); }
}pts[1000+10];

const double eps = 1e-9;
const double PI = 3.141592653589793;
const double R = 6378.00 ; // radius of earth

double GCDist(PT p1, PT p2) {
    p1 = p1*(PI/180.); p2 = p2*(PI/180.);
    double dlon = p2.lon - p1.lon;
    double dlat = p2.lat - p1.lat;
    double a = pow((sin(dlat/2)),2) + cos(p1.lat) * cos(p2.lat) * pow(sin(dlon/2), 2);
    double c = 2 * atan2(sqrt(a), sqrt(1-a));
    double d = R * c;
    return d*eps;
}

```

## 5 String Algorithms

### 5.1 Suffix Array 1

```

// Suffix array construction in O(L log^2 L) time. Routine for
// computing the length of the longest common prefix of any two
// suffixes in O(log L) time.
// INPUT:  string s
// OUTPUT: array suffix[] such that suffix[i] = index (from 0 to L-1)
//         of substring s[i...L-1] in the list of sorted suffixes.
//         That is, if we take the inverse of the permutation suffix[],
//         we get the actual suffix array.

#include <vector>
#include <iostream>
#include <string>

using namespace std;

struct SuffixArray {
    const int L;
    string s;
    vector<vector<int>> > P;
    vector<pair<pair<int,int>,int>> > M;

    SuffixArray(const string &s) : L(s.length()), s(s), P(1, vector<int>(L, 0)), M(L) {
        for (int i = 0; i < L; i++) P[0][i] = int(s[i]);
        for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
            P.push_back(vector<int>(L, 0));
            for (int i = 0; i < L; i++)
                M[i] = make_pair(make_pair(P[level-1][i], i + skip < L ? P[level-1][i + skip] : -1000), i);
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ? P[level][M[i-1].second] : i;
        }
    }

    vector<int> GetSuffixArray() { return P.back(); }

    // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
    int LongestCommonPrefix(int i, int j) {
        int len = 0;
        if (i == j) return L - i;
        for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
            }
        }
    }
}

```

```

        len += 1 << k;
    }
    return len;
}
};

// BEGIN CUT
// The following code solves UVA problem 11512: GATTACA.
#define TESTING
#ifdef TESTING
int main() {
    int T;
    cin >> T;
    for (int caseno = 0; caseno < T; caseno++) {
        string s;
        cin >> s;
        SuffixArray array(s);
        vector<int> v = array.GetSuffixArray();
        int bestlen = -1, bestpos = -1, bestcount = 0;
        for (int i = 0; i < s.length(); i++) {
            int len = 0, count = 0;
            for (int j = i+1; j < s.length(); j++) {
                int l = array.LongestCommonPrefix(i, j);
                if (l >= len) {
                    if (l > len) count = 2; else count++;
                    len = l;
                }
            }
            if (len > bestlen || len == bestlen && s.substr(bestpos, bestlen) > s.substr(i, len)) {
                bestlen = len;
                bestcount = count;
                bestpos = i;
            }
        }
        if (bestlen == 0) {
            cout << "No repetitions found!" << endl;
        } else {
            cout << s.substr(bestpos, bestlen) << " " << bestcount << endl;
        }
    }
}

#else
// END CUT
int main() {

    // bobocel is the 0'th suffix
    // obocel is the 5'th suffix
    // bocel is the 1'st suffix
    // ocel is the 6'th suffix
    // cel is the 2'nd suffix
    // el is the 3'rd suffix
    // l is the 4'th suffix
    SuffixArray suffix("bobocel");
    vector<int> v = suffix.GetSuffixArray();

    // Expected output: 0 5 1 6 2 3 4
    // 2
    for (int i = 0; i < v.size(); i++) cout << v[i] << " ";
    cout << endl;
    cout << suffix.LongestCommonPrefix(0, 2) << endl;
}
// BEGIN CUT
#endif
// END CUT

```

## 5.2 Suffix Array 2

```

/*
Suffix array O(n lg^2 n)
LCP table O(n)
*/
#include <cstdio>
#include <algorithm>
#include <cstring>

using namespace std;

#define REP(i, n) for (int i = 0; i < (int)(n); ++i)

const int MAXN = 1 << 21;
char * S;
int N, gap;
int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];

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 = strlen(S);
    REP(i, N) sa[i] = i, pos[i] = S[i];
    for (gap = 1;; gap *= 2)
    {
        sort(sa, sa + N, sufCmp);
        REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
        REP(i, N) pos[sa[i]] = tmp[i];
        if (tmp[N - 1] == N - 1) break;
    }
}

void buildLCP()
{
    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;
    }
}

```

## 5.3 Prefix Function

```

std::vector<int> prefix_function(const std::string& str) {
    std::vector<int> prefs(str.size(), 0);
    for (int i = 1; i < str.size(); ++i) {
        int pref = prefs[i - 1];
        while (pref > 0 && str[i] != str[pref]) {
            pref = prefs[pref - 1];
        }
        if (str[i] == str[pref]) {
            ++pref;
        }
        prefs[i] = pref;
    }
    return prefs;
}

std::vector<int> z_function(const std::string& str) {
    std::vector<int> zfunc(str.size(), 0);
    zfunc[0] = str.size();
    for (int i = 1, left = 0, right = 0; i < str.size(); ++i) {
        if (i <= right) {
            zfunc[i] = std::min(right - i + 1, zfunc[i - left]);
        }
        while (i + zfunc[i] < str.size() && str[zfunc[i]] == str[i + zfunc[i]]) {
            ++zfunc[i];
        }
        if (i + zfunc[i] - 1 > right) {
            left = i;
            right = i + zfunc[i] - 1;
        }
    }
    return zfunc;
}

std::string from_prefix_function(const std::vector<int>& prefs) {
    std::string str(prefs.size(), '.');
    char current_symbol = 'a';
    for (int i = 0; i < prefs.size(); ++i) {
        if (prefs[i] > 0) {
            str[i] = str[prefs[i] - 1];
        }
        else {
            str[i] = current_symbol++;
        }
    }
    return str;
}

std::vector<int> prefix_to_z(const std::vector<int>& prefs) {
    return z_function(from_prefix_function(prefs));
}

std::vector<int> z_to_prefix(const std::vector<int>& z_func) {
    std::vector<int> prefs(z_func.size(), 0);
    for (int i = 1; i < z_func.size(); ++i) {

```

```

    prefs[i + z_func[i] - 1] = std::max(prefs[i + z_func[i] - 1], z_func[i]);
}
for (int i = z_func.size() - 2; i >= 0; --i) {
    prefs[i] = std::max(prefs[i + 1] - 1, prefs[i]);
}
return prefs;
}

```

## 5.4 Infix to Postfix

```

import java.util.Stack;

public class ShuntingYard {

    public static void main(String[] args) {
        String infix = "3 + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3";
        System.out.printf("infix:  %s\n", infix);
        System.out.printf("postfix: %s\n", infixToPostfix(infix));
    }

    static String infixToPostfix(String infix) {
        final String ops = "+/*^";
        StringBuilder sb = new StringBuilder();
        Stack<Integer> s = new Stack<>();

        for (String token : infix.split("\\s")) {
            if (token.isEmpty())
                continue;
            char c = token.charAt(0);
            int idx = ops.indexOf(c);

            // check for operator
            if (idx != -1) {
                if (s.isEmpty())
                    s.push(idx);

                else {
                    while (!s.isEmpty()) {
                        int prec2 = s.peek() / 2;
                        int prec1 = idx / 2;
                        if (prec2 > prec1 || (prec2 == prec1 && c != '^'))
                            sb.append(ops.charAt(s.pop())).append(' ');
                        else break;
                    }
                    s.push(idx);
                }
            }
            else if (c == '(') {
                s.push(-2); // -2 stands for '('
            }
            else if (c == ')') {
                // until '(' on stack, pop operators.
                while (s.peek() != -2)
                    sb.append(ops.charAt(s.pop())).append(' ');
                s.pop();
            }
            else {
                sb.append(token).append(' ');
            }
        }
        while (!s.isEmpty())
            sb.append(ops.charAt(s.pop())).append(' ');
        return sb.toString();
    }
}

```

## 5.5 Knuth-Morris-Pratt (KMP)

```

public class Kmp {

    public static int[] prefixFunction(String s) {
        int[] p = new int[s.length()];
        int k = 0;
        for (int i = 1; i < s.length(); i++) {
            while (k > 0 && s.charAt(k) != s.charAt(i))
                k = p[k - 1];
            if (s.charAt(k) == s.charAt(i))
                ++k;
            p[i] = k;
        }
        return p;
    }
}

```

```

public static int kmpMatcher(String s, String pattern) {
    int m = pattern.length();
    if (m == 0)
        return 0;
    int[] p = prefixFunction(pattern);
    for (int i = 0, k = 0; i < s.length(); i++)
        for (; k = p[k - 1];) {
            if (pattern.charAt(k) == s.charAt(i)) {
                if (++k == m)
                    return i + 1 - m;
                break;
            }
            if (k == 0)
                break;
        }
    return -1;
}

// Competitive Programming KMP
#define MAX_N 100010
char T[MAX_N], P[MAX_N]; // T = text, P = pattern
int b[MAX_N], n, m; // b = back table, n = length of T, m = length of P

void kmpPreprocess() {
    int i = 0, j = -1; b[0] = -1; // call this before calling kmpSearch()
    while (i < m) { // starting values
        while (j >= 0 && P[i] != P[j]) j = b[j]; // pre-process the pattern string P
        i++; j++; // if different, reset j using b
        // if same, advance both pointers
        b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4
    }
}

// in the example of P = "SEVENTY SEVEN" above
void kmpSearch() {
    // this is similar as kmpPreprocess(), but on string T
    int i = 0, j = 0;
    // starting values
    while (i < n) { // search through string T
        while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
        i++; j++;
        if (j == m) { // if same, advance both pointers
            printf("P is found at index %d in T\n", i - j); // a match found when j == m
            j = b[j]; // prepare j for the next possible match
        }
    }
}

```

## 5.6 Simple Parser

```

const char * expressionToParse = "3+2+4*1+(4+9)*6";
char peek() {
    return *expressionToParse;
}

char get() {
    return *expressionToParse++;
}

int expression();
int number() {
    int result = get() - '0';
    while (peek() >= '0' && peek() <= '9') {
        result = 10 * result + get() - '0';
    }
    return result;
}

int factor() {
    if (peek() >= '0' && peek() <= '9')
        return number();
    else if (peek() == '(') {
        get(); // '('
        int result = expression();
        get(); // ')'
        return result;
    }
    else if (peek() == '-') {
        get();
        return -factor();
    }
    return 0; // error
}

int term() {
    int result = factor();
}

```

```

while (peek() == '*' || peek() == '/')
    if (get() == '*')
        result *= factor();
    else
        result /= factor();
return result;
}

int expression(){
    int result = term();
    while (peek() == '+' || peek() == '-')
        if (get() == '+')
            result += term();
        else
            result -= term();
    return result;
}

int _tmain(int argc, _TCHAR* argv[]){
    int result = expression();
    return 0;
}

```

## 5.7 Longest Palindrome

```

using namespace std;
template <class RAI1, class RAI2>
void fastLongestPalindromes(RAI1 seq, RAI1 seqEnd, RAI2 out)
{
    int seqLen = seqEnd - seq;
    int i = 0, j, d, s, e, lLen, k = 0;
    int palLen = 0;
    while (i < seqLen)
    {
        if (i > palLen && seq[i - palLen - 1] == seq[i])
        {
            palLen += 2;
            i++;
            continue;
        }
        out[k++] = palLen;
        s = k - 2;
        e = s - palLen;
        bool b = true;
        for (j = s; j > e; j--)
        {
            d = j - e - 1;
            if (out[j] == d){
                palLen = d;
                b = false;
                break;
            }
            out[k++] = min(d, out[j]);
        }
        if (b)
        {
            palLen = 1;
            i++;
        }
        out[k++] = palLen;
    }
}

```

```

lLen = k;
s = lLen - 2;
e = s - (2 * seqLen + 1 - lLen);
for (i = s; i > e; i--)
{
    d = i - e - 1;
    out[k++] = min(d, out[i]);
}

//Example
//opposes
//[0, 1, 0, 1, 4, 1, 0, 1, 0, 1, 0, 3, 0, 1, 0]
//Longest palindrome has length 4
int main()
{
    string s; cin >> s;
    vector<int> V(2 * s.length() + 1);
    fastLongestPalindromes(s.begin(), s.end(), V.begin());
    int best = 0;
    cout << "[";
    for (int i = 0; i < V.size(); i++)
    {
        if (i > 0) cout << ", ";
        cout << V[i];
        best = max(best, V[i]);
    }
    cout << "]" << endl << "Longest palindrome has length " << best << endl;
    return 0;
}

```

## 6 Miscellaneous

### 6.1 Notes

```

/* ----- Bitmask

bit & (1 << i) // bit i is 0 or 1
(bit >> j) & 1 // bit i is 0 or 1 // use this & multiplication to avoid TLE
bit | (1 << i) // set bit i to 1
bit ^ (1 << i) // toggle bit i
x & (x - 1) // check if x is a power of 2
string stmp; bitset<12> tmp; // Debuging
tmp = bit; stmp = tmp.to_string();

/* ----- Data Structure Ideas
- Hash Table + Lookup
- Sparse Table
- SQRT Decomposition
- Bucketing
- Integer Arrays as matrices
- Recursive Tree Building
- Shortest Cycles
- Problem DAG

```