Contents

1 Setting

1.1 /.vimrc

```
syntax on
color torte
set nu ts=4 sw=4 ai mouse=a bs=2 ci hls  ru nocp
    showmatch ar fencs=utf-8
set guifont=Consolas:h10
filetype plugin indent on
so $VIMRUNTIME/mswin.vim
behave mswin

autocmd CursorMoved  * exe printf('match VisualNOS
    /\V\<%s\>/', escape(expand('<cword>'), '/\'))
autocmd CursorMovedi * exe printf('match VisualNOS
    /\V\<%s\>/', escape(expand('<cword>'), '/\'))

map <F5> :r ~/sample.cpp<CR>
map <F9> :call Compile()<CR>
map! <F9> <ESC>:call Compile()<CR>
map <F10> :call Run()<CR>
map! <F10> <ESC>:call Run()<CR>

func! Compile()
    exec "w"
    exec "!g++ -Wall -Wshadow -std=gnu++0x % -o %<
        2>log.txt"
    exe "cg log.txt"
    cw 5
endfunc

func! Run()
    exec "!./%<"  # "!%<" if windows
endfunc

cd ~/Desktop # C:\Users\???\Desktop
```

1.2 /cp.sh

```
#!/bin/bash
clear
g++ $1.cpp -DDBG -o $1
if [[ "$?" == "0" ]]; then
  echo Running
  ./$1 < $1.in > $1$2.out
  echo END
fi
```

1.3 /new.sh

```
#!/bin/bash

clear
cat template.cpp > $1.cpp
touch $1.in $1.out
echo $1 Created
```

2 General

2.1 Template

```
#pragma GCC optimize("O2")
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
using ULL = unsigned long long;
using PII = pair<int,int>;
```

```
using PLL = pair<LL, LL>;
using VI = vector<int>;
using VVI = vector<vector<int>>;
using dvt = double;
const int INF = 1e9;
const int MXN = 0;
const int MXV = 0;
const double EPS = 1e-9;
const int MOD = 1e9+7;
#define MP make_pair
#define PB push_back
#define Fi first
#define Se second
#define FOR(i, L, R) for(int i = L; i < (int)R; ++i)
#define FORD(i, L, R) for(int i = L; i > (int)R; --i)
#define IOS cin.tie(nullptr); cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

int main()
{
    IOS;
}
```

2.2 /buglist

```
/*
cmp 不能 return true
變數宣告在迴圈費時，要小心使用
<<運算小心溢位，good way: (1LL << x)
prime_table小心i,j溢位
 */
```

2.3 Builtin

```
From 日月掛長
unsigned int / unsigned long +l / unsigned long long
    +ll
int __builtin_ffs: 返回右起第一個1的位置
int __builtin_clz: 返回左起第一個1之前0的個數
int __builtin_ctz: 返回右起第一個1之後的0的個數
int __builtin_popcount: 返回1的個數
int __builtin_parity: 返回1的個數的奇偶性(1的個數 mod
    2的值)
```

2.4 BinarySearch

```
lower_bound(a, a + n, k);      //最左邊 ≥ k 的位置
upper_bound(a, a + n, k);      //最左邊 > k 的位置
upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
[lower_bound, upper_bound) //等於 k 的範圍
equal_range(a, a+n, k);
```

2.5 int128

```
istream &operator>>(istream &in, __int128 &x)
{
    char buf[30];
    in >> buf;
    bool minus = false;
    int len = strlen(buf);
    x = 0;
    for (int i = 0; i < len; i++)
    {
        if (i == 0 && buf[i] == '-')
        {
            minus = true;
        }
        else
        {
            x = x * 10 + buf[i] - 48;
        }
    }
    if (minus)
    {
        x *= -1;
    }
    return in;
}

ostream &operator<<(ostream &out, __int128 &x)
{
    vector<int> v;
    __int128 tmp = x;
    bool minus = tmp < 0;
    if (minus)
        tmp *= -1;

    while (tmp > 0)
    {
        v.push_back(tmp % 10);
        tmp /= 10;
    }
    if (minus)
    {
        out << "-";
    }
    for (int i = (int)v.size() - 1; i >= 0; i--)
    {
        out << v[i];
    }
    return out;
}
```

2.6 StableMatching

```
int t, n, b[N][N], bi[N], g[N][N], bg[N], gb[N];

void sol()
{
    deque<int> dq;
    memset(gb, 0, sizeof(gb));
    memset(bi, 0, sizeof(bi));
    for (int i = 1; i <= n; i++)
        dq.push_back(i);
    while (!dq.empty())
    {
        int x = dq.front();
        dq.pop_front();
        int y = b[x][++bi[x]];
        if (!gb[y])
        {
            gb[y] = x;
            bg[x] = y;
        }
        else if (g[y][x] < g[y][gb[y]])
        {
            dq.push_back(gb[y]);
            gb[y] = x;
            bg[x] = y;
        }
        else
        {
            dq.push_back(x);
        }
    }
    for (int i = 1; i <= n; i++)
    {
        cout << bg[i] << '\n';
    }
}

int main()
{
```

```
    int x;
    cin >> t;
    for (int i = 0; i < t; i++)
    {
        cin >> n;
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                cin >> b[i][j];
            }
        }
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                cin >> x;
                g[i][x] = j;
            }
        }
        if (i)
            cout << '\n';
        sol();
    }
}
```

2.7 Mergesort

```
long long sol(int L, int R)
{
    if (R - L <= 1)
        return 0;
    int M = (R + L) / 2;
    long long ans = sol(L, M) + sol(M, R);
    int i = L, j = M, k = L;
    while (i < M || j < R)
    {
        if (i >= M)
            buf[k] = arr[j++];
        else if (j >= R)
            buf[k] = arr[i++];
        else
        {
            if (arr[i] <= arr[j])
                buf[k] = arr[i++];
            else
            {
                buf[k] = arr[j++];
                ans += M - i;
            }
        }
        k++;
    }
    for (int k = L; k < R; k++)
        arr[k] = buf[k];
    return ans;
}
```

2.8 Multi

```
multiset<int>
equal_range (T1 a)：回傳 iterator 的 pair<lower_bound (a),upper_bound (a)> ，為 a 所在範圍
erase (T1 a)：刪除所有元素 a，如果只要刪除一個，用 s.erase (s.find (a))
```

2.9 ThreeSearch

```
#include <bits/stdc++.h>
using namespace std;
#define N 20
int t, n, i, j;
struct happy
{
    double a, b, c;
} h[N];
double f2(double x, double a, double b, double c)
{
    return a * (x - b) * (x - b) + c;
}
double f(double x)
{
    double ans = 0;
    for (int i = 0; i < n; i++)
    {
        ans = max(ans, f2(x, h[i].a, h[i].b, h[i].c));
        //    cout<<ans<<'\n';
    }
    return ans;
}
int main()
{
    cin.tie(NULL);
    for (cin >> t; i < t; i++)
    {
        for (cin >> n, j = 0; j < n; j++)
        {
            cin >> h[j].a >> h[j].b >> h[j].c;
        }
        double L = 0, R = 300, M, MM;
        while (R - L > 1e-9)
        {
            M = L + (R - L) / 3;
            MM = (M + R) / 2;
            if (f(M) > f(MM))
            {
                L = M;
            }
            else
            {
                R = MM;
            }
        }
        cout << fixed << setprecision(5) << f(L) <<
            '\n';
    }
}
```

2.10 Tree Policy

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp> // Common
    file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> set_t;
set_t t;
int main() {
    t.insert(5);
    t.insert(6);
    t.insert(3);
    t.insert(1);
    // the smallest is (0), bigest is (n-1), kth
        small is (k-1)
    int num = *t.find_by_order(0);
    printf("%d\n", num); // print 1
    num = *t.find_by_order(t.size()-1);
    printf("%d\n", num); // print 6
    // find the index
    int index = t.order_of_key(6);
    printf("%d\n", index); // print 3
    // cheak if there exist x
    int x = 5;
    int check = t.erase(x);
```

```
    if(check == 0) printf("t not contain 5\n");
    else if(check == 1) printf("t conain 5\n");
    //tree policy like set
    t.insert(5); t.insert(5);
    // get the size of t
    printf("%d\n", t.size()); // print 4
}
```

3 Data and Structure

3.1 Mo

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100005;
int a[N];
int curmax;
int app[N], cnt[N];

struct Query
{
    int L, R, qid, bid;
    bool operator<(const Query &rhs) const
    {
        if (bid != rhs.bid)
            return bid < rhs.bid;
        return R < rhs.R;
    }
} q[N];

bool cmp(Query a, Query b) { return a.L < b.L; }

void add(int x)
{
    int now = ++app[x];
    cnt[now - 1]--;
    cnt[now]++;
    curmax = max(curmax, now);
}

void sub(int x)
{
    int now = --app[x];
    cnt[now + 1]--;
    cnt[now]++;
    if (!cnt[curmax])
        curmax--;
}

int main()
{
    int n, Q;
    int ans[N];
    cin >> n >> Q;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
    }
    int k = floor(sqrt(n / 1.0));
    for (int i = 0; i < Q; i++)
    {
        cin >> q[i].L >> q[i].R;
        q[i].qid = i;
    }
    sort(q, q + Q, cmp);
    for (int i = 0; i < Q; i++)
    {
        q[i].bid = i / k;
    }
    sort(q, q + Q);
    for (int i = 0, curL = 1, curR = 0; i < Q; i++)
    {
        while (curR < q[i].R)
        {
            curR++;
            add(a[curR]);
        }
        while (q[i].R < curR)
        {
            sub(a[curR]);
            curR--;
        }
        while (curL < q[i].L)
        {
            sub(a[curL]);
            curL++;
        }
        while (q[i].L < curL)
        {
            curL--;
            add(a[curL]);
        }
        ans[q[i].qid] = curmax;
    }
    for (int i = 0; i < Q; i++)
    {
        cout << ans[i] << '\n';
    }
}
```

3.2 Segment Tree

```
int bulit(int L, int R, int x)
{
    if (L == R)
        return heap[x - 1] = arr[L];
    int M = (L + R) >> 1;
    return heap[x - 1] = bulit(L, M, (x << 1)) +
        bulit(M + 1, R, (x << 1) + 1);
}
void modify(int L, int R, int x, int a, int b, int mo)
{
    if (b < L || R < a)
        return;
    if (L == R)
    {
        heap[x - 1] += mo;
        return;
    }
    int M = (L + R) >> 1;
    modify(L, M, (x << 1), a, b, mo);
    modify(M + 1, R, (x << 1) + 1, a, b, mo);
    heap[x - 1] += mo;
    return;
}
int quest(int L, int R, int x, int a, int b)
{
    if (b < L || R < a)
        return 0;
    if (a <= L && R <= b)
        return heap[x - 1];
    int M = (L + R) >> 1;
    return quest(L, M, (x << 1), a, b) + quest(M + 1,
        R, (x << 1) + 1, a, b);
}
```

3.3 Treap

```
struct Treap{
    int val, pri, sz;
    Treap *lc, *rc;
    Treap(){}
    Treap(int _val)
    {
        val = _val;
        pri = rand();
        sz = 1;
```

```
        lc = rc = NULL;
    }
};

int getSize(Treap *a){
    return (a == NULL ? 0 : a->sz);
}

void split(Treap *t, Treap *&a, Treap *&b, int k)
{
    if(t == NULL)
    {
        a = b = NULL;
        return;
    }
    if(getSize(t->lc) < k)
    {
        a = t;
        split(t->rc, a->rc, b, k - getSize(t->lc) -
            1);
    }
    else
    {
        b = t;
        split(t->lc, a, b->lc, k);
    }
}

Treap* merge(Treap *a, Treap *b)
{
    if(!a || !b)
    {
        return (a ? a : b);
    }
    if(a->pri > b->pri)
    {
        a->rc = merge(a->rc, b);
        return a;
    }
    else
    {
        b->lc = merge(a, b->lc);
        return b;
    }
}

void Insert(Treap *&t, int x, int p)
{
    Treap *a, *b;
    split(t, a, b, x);
    t = merge(a, merge(new Treap(p), b));
}

void Delete(Treap *&t, int x)
{
    Treap *a, *b, *c;
    split(t, b, c, x);
    split(b, a, b, x - 1);
    t = merge(a, c);
}

/*
Usage
Treap *root = NULL; // declare
root = merge(root, new Treap(val)); // push back
Insert(root, x, y); // insert y after x-th element
Delete(root, x); // delete x-th element
*/
```

4 DP

4.1 Backpack Limit

```
struct State{
    LL w, val;
};
struct Data{
    LL v,w,m;
};

int main() {
    LL n, W;
    cin >> n >> W;
    vector<Data> d(n);
    vector<LL> dp(W + 5, -INF);
    dp[0] = 0;
    for(LL i = 0; i < n; ++i)
    {
        cin >> d[i].v >> d[i].w >> d[i].m;
    }
    deque<State> dq[MXW];
    for(int i = 0; i < n; ++i)
    {
        LL v = d[i].v, w = d[i].w, m = d[i].m;
        for(int j = 0; j <= W; ++j)
        {
            if(j < w)
            {
                dq[j].clear();
                dq[j].pb({j,dp[j]});
                continue;
            }
            int id = j % w;
            while(dq[id].front().w + m * w <
                j)dq[id].pop_front();
            LL tmp = dq[id].front().val + (j -
                dq[id].front().w) / w * v;
            while(!dq[id].empty() &&
                dq[id].back().val + (j -
                dq[id].back().w) / w * v <= dp[j])
                dq[id].pop_back();
            dq[id].push_back({j,dp[j]});
            dp[j] = max(dp[j], tmp);
        }
    }
    LL ans = -INF;
    for(int i = 0; i <= W; i++){
        ans = max(ans, dp[i]);
    }
    cout << ans << '\n';
}
```

4.2 CounterLine

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1 << 15;
int n, m, cur;
long long int dp[2][N];

void update(int a, int b)
{
    if (b & (1 << m))
    {
        dp[cur][b ^ (1 << m)] += dp[1 - cur][a];
    }
}

int main()
{
    while (cin >> n >> m)
    {
        if ((n * m) & 1)
        {
            cout << "0\n";
            continue;
        }
        if (n == 1 || m == 1)
        {
            cout << "1\n";
```

```
            continue;
        }
        if (n < m)
            swap(n, m);
        memset(dp, 0, sizeof(dp));
        cur = 0;
        dp[0][(1 << m) - 1] = 1;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                cur ^= 1;
                memset(dp[cur], 0, sizeof(dp[cur]));
                for (int k = 0; k < (1 << m); k++)
                {
                    update(k, k << 1);
                    if (i && !(k & (1 << m - 1)))
                        update(k, (k << 1) ^ (1 << m)
                            ^ 1);
                    if (j && !(k & 1))
                        update(k, (k << 1) ^ 3);
                }
            }
        }
        cout << dp[cur][(1 << m) - 1] << '\n';
    }
}
```

4.3 LCS

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, m;
    vector<int> a, b, dp[2];
    cin >> n >> m;
    a.resize(n);
    b.resize(m);
    for (int i = 0; i < a.size(); i++)
    {
        cin >> a[i];
    }
    for (int i = 0; i < b.size(); i++)
    {
        cin >> b[i];
    }
    dp[0].resize(m + 1);
    dp[1].resize(m + 1);
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            if (a[i - 1] == b[j - 1])
                dp[i & 1][j] = dp[(i & 1) ^ 1][j - 1]
                    + 1;
            else
                dp[i & 1][j] = max(dp[i & 1][j - 1],
                    dp[(i & 1) ^ 1][j]);
        }
    }
    cout << dp[n & 1][m] << '\n';
}
```

4.4 LIS

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    while (cin >> n)
    {
        vector<int> v;
        for (int i = 0, x; i < n; i++)
        {
            cin >> x;
            if (!v.size() || x > v.back())
                v.push_back(x);
            else
                *lower_bound(v.begin(), v.end(), x) =
                    x;
        }
        cout << v.size() << '\n';
    }
}
```

4.5 ReRoot

```
LL dp[MXV], num[MXV], aa[MXV], sum = 0;
vector<LL> p[MXV];
bitset<MXV> vis;

void dfs(int s, LL depth)
{
    vis[s] = 1;
    num[s] = aa[s];
    dp[1] += depth * aa[s];
    for (int v : p[s])
    {
        if (!vis[v])
        {
            dfs(v, depth + 1);
            num[s] += num[v];
        }
    }
}

void solve(int s, int n)
{
    vis[s] = 1;
    for (int v : p[s])
    {
        if (!vis[v])
        {
            dp[v] = dp[s] + sum - num[v] * 2;
            solve(v, n);
        }
    }
}
```

4.6 TSP

```
void btb(int &x)
{
    x = 0;
    for (int i = 0, j = 1; i < n; i++, j *= 2)
        x += b[i] * j;
    return;
}
int main()
{
    memset(dp, 0, sizeof(dp));
    for (int i = 1, st; i <= n; i++)
    { // st:state
        for (int jj = 0; jj < n; jj++)
            b[n - jj - 1] = (jj < i);
        do
        {
            btb(st);
            for (int x = 0; x < n; x++)
            {
                if (!b[x])
                    continue;
```

```
                if (i == 1)
                    dp[x][st] = dis[x][0];
                for (int y = 0; y < n; y++)
                {
                    if (x != y && b[y] &&
                        (dp[x][st] == 0 ||
                         dp[x][st] > dp[y][st - (1 << x)] + dis[y][x]))
                    {
                        dp[x][st] = dp[y][st - (1 << x)] + dis[y][x];
                    }
                }
            }
        } while (next_permutation(b, b + n));
    }
    cout << dp[0][(1 << n) - 1] << '\n';
}
```

5 Geometry

5.1 Basic

```
struct dot
{
    dvt x, y;
};
struct Line
{
    dot st, ed;
};

dot operator+(dot a, dot b) { return {a.x + b.x, a.y + b.y}; }
dot operator-(dot a, dot b) { return {a.x - b.x, a.y - b.y}; }
dot operator*(dot a, dvt c) { return {a.x * c, a.y * c}; }
dot operator*(dvt c, dot a) { return a * c; }
dot operator/(dot a, dvt c) { return {a.x / c, a.y / c}; }
bool operator<(dot a, dot b) { return std::tie(a.x, a.y) < std::tie(b.x, b.y); }
bool operator==(dot a, dot b)
{
    return std::tie(a.x, a.y) == std::tie(b.x, b.y);
}
dvt iproduct(dot a, dot b) { return a.x * b.x + a.y * b.y; }
dvt cross(dot a, dot b) { return a.x * b.y - a.y * b.x; }
int dis(dot a, dot b)
{
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

int side(Line L, dot a)
{
    dvt cross_value = cross(a - L.st, L.ed - L.st);
    if (cross_value > EPS)
    {
        return 1;
    }
    else if (cross_value < -EPS)
    {
        return -1;
    }
    return 0;
}

bool has_jiao(Line AB, Line CD)
{
    int a = side(CD, AB.st);
    int b = side(CD, AB.ed);
    int c = side(AB, CD.st);
    int d = side(AB, CD.ed);
    if (a * b < 0 && c * d < 0)
    {
        return true;
    }
    if (a == 0 && iproduct(CD.st - AB.st, CD.ed - AB.st) <= 0)
    {
        return true;
    }
    if (b == 0 && iproduct(CD.st - AB.ed, CD.ed - AB.ed) <= 0)
    {
        return true;
    }
    if (c == 0 && iproduct(AB.st - CD.st, AB.ed - CD.st) <= 0)
    {
        return true;
    }
    if (d == 0 && iproduct(AB.st - CD.ed, AB.ed - CD.ed) <= 0)
    {
        return true;
    }
    return false;
}
```

5.2 Convex Hull

```
vector<dot> p, p1;

void convexhull()
{
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());
    p1.clear();
    p1.resize(p.size());
    int m = 0;
    FOR(i, 0, p.size())
    {
        while (m > 1 && cross(p1[m - 1] - p1[m - 2], p[i] - p1[m - 2]) <= 0)
            m--;
        p1[m++] = p[i];
    }
    int k = m;
    FORD(i, p.size() - 2, 0 - 1)
    {
        while (m > k && cross(p1[m - 1] - p1[m - 2], p[i] - p1[m - 2]) <= 0)
            m--;
        p1[m++] = p[i];
    }
    if (m > 1)
        m--;
    p1.resize(m);
}
```

6 Graph

6.1 Edge

```
struct Edge
{
    int from, to, w;
    bool operator<(const Edge& rhs) // optional
    {
        return w < rhs.w;
    }
};
```

6.2 Bloosom

```
int lca(int x, int y)
{
    MSET(vis, false);
    while (true)
    {
        x = base[x];
        vis[x] = true;
        if (match[x] ==-1)
            break;
        x = fr[match[x]];
    }
    while (true)
    {
        y = base[y];
        if (vis[y])
            return y;
        y = fr[match[y]];
    }
    return -1;
}
void set_path(int x, int fa)
{
    int y;
    while (x != fa)
    {
        y = match[x];
        blossom[base[x]] = true;
        blossom[base[y]] = true;
        y = fr[y];
        if (base[y] != fa)
            fr[y] = match[x];
        x = y;
    }
}
void flower(int x, int y)
{
    MSET(blossom, false);
    int fa = lca(x, y);
    set_path(x, fa);
    set_path(y, fa);
    if (base[x] != fa)
        fr[x] = y;
    if (base[y] != fa)
        fr[y] = x;
    REP(i, 1, n)
    if (blossom[base[i]])
    {
        base[i] = fa;
        if (!inq[i])
        {
            q.push(i);
            inq[i] = true;
        }
    }
}
bool bfs(int root)
{
    int cur, y, nxt;
    q = queue<int>();
    MSET(inq, false);
    MSET(fr,-1);
    REP(i, 1, n) base[i] = i;
    q.push(root);
    while (!q.empty())
    {
        cur = q.front();
        q.pop();
        inq[cur] = false;
        for (int i = first[cur]; ~i; i = in[i].next)
            if (base[cur] != base[in[i].t] &&
                match[cur] != in[i].t)
            {
                if (in[i].t == root ||
                    (~match[in[i].t] &&
                    ~fr[match[in[i].t]]))
                    flower(cur, in[i].t);
                else if (fr[in[i].t] == -1)
                {
                    fr[in[i].t] = cur;
                    if (match[in[i].t] == -1)
                    {
                        cur = in[i].t;
                        while (cur !=-1)
                        {
                            y = fr[cur];
                            nxt = match[y];
                            match[cur] = y;
                            match[y] = cur;
                            cur = nxt;
                        }
                        return true;
                    }
                    else
                    {
                        q.push(match[in[i].t]);
                        inq[match[in[i].t]] = true;
                    }
                }
            }
    }
    return false;
}
int do_match()
{
    int re = 0;
    MSET(match,-1);
    REP(i, 1, n) if (match[i] == -1 && bfs(i)) re++;
    return re;
}
```

6.3 CLE

```
struct Edge {
    int from;
    int to;
    int bdw;
    int cost;
};

int n, m, budget;
int in[MAXN], pre[MAXN], id[MAXN], vis[MAXN];
Edge edges[MAXN], tedges[MAXN];

int CLE(int root, int tn, int lowb) {
    copy(begin(edges), begin(edges) + m,
        begin(tedges));

    int res = 0;
    while (true) {
        for (int i = 0; i < tn; i++) {
            in[i] = INF;
        }

        //find in edge
        for (int i = 0; i < m; i++) {
            Edge e = tedges[i];
            if (e.from != e.to && e.bdw >= lowb &&
                e.cost < in[e.to]) {
                pre[e.to] = e.from;
                in[e.to] = e.cost;
            }
        }

        //check in edge
        for (int i = 0; i < tn; i++) {
            if (i == root) {
                continue;
            }
            if (in[i] == INF) {
                return -1;
            }
```

```
        }

        int nodenum = 0;
        memset(id, -1, sizeof(id));
        memset(vis, -1, sizeof(vis));
        in[root] = 0;

        //find cycles
        for (int i = 0; i < tn; i++) {
            res += in[i];
            int v = i;
            while (vis[v] != i && id[v] == -1 && v != root) {
                vis[v] = i;
                v = pre[v];
            }

            if (v != root && id[v] == -1) {
                for (int j = pre[v]; j != v; j = pre[j]) {
                    id[j] = nodenum;
                }
                id[v] = nodenum++;
            }
        }

        //no cycle
        if (nodenum == 0) {
            break;
        }

        for (int i = 0; i < tn; i++) {
            if (id[i] == -1) {
                id[i] = nodenum++;
            }
        }

        //grouping the vertices
        for (int i = 0; i < m; i++) {
            int from = tedges[i].from;
            int to = tedges[i].to;

            tedges[i].from = id[from];
            tedges[i].to = id[to];

            if (tedges[i].from != tedges[i].to) {
                tedges[i].cost -= in[to];
            }
        }

        tn = nodenum;
        root = id[root];
    }
    return res;
}
```

6.4 Disjoint Set

```
struct DisjointSet
{
    int p[MXV], sz[MXV];
    void init(int n)
    {
        for (int i = 0; i <= n; i++)
        {
            p[i] = i;
            sz[i] = 1;
        }
    }
    int find(int u) { return u == p[u] ? u : p[u] = find(p[u]); }
    void Union(int u, int v)
    {
        u = find(u);
        v = find(v);
        if(u == v)
        {
            return;
        }
        if(sz[u] < sz[v])
        {
            swap(u, v);
        }
        sz[u] += sz[v];
        p[v] = u;
    }
};

/*
Usage
DisjointSet djs; // declare
djs.init(int n); // initialize from vertex 0 to vertex n
djs.find(int u) // find the parent of vertex u
djs.Union(int u, int v) // union vertex u and v
*/
```

6.5 Longest Common Ancestor

```
const int LOG = 20;
vector<int> tin(MXV), tout(MXV), depth(MXV);
int par[MXV][LOG];
int timer = 0;
vector<int> G[MXV];

void dfs(int u, int f)
{
    tin[u] = ++timer;
    par[u][0] = f;
    for (int v : G[u])
    {
        if (v != f)
        {
            depth[v] = depth[u] + 1;
            dfs(v, u);
        }
    }
    tout[u] = ++timer;
}

void Doubling(int n)
{
    for (int j = 1; j < LOG; ++j)
    {
        for (int i = 1; i <= n; ++i)
        {
            par[i][j] = par[par[i][j - 1]][j - 1];
        }
    }
}

bool anc(int u, int v) { return tin[u] <= tin[v] && tout[v] <= tout[u]; }

int LCA(int u, int v)
{
    if (depth[u] > depth[v])
    {
        swap(u, v);
    }
    if (anc(u, v))
    {
        return u;
    }
    for (int j = LOG - 1; j >= 0; --j)
    {
        if (!anc(par[u][j], v))
            u = par[u][j];
    }
    return par[u][0];
}

```

```cpp
int dis(int u, int v)
{
    int lca = LCA(u, v);
    return depth[u] + depth[v] - 2 * depth[lca];
}

/*
dfs(root, root);
Doubling(n);
*/
```

6.6 MST

```cpp
int MST()
{
    DisjointSet djs;
    vector<Edge> edges;
    int n, m, ans = 0;
    cin >> n >> m;
    for (int i = 0, from, to, w; i < m; i++)
    {
        cin >> from >> to >> w;
        edges.push_back({from, to, w});
    }
    sort(edges.begin(), edges.end());
    djs.init(n);
    for (auto edge : edges)
    {
        // Union also check if to vertex haven't
            connected
        if (djs.Union(edge.from, edge.to))
        {
            ans += edge.w;
        }
    }
    return ans;
}
```

6.7 TopologicalSort

```cpp
#include <cstring>
#include <iostream>
#include <stack>
#include <vector>

#define S 50050

using namespace std;

vector<int> map[S];
stack<int> ans;
int state[S];
bool head[S];
bool valid;
int n, m;

void dfs(int cur)
{
    state[cur] = 1;

    for (auto next : map[cur])
        if (!state[next])
            dfs(next);
        else if (state[next] == 1)
        {
            valid = false;
            return;
        }

    state[cur] = 2;

    ans.push(cur);
}

void topology_sort()
{
    for (int i = 1; i <= n; i++)
        if (valid && head[i])
            dfs(i);

    if (!valid)
    {
        cout << -1 << endl;
        return;
    }

    while (!ans.empty())
    {
        cout << ans.top() << endl;
        ans.pop();
    }
}

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

    memset(head, true, sizeof(head));

    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;

        head[b] = false;

        map[a].push_back(b);
    }

    memset(state, 0, sizeof(state));
    valid = true;

    topology_sort();

    return 0;
}
```

6.8 TreeCentroid

```cpp
PII treeCentroid(int u, int f, int sz)
{
    // cout << u << ' ' << f << ' ' << sz << '\n';
    treeSz[u] = 1;
    PII res(__INT_MAX__, -1);
    int mx = 0;
    for (size_t i = 0; i != G[u].size(); ++i)
    {
        PII &e = G[u][i];
        int v = e.first;
        if (v == f || vis[v] == true)
        {
            continue;
        }
        res = min(res, treeCentroid(v, u, sz));
        // cout << u << ' ' << res.first << ' '<<
            res.second << '\n';
        treeSz[u] += treeSz[v];
        mx = max(mx, treeSz[v]);
    }
    mx = max(mx, sz - treeSz[u]);
    // cout << u << ':' << mx << ' ' << u << '\n';
    return min(res, {mx, u});
}
```

7 Graph Bipartite

7.1 Bipartite

```cpp
#include <cstring>
#include <iostream>
#include <stack>
#include <vector>

#define S 50050

using namespace std;

vector<int> map[S];
int visit[S];
bool valid;

void check(int start)
{
    stack<int> st;
    st.push(start);
    visit[start] = 1;

    while (valid && !st.empty())
    {
        int cur = st.top();
        st.pop();

        for (int i = 0; i < map[cur].size(); i++)
        {
            int next = map[cur][i];

            if (visit[next] == -1)
            {
                st.push(next);

                if (visit[cur] == 1)
                    visit[next] = 2;
                else
                    visit[next] = 1;
            }
            else if (visit[cur] == visit[next])
                valid = false;
        }
    }
}

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

    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;

        map[a].push_back(b);
        map[b].push_back(a);
    }

    // -1 : not visit, 1 : tsudere, 2 : proud
    memset(visit, -1, sizeof(visit));
    valid = true;

    for (int i = 1; i <= n; i++)
    {
        if (valid && visit[i] == -1)
        {
            check(i);
        }
    }

    if (valid)
        cout << "yes" << endl;
    else
        cout << "no" << endl;

    return 0;
}
```

7.2 BipartiteMatch

```cpp
int lhs, rhs, Left[MXV], G[MXV][MXV];
bitset<MXV> used;

bool dfs(int s)
{
    for (int i = 1; i <= rhs; i++)
    {
        if (!G[s][i] || used[i])
        {
            continue;
        }
        used[i] = true;
        if (Left[i] == -1 || dfs(Left[i]))
        {
            Left[i] = s;
            return true;
        }
    }
    return false;
}

int sol()
{
    int ret = 0;
    memset(Left, -1, sizeof(Left));
    for (int i = 1; i <= lhs; i++)
    {
        used.reset();
        if (dfs(i))
        {
            ret++;
        }
    }
    return ret;
}
```

7.3 KM

```cpp
template <typename T>
struct KM
{
    int n;
    int Left[N];
    T w[N][N], Lx[N], Ly[N];
    bitset<N> vx, vy;

    void init(int _n)
    {
        n = _n;
    }

    bool match(int i)
    {
        vx[i] = true;
        for (int j = 1; j <= n; j++)
        {
            if ((fabs(Lx[i] + Ly[j] - w[i][j]) <
                1e-9) && !vy[j])
            {
                vy[j] = 1;
                if (!Left[j] || match(Left[j]))
                {
                    Left[j] = i;
                    return true;
                }
            }
```

```
        }
        return false;
    }

    void update()
    {
        T a = 1e9;
        for (int i = 1; i <= n; i++)
        {
            if (vx[i])
            {
                for (int j = 1; j <= n; j++)
                {
                    if (!vy[j])
                    {
                        a = min(a, Lx[i] + Ly[j] -
                            w[i][j]);
                    }
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            if (vx[i])
            {
                Lx[i] -= a;
            }
            if (vy[i])
            {
                Ly[i] += a;
            }
        }
    }

    void hungarian()
    {
        for (int i = 1; i <= n; i++)
        {
            Left[i] = Lx[i] = Ly[i] = 0;
            for (int j = 1; j <= n; j++)
            {
                Lx[i] = max(Lx[i], w[i][j]);
            }
        }
        for (int i = 1; i <= n; i++)
        {
            while (1)
            {
                vx.reset();
                vy.reset();
                if (match(i))
                {
                    break;
                }
                update();
            }
        }
    }
};

/*
usage
KM<int> km; // declare with weight type
km.init(n); // initialize with vertex
km.hungarian(); // calculate
km.w[][]; // weight array
km.Left[i] // y_i match x_Left[i]
*/
```

7.4 Relation

```
1. 一般圖
|最大匹配|+|最小邊覆蓋|=|V|
|最大獨立集|+|最小點覆蓋|=|V|
|最大圖|=|補圖的最大獨立集|
2. 二分圖
|最大匹配|=|最小點覆蓋|
|最大獨立集|=|最小邊覆蓋|
|最大獨立集|=|V|-|最大匹配|
|最大圖|=|補圖的最大獨立集|
```

8 Graph Connectivity

8.1 decide

```
點雙連通
非根節點：low[i] >= depth[now]
根節點：如果子節點 >1 ，該點就是割點

邊雙連通：low[i] > depth[now]
```

8.2 low

```
bitset<MXV> is_cut_vertex, visit;
vector<int> G[MXV], low(MXV), depth(MXV);

void dfs(int now, int cur_depth)
{
    visit[now] = true;
    depth[now] = low[now] = cur_depth;
    for (auto i : G[now])
    {
        if (visit[i])
        { // ancestor
            low[now] = min(low[now], depth[i]);
        }
        else
        { // offspring
            dfs(i, cur_depth + 1);
            low[now] = min(low[now], low[i]);
        }
    }
    return;
}
```

9 Graph Flow

9.1 Dinic

```
using LL = long long;
struct Dinic
{
    int n, s, t, level[MXV], now[MXV];
    struct Edge
    {
        int v;
        LL rf; // rf: residual flow
        int re;
    };
    vector<Edge> e[MXV];
    void init(int _n, int _s, int _t)
    {
        n = _n;
        s = _s;
        t = _t;
        for (int i = 0; i <= n; i++)
            e[i].clear();
    }
    void add_edge(int u, int v, LL f)
    {
        e[u].push_back({v, f, (int)e[v].size()});
        e[v].push_back({u, f, (int)e[u].size() - 1});
        // for directional graph
```

```
        // e[v].push_back({u, 0, (int)e[u].size() - 1});
    }
    bool bfs()
    {
        fill(level, level + n + 1, -1);
        queue<int> q;
        q.push(s);
        level[s] = 0;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (auto it : e[u])
            {
                if (it.rf > 0 && level[it.v] == -1)
                {
                    level[it.v] = level[u] + 1;
                    q.push(it.v);
                }
            }
        }
        return level[t] != -1;
    }
    LL dfs(int u, LL limit)
    {
        if (u == t)
            return limit;
        LL res = 0;
        while (now[u] < (int)e[u].size())
        {
            Edge &it = e[u][now[u]];
            if (it.rf > 0 && level[it.v] == level[u] + 1)
            {
                LL f = dfs(it.v, min(limit, it.rf));
                res += f;
                limit -= f;
                it.rf -= f;
                e[it.v][it.re].rf += f;
                if (limit == 0)
                    return res;
            }
            else
                ++now[u];
        }
        if (!res)
            level[u] = -1;
        return res;
    }
    LL flow(LL res = 0)
    {
        while (bfs())
        {
            memset(now, 0, sizeof(now));
            res += dfs(s, INF);
        }
        return res;
    }
};

/*
usage
Dinic dinic; // declare
dinic.init(n, s, t); // initialize, n vertexs, start from s to t
dinic.add_edge(x, y, z); // add edge from x to y, weight is z
dinic.flow() // calculate max flow
*/
```

9.2 MCMF

```
using LL = long long;
struct MCMF
{
    struct Edge
    {
        int u, v;
        LL cost, cap;
    };
    int n, pre[MXV], cnt[MXV];
    vector<Edge> edges;
    vector<int> G[MXV];
    LL dis[MXV], ansFlow, ansCost;
    bitset<MXV> inque;
    void init(int _n)
    {
        n = _n;
        edges.clear();
        for (int i = 0; i <= n; ++i)
            G[i].clear();
    }
    void addEdge(int u, int v, LL cost, LL cap)
    {
        G[u].push_back(edges.size());
        edges.push_back({u, v, cost, cap});
        G[v].push_back(edges.size());
        edges.push_back({v, u, -cost, 0});
    }
    bool spfa(int s, int t)
    {
        queue<int> q;
        bool negative = false;
        fill(begin(dis), end(dis), INF);
        fill(begin(pre), end(pre), -1);
        fill(begin(cnt), end(cnt), 0);
        inque.reset();
        dis[s] = 0;
        cnt[s] = 1;
        q.push(s);
        inque[s] = true;
        while (!q.empty() && !negative)
        {
            int u = q.front();
            q.pop();
            inque[u] = false;
            for (int i : G[u])
            {
                Edge &e = edges[i];
                int v = e.v;
                LL cost = e.cost, cap = e.cap;
                if (dis[v] > dis[u] + cost && cap > 0)
                {
                    dis[v] = dis[u] + cost;
                    pre[v] = i;
                    if (inque[v])
                        continue;
                    q.push(v);
                    inque[v] = true;
                    ++cnt[v];
                    if (cnt[v] == n + 2)
                    {
                        negative = true;
                        break;
                    }
                }
            }
        }
        return dis[t] != INF;
    }
    LL update(int u, LL limit)
    {
        if (pre[u] == -1)
            return limit;
        int i = pre[u];
        Edge &e = edges[i];
        LL f = update(e.u, min(limit, e.cap));
        ansCost += f * e.cost;
        edges[i].cap -= f;
        edges[i ^ 1].cap += f;
        return f;
    }
```

```
    PLL sol(int s, int t, LL D)
    {
        ansFlow = ansCost = 0;
        while (spfa(s, t))
            ansFlow += update(t, INF);
        return make_pair(ansFlow, ansCost);
    }
};

/*
usage
MCMF<int> mcmf; // declare
mcmf.init(n, s, t); // initialize, n vertexs, start from s to t
mcmf.add_edge(x, y, z); // add edge from x to y, weight is z
mcmf.flow() // calculate max flow
*/
```

10 Graph Shortest Path

10.1 BellmanFord

```
struct Edge
{
    int t, w;
};
int v, e;
int d[N], cnt[N];
bitset<N> inq;
queue<int> Q;
vector<Edge> G[N];

void addEdge(int from, int to, int w) {
    G[from].push_back({to, w}); }

bool hasnegativeCycle()
{
    while (!Q.empty())
        Q.pop();
    for (int i = 1; i <= v; i++)
    {
        inq[i] = true;
        cnt[i] = d[i] = 0;
        Q.push(i);
    }
    while (!Q.empty())
    {
        int s = Q.front();
        Q.pop();
        inq[s] = false;
        for (Edge it : G[s])
        {
            if (d[it.t] > d[s] + it.w)
            {
                d[it.t] = d[s] + it.w;
                if (inq[it.t])
                    continue;
                Q.push(it.t);
                inq[it.t] = true;
                if (++cnt[it.t] > v)
                    return true;
            }
        }
    }
    return false;
}
```

10.2 Dijkstra

```
struct Dijkstra
{
    const int INF = 1000000000;
    int d[MXV], p[MXV];
    vector<Edge> E;
    vector<int> v[MXV];
    bitset<MXV> vis;

    void init()
    {
        fill(d, d + MXV, INF);
        memset(p, 0, sizeof(p));
        E.clear();
        for (int i = 0; i < MXV; i++)
        {
            v[i].clear();
        }
        vis.reset();
    }

    void addEdge(int from, int to, int w)
    {
        v[from].push_back(E.size());
        E.push_back(Edge{from, to, w});
    }

    void dijkstra(int s)
    {
        d[s] = 0;
        priority_queue<PII, vector<PII>,
            greater<PII>> states;
        vis.reset();
        states.push(MP(d[s], s));
        while (!states.empty())
        {
            PII state = states.top();
            states.pop();
            if (vis[state.second])
            {
                continue;
            }
            vis[state.second] = true;
            for (int u : v[state.second])
            {
                Edge e = E[u];
                if (d[e.to] > d[e.from] + e.w)
                {
                    d[e.to] = d[e.from] + e.w;
                    p[e.to] = e.from;
                    states.push(MP(d[e.to], e.to));
                }
            }
        }
    }
};

/*
Usage
Dijkstra dijkstra; // declare
dijkstra.init();
dijsktra.addEdge(int from, int to, int w); // add a directional Edge
dijkstra.dijkstra(int s) // calculation shortest distance from s
*/
```

10.3 FloydWarshall

```
template <typename T> struct FloydWarshall
{
    T d[MXV][MXV];
    void init() { memset(d, 0x3f, sizeof(d)); }
    void floydWarshall(int n)
    {
        for (int k = 1; k <= n; ++k)
        {
            for (int i = 1; i <= n; ++i)
            {
```

```
                for (int j = 1; j <= n; ++j)
                {
                    d[i][j] = d[j][i] = min(d[i][j],
                        d[i][k] + d[k][j]);
                }
            }
        }
    }
};

/*
usage
FloydWarshall<int> floydWarshall; // declare with distace's type
floydWarshall.init(); // initialize
floydWarshall.floydWarshall(); // calculate all-pair shortest path
*/
```

10.4 SPFA

```
struct Edge
{
    int t;
    long long w;
    Edge(){};
    Edge(int _t, long long _w) : t(_t), w(_w) {}
};

bool SPFA(int st)
{
    vector<int> cnt(n, 0);
    bitset<MXV> inq(0);
    queue<int> q;

    q.push(st);
    dis[st] = 0;
    inq[st] = true;
    while (!q.empty())
    {
        int cur = q.front();
        q.pop();
        inq[cur] = false;
        for (auto &e : G[cur])
        {
            if (dis[e.t] <= dis[cur] + e.w)
                continue;
            dis[e.t] = dis[cur] + e.w;
            if (inq[e.t])
                continue;
            ++cnt[e.t];
            if (cnt[e.t] > n)
                return false; // negtive cycle
            inq[e.t] = true;
            q.push(e.t);
        }
    }
    return true;
}
```

11 Math

11.1 Catalan

$C_0 = 1$ and $C_{n+1} = \frac{2(2n+1)}{n+2} C_n$,

11.2 Combination

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int M = 1000005;
int n, k;
LL m, phi;
vector<int> facs;
LL dp[M], dp2[M][32];
LL pw(LL x, LL y)
{
    LL ret = 1, tmp = x % m;
    while (y)
    {
        if (y & 1)
        {
            ret = ret * tmp % m;
        }
        tmp = tmp * tmp % m;
        y >>= 1;
    }
    return ret;
}
void init()
{
    facs.clear();
    LL x = m, sq = (LL)sqrt(m);
    phi = 1;
    for (LL i = 2; i <= sq; i++)
    {
        if (x % i)
            continue;
        phi *= i - 1;
        x /= i;
        facs.push_back(i);
        while (x % i == 0)
        {
            phi *= i;
            x /= i;
        }
    }
    if (x > 1)
    {
        phi *= x - 1;
        facs.push_back((int)x);
    }
    k = facs.size();
    dp[0] = 1;
    memset(dp2, 0, sizeof(dp2));
    for (int i = 1; i < M; i++)
    {
        LL tmp = i;
        for (int j = 0; j < k; j++)
        {
            dp2[i][j] = dp2[i - 1][j];
            while (tmp % facs[j] == 0)
            {
                tmp /= facs[j];
                dp2[i][j]++;
            }
        }
        dp[i] = dp[i - 1] * tmp % m;
    }
    return;
}
int main()
{
    while (cin >> n >> m)
    {
        init();
        while (n--)
        {
            LL ans = 1;
            int x, y;
            cin >> x >> y;
            for (int i = 0; i < k; i++)
            {
                ans = ans * pw(facs[i], dp2[x][i] -
                    dp2[x - y][i] - dp2[y][i]) %
                      m;
            }
```

```
            ans = ans * dp[x] % m;
            ans = ans * pw(dp[y], phi - 1) % m;
            ans = ans * pw(dp[x - y], phi - 1) % m;
            cout << ans << '\n';
        }
    }
}
```

11.3 Extend Euclidean.cpp

```
int extgcd(int a, int b, int &x, int &y)
{
    int d = a;
    if (b)
    {
        d = extgcd(b, a % b, y, x), y -= (a / b) * x;
    }
    else
        x = 1, y = 0;
    return d;
} // ax+by=1 ax同餘 1 mod b
```

11.4 FFT

```
struct Complex
{
    LD r, i;
    Complex(LD _r = 0.0, LD _i = 0.0)
    {
        r = _r;
        i = _i;
    }
    Complex operator+(Complex rhs) { return Complex(r + rhs.r, i + rhs.i); }
    Complex operator-(Complex rhs) { return Complex(r - rhs.r, i - rhs.i); }
    Complex operator*(Complex rhs)
    {
        return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs.r);
    }
};

template <typename T> struct FFT
{
    void fft(vector<Complex> &a, int n, int inv)
    {
        for (int i = 1, j = 0; i < n; ++i)
        {
            for (int k = (n >> 1); (j ^= k) < k; k >>= 1)
                ;
            if (i > j)
            {
                swap(a[i], a[j]);
            }
        }
        for (int m = 2; m <= n; m <<= 1)
        {
            Complex wm(cos(2 * PI * inv / m), sin(2 * PI * inv / m));
            for (int k = 0; k < n; k += m)
            {
                Complex w(1.0, 0.0);
                for (int j = 0; j < (m >> 1); ++j, w = w * wm)
                {
                    Complex u = a[k + j], t = w * a[k + j + (m >> 1)];
                    a[k + j] = u + t;
                    a[k + j + (m >> 1)] = u - t;
                }
            }
        }
        if (inv == -1)
        {
            FOR(i, 0, n)
            {
                a[i].r /= n;
                a[i].i /= n;
            }
        }
    }
    void convolution(vector<Complex> &A, vector<Complex> &B, vector<Complex> &C,
                     vector<Complex> &D, int n, vector<Complex> &ans)
    {
        fft(A, n, 1);
        fft(B, n, 1);
        fft(C, n, 1);
        fft(D, n, 1);
        FOR(i, 0, n) { ans[i] = A[i] * B[i] * C[i] * D[i]; }
        fft(ans, n, -1);
        return;
    }
};
```

11.5 GaussElimination

```
const int MAXN = 300;
const double EPS = 1e-8;
int n;
double A[MAXN][MAXN];
void Gauss()
{
    for (int i = 0; i < n; i++)
    {
        bool ok = 0;
        for (int j = i; j < n; j++)
        {
            if (fabs(A[j][i]) > EPS)
            {
                swap(A[j], A[i]);
                ok = 1;
                break;
            }
        }
        if (!ok)
            continue;
        double fs = A[i][i];
        for (int j = i + 1; j < n; j++)
        {
            double r = A[j][i] / fs;
            for (int k = i; k < n; k++)
            {
                A[j][k] -= A[i][k] * r;
            }
        }
    }
}
```

11.6 Matrix

```
struct Mat
{
    int sz;
    LL x[MXN][MXN];
    Mat() { memset(x, 0, sizeof(x)); }
    Mat(int _sz)
    {
        sz = _sz;
        memset(x, 0, sizeof(x));
    }
    Mat operator*(Mat a)
```

```
    {
        Mat res(sz);
        FOR(i, 1, sz + 1) FOR(j, 1, sz + 1) FOR(k, 1,
            sz + 1)
        {
            res.x[i][j] += x[i][k] * a.x[k][j];
            res.x[i][j] %= MOD;
        }
        return res;
    }
    void output()
    {
        FOR(i, 1, sz + 1) FOR(j, 1, sz + 1) cout <<
            x[i][j] << " \n"[j == sz];
    }
};
```

11.7 Phi

```
void phi_table(int n) // [1,n]
{
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (phi[i])
            continue;
        for (int j = i; j <= n; j += i)
        {
            if (!phi[j])
                phi[j] = j;
            phi[j] = phi[j] / i * (i - 1);
        }
    }
}
```

11.8 PowerTower

```
int POW(int a, int b, int mod)
{
    int ret = 1;
    int tmp = 1;
    for (int i = 0; i < b; i++)
    {
        tmp *= a;
        if (tmp > mod)
            break;
    }
    tmp = (tmp >= mod) ? mod : 0;
    for (; b; b >>= 1)
    {
        if (b & 1)
            ret = ret * a % mod;
        a = a * a % mod;
    }
    return ret + tmp;
}

int dfs(int d, int MOD)
{
    if (d == n - 1)
    {
        if (a[d] >= MOD)
            return (a[d] % MOD) + MOD;
        return a[d];
    }
    int k = dfs(d + 1, phi[MOD]);
    return POW(a[d], k, MOD);
}
```

11.9 Prime table

```
void primeTable()
{
    is_notp.reset();
    is_notp[0] = is_notp[1] = 1;
    for (int i = 2; i < N; i++)
    {
        if (!is_notp[i])
        {
            p.push_back(i);
        }
        for (int j = 0; j < (int)p.size() && i * p[j]
            < N; j++)
        {
            is_notp[i * p[j]] = 1;
            if (i % p[j] == 0)
            {
                break;
            }
        }
    }
}
```

12 String

12.1 Aho Corasick

```
struct Node
{
    char ch;
    int v;
    Node *next[MXW];
    Node *fail;
    Node(): v(0), fail(0) {
        memset(next,0,sizeof(next)); }
};

void insert(Node *root, char *s)
{
    int sz = strlen(s);
    FOR(i, 0, sz)
    {
        int v = s[i] - 'a';
        if (root->next[v] == NULL)
        {
            root->next[v] = new Node();
        }
        root = root->next[v];
        root->ch = s[i];
    }
    ++root->v;
}

queue<Node *> q;
void bulidAC(Node *root)
{
    Node *k, *tmp;
    FOR(i, 0, MXW)
    {
        if (root->next[i] != NULL)
        {
            root->next[i]->fail = root;
            q.push(root->next[i]);
        }
    }
    while (!q.empty())
    {
        k = q.front();
        q.pop();
        FOR(i, 0, MXW) if (k->next[i] != NULL)
        {
            tmp = k->fail;
            while (tmp != NULL)
            {
                if (tmp->next[i] != NULL)
```

```
                {
                    k->next[i]->fail = tmp->next[i];
                    break;
                }
                tmp = tmp->fail;
            }
            if (tmp == NULL)
            {
                k->next[i]->fail = root;
            }
            q.push(k->next[i]);
        }
    }
}

int ans;
void acAutomation(Node *root, char *s)
{
    Node *p = root;
    int sz = strlen(s);
    FOR(i, 0, sz)
    {
        int v = s[i] - 'a';
        while (p->next[v] == NULL && p != root)
        {
            p = p->fail;
        }
        p = p->next[v];
        if (p == NULL)
        {
            p = root;
        }
        Node *k = p;
        while (k != root)
        {
            if (k->v >= 0)
            {
                ans += k->v;
                k->v = -1;
            }
            else
            {
                break;
            }
            k = k->fail;
        }
    }
}

char s[MXS];
int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int n;
        Node *root =  new Node();
        scanf("%d", &n);
        while (n--)
        {
            scanf("%s", s);
            insert(root, s);
        }
        bulidAC(root);
        scanf("%s", s);
        ans = 0;
        acAutomation(root, s);
        printf("%d\n", ans);
    }
}
```

12.2 KMP

```
void bulid_fail_funtion(string B, int *fail)
{
    int len = B.length(), current_pos;
    current_pos = fail[0] = -1;
    for (int i = 1; i < len; i++)
    {
        while (current_pos != -1 && B[current_pos +
            1] != B[i])
        {
            current_pos = fail[current_pos];
        }
        if (B[current_pos + 1] == B[i])
        {
            current_pos++;
        }
        fail[i] = current_pos;
    }
}
void match(string A, string B, int *fail)
{
    int lenA = A.length(), lenB = B.length();
    int current_pos = -1;
    for (int i = 0; i < lenA; i++)
    {
        while (current_pos != -1 && B[current_pos +
            1] != A[i])
        {
            current_pos = fail[current_pos];
        }
        if (B[current_pos + 1] == A[i])
            current_pos++;
        if (current_pos == lenB - 1)
        { // match! A[i-lenB+1,i]=B
            current_pos = fail[current_pos];
        }
    }
}
int main()
{
    int t, i;
    string s;
    for (i = 0, cin >> t; i < t; i++)
    {
        cin >> s;
        int fail[N];
        bulid_fail_funtion(s, fail);
        int p = s.length() - 1;
        if (fail[p] != -1 && (p + 1) % (p - fail[p])
            == 0)
            printf("%d\n", p - fail[p]);
        else
            printf("%d\n", p + 1);
    }
}
```

12.3 Manacher

```
void sol(char *s)
{
    int sz = strlen(s);
    si = 0;
    ss[si++] = '$';
    ss[si++] = '#';
    FOR(i, 0, sz)
    {
        ss[si++] = s[i];
        ss[si++] = '#';
    }
    ss[si++] = '_';
    int mx = 0, id = 0;
    FOR(i, 0, si)
    {
        if (mx > i)
        {
            ma[i] = min(ma[2 * id - i], mx - i);
        }
        else
        {
```

```
            ma[i] = 1;
        }
        while (ss[i + ma[i]] == ss[i - ma[i]])
        {
            ++ma[i];
        }
        if (i + ma[i] > mx)
        {
            id = i;
            mx = i + ma[i];
        }
    }
}
```

12.4 Trie

```
struct Node
{
    char ch;
    int v;
    Node *next[26];
    Node()
    {
        v = 0;
        FOR(i, 0, 26) next[i] = NULL;
    }
};

void insert(Node *root, string s)
{
    FOR(i, 0, s.size())
    {
        int v = s[i] - 'a';
        if (root->next[v] == NULL)
        {
            root->next[v] = new Node();
        }
        root = root->next[v];
        ++root->v;
        root->ch = s[i];
    }
    return;
}
void search(Node *root, string s)
{
    FOR(i, 0, s.size())
    {
        int v = s[i] - 'a';
        root = root->next[v];
        if (root->v == 1)
        {
            cout << s << ' ' << s.substr(0, i + 1) <<
                '\n';
            return;
        }
    }
    cout << s << ' ' << s << '\n';
}

int main()
{
    vector<string> v;
    string s;
    Node *root = new Node();
    while (cin >> s)
    {
        insert(root, s);
        v.push_back(s);
    }
    FOR(i, 0, v.size()) { search(root, v[i]); }
}
```

12.5 Zvalue

```
void z_value(string s)
{
    int L = 0, R = 0;
    z[0] = 0;
    for (int i = 1; i < (int)s.size(); i++)
    {
        if (i > R)
        {
            z[i] = 0;
        }
        else
        {
            int ip = i - L;
            if (ip + z[ip] < z[L])
            {
                z[i] = z[ip];
            }
            else
            {
                z[i] = R - i + 1;
            }
        }
        while (i + z[i] < (int)s.size() && s[i +
            z[i]] == s[z[i]])
        {
            z[i]++;
        }
        if (i + z[i] - 1 > R)
        {
            L = i;
            R = L + z[i] - 1;
        }
    }
}
```