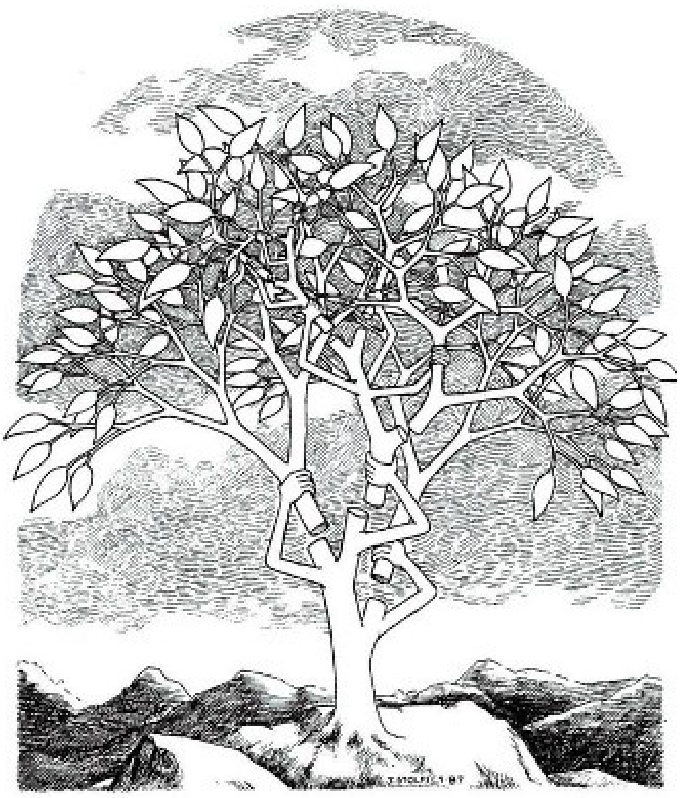
[Leewings] Algorithm Library

2012.07.17



by Leewings Ac
Nankai University

Table Of Contents

Vimrc

```
set nocp backup ru sc is et nu acd scs hid hls ai sm ignorecase sw=4 cot=longest,menu backspace=indent,eol,start mouse=a
syntax on
filetype plugin indent on
setlocal makeprg=g++\ -g\ -Wall\ -o\ %:r\ %
command -bar MAKE w | make | cw
command CRUN MAKE | !./%:r
command CRUNIN MAKE | !./%:r < in
command GDB !gnome-terminal -e gdb %:r &

" winsize 100 100
" winpos 600 0

" Use Vim settings, rather then Vi settings (much better!).
" This must be first, because it changes other options as a side effect.
" set nocompatible

" allow backspacing over everything in insert mode
" set backspace=indent,eol,start

" set backup  " keep a backup file
" set ruler  " show the cursor position all the time
" set showcmd  " display incomplete commands
" set incsearch  " do incremental searching
" set shiftwidth=4
" set expandtab
" set number
" set completeopt=longest,menu
" set showmatch
" set autochdir
" set ignorecase smartcase
" set hidden
" set hlsearch
" set autoindent  " always set autoindenting on

" syntax on
" filetype plugin indent on

" inoremap {<CR>    {<CR>}<Esc>O
" vnoremap {<CR>   S{<CR>}<Esc>Pk=iB

" In many terminal emulators the mouse works just fine, thus enable it.
" if has('mouse')
"     set mouse=a
" endif
```

Java Template

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

public class Main {
    public static void main(String[] args) throws IOException
    {
        new Prob().solve();
    }
}

class Prob {
    static final MyReader in = new MyReader();
    static final PrintWriter out = new PrintWriter(System.out);

    void solve() throws IOException
    {

        out.flush();
    }

    static void debug(Object...o)
    {
        System.err.println(Arrays.deepToString(o));
    }
}

class MyReader {
    static final BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    static StringTokenizer in;

    boolean hasNext() throws IOException
    {
        if (in == null || in.hasMoreTokens()) return true;
        String line = br.readLine();
        if (line == null) return false;
        in = new StringTokenizer(line);
        return true;
    }

    String next() throws IOException
    {
        while (in == null || !in.hasMoreTokens())
            in = new StringTokenizer(br.readLine());
        return in.nextToken();
    }

    int nextInt() throws IOException { return Integer.parseInt(next()); }
    long nextLong() throws IOException { return Long.parseLong(next()); }
    double nextDouble() throws IOException { return Double.parseDouble(next()); }
    BigInteger nextBigInteger() throws IOException { return new BigInteger(next()); }
    BigDecimal nextBigDecimal() throws IOException { return new BigDecimal(next()); }
}

// Usage: Arrays.sort(test, new ProbComparator());
class ProbComparator implements Comparator<Prob> {
    public int compare(Prob a, Prob b)
    {
        // return:
        //        1: a > b
        //        0: a = b
        //       -1: a < b
        return 0;
    }
}
```

Math

gcd

```
// we can use __gcd() in <algorithm>
int gcd(int a, int b)
{
    while (b) {
        int t = a % b;
        a = b;
        b = t;
    }
    return a;
}

// ax + by = gcd(a, b)
int extended_euclid(int a, int b, int *x, int *y)
{
    if (b == 0) {
        *x = 1;
        *y = 0;

        return a;
    }

    int r = extended_euclid(b, a % b, x, y);
    int t = *x;
    *x = *y;
    *y = t - a / b * *y;

    return r;
}

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

Euler

```
void euler(int n, int *phi, int prm_cnt, int *prm, int *is_prm)
{
    phi[1] = 1;
    for (int i = 2; i < n; i++) {
        if (is_prm[i]) phi[i] = i - 1;
        else {
            for (int j = 0; j < prm_cnt; j++)
            if (i % prm[j] == 0) {
                int k = i / prm[j];
                if (k % prm[j] == 0) phi[i] = phi[k] * prm[j];
                else phi[i] = phi[k] * (prm[j] - 1);
                break;
            }
        }
    }
}
```

Hash

```
unsigned int BKDRHash(char *str)
{
    unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;
    while (*str)
        hash = hash * seed + *str++;

    return (hash & 0x7FFFFFFF);
}

int ELFHash(char* key)
{
    unsigned int h = 0;
    while(*key) {
        h = (h << 4) + *key++;
        unsigned int g = h & 0xF0000000;
        if (g) h ^= g >> 24;
        h &= ~g;
    }
    return h % MOD;
}
```

Count How Many Ones in Binary Notation

```
inline int count_bit_one(int x)
{
    x = (x & 0x55555555) + ((x >>  1) & 0x55555555);
    x = (x & 0x33333333) + ((x >>  2) & 0x33333333);
    x = (x & 0x0f0f0f0f) + ((x >>  4) & 0x0f0f0f0f);
    x = (x & 0x00ff00ff) + ((x >>  8) & 0x00ff00ff);
    x = (x & 0x0000ffff) + ((x >> 16) & 0x0000ffff);
    return x;
}
```

Prime

```
int get_prime(int n, int* p, bool* b)
{
    int cnt = 0;
    memset(b, true, sizeof(bool) * n);
    b[0] = b[1] = false;
    for (int i = 2; i < n; i++) {
        if (b[i]) p[cnt++] = i;
        for (int j = 0; j < cnt && i * p[j] < n; j++) {
            b[i * p[j]] = false;
            if (i % p[j] == 0) break;
        }
    }

    return cnt;
}
```

Divisors

```
/*
 *  SRC: POJ 2992
 * PROB: Divisors
 * ALGO: Prime
 * DATE: Jul 13, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>

int sum, p[100], cnt[100];
bool b[500];

void prime()
{
    memset(b, true, sizeof(b));
    b[0] = b[1] = false;

    for (int i = 0; i < 500; i++) {
        if (b[i]) p[sum++] = i;
        for (int j = 0; j < sum && i * p[j] < 500; j++) {
            b[i * p[j]] = false;
            if (i % p[j] == 0) break;
        }
    }
}

// C(n, k)
void divide(int n, int k)
{
    memset(cnt, 0, sizeof(cnt));

    for (int i = 0; i < sum && n >= p[i]; i++) {
        int a = n, b = k, c = n - k;

        while (a > 1) {
            a /= p[i];
            cnt[i] += a;
        }

        while (b > 1) {
            b /= p[i];
            cnt[i] -= b;
        }

        while (c > 1) {
            c /= p[i];
            cnt[i] -= c;
        }
    }
}
```

```
bool solve()
{
    int n, k;
    if (scanf("%d%d", &n, &k) != 2) return false;
    divide(n, k);

    long long ans = 1;
    for (int i = 0; i < sum; i++)
        if (cnt[i]) ans *= (cnt[i] + 1LL);

    printf("%lld\n", ans);

    return true;
}

int main()
{
    prime();

    while (solve()) ;

    return 0;
}
```

Josephus

```
int josephus(int n, int m)
{

/*
 * orignal:         1,          2, ... , m - 1, m, m + 1, m + 2, ... , i
 * relabel: i - m + 1, i - m + 2, ... , i - 1, 0,     1,     2, ... , i - m
 * if No.x is killed in next round,
 * its actually label before relabeling is (x + m - 1) % i + 1
 *
 * Hence, we get:
 * f[1] = 1 (the last person must be relabeled 1)
 * f[i] = (f[i - 1] + m - 1) % i + 1, for i in 2, 3, ... , n
 */

    int x = 1;
    for (int i = 2; i <= n; i++) x = (x + m - 1) % i + 1;
    return x;
}
```

Gauss Elimination

```
class GaussElimination {
    private BigFraction[] X;

    //  1: infinitely many solutions
    //  0: one solution
    // -1: no solution
    public int solve(int n, BigFraction[][] A, BigFraction[] B)
    {
        X = new BigFraction[n];

        for (int xc = 0; xc < n; xc++) {
            int row = xc;
            for (int i = row + 1; i < n; i++)
                if (A[i][xc].abs().compareTo(A[row][xc].abs()) > 0) row = i;
            if (A[row][xc].isZero()) {
                if (B[row].isZero()) return 1;
                return -1;
            }

            if (row != xc) {
                for (int j = xc; j < n; j++) {
                    BigFraction tmp = A[xc][j];
                    A[xc][j] = A[row][j];
                    A[row][j] = tmp;
                }
                BigFraction tmp = B[xc];
                B[xc] = B[row];
                B[row] = tmp;
            }

            for (int i = xc + 1; i < n; i++) {
                BigFraction rate = A[i][xc].divide(A[xc][xc]);
                for (int j = xc; j < n; j++)
                    A[i][j] = A[i][j].subtract(rate.multiply(A[xc][j]));
                B[i] = B[i].subtract(rate.multiply(B[xc]));
            }
        }

        for (int i = n - 1; i >= 0; i--) {
            X[i] = B[i];
            for (int j = n - 1; j > i; j--)
                X[i] = X[i].subtract(A[i][j].multiply(X[j]));
            X[i] = X[i].divide(A[i][i]);
        }

        return 0;
    }

    public BigFraction[] getAns() { return X; }

    public String toString()
    {
        String res = "";
        for (int i = 0; i < X.length - 1; i++)
            res += X[i] + " ";
        return res + X[X.length - 1];
    }
}
```

Big Fraction

```
import java.util.*;
import java.math.BigInteger;

class BigFraction implements Comparable {
    private BigInteger numerator, denominator;
    public static BigFraction ZERO = new BigFraction(BigInteger.ZERO, BigInteger.ONE);
    public static BigFraction ONE  = new BigFraction(BigInteger.ONE,  BigInteger.ONE);

    public BigFraction()
    {
        numerator = BigInteger.ZERO;
        denominator = BigInteger.ONE;
    }

    public BigFraction(BigInteger _numerator, BigInteger _denominator)
    {
        numerator = _numerator;
        denominator = _denominator;
    }

    public BigFraction add(BigFraction other)
    {
        BigInteger new_numerator = numerator.multiply(other.denominator).add(other.numerator.multiply(denominator));
        BigInteger new_denominator = denominator.multiply(other.denominator);
        return new BigFraction(new_numerator, new_denominator).simplify();
    }

    public BigFraction subtract(BigFraction other)
    {
        BigInteger new_numerator = numerator.multiply(other.denominator).subtract(other.numerator.multiply(denominator));
        BigInteger new_denominator = denominator.multiply(other.denominator);
        return new BigFraction(new_numerator, new_denominator).simplify();
    }

    public BigFraction multiply(BigFraction other)
    {
        BigInteger new_numerator = numerator.multiply(other.numerator);
        BigInteger new_denominator = denominator.multiply(other.denominator);
        return new BigFraction(new_numerator, new_denominator).simplify();
    }

    public BigFraction divide(BigFraction other)
    {
        BigInteger new_numerator = numerator.multiply(other.denominator);
        BigInteger new_denominator = denominator.multiply(other.numerator);
        return new BigFraction(new_numerator, new_denominator).simplify();
    }

    public BigFraction abs()
    {
        return new BigFraction(numerator.abs(), denominator.abs());
    }

    public int compareTo(Object other)
    {
        return subtract((BigFraction)other).signum();
    }

    public boolean isZero()
    {
        return numerator.equals(BigInteger.ZERO);
    }

    public boolean isInteger()
    {
        return denominator.equals(BigInteger.ONE);
    }

    public BigFraction negate()
    {
        return new BigFraction(numerator.negate(), denominator);
    }

    public int signum()
    {
        return numerator.signum();
    }

    public BigFraction simplify()
    {
        if (isZero()) return BigFraction.ZERO;
        BigInteger gcd = numerator.gcd(denominator);
        if (denominator.signum() == -1)
            return new BigFraction(numerator.divide(gcd).negate(),
                    denominator.divide(gcd).negate());
        return new BigFraction(numerator.divide(gcd), denominator.divide(gcd));
    }

    public String toString()
    {
        if (isZero()) return "0";
        if (isInteger()) return numerator.toString();
        return numerator + "/" + denominator;
    }
}
```

String

Trie Tree

```
/*
 *  SRC: POJ 3630
 * PROB: Phone List
 * ALGO: Trie
 * DATE: Jul 22, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstdlib>
#include <cstring>

class Trie {
    private:
        const static int CHARSET_SIZE = 10;
        const static int NODE_MAX_SIZE = 200000;

        struct Tnode {
            Tnode *next[CHARSET_SIZE];
            int exist;
        };
        Tnode node[NODE_MAX_SIZE],
              *node_tail,
              *root;

    public:
        Trie()
        {
            reset();
        }

        void reset()
        {
            memset(node, 0, sizeof(node));
            node_tail = node;
            root = node_tail++;
            root->exist = false;
        }

        int insert(char *s)
        {
            Tnode *p = root;

            while (*s) {
                int idx = *s++ - '0';
                if (!p->next[idx]) p->next[idx] = node_tail++;
                p = p->next[idx];
                if (p->exist) return false;
            }

            p->exist++;

            return true;
        }
};

Trie trie;

int n;
char phone[10000][20];

int cmp(const void *a, const void *b)
{
    return strcmp((char *)a, (char *)b);
}

void work()
{
    trie.reset();

    scanf("%d", &n);

    for (int i = 0; i < n; i++) scanf("%s", phone[i]);

    qsort(phone, n, sizeof(char) * 20, cmp);

    for (int i = 0; i < n; i++)
        if (!trie.insert(phone[i])) {
            puts("NO");
            return ;
        }

    puts("YES");
}
```

```
int main()
{
    int t;
    scanf("%d", &t);

    while (t--) work();

    return 0;
}
```

KMP

```
void kmp_init(int *prn, char *b, int m)
{
    prn[0] = 0;
    for (int i = 1, j = 0; i < m; i++) {
        while (j > 0 && b[j] != b[i]) j = prn[j - 1];
        if (b[j] == b[i]) j++;
        prn[i] = j;
    }
}

int kmp(int *prn, char *a, char *b, int n, int m)
{
    int cnt = 0;
    for (int i = 0, j = 0; i < n; i++) {
        while (j > 0 && b[j] != a[i]) j = prn[j - 1];
        if (b[j] == a[i]) j++;
        if (j == m) {
            cnt++;
            j = prn[j - 1];
        }
    }

    return cnt;
}

template<typename ForwardIterator>
void kmp_init(int* prn, ForwardIterator first, ForwardIterator last)
{
    prn[0] = 0;
    ForwardIterator curr = first + 1;
    for (int j = 0; curr != last; curr++) {
        while (j > 0 && *(first + j) != *curr) j = prn[j - 1];
        if (*(first + j) == *curr) j++;
        prn[curr - first] = j;
    }
}

template<typename ForwardIterator>
void kmp_init(vector<int> *prn, ForwardIterator first, ForwardIterator last)
{
    prn->clear();
    prn->push_back(0);
    ForwardIterator curr = first + 1;
    for (int j = 0; curr != last; curr++) {
        while (j > 0 && *(first + j) != *curr) j = (*prn)[j - 1];
        if (*(first + j) == *curr) j++;
        prn->push_back(j);
    }
}

template<typename ForwardIterator, typename Pattern>
int kmp(const Pattern &prn,                                             \
        ForwardIterator first_a, ForwardIterator last_a,                \
        ForwardIterator first_b, ForwardIterator last_b)
{
    int cnt = 0;
    ForwardIterator curr_a = first_a;
    for (int j = 0; curr_a != last_a; curr_a++) {
        while (j > 0 && *(first_b + j) != *curr_a) j = prn[j - 1];
        if (*(first_b + j) == *curr_a) j++;
        if (first_b + j == last_b) {
            cnt++;
            j = prn[j - 1];
        }
    }

    return cnt;
}
```

AC Automata

```
/*
 *  SRC: POJ 1204
 * PROB: Word Puzzles
 * ALGO: AC Automata
 * DATE: Jul 23, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <queue>

using std::queue;

int l, c, w, ans_cnt, len[1010];
char map[1010][1010], word[1010];

struct Answers {
    int x, y;
    char dir;
} ans[1010];

class ACAutomata {
    private:
        const static int CHARSET_SIZE = 26;
        const static int NODE_MAX_SIZE = 200000;

        struct Tnode {
            Tnode *next[CHARSET_SIZE];
            Tnode *fail;
            int exist;
            int id;
        };
        Tnode node[NODE_MAX_SIZE],
              *node_tail,
              *root;

    public:
        ACAutomata() { reset(); }

        void reset()
        {
            memset(node, 0, sizeof(node));
            node_tail = node;
            root = node_tail++;
        }

        int insert(char *s, int id)
        {
            Tnode *p = root;

            while (*s) {
                int idx = *s++ - 'A';
                if (!p->next[idx]) p->next[idx] = node_tail++;
                p = p->next[idx];
            }

            p->exist++;
            p->id = id;

            return true;
        }

        void build_fail()
        {
            queue<Tnode *> Q;

            for (int i = 0; i < CHARSET_SIZE; i++) {
                if (root->next[i]) {
                    root->next[i]->fail = root;
                    Q.push(root->next[i]);
                }
                else root->next[i] = root;
            }

            while (!Q.empty()) {
                Tnode *curr = Q.front();
                Q.pop();

                for (int i = 0; i < CHARSET_SIZE; i++) {
                    Tnode *u = curr->next[i];
                    if (u) {
                        Tnode *v = curr->fail;
                        while (!v->next[i]) v = v->fail;
                        u->fail = v->next[i];

                        Q.push(u);
                    }
                }

                // for nesting case
                // if (!curr->id) curr->id = curr->fail->id;
            }
        }

//        int query(char* s)
        void query(int x, int y, int dx, int dy, char dir)
        {
//            int res = 0;

            Tnode* p = root;
            while (map[x][y]) {
                int idx = map[x][y] - 'A';
                while (!p->next[idx] && p != root) p = p->fail;
                p = p->next[idx];

                if (p->id) {
                    Tnode* t = p;
                    while (t != root && t->id != -1) {
                        if (t->id) {
                            ans_cnt++;
                            ans[t->id].x = x - (len[t->id] -1) * dx;
                            ans[t->id].y = y - (len[t->id] -1) * dy;
                            ans[t->id].dir = dir;
                        }
                        t->id = -1;
                        t = t->fail;
                    }
                }

                x += dx;
                y += dy;
            }

//            return res;
        }
};

ACAutomata aca;

int main()
{
    scanf("%d%d%d", &l, &c, &w);

    for (int i = 1; i <= l; i++) scanf("%s", map[i] + 1);

    for (int i = 1; i <= w; i++) {
        scanf("%s", word);
        aca.insert(word, i);
        len[i] = strlen(word);
    }

    aca.build_fail();

    for (int j = 1; j <= c; j++) {
        aca.query(l, j, -1, 0, 'A');
        if (ans_cnt == w) break;

        aca.query(l, j, -1, 1, 'B');
        if (ans_cnt == w) break;

        aca.query(1, j, 1, 1, 'D');
        if (ans_cnt == w) break;

        aca.query(1, j, 1, 0, 'E');
        if (ans_cnt == w) break;

        aca.query(1, j, 1, -1, 'F');
        if (ans_cnt == w) break;

        aca.query(l, j, -1, -1, 'H');
        if (ans_cnt == w) break;
    }

    for (int i = 1; i <= l; i++) {
        if (ans_cnt == w) break;

        aca.query(i, 1, -1, 1, 'B');
        if (ans_cnt == w) break;

        aca.query(i, 1, 0, 1, 'C');
        if (ans_cnt == w) break;

        aca.query(i, 1, 1, 1, 'D');
        if (ans_cnt == w) break;

        aca.query(i, c, 1, -1, 'F');
        if (ans_cnt == w) break;

        aca.query(i, c, 0, -1, 'G');
        if (ans_cnt == w) break;

        aca.query(i, c, -1, -1, 'H');
        if (ans_cnt == w) break;
    }

    for (int i = 1; i <= w; i++)
        printf("%d %d %c\n", ans[i].x - 1, ans[i].y - 1, ans[i].dir);

    return 0;
}
```

DP on AC Automata

```
/*
 *  SRC: ZOJ 3545
 * PROB: Rescue the Rabbit
 * ALGO: DP on AC Automata
 * DATE: Oct 05, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>

using std::queue;
using std::max;

int n, l;
int w[20];
char dna[20][200];

inline int gene_to_id(char c)
{
    switch (c) {
        case 'A': return 0;
        case 'G': return 1;
        case 'T': return 2;
        case 'C': return 3;
    }
}

class ACAutomata {
    private:
        const static int CHARSET_SIZE = 4;
        const static int NODE_MAX_SIZE = 1024;

        struct Tnode {
            Tnode* next[CHARSET_SIZE];
            Tnode* fail;
            int id;
        };
        Tnode* root;

        int node_cnt;
        Tnode node[NODE_MAX_SIZE];

        int f[2][1024][1024];
        bool vis[2][1024][1024];
        int curr, next;

    public:
        ACAutomata() { reset(); }

        void reset()
        {
            memset(node, 0, sizeof(node));
            node_cnt = 0;
            root = &node[node_cnt++];

            memset(f, 0xaf, sizeof(f));
            memset(vis, 0, sizeof(vis));
            curr = 0, next = 1;
        }

        int insert(char *s, int id)
        {
            Tnode* p = root;

            while (*s) {
                int idx = gene_to_id(*s);
                if (!p->next[idx])
                    p->next[idx] = &node[node_cnt++];
                p = p->next[idx];
                s++;
            }

            p->id = id;

            return true;
        }

        void build_fail()
        {
            queue<Tnode*> Q;

            for (int i = 0; i < CHARSET_SIZE; i++) {
                if (root->next[i]) {
                    root->next[i]->fail = root;
                    Q.push(root->next[i]);
                }
                else root->next[i] = root;
            }

            while (!Q.empty()) {
                Tnode* curr = Q.front();
                Q.pop();

                for (int i = 0; i < CHARSET_SIZE; i++) {
                    Tnode* u = curr->next[i];
                    if (u) {
                        Tnode* v = curr->fail;
                        while (!v->next[i]) v = v->fail;
                        u->fail = v->next[i];

                        Q.push(u);
                    }
                }

                // for nesting case
                if (!curr->id) curr->id = curr->fail->id;
            }
        }

        void query()
        {
            int ans = 0xafafafaf;

            f[curr][0][0] = 0;
            vis[curr][0][0] = 1;
            for (int i = 0; i < l; i++) {
                for (int j = 0; j < node_cnt; j++)
                    for (int k = 0, final = 1 << n; k < final; k++)
                        if (vis[curr][j][k])
                            for (int idx = 0; idx < CHARSET_SIZE; idx++) {
                                Tnode *p = &node[j];
                                while (!p->next[idx] && p != root) p = p->fail;
                                p = p->next[idx];

                                int state = k,
                                    offset = 0;
                                Tnode *t = p;
                                while (t->id) {
                                    int t_state = 1 << (t->id - 1);
                                    if (!(state & t_state)) {
                                        offset += w[t->id];
                                        state  |= t_state;
                                    }
                                    t = t->fail;
                                }

                                vis[next][p - node][state] = true;
                                f[next][p - node][state] = max(f[next][p - node][state], f[curr][j][k] + offset);
                                ans = max(ans, f[next][p - node][state]);
                            }
                curr ^= 1;
                next ^= 1;
                memset(f[next], 0xaf, sizeof(f[next]));
                memset(vis[next], 0, sizeof(vis[next]));
            }

            if (ans < 0) puts("No Rabbit after 2012!");
            else printf("%d\n", ans);
        }
};

ACAutomata aca;

int main()
{
    while (scanf("%d%d", &n, &l) != EOF) {
        for (int i = 1; i <= n; i++) {
            scanf("%s%d", dna[i], w + i);
            aca.insert(dna[i], i);
        }

        aca.build_fail();
        aca.query();
        aca.reset();
    }

    return 0;
}
```

Suffix Array

```
/*
 *  SRC: POJ 3882
 * PROB: Stammering Aliens
 * ALGO: Suffix Array
 * DATE: Aug 21, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <utility>
#include <algorithm>
#include <cmath>

using std::pair;
using std::make_pair;
using std::max;
using std::min;

class Suffix_Array {
    // use Doubling Algorithm

    private:
        const static int MAX_LEN = 40010;
        const static int MAX_CHAR = 128;

        int* n_rank;
        int int_buf_1[MAX_LEN], int_buf_2[MAX_LEN]; // used for ranking

        int cnt[MAX_LEN]; // used for counting sort

        void c_sort(int k, int range)
        {
            int* t_rank = n_rank;
            int tR = 0;
            for (int i = len - k; i < len; i++) t_rank[tR++] = i;
            for (int i = 0; i < len; i++)
                if (suff[i] >= k) t_rank[tR++] = suff[i] - k;

            for (int i = 0; i < len; i++) cnt[rank[i]]++;
            for (int i = 1; i < range; i++) cnt[i] += cnt[i - 1];
            for (int i = len - 1, j; i >= 0; i--) {
                j = t_rank[i];
                suff[--cnt[rank[j]]] = j;
            }

            memset(cnt, 0, sizeof(int) * range);
        }

    public:
        char str[MAX_LEN];
        int len;

        int suff[MAX_LEN]; // suff[i]: the i-th *sorted* suffix
        int* rank;         // rank[i]: the rank of the i-th *original* suffix
        // suff[i] = j <=> rank[j] = i

        // LCP(i, j): the Longest Common Prefix of the i-th and j-th *sorted* suffixes
        int hgt[MAX_LEN];  //  hgt[i]: LCP(i - 1, i)

        int max_suff[20][MAX_LEN], min_hgt[20][MAX_LEN];

        void reset() {
            memset(int_buf_1, 0, sizeof(int_buf_1));
            memset(int_buf_2, 0, sizeof(int_buf_2));
            memset(cnt, 0, sizeof(cnt));
            memset(suff, 0, sizeof(suff));
            memset(hgt, 0, sizeof(hgt));
        }

        void build()
        {
            len = strlen(str);
            rank = int_buf_1;
            n_rank = int_buf_2;

            for (int i = 0; i < len; i++) {
                rank[i] = str[i];
                suff[i] = i;
            }
            c_sort(0, MAX_CHAR);

            for (int k = 1, max_rank = MAX_CHAR; max_rank != len; k <<= 1) {
                c_sort(k, max_rank + 1);

                max_rank = n_rank[suff[0]] = 1;
                for (int i = 1; i < len; i++) {
                    if (rank[suff[i - 1]] == rank[suff[i]] && rank[suff[i - 1] + k] == rank[suff[i] + k])
                        n_rank[suff[i]] = max_rank;
                    else n_rank[suff[i]] = ++max_rank;
                }

                int* tp = rank;
                rank = n_rank;
                n_rank = tp;
            }
            for (int i = 0; i < len; i++) rank[i]--;
        }

        void calc_hgt()
        {
            for (int i = 0, j, k = 0; i < len; hgt[rank[i++]] = k)
                for (k ? k-- : 0, j = suff[rank[i] - 1]; str[i + k] == str[j + k]; k++) ;
        }

        void calc_lcp()
        {
            memcpy(max_suff[0], suff, sizeof(int) * len);
            memcpy(min_hgt[0], hgt, sizeof(int) * len);
            for (int i = 1; 1 << i <= len; i++)
                for (int j = 0; j + (1 << i) <= len; j++) {
                    max_suff[i][j] = max(max_suff[i - 1][j], max_suff[i - 1][j + (1 << (i - 1))]);
                    min_hgt[i][j] = min(min_hgt[i - 1][j], min_hgt[i - 1][j + (1 << (i - 1))]);
                }
        }

        pair<int, int> lcp(int l, int r) // [l, r)
        {
            int idx = log2(r - l);
            int pos = max(max_suff[idx][l], max_suff[idx][r - (1 << idx)]);

            l++;
            idx = log2(r - l);
            int res = min(min_hgt[idx][l], min_hgt[idx][r - (1 << idx)]);

            return make_pair(res, pos);
        }
};

Suffix_Array sa;

int main()
{
    int m;
    while (scanf("%d", &m), m) {
        scanf("%s", sa.str);
        if (m == 1) {
            printf("%d %d\n", strlen(sa.str), 0);
            continue;
        }

        sa.build();
        sa.calc_hgt();
        sa.calc_lcp();

        int longest = 0, pos;
        for (int i = 0; i + m <= sa.len; i++) {
            pair<int, int> p = sa.lcp(i, i + m);
            if (p.first > longest) {
                longest = p.first;
                pos = p.second;
            }
            else if (p.first == longest && p.second > pos) pos = p.second;
        }

        if (longest) printf("%d %d\n", longest, pos);
        else puts("none");

        sa.reset();
    }

    return 0;
}
```

Computational Geometry

Common

```
// Constants
const double eps = 1e-8;
const double pi = acos(-1.0f);
const double DINF = 1.0/0.0f;

// Functions
inline bool eq0(double x) { return fabs(x) < eps; }
inline bool eq(double x, double y) { return fabs(x - y) < eps; }
inline bool ls(double x, double y) { return x + eps < y; }
inline bool gr(double x, double y) { return x - eps > y; }
inline bool greq(double x, double y) { return x + eps >= y; }
inline bool lseq(double x, double y) { return x - eps <= y; }
inline double fmax(double x, double y) { return gr(x, y) ? x : y; }
inline double fmin(double x, double y) { return ls(x, y) ? x : y; }

inline double dot(const Vec &u, const Vec &v) { return u.x * v.x + u.y * v.y; }
inline double cross(const Vec &u, const Vec &v) { return u.x * v.y - u.y * v.x; }

inline double dot(const Vec &u, const Vec &v)
{
    return u.x * v.x + u.y * v.y + u.z * v.z;
}
inline Vec cross(const Vec &u, const Vec &v)
{
    return Vec(u.y * v.z - u.z * v.y,
               u.z * v.x - u.x * v.z,
               u.x * v.y - u.y * v.x);
}
```

Distance from Point to Segment

```
inline double dist_p2seg(const Point &p, const Point &a, const Point &b)
{
    if (ls(dot(p - a, b - a), 0.0)) return dist(p, a);
    if (ls(dot(p - b, a - b), 0.0)) return dist(p, b);
    double area = fabs(cross(p - a, b - a));
    if (eq0(area)) return fmin(dist(p, a), dist(p, b));
    return area / dist(a, b);
}
```

Gravity Center

```
// ver[] has been sorted by clockwise or counterclockwise
Point g_center(Point *ver, int n)
{
    Point res(0, 0);
    ver[n] = ver[0];

    double area = 0;
    for (int i = 1; i <= n; i++) {
        Point &a = ver[i - 1],
              &b = ver[i];
        double t = cross(a, b);
        area += t / 2.0;
        res.x += (a.x + b.x) * t;
        res.y += (a.y + b.y) * t;
    }
    res.x /= (6.0 * area);
    res.y /= (6.0 * area);

    return res;
}
```

Graham Scan

```
/*
 *  SRC: POJ 1113
 * PROB: Wall
 * ALGO: Graham Scan(Convex Hull)
 * DATE: Jul 26, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */
```

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>

using std::sort;
using std::swap;

const int MAXN = 1000;
const double pi = acos(-1.0);
const double eps = 1e-12;

inline bool eq0(double x) { return fabs(x) < eps; }
inline bool eq(double x, double y) { return fabs(x - y) < eps; }
inline bool ls(double x, double y) { return x + eps < y; }
inline bool gr(double x, double y) { return x - eps > y; }

struct Point {
    double x, y;
    double agl;

    Point() { }
    Point(double _x, double _y) : x(_x), y(_y) { }

    bool operator<(const Point &other) const
    {
        if (eq(y, other.y)) return ls(x, other.x);
        return ls(y, other.y);
    }

    Point operator-(const Point &other) const
    {
        return Point(x - other.x, y - other.y);
    }

    double sqlen() const { return x * x + y * y; }
    double length() const { return sqrt(sqlen()); }
};
typedef Point Vec;

inline double cross(const Vec &u, const Vec &v)
{
    return u.x * v.y - u.y * v.x;
}

Point first_ver;

inline bool cmp(const Point &a, const Point &b)
{
    if (eq(a.agl, b.agl))
        return ls((a - first_ver).sqlen(), (b - first_ver).sqlen());
    return ls(a.agl, b.agl);
}

inline bool check(const Point &a, const Point &b, const Point &c)
{
    return cross(b - c, a - c) > eps;
}

double graham_scan(Point *ver, Point *stack, int n, int &top)
{
    int min_ver = 0;
    for (int i = 1; i < n; i++)
        if (ver[i] < ver[min_ver]) min_ver = i;
    swap(ver[0], ver[min_ver]);
    first_ver = ver[0];

    for (int i = 1; i < n; i++)
        ver[i].agl = atan2(ver[i].y - ver[0].y, ver[i].x - ver[0].x);
    sort(ver + 1, ver + n, cmp);

    top = 0;
    stack[top++] = ver[0];
    stack[top++] = ver[1];
    for (int i = 2; i < n; i++) {
        while (top > 1 && !check(ver[i], stack[top - 1], stack[top - 2]))
            top--;
        stack[top++] = ver[i];
    }
    stack[top] = ver[0];

    double res = 0;
    for (int i = 0; i < top; i++)
        res += (stack[i] - stack[i + 1]).length();

    return res;
}

Point ver[MAXN], stack[MAXN];
int top;

int main()
{
    int n, l;
    scanf("%d%d", &n, &l);
    for (int i = 0; i < n; i++)
        scanf("%lf%lf", &ver[i].x, &ver[i].y);

    printf("%.0f\n", graham_scan(ver, stack, n, top) + 2.0 * pi * l);
    return 0;
}
```

3d Convex Hull

```
/*
 *  SRC: HDOJ 3662
 * PROB: 3D Convex Hull
 * ALGO: 3D Convex Hull
 * DATE: Nov 12, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>

using std::swap;

const int MAXV = 333;
const double eps = 1e-8;

inline bool eq0(double x) { return fabs(x) < eps; }
inline bool gr(double x, double y) { return x - eps > y; }

struct Point {
    double x, y, z;
    int id;

    Point() { }
    Point(double _x, double _y, double _z, int _id = 0)
        : x(_x), y(_y), z(_z), id(_id)
    { }

    Point operator-(const Point &other) const
    {
        return Point(x - other.x, y - other.y, z - other.z);
    }

    double sqlen() const { return x * x + y * y + z * z; }
    double len() const { return sqrt(sqlen()); }
    void norm() { double l = len(); x /= l; y /= l; z /= l; }
};
typedef Point Vec;

inline double dot(const Vec &u, const Vec &v)
{
    return u.x * v.x + u.y * v.y + u.z * v.z;
}
inline Vec cross(const Vec &u, const Vec &v)
{
    return Vec(u.y * v.z - u.z * v.y,
               u.z * v.x - u.x * v.z,
               u.x * v.y - u.y * v.x);
}

inline bool coline(const Point &a, const Point &b, const Point &c)
{
    return eq0(cross(b - a, c - a).sqlen());
}
inline bool coplaner(const Point &a, const Point &b, const Point &c, const Point &d)
{
    return eq0(dot(cross(b - a, c - a), d - a));
}

struct Plane {
    Point p0, p1, p2;

    Plane() { }
    Plane(const Point &_p0, const Point &_p1, const Point &_p2)
        : p0(_p0), p1(_p1), p2(_p2)
    { }

    Vec norm_vec() const
    {
        Vec n = cross(p1 - p0, p2 - p0);
        n.norm();
        return n;
    }
};

inline bool coplaner(const Plane &a, const Plane &b)
{
    return coplaner(a.p0, a.p1, a.p2, b.p0) && \
           coplaner(a.p0, a.p1, a.p2, b.p1) && \
           coplaner(a.p0, a.p1, a.p2, b.p2);
}

inline bool point_above_plane(const Point &p, const Plane &f)
{
    return gr(dot(p - f.p0, f.norm_vec()), 0.0);
}

int edge[MAXV][MAXV];
Plane tf[MAXV];

void convex_hull(int n, Point *ver, Plane *ch, int *ch_cnt)
{
    memset(edge, 0, sizeof(edge));

    if (coline(ver[0], ver[1], ver[2]))
        for (int i = 3; i < n; i++) {
            if (!coline(ver[0], ver[1], ver[i])) {
                swap(ver[2], ver[i]);
                break;
            }
        }
    if (coplaner(ver[0], ver[1], ver[2], ver[3]))
        for (int i = 4; i < n; i++) {
            if (!coplaner(ver[0], ver[1], ver[2], ver[i])) {
                swap(ver[3], ver[i]);
                break;
            }
        }

    int cnt = 0;
    ch[cnt++] = Plane(ver[0], ver[1], ver[2]);
    ch[cnt++] = Plane(ver[2], ver[1], ver[0]);

    for (int i = 3; i < n; i++) {
        Point curr = ver[i];
        for (int j = 0; j < cnt; j++) {
            Plane f = ch[j];
            if (point_above_plane(curr, f)) {
                edge[f.p0.id][f.p1.id] = 1;
                edge[f.p1.id][f.p2.id] = 1;
                edge[f.p2.id][f.p0.id] = 1;
            } else {
                edge[f.p0.id][f.p1.id] = -1;
                edge[f.p1.id][f.p2.id] = -1;
                edge[f.p2.id][f.p0.id] = -1;
            }
        }
        int tf_cnt = 0;
        for (int j = 0; j < cnt; j++) {
            Plane f = ch[j];
            if (edge[f.p0.id][f.p1.id] == 1) {
                if (edge[f.p1.id][f.p0.id] == -1)
                    tf[tf_cnt++] = Plane(f.p0, f.p1, curr);
                if (edge[f.p2.id][f.p1.id] == -1)
                    tf[tf_cnt++] = Plane(f.p1, f.p2, curr);
                if (edge[f.p0.id][f.p2.id] == -1)
                    tf[tf_cnt++] = Plane(f.p2, f.p0, curr);
            } else tf[tf_cnt++] = f;
        }
        for (int i = 0; i < tf_cnt; i++) ch[i] = tf[i];
        cnt = tf_cnt;
    }

    *ch_cnt = cnt;
}

int cnt_plane(const Plane *ch, int ch_cnt)
{
    int plane_cnt = 0;
    for (int i = 0; i < ch_cnt; i++) {
        bool unique = true;
        for (int j = 0; j < i; j++)
            if (coplaner(ch[i], ch[j])) {
                unique = false;
                break;
            }
        plane_cnt += unique;
    }
    return plane_cnt;
}

double surface_area(const Plane *ch, int ch_cnt)
{
    double s = 0.0;
    for (int i = 0; i < ch_cnt; i++)
        s += 0.5 * cross(ch[i].p1 - ch[i].p0, ch[i].p2 - ch[i].p0).len();
    return s;
}

double volume(const Plane *ch, int ch_cnt)
{
    double v = 0.0;
    const Point O(0, 0, 0);
    for (int i = 0; i < ch_cnt; i++)
        v += dot(cross(ch[i].p0 - O, ch[i].p1 - O), ch[i].p2 - O);
    return fabs(v) / 6.0;
}

Point ver[MAXV];
Plane ch[MAXV]; // convex hull, actually, it stores *triangles*

int main()
```

```
{
    int n;
    while (~scanf("%d", &n)) {
        int ch_cnt = 0;
        for (int i = 0; i < n; i++) {
            double x, y, z;
            scanf("%lf%lf%lf", &x, &y, &z);
            ver[i] = Point(x, y, z, i);
        }

        convex_hull(n, ver, ch, &ch_cnt);

        printf("%d\n", cnt_plane(ch, ch_cnt));
    }

    return 0;
}
```

Rotating Calipers

```
/*
 *  SRC: POJ 2187
 * PROB: Beauty Contest
 * ALGO: Rotating Calipers
 * DATE: Nov 09, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cmath>
#include <algorithm>

using std::sort;
using std::swap;

const int MAXN = 50010;
const double eps = 1e-12;

inline bool eq0(double x) { return fabs(x) < eps; }
inline bool eq(double x, double y) { return fabs(x - y) < eps; }
inline bool ls(double x, double y) { return x + eps < y; }
inline bool gr(double x, double y) { return x - eps > y; }
inline double fmax(double x, double y) { return gr(x, y) ? x : y; }

struct Point {
    double x, y;
    double agl;

    Point() { }
    Point(double _x, double _y) : x(_x), y(_y) { }

    bool operator<(const Point &other) const
    {
        if (eq(y, other.y)) return ls(x, other.x);
        return ls(y, other.y);
    }

    Point operator-(const Point &other) const
    {
        return Point(x - other.x, y - other.y);
    }

    double sqlen() const { return x * x + y * y; }
    double length() const { return sqrt(sqlen()); }
};
typedef Point Vec;

inline double cross(const Vec &u, const Vec &v)
{
    return u.x * v.y - u.y * v.x;
}

inline double dist(const Point &u, const Point &v)
{
    return (u - v).length();
}

Point first_ver;

inline bool cmp(const Point &a, const Point &b)
{
    if (eq(a.agl, b.agl))
        return ls((a - first_ver).sqlen(), (b - first_ver).sqlen());
    return ls(a.agl, b.agl);
}

inline bool check(const Point &a, const Point &b, const Point &c)
{
    return cross(b - c, a - c) > eps;
}

void graham_scan(Point *ver, Point *stack, int n, int &top)
```

```
{
    int min_ver = 0;
    for (int i = 1; i < n; i++)
        if (ver[i] < ver[min_ver]) min_ver = i;
    swap(ver[0], ver[min_ver]);
    first_ver = ver[0];

    for (int i = 1; i < n; i++)
        ver[i].agl = atan2(ver[i].y - ver[0].y, ver[i].x - ver[0].x);
    sort(ver + 1, ver + n, cmp);

    top = 0;
    stack[top++] = ver[0];
    stack[top++] = ver[1];
    for (int i = 2; i < n; i++) {
        while (top > 1 && !check(ver[i], stack[top - 1], stack[top - 2]))
            top--;
        stack[top++] = ver[i];
    }
    stack[top] = ver[0];
}

double rotating_calipers(const Point *ver, int n)
{
    double res = 0.0;
    for (int curr = 0, next = 1; curr < n; curr++) {
        while (gr(cross(ver[curr + 1] - ver[curr], ver[next + 1] - ver[curr]),
                  cross(ver[curr + 1] - ver[curr], ver[next] - ver[curr])))
            next = (next + 1) % n;
        res = fmax(res, dist(ver[curr], ver[next]));
    }
    return res;
}

Point ver[MAXN], stack[MAXN];
int top;

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%lf%lf", &ver[i].x, &ver[i].y);

    graham_scan(ver, stack, n, top);
    double ans = rotating_calipers(stack, top);
    printf("%.0f\n", ans * ans);

    return 0;
}
```

Cutting Rectangle

```
/*
ID: os.idea1
LANG: C
TASK: window
*/
#include <stdio.h>

#define MAX_SIZE 1000

FILE *fin, *fout;

struct Point {
    int x, y;
};

struct Lnode {
    int pos;

    char id;
    struct Point a, b;
    struct Lnode *prev, *next;
} link[MAX_SIZE];
struct Lnode *head, *tail;
int link_cnt;
int pos[256];

struct Rectangle {
    struct Point a, b;
    int del;
} rec[MAX_SIZE];
int rec_cnt;

inline int area(struct Point a, struct Point b)
{
    return (b.x - a.x) * (b.y - a.y);
}

inline void rec_insert(struct Point a, struct Point b)
{
    rec[rec_cnt].a = a;
    rec[rec_cnt].b = b;
    rec[rec_cnt].del = 0;
    rec_cnt++;
}

inline int min(int a, int b) { return a < b ? a : b; }
inline int max(int a, int b) { return a > b ? a : b; }
inline void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; }

void create(char *s)
{
    struct Lnode *ptr = &link[link_cnt++];
    ptr->id = s[2];
    pos[s[2]] = ptr->pos;

    sscanf(s + 4, "%d%*c%d%*c%d%*c%d",
           &ptr->a.x, &ptr->a.y, &ptr->b.x, &ptr->b.y);
    if (ptr->a.x > ptr->b.x) swap(&ptr->a.x, &ptr->b.x);
    if (ptr->a.y > ptr->b.y) swap(&ptr->a.y, &ptr->b.y);

    ptr->next = head;
    if (head) head->prev = ptr;
    else tail = ptr;
    head = ptr;
}

void destory(char id)
{
    struct Lnode *ptr = &link[pos[id]];
    if (ptr == head) {
        head = ptr->next;
        if (head) head->prev = 0;
        return ;
    }

    if (ptr == tail) {
        tail = ptr->prev;
        if (tail) tail->next = 0;
        return ;
    }

    ptr->prev->next = ptr->next;
    ptr->next->prev = ptr->prev;
}

void to_top(char id)
{
    struct Lnode *ptr = &link[pos[id]];
    if (ptr == head) return ;

    destory(id);
    ptr->next = head;
    ptr->prev = 0;
    head->prev = ptr;
    head = ptr;
}

void to_bottom(char id)
{
    struct Lnode *ptr = &link[pos[id]];
    if (ptr == tail) return ;

    destory(id);
    ptr->prev = tail;
    ptr->next = 0;
    tail->next = ptr;
    tail = ptr;
}

void show(char id)
{
    rec_cnt = 0;

    rec_insert(link[pos[id]].a, link[pos[id]].b);

    struct Lnode *ptr = head;
    while (ptr->id != id) {
        int i;
        for (i = 0; i < rec_cnt; i++)
            if (!rec[i].del) {
                if (rec[i].a.x >= ptr->b.x || rec[i].b.x <= ptr->a.x) continue;
                if (rec[i].a.y >= ptr->b.y || rec[i].b.y <= ptr->a.y) continue;

                int x1 = max(rec[i].a.x, ptr->a.x),
                    x2 = min(rec[i].b.x, ptr->b.x);
                if (rec[i].a.x < x1)
                    rec_insert(rec[i].a, (struct Point){x1, rec[i].b.y});
                if (rec[i].b.x > x2)
                    rec_insert((struct Point){x2, rec[i].a.y}, rec[i].b);

                int y1 = max(rec[i].a.y, ptr->a.y),
                    y2 = min(rec[i].b.y, ptr->b.y);
                if (rec[i].a.y < y1)
                    rec_insert((struct Point){x1, rec[i].a.y}, \
                               (struct Point){x2, y1});
                if (rec[i].b.y > y2)
                    rec_insert((struct Point){x1, y2}, \
                               (struct Point){x2, rec[i].b.y});
                rec[i].del = 1;
            }

        ptr = ptr->next;
    }

    int area_sum = area(rec[0].a, rec[0].b), area_rest = 0;
    int i;
    for (i = 0; i < rec_cnt; i++)
        if (!rec[i].del) area_rest += area(rec[i].a, rec[i].b);

    fprintf(fout, "%.3f\n", (double)area_rest / area_sum * 100);
}

int main()
{
    int i;
    for (i = 0; i < MAX_SIZE; i++) link[i].pos = i;

    fin = fopen("window.in", "r");
    fout = fopen("window.out", "w");

    char str[50];
    while (fscanf(fin, "%s", str) != EOF) {
        switch (str[0]) {
            case 'w': create(str);       break;
            case 't': to_top(str[2]);    break;
            case 'b': to_bottom(str[2]); break;
            case 'd': destory(str[2]);   break;
            case 's': show(str[2]);
        }
    }

    return 0;
}
```

Cloest Pair of Points on Plane

```
/*
 *  SRC: NKOJ 2185
 * PROB: Exercise 5
 * ALGO: D&C
 * DATE: Nov 23, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>

using std::sort;
using std::min;

const int MAX_P = 30010;

struct Point {
    int id;
    int x, y;
};
Point X[MAX_P], Y[MAX_P], Z[MAX_P];

inline bool cmp_x(const Point &a, const Point &b) { return a.x < b.x; }
inline bool cmp_y(const Point &a, const Point &b) { return a.y < b.y; }
inline int sqr(int x) { return x * x; }
inline int sqr_dist(const Point &a, const Point &b)
{
    return sqr(a.x - b.x) + sqr(a.y - b.y);
}

void merge(Point *p, Point *q, int left, int mid, int right)
{
    int l = left, r = mid + 1, cnt = left;
    while (l <= mid && r <= right) {
        if (q[l].y < q[r].y) p[cnt++] = q[l++];
        else p[cnt++] = q[r++];
    }
    while (l <= mid)   p[cnt++] = q[l++];
    while (r <= right) p[cnt++] = q[r++];
}

int closest(Point *X, Point *Y, Point *Z, int l, int r) // X[l, r]
{
    if (r - l == 1) return sqr_dist(X[l], X[r]);
    if (r - l == 2) {
        int d1 = sqr_dist(X[l], X[l + 1]),
            d2 = sqr_dist(X[l], X[r]),
            d3 = sqr_dist(X[l + 1], X[r]);
        return min(d1, min(d2, d3));
    }

    int mid = (l + r) / 2;
    for (int i = l, j = l, k = mid + 1; i <= r; i++) {
        if (Y[i].id <= mid) Z[j++] = Y[i];
        else Z[k++] = Y[i];
    }

    int dl = closest(X, Z, Y, l, mid),
        dr = closest(X, Z, Y, mid + 1, r),
        d  = min(dl, dr);
    merge(Y, Z, l, mid, r);

    int z_cnt = l;
    for (int i = l; i <= r; i++)
        if (sqr(abs(X[mid].x - Y[i].x)) < d)
            Z[z_cnt++] = Y[i];

    for (int i = l; i < z_cnt; i++)
        for (int j = i + 1; j < z_cnt && sqr(Z[j].y - Z[i].y) < d; j++)
            d = min(d, sqr_dist(Z[i], Z[j]));

    return d;
}

int closest_pair(Point *X, int n)
{
    sort(X, X + n, cmp_x);
    for (int i = 0; i < n; i++) X[i].id = i;
    memcpy(Y, X, n * sizeof(X[0]));
    sort(Y, Y + n, cmp_y);

    return closest(X, Y, Z, 0, n - 1);
}

int main()
{
    int n;
    scanf("%d\n", &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d", &X[i].x, &X[i].y);

    int ans = closest_pair(X, n);
    printf("%d\n", ans);

    return 0;
}
```

Graph Theory

Prim

```
/*
 *  SRC: POJ 1258
 * PROB: Agri-Net
 * ALGO: Prim
 * DATE: Jul 24, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>

using std::vector;
using std::priority_queue;

const int MAXN = 110;

struct Edge
{
    int v, d;

    Edge(int _v, int _d)
        : v(_v), d(_d)
    { }

    bool operator<(const Edge& other) const
    {
        return d > other.d;
    }
};

int dist[MAXN];

bool prim()
{
    int n;
    if (scanf("%d", &n) == EOF) return false;

    vector<Edge> edge[MAXN];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            int d;
            scanf("%d", &d);
            if (i != j) edge[i].push_back(Edge(j, d));
        }

    memset(dist, 0x3f, sizeof(dist));
    dist[0] = 0;

    priority_queue<Edge> Q;
    Q.push(Edge(0, 0));

    int sum = 0;
    while (!Q.empty()) {
        int u = Q.top().v;
        int td = Q.top().d;
        Q.pop();
        if (!dist[u] && u != 0) continue;
        sum += td;
        dist[u] = 0;
        for (int i = 0; i < edge[u].size(); i++) {
            int v = edge[u][i].v;
            int d = edge[u][i].d;
            if (d < dist[v]) {
                dist[v] = d;
                Q.push(Edge(v, dist[v]));
            }
        }
    }

    printf("%d\n", sum);

    return true;
}

int main()
{
    while (prim()) ;

    return 0;
}
```

Kruskal

```
/*
 *  SRC: POJ 1258
 * PROB: Agri-Net
 * ALGO: Kruskal + Disjoint Set
 * DATE: Jul 24, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <algorithm>

using std::sort;

const int MAXN = 110;
const int MAXM = 10010;

struct Edge {
    int u, v, d;

    bool operator<(const Edge &other) const { return d < other.d; }
} e[MAXM];

class Disjoint_Set {
    public:
        int a[MAXN];

        Disjoint_Set() { reset(); }
        void reset() { memset(a, 0xff, sizeof(a)); }

        int find(int u)
        {
            int x = u, y = u;
            while (a[u] >= 0) u = a[u];
            while (a[y] >= 0) {
                x = a[y];
                a[y] = u;
                y = x;
            }
            return u;
        }

        void join(int u, int v)
        {
            int x = find(u),
                y = find(v);
            if (x != y) {
                a[x] += a[y];
                a[y] = x;
            }
        }
};
Disjoint_Set ds;

int kruskal(int m)
{
    sort(e, e + m);

    int sum = 0;
    for (int i = 0; i < m; i++)
        if (ds.find(e[i].u) != ds.find(e[i].v)) {
            ds.join(e[i].u, e[i].v);
            sum += e[i].d;
        }

    return sum;
}

int main()
{
    int n;
    while (~scanf("%d", &n)) {
        int cnt = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
                int d;
                scanf("%d", &d);
                if (i < j) e[cnt++] = (Edge){i, j, d};
            }

        printf("%d\n", kruskal(cnt));

        ds.reset();
    }

    return 0;
}
```

Bellman Ford

```
/*
 *  SRC: POJ 3259
 * PROB: Wormholes
 * ALGO: Bellman Ford
 * DATE: Jul 24, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>

using std::vector;

struct Edge {
    int u, v, d;

    Edge(int _u, int _v, int _d)
        : u(_u), v(_v), d(_d)
    { }
};

const int MAXN = 510;

int dist[MAXN];
vector<Edge> edge;
typedef vector<Edge>::const_iterator vci;

bool bellmanFord()
{
    memset(dist, 0x3f, sizeof(dist));
    edge.clear();

    int n, m, w;
    scanf("%d%d%d", &n, &m, &w);

    for (int i = 0; i < m; i++) {
        int s, e, t;
        scanf("%d%d%d", &s, &e, &t);
        edge.push_back(Edge(s, e, t));
        edge.push_back(Edge(e, s, t));
    }
    for (int i = 0; i < w; i++) {
        int s, e, t;
        scanf("%d%d%d", &s, &e, &t);
        edge.push_back(Edge(s, e, -t));
    }

    dist[1] = 0;
    for (int i = 0; i < n - 1; i++)
        for (vci e = edge.begin(); e != edge.end(); e++)
            if (dist[e->v] > dist[e->u] + e->d)
                dist[e->v] = dist[e->u] + e->d;

    for (vci e = edge.begin(); e != edge.end(); e++)
        if (dist[e->v] > dist[e->u] + e->d)
            return true;

    return false;
}

int main()
{
    int F;
    scanf("%d", &F);

    while (F--) {
        if (bellmanFord()) puts("YES");
        else puts("NO");
    }

    return 0;
}
```

Dijkstra

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>

using std::vector;
using std::priority_queue;

const int MAXN = ;

struct Edge {
    int v, d;

    Edge(int _v, int _d)
        : v(_v), d(_d)
    { }

    bool operator<(const Edge& other) const
    {
        return d > other.d;
    }
};

vector<Edge> edge[MAXN];

int dist[MAXN];

int dijkstra()
{
    memset(dist, 0x3f, sizeof(dist));
    dist[0] = 0;

    priority_queue<Edge> que;
    que.push(Edge(0, 0));

    while (!que.empty()) {
        int u = que.top().v;
        int td = que.top().d;
        que.pop();
        if (td > dist[u]) continue;
        for (int i = 0; i < edge[u].size(); i++) {
            int v = edge[u][i].v;
            int d = edge[u][i].d;
            if (d + dist[u] < dist[v]) {
                dist[v] = d + dist[u];
                que.push(Edge(v, dist[v]));
            }
        }
    }

    return something;
}
```

SPFA

```
/*
 *  SRC: POJ 2983
 * PROB: Is the Information Reliable?
 * ALGO: SPFA
 * DATE: Apr 12, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <queue>

using std::queue;

const int MAXV = 1010;
const int MAXE = 300010;

int dist[MAXV];
int cnt[MAXV];
bool in_queue[MAXV];

struct Edge {
    int v, d;
    Edge *next;
};
Edge e_buf[MAXE];
Edge *e_head[MAXV];
Edge *e_tail = e_buf;

inline void add_edge(int u, int v, int d)
{
    *e_tail = (Edge){v, d, e_head[u]};
    e_head[u] = e_tail++;
}

bool spfa(int n, int src)
{
    memset(dist, 0x3f, sizeof(dist));
    memset(cnt, 0, sizeof(cnt));
    memset(in_queue, false, sizeof(in_queue));
    dist[src] = 0;

    queue<int> que;
    que.push(src);
    while (!que.empty()) {
        int u = que.front();
        que.pop();
        in_queue[u] = false;
        for (Edge *e = e_head[u]; e; e = e->next) {
            int v = e->v;
            int d = e->d;
            if (d + dist[u] < dist[v]) {
                dist[v] = d + dist[u];
                if (!in_queue[v]) {
                    if (cnt[v]++ >= n) return false;
                    que.push(v);
                    in_queue[v] = true;
                }
            }
        }
    }

    return true;
}

int main()
{
    int n, m;
    while (~scanf("%d%d", &n, &m)) {
        memset(e_head, 0, sizeof(e_head));
        e_tail = e_buf;

        // some points may be isolate, so add a super source 0
        for (int i = 1; i <= n; i++) {
            add_edge(0, i, 0);
        }
        for (int i = 0; i < m; i++) {
            char c;
            int u, v;
            scanf(" %c%d%d", &c, &u, &v);
            if (c == 'P') {
                int x;
                scanf("%d", &x);
                add_edge(u, v, -x);
                add_edge(v, u, x);
            } else {
                add_edge(u, v, -1);
            }
        }

        // n points plus a super souce, the 0
        puts(spfa(n + 1, 0) ? "Reliable" : "Unreliable");
    }

    return 0;
}
```

SPFA with SLF and LLL

```
/*
 *  SRC: POJ 2983
 * PROB: Is the Information Reliable?
 * ALGO: SPFA
 * DATE: Apr 12, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <deque>

using std::deque;

const int MAXV = 1010;
const int MAXE = 300010;
const int eps = 1e-12;

inline bool gr(double x, double y) { return x - eps > y; }

int dist[MAXV];
int cnt[MAXV];
bool in_queue[MAXV];

struct Edge {
    int v, d;
    Edge *next;
};
Edge e_buf[MAXE];
Edge *e_head[MAXV];
Edge *e_tail = e_buf;

inline void add_edge(int u, int v, int d)
{
    *e_tail = (Edge){v, d, e_head[u]};
    e_head[u] = e_tail++;
}

bool spfa(int n, int src)
{
    memset(dist, 0x3f, sizeof(dist));
    memset(cnt, 0, sizeof(cnt));
    memset(in_queue, false, sizeof(in_queue));
    dist[src] = 0;

    deque<int> que;
    que.push_back(src);
    double avg = 0;
    while (!que.empty()) {
        while (gr(dist[que.front()], avg)) {
            que.push_back(que.front());
            que.pop_front();
        }
        int u = que.front();
        double tot = avg * que.size() - dist[u];
        que.pop_front();
        in_queue[u] = false;

        for (Edge *e = e_head[u]; e; e = e->next) {
            int v = e->v;
            int d = e->d;
            if (d + dist[u] < dist[v]) {
                dist[v] = d + dist[u];
                if (!in_queue[v]) {
                    if (cnt[v]++ >= n) return false;
                    if (!que.empty() && dist[v] < dist[que.front()]) {
                        que.push_front(v);
                    } else {
                        que.push_back(v);
                    }
                    tot += dist[v];
                    in_queue[v] = true;
                }
            }
        }

        avg = tot / que.size();
    }

    return true;
}

int main()
{
    int n, m;
    while (~scanf("%d%d", &n, &m)) {
        memset(e_head, 0, sizeof(e_head));
        e_tail = e_buf;

        // some points may be isolate, so add a super source 0
        for (int i = 1; i <= n; i++) {
            add_edge(0, i, 0);
        }
        for (int i = 0; i < m; i++) {
            char c;
            int u, v;
            scanf(" %c%d%d", &c, &u, &v);
            if (c == 'P') {
                int x;
                scanf("%d", &x);
                add_edge(u, v, -x);
                add_edge(v, u, x);
            } else {
                add_edge(u, v, -1);
            }
        }

        // n points plus a super souce, the 0
        puts(spfa(n + 1, 0) ? "Reliable" : "Unreliable");
    }

    return 0;
}
```

Tarjan for Strongly Connected Component

```
/*
 *  SRC: POJ 2186
 * PROB: Popular Cows
 * ALGO: Tarjan SCC (Strongly Connected Component)
 * DATE: Jul 23, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

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

using std::vector;
using std::min;
```

```
typedef vector<int>::const_iterator vci;
const int MAXN = 10010;

vector<int> edge[MAXN];

vector<int> stack;
int idx, scc_cnt;
int dfn[MAXN], low[MAXN], scc_id[MAXN], scc_size[MAXN];
bool in_stack[MAXN];

void tarjan_dfs(int u)
{
    stack.push_back(u);
    in_stack[u] = true;
    dfn[u] = low[u] = idx++;

    for (vci v = edge[u].begin(); v != edge[u].end(); v++) {
        if (dfn[*v] == -1) {
            tarjan_dfs(*v);
            low[u] = min(low[u], low[*v]);
        } else if (in_stack[*v]) {
            low[u] = min(low[u], dfn[*v]);
        }
    }

    if (dfn[u] == low[u]) {
        int v;
        do {
            v = stack.back();
            stack.pop_back();
            in_stack[v] = false;
            scc_id[v] = scc_cnt;
            scc_size[scc_cnt]++;
        } while (v != u) ;
        scc_cnt++;
    }
}

void tarjan(int n)
{
    idx = scc_cnt = 0;
    memset(dfn, 0xff, sizeof(dfn));
    memset(low, 0xff, sizeof(low));
    memset(scc_id, 0xff, sizeof(scc_id));
    memset(scc_size, 0, sizeof(scc_size));

    for (int i = 0; i < n; i++)
        if (dfn[i] == -1) tarjan_dfs(i);
}

int n, m;
int out_deg[MAXN];

int solve()
{
    if (scc_cnt == 1) return n;

    for (int u = 0; u < n; u++)
        for (vci v = edge[u].begin(); v != edge[u].end(); v++)
            if (scc_id[u] != scc_id[*v]) out_deg[scc_id[u]]++;

    int ans = 0, ans_cnt = 0;
    for (int i = 0; i < scc_cnt; i++)
        if (out_deg[i] == 0) {
            ans = scc_size[i];
            ans_cnt++;
        }

    return ans_cnt == 1 ? ans : 0;
}

int main()
{
    scanf("%d%d", &n, &m);

    for (int i = 0; i < m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        edge[a - 1].push_back(b - 1);
    }

    tarjan(n);

    printf("%d\n", solve());

    return 0;
}
```

Tarjan for Vertex Biconnected Component

```
/*
 *  SRC: poj 2942
 * PROB: Knights of the Round Table
 * ALGO: Tarjan Vertex BCC (Vertex Biconnected Component
 * DATE: Jul 09, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

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

using namespace std;

typedef vector<int>::const_iterator vci;
const int MAXN = 1010;

vector<int> edge[MAXN];

vector<int> stack;
int idx, bcc_cnt;
int dfn[MAXN], low[MAXN];
vector<int> bcc_block[MAXN];
bool in_stack[MAXN];

void tarjan_dfs(int u, int parent)
{
    stack.push_back(u);
    in_stack[u] = true;
    dfn[u] = low[u] = idx++;

    for (vci v = edge[u].begin(); v != edge[u].end(); v++) {
        if (*v == parent) continue;
        if (dfn[*v] == -1) {
            tarjan_dfs(*v, u);
            low[u] = min(low[u], low[*v]);
            if (dfn[u] <= low[*v]) {
                int p;
                do {
                    p = stack.back();
                    stack.pop_back();
                    in_stack[p] = false;
                    bcc_block[bcc_cnt].push_back(p);
                } while (p != *v) ;
                bcc_block[bcc_cnt++].push_back(u);
            }
        } else if (in_stack[*v]) {
            low[u] = min(low[u], dfn[*v]);
        }
    }
}

void tarjan(int n)
{
    for (int i = 0; i < bcc_cnt; i++) bcc_block[i].clear();
    stack.clear();
    idx = bcc_cnt = 0;
    memset(dfn, 0xff, sizeof(dfn));
    memset(low, 0xff, sizeof(low));
    for (int i = 0; i < n; i++)
        if (dfn[i] == -1) tarjan_dfs(i , -1);
}

int mat[MAXN][MAXN];
bool color[MAXN], to_dye[MAXN], vis[MAXN];
bool expel[MAXN];

bool dye_dfs(int u, bool col)
{
    vis[u] = true;
    color[u] = col;
    for (vci v = edge[u].begin(); v != edge[u].end(); v++) {
        if (to_dye[*v]) {
            if (!vis[*v]) {
                if (dye_dfs(*v, col ^ 1)) return true;
            } else if (col == color[*v]) {
                return true;
            }
        }
    }
    return false;
}

void dye()
{
    memset(expel, true, sizeof(expel));
    for (int i = 0; i < bcc_cnt; i++) {
        memset(to_dye, false, sizeof(to_dye));
        memset(vis, false, sizeof(vis));
        for (int j = 0; j < bcc_block[i].size(); j++) to_dye[bcc_block[i][j]] = true;
        if (dye_dfs(bcc_block[i].back(), 0)) {
            for (int j = 0; j < bcc_block[i].size(); j++) expel[bcc_block[i][j]] = false;
        }
    }
}

int main()
{
    int n, m;
    while (scanf("%d%d", &n, &m), n || m) {
        memset(mat, 0, sizeof(mat));
        for (int i = 0; i < m; i++) {
            int x, y;
            scanf("%d%d", &x, &y);
            x--; y--;
            mat[x][y] = mat[y][x] = 1;
        }

        for (int i = 0; i < n; i++) {
            edge[i].clear();
            for (int j = 0; j < n; j++) {
                if (i != j && !mat[i][j]) edge[i].push_back(j);
            }
        }

        tarjan(n);
        dye();

        int ans = 0;
        for (int i = 0; i < n; i++)
            if (expel[i]) ans++;
        printf("%d\n", ans);
    }

    return 0;
}
```

Tarjan for Edge Biconnected Component

```
/*
 *  SRC: POJ 3352
 * PROB: Road Construction
 * ALGO: Tarjan Edge BCC (Edge Biconnected Component)
 * DATE: Apr 12, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

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

using std::vector;
using std::min;

typedef vector<int>::const_iterator vci;
const int MAXN = 10010;

vector<int> edge[MAXN];

vector<int> stack;
int idx, bcc_cnt;
int dfn[MAXN], low[MAXN], bcc_id[MAXN], bcc_size[MAXN];
bool in_stack[MAXN];

void tarjan_dfs(int u, int parent)
{
    stack.push_back(u);
    in_stack[u] = true;
    dfn[u] = low[u] = idx++;

    for (vci v = edge[u].begin(); v != edge[u].end(); v++) {
        if (*v == parent) continue;
        if (dfn[*v] == -1) {
            tarjan_dfs(*v, u);
            low[u] = min(low[u], low[*v]);
        } else if (in_stack[*v]) {
            low[u] = min(low[u], dfn[*v]);
        }
    }

    if (dfn[u] == low[u]) {
        int v;
        do {
            v = stack.back();
            stack.pop_back();
            in_stack[v] = false;
            bcc_id[v] = bcc_cnt;
            bcc_size[bcc_cnt]++;
        } while (v != u) ;
        bcc_cnt++;
    }
}

void tarjan(int n)
{
    idx = bcc_cnt = 0;
    memset(dfn, 0xff, sizeof(dfn));
    memset(low, 0xff, sizeof(low));
    memset(bcc_id, 0xff, sizeof(bcc_id));
    memset(bcc_size, 0, sizeof(bcc_size));
    for (int i = 0; i < n; i++)
        if (dfn[i] == -1) tarjan_dfs(i, -1);
}

int degree[MAXN];

int count_leaves(int n)
{
    memset(degree, 0, sizeof(degree));

    for (int u = 0; u < n; u++) {
        for (vci v = edge[u].begin(); v != edge[u].end(); v++) {
            if (bcc_id[u] != bcc_id[*v]) {
                degree[bcc_id[u]]++;
            }
        }
    }

    int cnt = 0;
    for (int i = 0; i < bcc_cnt; i++) {
        if (degree[i] == 1) cnt++;
    }

    return cnt;
}

int main()
{
    int n, r;
    scanf("%d%d", &n, &r);
    for (int i = 0; i < r; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        edge[u - 1].push_back(v - 1);
        edge[v - 1].push_back(u - 1);
    }

    tarjan(n);

    printf("%d\n", (count_leaves(n) + 1) >> 1);

    return 0;
}
```

Tarjan for Lowest Common Ancestor

```
/*
 *  SRC: POJ 1986
 * PROB: Distance Queries
 * ALGO: Tarjan LCA (Lowest Common Ancestor)
 * DATE: May 28, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>

using std::vector;

const int MAXN = 40000;

class Disjoint_Set {
    public:
        int a[MAXN];

        Disjoint_Set() { reset(); }
        void reset() { memset(a, 0xff, sizeof(a)); }

        int find(int u)
        {
            int x = u, y = u;
            while (a[u] >= 0) u = a[u];
            while (a[y] >= 0) {
                x = a[y];
                a[y] = u;
                y = x;
            }
            return u;
        }

        void join(int u, int v)
        {
            int x = find(u),
                y = find(v);
            if (x != y) {
                a[x] += a[y];
                a[y] = x;
            }
        }
};
```

```
struct Edge {
    int v, d;
    Edge(int _v, int _d) : v(_v), d(_d) { }
};

typedef vector<Edge>::const_iterator vci;
vector<Edge> edge[MAXN];
vector<Edge> query[MAXN];
Disjoint_Set ds;
int parent[MAXN], dist[MAXN], ans[MAXN];
bool vis[MAXN];

void tarjan(int u, int length)
{
    vis[u] = true;
    dist[u] = length;

    for (vci e = query[u].begin(); e != query[u].end(); e++) {
        if (vis[e->v]) {
            // ds.find(e->v) is exactly the lowest common ancestor of u and e->v
            ans[e->d] = dist[u] + dist[e->v] - dist[ds.find(e->v)] * 2;
        }
    }

    for (vci e = edge[u].begin(); e != edge[u].end(); e++) {
        if (e->v != parent[u]) {
            parent[e->v] = u;
            tarjan(e->v, length + e->d);
        }
    }

    ds.join(parent[u], u);
}

int main()
{
    int n;
    scanf("%d%*d", &n);
    for (int i = 0; i < n; i++) edge[i].clear();
    for (int i = 0; i < n; i++) query[i].clear();
    for (int i = 1; i < n; i++) {
        int u, v, d;
        scanf("%d%d%d%*s", &u, &v, &d);
        edge[u - 1].push_back(Edge(v - 1, d));
        edge[v - 1].push_back(Edge(u - 1, d));
    }

    int k;
    scanf("%d", &k);
    for (int i = 0; i < k; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        query[u - 1].push_back(Edge(v - 1, i));
        query[v - 1].push_back(Edge(u - 1, i));
    }

    memset(vis, false, sizeof(vis));
    parent[0] = 0;
    tarjan(0, 0);
    for (int i = 0; i < k; i++) printf("%d\n", ans[i]);

    return 0;
}
```

2-SAT

```
/*
 *  SRC: POJ 3648
 * PROB: Wedding
 * ALGO: 2-SAT
 * DATE: Oct 16, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>
#include <algorithm>

using std::vector;
using std::queue;
using std::min;

typedef vector<int>::const_iterator vci;
const int MAXN = 1010 << 1;

vector<int> edge[MAXN];
vector<int> new_edge[MAXN];

vector<int> stack;
```

```
int idx, scc_cnt;
int dfn[MAXN], low[MAXN], scc_id[MAXN], scc_size[MAXN];
bool in_stack[MAXN];

int in_deg[MAXN], color[MAXN];

void tarjan_dfs(int u)
{
    stack.push_back(u);
    in_stack[u] = true;
    dfn[u] = low[u] = idx++;

    for (vci v = edge[u].begin(); v != edge[u].end(); v++) {
        if (dfn[*v] == -1) {
            tarjan_dfs(*v);
            low[u] = min(low[u], low[*v]);
        } else if (in_stack[*v])
            low[u] = min(low[u], dfn[*v]);
    }

    if (dfn[u] == low[u]) {
        int v;
        do {
            v = stack.back();
            stack.pop_back();
            in_stack[v] = false;
            scc_id[v] = scc_cnt;
            scc_size[scc_cnt]++;
        } while (v != u) ;
        scc_cnt++;
    }
}

void tarjan(int n)
{
    idx = scc_cnt = 0;
    memset(dfn, 0xff, sizeof(dfn));
    memset(low, 0xff, sizeof(low));
    memset(scc_id, 0xff, sizeof(scc_id));
    memset(scc_size, 0, sizeof(scc_size));

    for (int i = 0; i < n; i++)
        if (dfn[i] == -1) tarjan_dfs(i);
}

void color_dfs(int u)
{
    color[u] = 2;
    for (vci v = new_edge[u].begin(); v != new_edge[u].end(); v++)
        if (!color[*v]) color_dfs(*v);
}

bool sat(int n)
{
    tarjan(n);

    for (int i = 0; i < n; i += 2)
        if (scc_id[i] == scc_id[i ^ 1]) return false;

    memset(in_deg, 0, sizeof(in_deg));
    for (int u = 0; u < n; u++)
        for (vci v = edge[u].begin(); v != edge[u].end(); v++)
            if (scc_id[u] != scc_id[*v]) {
                new_edge[scc_id[*v]].push_back(scc_id[u]);
                in_deg[scc_id[u]]++;
            }

    queue<int> Q;
    vector<int> topo;
    for (int i = 0; i < scc_cnt; i++)
        if (!in_deg[i]) Q.push(i);
    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        topo.push_back(u);
        for (vci v = new_edge[u].begin(); v != new_edge[u].end(); v++)
            if (--in_deg[*v] == 0) Q.push(*v);
    }

    memset(color, 0, sizeof(color));
    for (vci u = topo.begin(); u != topo.end(); u++)
        if (!color[*u]) {
            color[*u] = 1;
            for (int i = 0; i < n; i++)
                if (scc_id[i] == *u)
                    if (!color[scc_id[i ^ 1]])
                        color_dfs(scc_id[i ^ 1]);
        }

    return true;
}

inline int which(char c) { return c == 'w' ? 0 : 1; }

int main()
{
    int n, m;
    while (scanf("%d%d", &n, &m), n || m) {
        for (int i = 0; i < n * 2; i++) {
            edge[i].clear();
            new_edge[i].clear();
        }

        for (int i = 0; i < m; i++) {
            int  id1, id2;
            char hw1, hw2;
            scanf("%d%c%d%c", &id1, &hw1, &id2, &hw2);
            edge[(id1 << 1) ^ which(hw1)].push_back((id2 << 1) ^ (!which(hw2)));
            edge[(id2 << 1) ^ which(hw2)].push_back((id1 << 1) ^ (!which(hw1)));
        }
        edge[0].push_back(1);

        if (!sat(n * 2)) {
            puts("bad luck");
            continue;
        }

        int c = color[scc_id[0]];
        bool first = true;
        for (int i = 2; i < n * 2; i++)
            if (color[scc_id[i]] == c) {
                if (!first) putchar(' ');
                printf("%d%c", i >> 1, (i & 1) ? 'h' : 'w');
                first = false;
            }
        putchar(10);
    }

    return 0;
}
```

Hungarian

```
/*
 *  SRC: POJ 3041
 * PROB: Asteroids
 * ALGO: Hungarian
 * DATE: Mar 05, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>

using std::vector;

const int MAXN = 555;

vector<int> edge[MAXN];
typedef vector<int>::const_iterator vci;

int match[MAXN];
bool vis[MAXN];

bool find_path(int u)
{
    for (vci v = edge[u].begin(); v != edge[u].end(); v++)
        if (!vis[*v]) {
            vis[*v] = true;
            if (match[*v] == -1 || find_path(match[*v])) {
                match[*v] = u;
                return true;
            }
        }

    return false;
}

int hungarian(int n)
{
    int res = 0;
    memset(match, 0xff, sizeof(match));
    for (int i = 0; i < n; i++) {
        memset(vis, 0, sizeof(vis));
        if (find_path(i)) res++;
    }

    return res;
}

int main()
{
    int n, k;
    scanf("%d%d", &n, &k);
    for (int i = 0; i < k; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        edge[x - 1].push_back(y - 1);
    }
    printf("%d\n", hungarian(n));

    return 0;
}
```

KM

```
/*
 *  SRC: HDOJ 3722
 * PROB: Card Game
 * ALGO: KM
 * DATE: Oct 16, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>

const int MAXN = 300;
const int INF = 0x3f3f3f3f;

int n;
int  w[MAXN][MAXN],
     lx[MAXN], ly[MAXN],
     match[MAXN], slack[MAXN];
bool visx[MAXN], visy[MAXN];

bool find_path(int x)
{
    visx[x] = true;
    for (int y = 0; y < n; y++) {
        if (visy[y]) continue;
        int t = lx[x] + ly[y] - w[x][y];
        if (t == 0) {
            visy[y] = true;
            if (match[y] == -1 || find_path(match[y])) {
                match[y] = x;
                return true;
            }
        }
        else if (slack[y] > t) slack[y] = t;
    }
    return false;
}

int KM()
{
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    memset(match, 0xff, sizeof(match));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (lx[i] < w[i][j]) lx[i] = w[i][j];

    for (int x = 0; x < n; x++) {
        memset(slack, 0x3f, sizeof(slack));
        while (1) {
            memset(visx, 0, sizeof(visx));
            memset(visy, 0, sizeof(visy));
            if (find_path(x)) break;

            int d = INF;
            for (int i = 0; i < n; i++)
                if (!visy[i] && slack[i] < d) d = slack[i];
            for (int i = 0; i < n; i++)
                if (visx[i]) lx[i] -= d;
            for (int i = 0; i < n; i++) {
                if (visy[i]) ly[i] += d;
                else slack[i] -= d;
            }
        }
    }

    int res = 0;
    for (int i = 0; i < n; i++) res += w[match[i]][i];
    return res;
}

char card[MAXN][1010];

int eval_w(char *a, char *b)
{
    int len_a = strlen(a) - 1;

    int res = 0;
    while (len_a >= 0 && *b && a[len_a] == *b) res++, len_a--, b++;
    return res;
}

int main()
{
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; i++) scanf("%s", card[i]);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
                if (i == j) w[i][j] = 0;
                else w[i][j] = eval_w(card[i], card[j]);
            }
        printf("%d\n", KM());
    }

    return 0;
}
```

Dinic (Pointer)

```
/*
 *  SRC: POJ 3281
 * PROB: Dining
 * ALGO: Dinic
 * DATE: Jun 2, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <algorithm>

using std::min;

const int INF = 0x3f3f3f3f;
const int MAXV = 1010;
const int MAXE = 1000000;
const int orig = 0, dest = MAXV - 1;

struct Edge {
    int v;
    int c, f; // capa, flow
    Edge *next;
    Edge *rev; // revese edge
};
Edge e_buf[MAXE];
Edge *e_head[MAXV];
Edge *e_work[MAXV];
Edge *e_tail;

int que[MAXV], lev[MAXV];

inline void add_edge(int u, int v, int c)
{
    *e_tail = (Edge){v, c, 0, e_head[u]};
    e_head[u] = e_tail++;

    *e_tail = (Edge){u, 0, 0, e_head[v]};
    e_head[v] = e_tail++;

    e_head[u]->rev = e_head[v];
    e_head[v]->rev = e_head[u];
}

bool bfs()
{
    memset(lev, 0xff, sizeof(lev));

    int *head = que, *tail = que;
    *tail++ = orig;
    lev[orig] = 0;

    while (head != tail) {
        int u = *head++;
        for (Edge *e = e_head[u]; e; e = e->next)
            if (lev[e->v] == -1 && e->f < e->c) {
                lev[e->v] = lev[u] + 1;
                *tail++ = e->v;
            }
    }

    return lev[dest] > -1;
}

int dfs(int u, int f)
{
    if (u == dest) return f;

    int res = 0;
    for (Edge *&e = e_work[u]; e; e = e->next)
        if (lev[e->v] == lev[u] + 1 && e->f < e->c) {
            int tmp = dfs(e->v, min(f - res, e->c - e->f));
            res += tmp;
            e->f += tmp;
            e->rev->f = -e->f;
            if (res == f) break;
        }
    return res;
}

int dinic()
{
    int res = 0;
    while (bfs()) {
        memcpy(e_work, e_head, sizeof(e_head));
        int tmp = dfs(orig, INF);
        if (tmp) res += tmp;
        else break;
    }

    return res;
}

void build_graph()
{
    int N, F, D;
    scanf("%d%d%d", &N, &F, &D);

    /*
     * 0: orig
     * 1 to F: FF
     * F + 1 to F + N: C1
     * F + N + 1 to F + 2N: C2
     * F + 2N + 1 to F + 2N + D: DD
     * MAXV: dest
     */

    const int FF = 0;
    const int C1 = F, C2 = F + N;
    const int DD = F + 2 * N;

    for (int i = 1; i <= F; i++) {
        // orig -> FF, capa = 1
        // FF -> orig, capa = 0
        add_edge(orig, FF + i, 1);
    }
    for (int i = 1; i <= N; i++) {
        // C1 -> C2, capa = 1
        // C2 -> C1, capa = 0
        add_edge(C1 + i, C2 + i, 1);
    }
    for (int i = 1; i <=D; i++) {
        // DD -> dest, capa = 1
        // dest -> DD, capa = 0
        add_edge(DD + i, dest, 1);
    }

    for (int i = 1; i <= N; i++) {
        int f, d;
        scanf("%d%d", &f, &d);
        for (int j = 0; j < f; j++) {
            int tmp;
            scanf("%d", &tmp);
            // FF -> C1, capa = 1
            // C1 -> FF, capa = 0
            add_edge(FF + tmp, C1 + i, 1);
        }
        for (int j = 0; j < d; j++) {
            int tmp;
            scanf("%d", &tmp);
            // C2 -> DD, capa = 1
            // DD -> C2, capa = 0
            add_edge(C2 + i, DD + tmp, 1);
        }
    }
}

int main()
{
    // memset(e_buf, 0, sizeof(e_buf));
    memset(e_head, 0, sizeof(e_head));
    e_tail = e_buf;

    build_graph();

    printf("%d\n", dinic());

    return 0;
}
```

Dinic (STL)

```
/*
 *  SRC: POJ 3281
 * PROB: Dining
 * ALGO: Dinic
 * DATE: Jun 2, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>
#include <algorithm>

using std::vector;
using std::queue;
using std::min;

const int INF = 0x3f3f3f3f;
const int MAXN = 1010;
const int orig = 0, dest = MAXN - 1;

struct Edge {
    int v;
    int rev; // the position of revese edge
    int c, f; // capa, flow

    Edge(int _v, int _rev, int _c)
        : v(_v), rev(_rev), c(_c), f(0)
    { }
};
vector<Edge> edge[MAXN];

int lev[MAXN + 1];

inline void add_edge(int u, int v, int capa)
{
    edge[u].push_back(Edge(v, edge[v].size(), capa));
    edge[v].push_back(Edge(u, edge[u].size() - 1, 0));
}

bool bfs()
{
    memset(lev, 0xff, sizeof(lev));

    queue<int> Q;
    Q.push(orig);
    lev[orig] = 0;

    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        for (unsigned int i = 0; i < edge[u].size(); i++) {
            int v = edge[u][i].v;
            if (lev[v] == -1 && edge[u][i].f < edge[u][i].c) {
                lev[v] = lev[u] + 1;
                Q.push(v);
            }
        }
    }

    return lev[dest] > -1;
}

int dfs(int u, int f)
{
    if (u == dest) return f;

    int res = 0;
    for (unsigned int i = 0; i < edge[u].size(); i++) {
        int v = edge[u][i].v;
        if (lev[v] == lev[u] + 1 && edge[u][i].f < edge[u][i].c) {
            int tmp = dfs(v, min(f - res, edge[u][i].c - edge[u][i].f));
            res += tmp;
            edge[u][i].f += tmp;
            int j = edge[u][i].rev;
            edge[v][j].f = -edge[u][i].f;
            if (res == f) break;
        }
    }

    return res;
}

int dinic()
{
    int res = 0;
    while (bfs()) {
        int tmp = dfs(orig, MAXN);
        if (tmp) res += tmp;
        else break;
    }

    return res;
}

void build_graph()
{
    int N, F, D;
    scanf("%d%d%d", &N, &F, &D);

    /*
     * 0: orig
     * 1 to F: FF
     * F + 1 to F + N: C1
     * F + N + 1 to F + 2N: C2
     * F + 2N + 1 to F + 2N + D: DD
     * MAXN: dest
     */

    const int FF = 0;
    const int C1 = F, C2 = F + N;
    const int DD = F + 2 * N;

    for (int i = 1; i <= F; i++) {
        // orig -> FF, capa = 1
        // FF -> orig, capa = 0
        add_edge(orig, FF + i, 1);
    }
    for (int i = 1; i <= N; i++) {
        // C1 -> C2, capa = 1
        // C2 -> C1, capa = 0
        add_edge(C1 + i, C2 + i, 1);
    }
    for (int i = 1; i <=D; i++) {
        // DD -> dest, capa = 1
        // dest -> DD, capa = 0
        add_edge(DD + i, dest, 1);
    }

    for (int i = 1; i <= N; i++) {
        int f, d;
        scanf("%d%d", &f, &d);
        for (int j = 0; j < f; j++) {
            int tmp;
            scanf("%d", &tmp);
            // FF -> C1, capa = 1
            // C1 -> FF, capa = 0
            add_edge(FF + tmp, C1 + i, 1);
        }
        for (int j = 0; j < d; j++) {
            int tmp;
            scanf("%d", &tmp);
            // C2 -> DD, capa = 1
            // DD -> C2, capa = 0
            add_edge(C2 + i, DD + tmp, 1);
        }
    }
}

int main()
{
    build_graph();

    printf("%d\n", dinic());

    return 0;
}
```

Minimum Cost Flow (pointer)

```
/*
 *  SRC: POJ 2135
 * PROB: Farm Tour
 * ALGO: MCMF
 * DATE: Jul 25, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <algorithm>

using std::min;

const int INF = 0x3f3f3f3f;
const int MAXV = 1010;
const int MAXE = 1000000;
const int orig = 0, dest = 1000;

struct Edge {
    int v;
    int c, f; // capa, flow
    int cpf; // cost per flow
    Edge *next,
         *rev; // revese edge
};
Edge e_buf[MAXE],
     *e_tail,
     *e_head[MAXV];

struct RoadNode {
    RoadNode *next;
    Edge *which; // which edge
};
RoadNode road[MAXV];
```

```
int que[MAXE], dist[MAXV];
bool vis[MAXV];

inline void add_edge(int u, int v, int c, int cpf)
{
    *e_tail = (Edge){v, c, 0, cpf, e_head[u]};
    e_head[u] = e_tail++;

    *e_tail = (Edge){u, 0, 0, -cpf, e_head[v]};
    e_head[v] = e_tail++;

    e_head[u]->rev = e_head[v];
    e_head[v]->rev = e_head[u];
}

bool spfa()
{
    memset(vis, false, sizeof(vis));
    memset(dist, 0x3f, sizeof(dist));

    int *head = que,
        *tail = que;
    *tail++ = orig;
    vis[orig] = true;
    dist[orig] = 0;
    road[orig].next = 0;

    while (head != tail) {
        int u = *head++;
        vis[u] = false;
        for (Edge *e = e_head[u]; e; e = e->next)
            if (e->cpf + dist[u] < dist[e->v] && e->f < e->c) {
                dist[e->v] = e->cpf + dist[u];
                road[e->v].next = &road[u];
                road[e->v].which = e;
                if (vis[e->v] == false) {
                    vis[e->v] = true;
                    *tail++ = e->v;
                }
            }
    }

    return dist[dest] < INF;
}

int flow()
{
    int min_flow = INF;
    for (RoadNode *r = &road[dest]; r->next; r = r->next)
        min_flow = min(min_flow, r->which->c - r->which->f);

    int res = 0;
    for (RoadNode *r = &road[dest]; r->next; r = r->next) {
        r->which->f += min_flow;
        res += r->which->cpf;
        r->which->rev->f = -r->which->f;
    }
    res *= min_flow;

    return res;
}

int mcmf()
{
    int res = 0;
    while (spfa()) res += flow();

    return res;
}

void build_graph()
{
    int n, m;
    scanf("%d%d", &n, &m);
    add_edge(orig, 1, 2, 0);
    for (int i = 0; i < m; i++) {
        int s, e, v;
        scanf("%d%d%d", &s, &e, &v);
        add_edge(s, e, 1, v);
        add_edge(e, s, 1, v);
    }
    add_edge(n, dest, 2, 0);
}

int main()
{
    memset(e_buf, 0, sizeof(e_buf));
    memset(e_head, 0, sizeof(e_head));
    e_tail = e_buf;

    build_graph();

    printf("%d\n", mcmf());

    return 0;
}
```

Minimum Cost Flow (STL)

```
/*
 *  SRC: POJ 2135
 * PROB: Farm Tour
 * ALGO: MCMF
 * DATE: Jul 25, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <vector>
#include <queue>

using std::vector;
using std::queue;

inline int fmin(int a, int b) { return a < b ? a : b; }

const int INF = 0x3f3f3f3f;
const int MAXN = 1010;
const int orig = 0, dest = MAXN;

struct Edge {
    int v;
    int rev; // the position of revese edge
    int c, f; // capa, flow
    int cpf; // cost per flow

    Edge(int _v, int _rev, int _c, int _cpf)
        : v(_v), rev(_rev), c(_c), f(0), cpf(_cpf)
    { }
};
vector<Edge> edge[MAXN + 1];

struct Route {
    int u, v, which;

    Route(int _u = 0, int _v = 0, int _which = 0)
        : u(_u), v(_v), which(_which)
    { }
};
Route bfs_route[MAXN + 1];
bool vis[MAXN + 1];
int dist[MAXN + 1];

inline void add_edge(int u, int v, int capa, int cpf)
{
    edge[u].push_back(Edge(v, edge[v].size(), capa, cpf));
    edge[v].push_back(Edge(u, edge[u].size() - 1, 0, -cpf));
}

bool spfa()
{
    memset(vis, false, sizeof(vis));
    memset(dist, 0x3f, sizeof(dist));

    queue<int> Q;
    Q.push(orig);
    vis[orig] = true;
    dist[orig] = 0;
    bfs_route[orig] = Route(-1, 0, -1);

    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        vis[u] = false;
        for (unsigned int i = 0; i < edge[u].size(); i++) {
            int v = edge[u][i].v,
                cpf = edge[u][i].cpf;
            if (cpf + dist[u] < dist[v] && edge[u][i].f < edge[u][i].c) {
                dist[v] = cpf + dist[u];
                bfs_route[v] = Route(u, v, i);
                if (vis[v] == false) {
                    vis[v] = true;
                    Q.push(v);
                }
            }
        }
    }

    return dist[dest] < INF;
}

int flow()
{
    int min_flow = INF;
    Route *r = &bfs_route[dest];
    while (r->u != -1) {
        min_flow = fmin(min_flow, edge[r->u][r->which].c - edge[r->u][r->which].f);
        r = &bfs_route[r->u];
    }
    int res = 0;
    r = &bfs_route[dest];
    while (r->u != -1) {
        edge[r->u][r->which].f += min_flow;
        res += edge[r->u][r->which].cpf;
        int j = edge[r->u][r->which].rev;
        edge[r->v][j].f = -edge[r->u][r->which].f;
        r = &bfs_route[r->u];
    }
    res *= min_flow;

    return res;
}

int mcmf()
{
    int res = 0;
    while (spfa()) res += flow();

    return res;
}

void build_graph()
{
    int n, m;
    scanf("%d%d", &n, &m);
    add_edge(orig, 1, 2, 0);
    for (int i = 0; i < m; i++) {
        int s, e, v;
        scanf("%d%d%d", &s, &e, &v);
        add_edge(s, e, 1, v);
        add_edge(e, s, 1, v);
    }
    add_edge(n, dest, 2, 0);
}

int main()
{
    build_graph();

    printf("%d\n", mcmf());

    return 0;
}
```

Stoer Wagner

```
/*
 *  SRC: HDOJ 3691
 * PROB: Nubulsa Expo
 * ALGO: Stoer-Wagner
 * DATE: Oct 12, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>

const int INF = 0x3f3f3f3f;
const int MAXN = 333;

int n, m;
int mat[MAXN][MAXN];

bool A[MAXN];
int v[MAXN], w[MAXN];

int stoer_wagner(int n)
{
    int res = INF;
    for (int i = 0; i < n; i++) v[i] = i;

    while (n > 1) {
        A[v[0]] = true;

        for (int i = 1; i < n; i++) {
            A[v[i]] = false;
            w[v[i]] = mat[v[0]][v[i]];
        }

        for (int prev = 0, i = 1; i < n; i++) {
            int curr = -1;
            for (int j = 1; j < n; j++)
                if (!A[v[j]] && (curr == -1 || w[v[j]] > w[v[curr]]))
                    curr = j;
            A[v[curr]] = true;

            if (i == n - 1) {
                if (res > w[v[curr]]) res = w[v[curr]];
                for (int j = 0; j < n; j++)
                    mat[v[j]][v[prev]] = mat[v[prev]][v[j]] += mat[v[curr]][v[j]];
                v[curr] = v[--n];
                break;
            }

            prev = curr;
            for (int j = 1; j < n; j++)
                if (!A[v[j]]) w[v[j]] += mat[v[curr]][v[j]];
        }
    }

    return res;
}

int main()
{
    while (scanf("%d%d%*d", &n, &m), n && m) {
        memset(mat, 0, sizeof(mat));

        for (int i = 0; i < m; i++) {
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
            a--, b--;
            mat[a][b] = mat[b][a] += c;
        }

        printf("%d\n", stoer_wagner(n));
    }

    return 0;
}
```

Tree

BIT

```
/*
 *  SRC: POJ 2352
 * PROB: Stars
 * ALGO: BIT
 * DATE: Jul 20, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <algorithm>

using std::sort;

class BIT {
    private:
        const static int bound = 320010;
        int c[bound + 1];
        int bit_mask;

        // x must *not* be *zero*
        int lowbit(int x) { return x & -x; }

        void get_bit_mask()
        {
            int cnt = 0;
            for ( ; bound >> cnt; cnt++)
                ;

            bit_mask = 1 << (cnt - 1);
        }

    public:
        void update(int x, int v)
        {
            for ( ; x <= bound; x += lowbit(x))
                c[x] += v;
        }

        int sum(int x)
        {
            int res = 0;
            for ( ; x > 0; x -= lowbit(x))
                res += c[x];
            return res;
        }

        // get_bit_mask() first
        int find(int tot)
        {
            int res = 0, cnt = 0, bm = bit_mask;
            while (bm != 0) {
                if (res + bm < bound && cnt + c[res + bm] < tot) { // find the left one
             // if (res + bm <=bound && cnt + c[res + bm] <=tot)   // find the right one
                    res += bm;
                    cnt += c[res];
                }
                bm >>= 1;
            }

            return res + 1; // left
         // return res;     // right
        }
};
BIT bit;

struct Point {
    int x, y;

    bool operator<(const Point& other) const
    {
        if (x == other.x) return y < other.y;
        return x < other.x;
    }
} star[15010];

int n;
int level[15010];

int main()
{
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        scanf("%d%d", &star[i].x, &star[i].y);
    sort(star, star + n);

    for (int i = 0; i < n; i++) {
        bit.update(star[i].y + 1, 1);
        level[bit.sum(star[i].y + 1) - 1]++;
    }

    for (int i = 0; i < n; i++)
        printf("%d\n", level[i]);

    return 0;
}
```

BIT-2D

```
/*
 *  SRC: POJ 1195
 * PROB: Mobile phones
 * ALGO: 2D BIT(Binary Indexed Tree)
 * DATE: Jul 19, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>

class BIT_2D {
    private:
        const static int bound_x = 1024, bound_y = 1024;
        int c[bound_x + 1][bound_y + 1];

        // x must *not* be *zero*
        int lowbit(int x) { return x & -x; }

    public:
        void update(int x, int y, int v)
        {
            for ( ; x <= bound_x; x += lowbit(x))
                for (int ty = y; ty <= bound_y; ty += lowbit(ty))
                    c[x][ty] += v;
        }

        int sum(int x, int y)
        {
            int res = 0;
            for ( ; x; x -= lowbit(x))
                for (int ty = y; ty; ty -= lowbit(ty))
                    res += c[x][ty];
            return res;
        }
};
BIT_2D bit;

int n;

int main()
{
    int ins;
    scanf("%d", &ins);
    scanf("%d", &n);

    while (scanf("%d", &ins) != EOF && ins != 3) {
        if (ins == 1) {
            int x, y, a;
            scanf("%d%d%d", &x, &y, &a);
            bit.update(x + 1, y + 1, a);
        }
        else {
            int l, b, r, t;
            scanf("%d%d%d%d", &l, &b, &r, &t);
            printf("%d\n", bit.sum(r + 1, t + 1) - bit.sum(l, t + 1) - \
                           bit.sum(r + 1, b) + bit.sum(l, b));
        }
    }

    return 0;
}
```

Segment Tree (heap)

```
/*
 *  SRC: POJ 3264
 * PROB: Balanced Lineup
 * ALGO: Segment Tree
 * DATE: Jul 21, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */
```

```
#include <cstdio>
#include <algorithm>

using namespace std;

inline int LC(int x) { return x << 1; }
inline int RC(int x) { return (x << 1) | 1; }

class Segment_Tree {
    private:
        const static int INF = 0x3FFFFFFF;

        struct Tnode {
            int a, b;
            int mn, mx; // min, max
        };
        Tnode node[200000];

    public:
        void build(int a, int b, int idx = 1)
        {
            node[idx].a = a;
            node[idx].b = b;
            node[idx].mn = INF;
            node[idx].mx = -INF;

            if (a + 1 < b) {
                build(a, (a + b) >> 1, LC(idx));
                build((a + b) >> 1, b, RC(idx));
            }
        }

        void insert(int c, int d, int v, int idx = 1)
        {
            if (c <= node[idx].a && node[idx].b <= d) {
                node[idx].mn = min(node[idx].mn, v);
                node[idx].mx = max(node[idx].mx, v);
                return ;
            }

            if (c < (node[idx].a + node[idx].b) >> 1) insert(c, d, v, LC(idx));
            if (d > (node[idx].a + node[idx].b) >> 1) insert(c, d, v, RC(idx));
            node[idx].mn = min(node[LC(idx)].mn, node[RC(idx)].mn);
            node[idx].mx = max(node[LC(idx)].mx, node[RC(idx)].mx);
        }

        void query(int c, int d, int* mn, int* mx, int idx = 1)
        {
            if (c <= node[idx].a && node[idx].b <= d) {
                *mn = node[idx].mn;
                *mx = node[idx].mx;
                return ;
            }

            int l_mn = INF, l_mx = -INF, r_mn = INF, r_mx = -INF;
            if (c < (node[idx].a + node[idx].b) >> 1) query(c, d, &l_mn, &l_mx, LC(idx));
            if (d > (node[idx].a + node[idx].b) >> 1) query(c, d, &r_mn, &r_mx, RC(idx));
            *mn = min(l_mn, r_mn);
            *mx = max(l_mx, r_mx);
        }
};

Segment_Tree st;

int main()
{
    int n, q;
    scanf("%d%d", &n, &q);

    st.build(0, n);

    for (int i = 0; i < n; i++) {
        int h;
        scanf("%d", &h);
        st.insert(i, i + 1, h);
    }

    for (int i = 0; i< q; i++) {
        int a, b;
        scanf("%d%d", &a, &b);

        int mn, mx;
        st.query(a - 1, b, &mn, &mx);

        printf("%d\n", mx - mn);
    }

    return 0;
}
```

Segment Tree (pointer)

```
/*
```

```
 *  SRC: POJ 2777
 * PROB: Count Color
 * ALGO: Segment Tree
 * DATE: Jul 20, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>

class Segment_Tree {
    private:
        struct Tnode {
            int a, b; // segment [a, b)
            int cover;
            int pure;
            Tnode *lc;
            Tnode *rc;

            Tnode(int _a, int _b)
                : a(_a), b(_b), cover(1), pure(1), lc(0), rc(0)
            { }
        };
        Tnode *root;

        Tnode *build(int a, int b)
        {
            Tnode *p = new Tnode(a, b);

            if (a + 1 < b) {
                p->lc = build(a, (a + b) / 2);
                p->rc = build((a + b) / 2, b);
            }

            return p;
        }

        void _insert(int c, int d, int color, Tnode *p)
        {
            if (c <= p->a && p->b <= d) {
                p->pure = 1;
                p->cover = color;

                return ;
            }

            if (p->pure) {
                p->pure = 0;

                p->lc->cover = p->cover;
                p->lc->pure = 1;

                p->rc->cover = p->cover;
                p->rc->pure = 1;
            }

            if (c < (p->a + p->b) / 2) _insert(c, d, color, p->lc);
            if (d > (p->a + p->b) / 2) _insert(c, d, color, p->rc);
            p->cover = p->lc->cover | p->rc->cover;
        }

        int _query(int c, int d, Tnode *p)
        {
            if ((c <= p->a && p->b <= d) || p->pure)
                return p->cover;

            int res = 0;
            if (c < (p->a + p->b) / 2) res |= _query(c, d, p->lc);
            if (d > (p->a + p->b) / 2) res |= _query(c, d, p->rc);

            return res;
        }

    public:
        Segment_Tree(int l, int r)
        {
            root = build(l, r);
        }

        void insert(int c, int d, int color)
        {
            _insert(c, d, color, root);
        }

        int query(int c, int d)
        {
            return _query(c, d, root);
        }
};

int main()
{
    int l, t, o;
    scanf("%d%d%d", &l, &t, &o);

    Segment_Tree st(1, l + 1);
    for (int i = 0; i < o; i++) {
        char ctrl[10];
        int a, b;
        scanf("%s%d%d", ctrl, &a, &b);
        if (a > b) {
            int t = a;
            a = b;
            b = t;
        }
        if (ctrl[0] == 'C') {
            int c;
            scanf("%d", &c);
            st.insert(a, b + 1, 1 << (c - 1));
        }
        else {
            int colors = st.query(a, b + 1);
            int cnt = 0;
            while (colors) {
                cnt++;
                colors -= colors & -colors;
            }

            printf("%d\n", cnt);
        }
    }

    return 0;
}
```

Segment Tree (discrete)

```
/*
 *  SRC: POJ 2482
 * PROB: Stars in Your Window
 * ALGO: Segment Tree
 * DATE: Sep 07, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

const double eps = 1e-8;
const int MAXN = 10010;
const int INF = 0x3FFFFFFF;

inline bool eq(double a, double b) { return fabs(a - b) < eps; }
inline bool ls(double a, double b) { return a + eps < b; }
inline int LC(int x) { return x << 1; }
inline int RC(int x) { return (x << 1) | 1; }

class Segment_Tree {
    private:
        struct Tnode {
            int a, b;
            int v, mx;
        };
        Tnode node[MAXN * 20];

    public:
        void build(int a, int b, int idx = 1)
        {
            node[idx].a = a;
            node[idx].b = b;
            node[idx].v = 0;
            node[idx].mx = -INF;

            if (a + 1 < b) {
                build(a, (a + b) >> 1, LC(idx));
                build((a + b) >> 1, b, RC(idx));
            }
        }

        void insert(int c, int d, int v, int idx = 1)
        {
            if (c <= node[idx].a && node[idx].b <= d) {
                node[idx].v += v;
                node[idx].mx = max(node[LC(idx)].mx, node[RC(idx)].mx) \
                               + node[idx].v;
                return ;
            }

            if (c < (node[idx].a + node[idx].b) >> 1) insert(c, d, v, LC(idx));
            if (d > (node[idx].a + node[idx].b) >> 1) insert(c, d, v, RC(idx));
            node[idx].mx = max(node[LC(idx)].mx, node[RC(idx)].mx) \
                           + node[idx].v;
        }

        int query()
        {
            return node[1].mx;
        }
};

struct Point {
    int x, y, v;
};

struct Discrete {
    double y;
    int id, new_y;
};

inline bool cmp_y(const Discrete &a, const Discrete &b) { return ls(a.y, b.y); }
inline bool cmp_id(const Discrete &a, const Discrete &b) { return a.id < b.id; }

struct Segment {
    double x;
    int b, e, v;

    bool operator<(const Segment &other) const
    {
        if (eq(x, other.x)) { return v < other.v; }
        return x < other.x;
    }
};

Segment_Tree st;
Point p[MAXN];
Discrete dy[MAXN * 2];
Segment seg[MAXN * 2];

int main()
{
    int n, W, H;
    while (scanf("%d%d%d", &n, &W, &H) != EOF) {
        double dw = W / 2.0, dh = H / 2.0;
        for (int i = 0; i < n; i++) {
            scanf("%d%d%d", &p[i].x, &p[i].y, &p[i].v);
            dy[LC(i)].id = LC(i);
            dy[LC(i)].y = p[i].y - dh;
            dy[RC(i)].id = RC(i);
            dy[RC(i)].y = p[i].y + dh;
        }

        sort(dy, dy + n * 2, cmp_y);
        dy[0].new_y = 0;
        int cnt = 0;
        for (int i = 1; i < n * 2; i++) {
            if (!eq(dy[i].y, dy[i - 1].y)) cnt++;
            dy[i].new_y = cnt;
        }
        sort(dy, dy + n * 2, cmp_id);

        for (int i = 0; i < n; i++) {
            seg[LC(i)].x = p[i].x - dw;
            seg[LC(i)].b = dy[LC(i)].new_y;
            seg[LC(i)].e = dy[RC(i)].new_y;
            seg[LC(i)].v = p[i].v;

            seg[RC(i)].x = p[i].x + dw;
            seg[RC(i)].b = dy[LC(i)].new_y;
            seg[RC(i)].e = dy[RC(i)].new_y;
            seg[RC(i)].v = -p[i].v;
        }
        sort(seg, seg + n * 2);

        int ans = -INF;
        st.build(0, cnt + 1);
        for (int i = 0; i < n * 2; i++) {
            st.insert(seg[i].b, seg[i].e, seg[i].v);
            ans = max(ans, st.query());
        }

        printf("%d\n", ans);
    }

    return 0;
}
```

Segment Tree (perimeter)

```
/*
 *  SRC: POJ 1177
 * PROB: Picture
 * ALGO: Segment Tree
 * DATE: Jul 21, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
```

```
#include <cstdlib>
#include <algorithm>

using std::sort;

class Segment_Tree {
    private:
        struct Tnode {
            int a, b; // segment [a, b)
            int cover;
            int len;
            Tnode *lc;
            Tnode *rc;

            Tnode(int _a, int _b)
                : a(_a), b(_b), cover(0), len(0), lc(0), rc(0)
            { }
        };
        Tnode *root;

        void _insert(int c, int d, int delta, Tnode *p)
        {
            if (c <= p->a && p->b <= d) {
                p->cover += delta;
            } else {
                if (!p->lc) {
                    p->lc = new Tnode(p->a, (p->a + p->b) / 2);
                    p->rc = new Tnode((p->a + p->b) / 2, p->b);
                }
                if (c < (p->a + p->b) / 2) _insert(c, d, delta, p->lc);
                if (d > (p->a + p->b) / 2) _insert(c, d, delta, p->rc);
            }

            if (p->cover) p->len = p->b - p->a;
            else if (p->lc) p->len = p->lc->len + p->rc->len;
            else p->len = 0;
        }

    public:
        Segment_Tree(int l, int r)
        {
            root = new Tnode(l, r);
        }

        void insert(int c, int d, int delta)
        {
            _insert(c, d, delta, root);
        }

        int query()
        {
            return root->len;
        }

};

struct Line {
    int x, y1, y2;
    int delta;

    bool operator<(const Line& other) const
    {
        // if two edges are on the same line, first add, then delete;
        if (x == other.x) return delta > other.delta;
        return x < other.x;
    }
};

Line X[10010], Y[10010];

int main()
{
    int n;
    scanf("%d", &n);

    int cnt = 0;
    for (int i = 0; i < n; i++) {
        int x1, y1, x2, y2;
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        X[cnt].x = x1; X[cnt].y1 = y1; X[cnt].y2 = y2; X[cnt].delta = 1;
        Y[cnt].x = y1; Y[cnt].y1 = x1; Y[cnt].y2 = x2; Y[cnt++].delta = 1;

        X[cnt].x = x2; X[cnt].y1 = y1; X[cnt].y2 = y2; X[cnt].delta = -1;
        Y[cnt].x = y2; Y[cnt].y1 = x1; Y[cnt].y2 = x2; Y[cnt++].delta = -1;
    }

    sort(X, X + cnt);
    sort(Y, Y + cnt);

    int ans = 0;
    Segment_Tree st_x(-10000, 10001);
    for (int i = 0, curr = 0; i < cnt; i++) {
        st_x.insert(X[i].y1, X[i].y2, X[i].delta);
        ans += abs(curr - st_x.query());
        curr = st_x.query();
    }

    Segment_Tree st_y(-10000, 10001);
    for (int i = 0, curr = 0; i < cnt; i++) {
        st_y.insert(Y[i].y1, Y[i].y2, Y[i].delta);
        ans += abs(curr - st_y.query());
        curr = st_y.query();
    }

    printf("%d\n", ans);

    return 0;
}
```

Segment Tree (area)

```
/*
 *  SRC: POJ 1389
 * PROB: Area of Simple Polygons
 * ALGO: Segment Tree
 * DATE: Jul 21, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <algorithm>

using std::sort;

class Segment_Tree {
    private:
        struct Tnode {
            int a, b; // segment [a, b)
            int cover;
            int len;
            Tnode *lc;
            Tnode *rc;

            Tnode(int _a, int _b)
                : a(_a), b(_b), cover(0), len(0), lc(0), rc(0)
            { }
        };
        Tnode *root;

        void insert(int c, int d, int delta, Tnode *p)
        {
            if (c <= p->a && p->b <= d) {
                p->cover += delta;
            } else {
                if (!p->lc) {
                    p->lc = new Tnode(p->a, (p->a + p->b) / 2);
                    p->rc = new Tnode((p->a + p->b) / 2, p->b);
                }
                if (c < (p->a + p->b) / 2) insert(c, d, delta, p->lc);
                if (d > (p->a + p->b) / 2) insert(c, d, delta, p->rc);
            }

            if (p->cover) p->len = p->b - p->a;
            else if (p->lc) p->len = p->lc->len + p->rc->len;
            else p->len = 0;
        }

    public:
        Segment_Tree(int l, int r)
        {
            root = new Tnode(l, r);
        }

        void insert(int c, int d, int delta)
        {
            insert(c, d, delta, root);
        }

        int query()
        {
            return root->len;
        }
};

struct Line {
    int x, y1, y2;
    int delta;

    bool operator<(const Line& other) const
    {
        // if two edges are on the same line, first add, then delete;
        if (x == other.x) return delta > other.delta;
        return x < other.x;
    }
};

inline int fmax(int a, int b) { return a > b ? a : b; }

Line X[2012];
```

```
int main()
{
    int x1, y1, x2, y2;
    while (1 + 1 == 2) {
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        if (x1 == -1 && y2 == -1 && x2 == -1 && y2 == -1) break;

        int cnt = 0, maxY = 0;
        while (1 + 1 == 2) {
            if (x1 == -1 && y2 == -1 && x2 == -1 && y2 == -1) break;

            X[cnt].x = x1; X[cnt].y1 = y1; X[cnt].y2 = y2; X[cnt++].delta = 1;
            X[cnt].x = x2; X[cnt].y1 = y1; X[cnt].y2 = y2; X[cnt++].delta = -1;

            maxY = fmax(maxY, fmax(y1, y2));

            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        }

        sort(X, X + cnt);

        int ans = 0;
        Segment_Tree st(0, maxY + 1);
        for (int i = 0, preL = 0, preX = 0; i < cnt; i++) {
            st.insert(X[i].y1, X[i].y2, X[i].delta);
            ans += preL * (X[i].x - preX);
            preL = st.query();
            preX = X[i].x;
        }

        printf("%d\n", ans);
    }

    return 0;
}
```

Partition Tree

```
/*
 *  SRC: POJ 2104
 * PROB: K-th Number
 * ALGO: Partition Tree
 * DATE: Sep 08, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <algorithm>

using std::sort;

const int MAXN = 100010;

class Partition_Tree {
    private:
        struct Tnode {
            int l, r;
        };
        Tnode node[MAXN << 2];

        int lcnt[20][MAXN], seg[20][MAXN];

        void _build(int a, int b, int d, int idx)
        {
            node[idx].l = a;
            node[idx].r = b;

            int mid = (a + b - 1) >> 1;
            int lmid = mid - a + 1; // the number of numbers which equal sorted[mid]
            for (int i = a; i < b; i++) {
                if (seg[d][i] < sorted[mid]) lmid--;
            }

            int lpos = a, rpos = mid + 1;
            for (int i = a; i < b; i++) {
                lcnt[d][i] = (i == a ? 0 : lcnt[d][i - 1]);
                if (seg[d][i] < sorted[mid]) {
                    lcnt[d][i]++;
                    seg[d + 1][lpos++] = seg[d][i];
                } else if (seg[d][i] > sorted[mid]) {
                    seg[d + 1][rpos++] = seg[d][i];
                } else if (lmid > 0) {
                    lmid--;
                    lcnt[d][i]++;
                    seg[d + 1][lpos++] = seg[d][i];
                } else {
                    seg[d + 1][rpos++] = seg[d][i];
                }
            }

            if (a + 1 < b) {
                _build(a, mid + 1, d + 1, idx << 1);
                _build(mid + 1, b, d + 1, (idx << 1) + 1);
            }
        }

        int _query(int a, int b, int k, int d, int idx)
        {
            if (b - a == 1) return seg[d][a];

            int ld = (a == node[idx].l ? 0 : lcnt[d][a - 1]),
                lseg = lcnt[d][b - 1] - ld;
            if (lseg >= k) {
                return _query(node[idx].l + ld, node[idx].l + ld + lseg, k, d + 1, idx << 1);
            }

            int mid = (node[idx].l + node[idx].r - 1) >> 1,
                rd = a - node[idx].l - ld,
                rseg = b - a - lseg;
            return _query(mid + rd + 1, mid + rd + rseg + 1, k - lseg, d + 1, (idx << 1) + 1);
        }

    public:
        int sorted[MAXN];

        void build(int a, int b)
        {
            memcpy(seg[0], sorted, sizeof(int) * (b - a));
            sort(sorted, sorted + b - a);
            _build(a, b, 0, 1);
        }

        int query(int a, int b, int k)
        {
            return _query(a, b, k, 0, 1);
        }
};

Partition_Tree pt;

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) scanf("%d", &pt.sorted[i]);
    pt.build(0, n);

    for (int i = 0; i < m; i++) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", pt.query(l - 1, r, k));
    }

    return 0;
}
```

Splay

```
/*
 *  SRC: POJ 2761
 * PROB: Feed the dogs
 * ALGO: Splay
 * DATE: Sep 08, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <algorithm>

using std::swap;
using std::sort;

const int MAXN = 100010;

class Splay {
    private:
        const static int BUF_SIZE = MAXN;
        int tree_size;

        struct Tnode {
            int key;
            int size, cnt;
            Tnode *l;
            Tnode *r;
        };

        Tnode buf[BUF_SIZE];
        Tnode *root;
        Tnode *buf_tail;

        Tnode *get_max(Tnode *x) const
        {
            while (x->r) x = x->r;
            return x;
        }

        Tnode *get_min(Tnode *x) const
        {
            while (x->l) x = x->l;
            return x;
        }

        void update_size(Tnode *x)
        {
            x->size = x->cnt;
            if (x->l) x->size += x->l->size;
            if (x->r) x->size += x->r->size;
        }

        void update_larger_size(Tnode *x, const Tnode *end)
        {
            if (x == end) return ;
            update_larger_size(x->l, end);
            update_size(x);
        }

        void update_smaller_size(Tnode *x, const Tnode *end)
        {
            if (x == end) return ;
            update_smaller_size(x->r, end);
            update_size(x);
        }

        Tnode *left_rotate(Tnode *x)
        {
            Tnode *y = x->r;
            x->r = y->l;
            y->l = x;
            update_size(x);
            return y;
        }

        Tnode *right_rotate(Tnode *x)
        {
            Tnode *y = x->l;
            x->l = y->r;
            y->r = x;
            update_size(x);
            return y;
        }

        Tnode *splay(Tnode *x, int key)
        {
            if (!x) return x;

            Tnode *y = new Tnode;
            y->l = y->r = 0;
            Tnode *larger = y;
            Tnode *smaller = y;

            while (key != x->key) {
                if (key < x->key) {
                    if (x->l && key < x->l->key) x = right_rotate(x);
                    if (!x->l) break;
                    larger->l = x;
                    larger = x;
                    x = x->l;
                } else {  // key > x->key
                    if (x->r && key > x->r->key) x = left_rotate(x);
                    if (!x->r) break;
                    smaller->r = x;
                    smaller = x;
                    x = x->r;
                }
            }

            larger->l = x->r;
            smaller->r = x->l;
            update_larger_size(y->l, larger->l);
            update_smaller_size(y->r, smaller->r);

            x->l = y->r;
            x->r = y->l;
            update_size(x);

            delete y;

            return x;
        }

        Tnode *insert(Tnode *x, int key)
        {
            if (x) {
                x = splay(x, key);
                if (key == x->key) {
                    x->size++;
                    x->cnt++;
                    return x;
                }
            }
            tree_size++;
            Tnode *y = buf_tail++;
            y->key = key;
            y->size = y->cnt = 1;
            y->l = y->r = 0;

            if (!x) return y;

            if (key < x->key) {
                y->l = x->l;
                y->r = x;
                x->l = 0;
            } else {
                y->r = x->r;
                y->l = x;
                x->r = 0;
            }
            update_size(x);
            update_size(y);

            return y;
        }

        Tnode *erase(Tnode *x, int key)
        {
            if (!x) return 0;

            x = splay(x, key);
            if (key == x->key) {
                if (x->cnt > 1) {
                    x->size--;
                    x->cnt--;
                    return x;
                }

                tree_size--;
                Tnode *y;
                if (!x->l) y = x->r;
                else {
                    y = splay(x->l, key);
                    y->r = x->r;
                    update_size(y);
                }

                return y;
            }

            return x;
        }

    public:
        Splay() { reset(); }
        bool empty() { return !root; }
        void insert(int key) { root = insert(root, key); }
        void erase(int key)  { root = erase(root, key); }

        void reset()
        {
            tree_size = 0;
            memset(buf, 0, sizeof(buf));
            buf_tail = buf;
            root = 0;
        }

        /* find the kth smallest one */
        int query(int k) const
        {
            Tnode *x = root;
            while (1 + 1 == 2) {
                if (x->l && x->l->size >= k) {
                    x = x->l;
                    continue;
                }
                if (x->l) k -= x->l->size;
                if (k <= x->cnt) return x->key;
                k -= x->cnt;
                x = x->r;
            }
        }
};

struct Query {
    int l, r, k;
    int id, ans;

    bool operator <(const Query &other) const
    {
        if (l == other.l) return r < other.r;
        return l < other.l;
    }
};
bool cmp_id(const Query &a, const Query &b) { return a.id < b.id; }

Splay splay;
int v[MAXN];
Query q[50010];
```

```
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", v + i);
    for (int i = 0; i < m; i++) {
        scanf("%d%d%d", &q[i].l, &q[i].r, &q[i].k);
        q[i].id = i;
    }
    sort(q, q + m);

    for (int i = q[0].l; i <= q[0].r; i++)
        splay.insert(v[i]);
    q[0].ans = splay.query(q[0].k);

    for (int i = 1; i < m; i++) {
        if (q[i - 1].r < q[i].l) {
            for (int j = q[i - 1].l; j <= q[i - 1].r; j++)
                splay.erase(v[j]);
            for (int j = q[i].l; j <= q[i].r; j++)
                splay.insert(v[j]);
        } else {
            for (int j = q[i - 1].l; j < q[i].l; j++)
                splay.erase(v[j]);
            if (q[i - 1].r < q[i].r) {
                for (int j = q[i - 1].r + 1; j <= q[i].r; j++)
                    splay.insert(v[j]);
            } else {
                for (int j = q[i].r + 1; j <= q[i - 1].r; j++)
                    splay.erase(v[j]);
            }
        }
        q[i].ans = splay.query(q[i].k);
    }

    sort(q, q + m, cmp_id);
    for (int i = 0; i < m; i++)
        printf("%d\n", q[i].ans);

    return 0;
}
```

Treap

```
/*
 *  SRC: POJ 2985
 * PROB: The k-th Largest Group
 * ALGO: Treap
 * DATE: Sep 05, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <algorithm>

using std::swap;

const int MAXN = 200010;

class Treap {
    private:
        const static int BUF_SIZE = MAXN;
        int tree_size;

        struct Tnode {
            int key, fix;
            int size, cnt;
            Tnode *l;
            Tnode *r;
        };

        Tnode buf[BUF_SIZE];
        Tnode *root;
        Tnode *buf_tail;

        Tnode *get_max(Tnode *x) const
        {
            while (x->r) x = x->r;
            return x;
        }

        Tnode *get_min(Tnode *x) const
        {
            while (x->l) x = x->l;
            return x;
        }

        void update_size(Tnode *x)
        {
            x->size = x->cnt;
            if (x->l) x->size += x->l->size;
            if (x->r) x->size += x->r->size;
        }

        Tnode *left_rotate(Tnode *x)
        {
            Tnode *y = x->r;
            x->r = y->l;
            y->l = x;

            update_size(x);
            update_size(y);

            return y;
        }

        Tnode *right_rotate(Tnode *x)
        {
            Tnode *y = x->l;
            x->l = y->r;
            y->r = x;

            update_size(x);
            update_size(y);

            return y;
        }

        Tnode *insert(Tnode *x, int key)
        {
            if (!x) {
                tree_size++;
                x = buf_tail++;
                x->key = key;
                x->fix = rand() * rand();
                x->size = x->cnt = 1;
                x->l = x->r = 0;

                return x;
            }

            if (key < x->key) {
                x->l = insert(x->l, key);
                if (x->l->fix > x->fix) x = right_rotate(x);
            } else if (key > x->key) {
                x->r = insert(x->r, key);
                if (x->r->fix > x->fix) x = left_rotate(x);
            } else {
                x->cnt++;
            }

            update_size(x);

            return x;
        }

        Tnode *erase(Tnode *x, int key)
        {
            if (!x) return 0;

            if (key < x->key) x->l = erase(x->l, key);
            else if (key > x->key) x->r = erase(x->r, key);
            else {
                if (x->cnt > 1) {
                    x->cnt--;
                    x->size--;
                    return x;
                }

                if (!x->l && !x->r) {
                    tree_size--;
                    return 0;
                }

                if (!x->l) x = left_rotate(x);
                else if (!x->r) x = right_rotate(x);
                else if (x->l->fix < x->r->fix) x = right_rotate(x);
                else x = left_rotate(x);
                x = erase(x, key);
            }

            update_size(x);

            return x;
        }

    public:
        Treap() { reset(); }
        bool empty() { return !root; }
        void insert(int key) { root = insert(root, key); }
        void erase(int key) { root = erase(root, key); }

        void reset()
        {
            tree_size = 0;
            memset(buf, 0, sizeof(buf));
            buf_tail = buf;
            root = 0;
        }

        /* find the kth largest one */
        int query(int k)
        {
            Tnode *x = root;
            while (1 + 1 == 2) {
                if (x->r && x->r->size >= k) {
                    x = x->r;
                    continue;
                }
                if (x->r) k -= x->r->size;
                if (k <= x->cnt) return x->key;
                k -= x->cnt;
                x = x->l;
            }
        }
};

class Disjoint_Set {
    private:

    public:
        int a[MAXN];

        Disjoint_Set() { reset(); }
        void reset() { memset(a, 0xff, sizeof(a)); }

        int find(int u)
        {
            if (a[u] < 0) return u;
            return a[u] = find(a[u]);
        }

        void join(int u, int v)
        {
            int x = find(u),
                y = find(v);
            if (x != y) {
                a[x] += a[y];
                a[y] = x;
            }
        }
};

Treap treap;
Disjoint_Set ds;

void combine(int x, int y)
{
    x = ds.find(x), y = ds.find(y);
    if (x != y) {
        treap.erase(-ds.a[x]);
        treap.erase(-ds.a[y]);
        ds.join(x, y);
        treap.insert(-ds.a[x]);
    }
}

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        treap.insert(1);
    }

    int c, i, j, k;
    while (m--) {
        scanf("%d", &c);
        if (!c) {
            scanf("%d%d", &i, &j);
            combine(i, j);
        } else {
            scanf("%d", &k);
            printf("%d\n", treap.query(k));
        }
    }

    return 0;
}
```

Other

Disjoint Set

```
class Disjoint_Set {
    public:
        // a[i] > 0: a[i] is i's ancestor;
        // a[i] < 0: -a[i] is the number of elements sharing the same ancestor i
        int a[MAXN];

        Disjoint_Set() { reset(); }
        void reset() { memset(a, 0xff, sizeof(a)); }

        int find(int u)
        {
            int x = u, y = u;
            while (a[u] >= 0) u = a[u];
            while (a[y] >= 0) {
                x = a[y];
                a[y] = u;
                y = x;
            }
            return u;
        }

        void join(int u, int v)
        {
            int x = find(u),
                y = find(v);
            if (x != y) {
                a[x] += a[y];
                a[y] = x;
            }
        }
};
```

Heap

```
void adjust_heap(int* first, int* last, int curr)
{
    int size = last - first;
    while (curr < (size >> 1)) {
        int child = (curr << 1) + 1;
        if (child + 1 < size && *(first + child) < *(first + child + 1))
            child++;
        if (*(first + curr) < *(first + child))
            swap(*(first + curr), *(first + child));
        else break;

        curr = child;
    }
}

void push_heap(int* first, int* last)
{
    int curr = last - 1 - first;
    while (curr) {
        int parent = (curr - 1) >> 1;
        if (*(first + parent) < *(first + curr))
            swap(*(first + curr), *(first + parent));
        else break;

        curr = parent;
    }
}

void pop_heap(int* first, int* last)
{
    swap(*first, *(last - 1));
    adjust_heap(first, last - 1, 0);
}

void make_heap(int* first, int* last)
{
    int* curr = first + 1;
    while (curr != last) push_heap(first, curr++);
}

void sort_heap(int* first, int* last)
{
    while (first != last) pop_heap(first, last--);
}
```

Merge Sort

```
void merge_sort(int *a, int l, int r) // [l, r)
{
    if (l + 1 == r) return ;

    int mid = (l + r) / 2;
    merge_sort(a, l, mid);
    merge_sort(a, mid, r);

    int *b = new int[r - l];
    int p = l, q = mid, cnt = 0;
    while (p < mid && q < r) {
        if (a[p] < a[q]) b[cnt++] = a[p++];
        else b[cnt++] = a[q++];
    }
    while (p < mid) b[cnt++] = a[p++];
    while (q < r)   b[cnt++] = a[q++];

    for (int i = 0; i < cnt; i++) a[l + i] = b[i];

    delete[] b;
}
```

Inversion Pair

```
/*
 *  SRC: NKOJ p1121
 * PROB: Ultra-QuickSort
 * ALGO: Merge Sort
 * DATE: Jun 7, 2011
 * COMP: jdk 6
 *
 * Created by Leewings Ac
 */

import java.util.*;

class Main {
    public static void main(String[] args)
    {
        new Prob().solve();
    }
}

class Prob {
    private long ans = 0;

    public void mergeSort(int a[], int l, int r) // [l, r)
    {
        if (l + 1 == r) return ;

        int mid = (l + r) / 2;
        mergeSort(a, l, mid);
        mergeSort(a, mid, r);

        int b[] = new int[r - l];
        int p = l, q = mid, cnt = 0;
        while (p < mid && q < r) {
            if (a[p] < a[q]) b[cnt++] = a[p++];
            else {
                b[cnt++] = a[q++];
                ans += mid - p;
            }
        }
        while (p < mid) b[cnt++] = a[p++];
        while (q < r)   b[cnt++] = a[q++];

        for (int i = 0; i < cnt; i++) a[l + i] = b[i];
    }

    public void solve()
    {
        Scanner cin = new Scanner(System.in);

        while (true) {
            int n = cin.nextInt();
            if (n == 0) break;

            int a[] = new int[n];
            for (int i = 0; i < n; i++) a[i] = cin.nextInt();

            ans = 0;
            mergeSort(a, 0, n);

            System.out.println(ans);
        }
    }
}
```

Monotone Priority Queue

```
/*
 *  SRC: POJ 1821
 * PROB: Fence
 * ALGO: DP + Monotone Priority Queue(decrease)
 * DATE: Jul 21, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <algorithm>

using namespace std;

int n, m;
int f[101][16001], que[16001];

struct Node {
    int l, p, s;

    bool operator<(const Node& other) const
    {
        return s < other.s;
    }
} worker[101];

inline int key(int i, int k)
{
    return f[i - 1][k] - k * worker[i].p;
}

/*
 * f[i][j] = {
 *              f[i - 1][j],
 *              f[i][j - 1],
 *              f[i - 1][k] + (j - k) * worker[i].p => (f[i - 1][k] - k * worker[i].p) + j * worker[i].p
 *           }
 */

int main()
{
    scanf("%d%d", &n, &m);

    for (int i = 1; i <= m; i++)
        scanf("%d%d%d", &worker[i].l, &worker[i].p, &worker[i].s);

    sort(worker + 1, worker + m + 1);

    for (int i = 1; i <= m; i++) {
        for (int j = 0; j < worker[i].s; j++) f[i][j] = f[i - 1][j];

        int head = 0, tail = 0;
        for (int k = max(worker[i].s - worker[i].l, 0); k < worker[i].s; k++) {
            while (head < tail && key(i, que[tail - 1]) <= key(i, k)) tail--;
            que[tail++] = k;
        }

        for (int j = worker[i].s, finish = min(worker[i].s + worker[i].l, n + 1); j < finish; j++) {
            while (head < tail && que[head] < j - worker[i].l) head++;
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            f[i][j] = max(f[i][j], key(i, que[head]) + j * worker[i].p);
        }

        for (int j = worker[i].s + worker[i].l; j <= n; j++)
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
    }

    printf("%d\n", f[m][n]);

    return 0;
}
```

Multiple Pack

```
/*
 *  SRC: POJ 2581
 * PROB: Exact Change Only
 * ALGO: DP(Multiple Pack)
 * DATE: Jul 28, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>

const double eps = 1e-12;

int f[510];

void zero_one_pack(int value, int max_value, int task)
```

```
{
    for (int i = max_value; i >= value; i--)
 if (f[i - value] == task) f[i] = task;
}

void complete_pack(int value, int max_value, int task)
{
    for (int i = value; i <= max_value; i++)
 if (f[i - value] == task) f[i] = task;
}

void multiple_pack(int value, int amount, int max_value, int task)
{
    if (value * amount >= max_value)
 complete_pack(value, max_value, task);
    else {
 for (int k = 1; k < amount; amount -= k, k <<= 1)
     zero_one_pack(k * value, max_value, task);

 zero_one_pack(value * amount, max_value, task);
    }
}

int main()
{
    double ta;
    int a, b, c, d, e;
    for (int task = 1; scanf("%lf%d%d%d%d", &ta, &b, &c, &d, &e) == 5; task++) {
        a = (ta + eps) * 100;
        f[0] = task;
        multiple_pack(25, b, a, task);
        multiple_pack(10, c, a, task);
        multiple_pack( 5, d, a, task);
        multiple_pack( 1, e, a, task);
        if (f[a] != task) puts("NO EXACT CHANGE");
        else {
            int B = 0, C = 0, D = 0, E = 0, now = a;
            while (now && B < b && f[now - 25] == task) { B++; now -= 25; }
            while (now && C < c && f[now - 10] == task) { C++; now -= 10; }
            while (now && D < d && f[now -  5] == task) { D++; now -=  5; }
            while (now && E < e && f[now -  1] == task) { E++; now -=  1; }
            printf("%d %d %d %d\n", B, C, D, E);
        }
    }

    return 0;
}
```

RMQ ST

```
/*
 *  SRC: POJ 3264
 * PROB: Balanced Lineup
 * ALGO: RMQ_ST
 * DATE: Jul 22, 2011
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>

using std::max;
using std::min;

const int MAXN = 50010;
int height[MAXN];
int mx[20][MAXN], mn[20][MAXN];

void RMQ_init(int *a, int n)
{
    memcpy(mx[0], a, sizeof(int) * n);
    memcpy(mn[0], a, sizeof(int) * n);
    for (int i = 1; 1 << i <= n; i++) {
        for (int j = 0; j + (1 << i) <= n; j++) {
            mx[i][j] = max(mx[i - 1][j], mx[i - 1][j + (1 << (i - 1))]);
            mn[i][j] = min(mn[i - 1][j], mn[i - 1][j + (1 << (i - 1))]);
        }
    }
}

int RMQ_max(int l, int r) // [l, r)
{
    int idx = log2(r - l);
    return max(mx[idx][l], mx[idx][r - (1 << idx)]);
}

int RMQ_min(int l, int r) // [l, r)
{
    int idx = log2(r - l);
    return min(mn[idx][l], mn[idx][r - (1 << idx)]);
}

int main()
{
    int n, q;
    scanf("%d%d", &n, &q);
    for (int i = 0; i < n; i++) {
        scanf("%d", height + i);
    }

    RMQ_init(height, n);

    for (int i = 0; i < q; i++) {
        int l, r;
        scanf("%d%d", &l, &r); // [l, r], starting from 1
        printf("%d\n", RMQ_max(l - 1, r) - RMQ_min(l - 1, r));
    }

    return 0;
}
```

RMQ ST for Lowest Common Ancestor

```
/*
 *  SRC: POJ 1986
 * PROB: Distance Queries
 * ALGO: RMQ-ST-LCA (Lowest Common Ancestor)
 * DATE: May 28, 2012
 * COMP: g++
 *
 * Created by Leewings Ac
 */

#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
#include <algorithm>

using std::vector;
using std::min;
using std::swap;

const int MAXN = 40000;

struct Edge {
    int v, d;
    Edge(int _v, int _d) : v(_v), d(_d) { }
};

typedef vector<Edge>::const_iterator vci;
vector<Edge> edge[MAXN];
int idx, cnt;
int parent[MAXN], dist[MAXN], label[MAXN], rev_label[MAXN], pos[MAXN], seq[MAXN << 1];
int mn[20][MAXN << 1];

void dfs(int u, int length)
{
    dist[u] = length;
    label[u] = cnt;
    rev_label[cnt++] = u;
    pos[label[u]] = idx;
    seq[idx++] = label[u];

    for (vci e = edge[u].begin(); e != edge[u].end(); e++) {
        if (e->v != parent[u]) {
            parent[e->v] = u;
            dfs(e->v, length + e->d);
            seq[idx++] = label[u];
        }
    }
}

void RMQ_init(int *a, int n)
{
    memcpy(mn, a, sizeof(int) * n);
    for (int i = 1; 1 << i <= n; i++) {
        for (int j = 0; j + (1 << i) <= n; j++) {
            mn[i][j] = min(mn[i - 1][j], mn[i - 1][j + (1 << (i - 1))]);
        }
    }
}

int RMQ_min(int l, int r) // [l, r)
{
    int idx = log2(r - l);
    return min(mn[idx][l], mn[idx][r - (1 << idx)]);
}

int RMQ_lca(int u, int v)
{
    u = label[u];
    v = label[v];
    if (pos[u] > pos[v]) swap(u, v);
    return rev_label[RMQ_min(pos[u], pos[v] + 1)];
}

int main()
{
    int n;
    scanf("%d%*d", &n);
    for (int i = 1; i < n; i++) {
        int u, v, d;
        scanf("%d%d%d%*s", &u, &v, &d);
        edge[u - 1].push_back(Edge(v - 1, d));
        edge[v - 1].push_back(Edge(u - 1, d));
    }

    idx = cnt = 0;
    parent[0] = 0;
    dfs(0, 0);
    RMQ_init(seq, idx);

    int k;
    scanf("%d", &k);
    for (int i = 0; i < k; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        u--; v--;
        printf("%d\n", dist[u] + dist[v] - dist[RMQ_lca(u, v)] * 2);
    }

    return 0;
}
```