2024.11.15

一、常用

头文件

```
/*

_/       _/     _/       _/     _/       _/    _/_/_/_/_/      _/_/        _/
_/
 _/      _/      _/       _/      _/     _/          _/        _/     _/      _/
_/
   _/   _/       _/       _/       _/   _/          _/       _/       _/     _/_/
_/_/
     _/_/        _/_/_/_/_/_/          _/            _/       _/       _/     _/   _/
_/
   _/   _/       _/       _/          _/            _/       _/       _/     _/
_/
```

```
 _/    _/    _/      _/        _/          _/       _/    _/     _/
_/
_/      _/    _/      _/        _/          _/         _/_/       _/
_/

*/
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
#define rep(i,a,n) for(int i=a;i<n;i++)
#define per(i,a,n) for(int i=n-1;i>=a;i--)
#define fastio ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
#define multi int _;cin>>_;while(_--)
#define debug(x) cerr << #x << " = " << (x) << endl;
#define int long long
#define pb push_back
#define eb emplace_back
ll gcd(ll a,ll b){ return b?gcd(b,a%b):a;}
mt19937_64 mrand(chrono::steady_clock().now().time_since_epoch().count());
int rnd(int l,int r){ return mrand() % (r - l + 1) + l;}
void test() {cerr << "\n";}
template<typename T, typename... Args>
void test(T x, Args... args) {cerr << x << " ";test(args...);}
const ll MOD = 998244353;
// const ll MOD = 1e9+7;
ll ksm(ll x,ll y){ll ans=1;x%=MOD;while(y)
{if(y&1)ans=ans*x%MOD;x=x*x%MOD,y/=2;}return ans;}

const int P1 = 972152273, base1 = 809;
const int P2 = 905563261, base2 = 919;
const ll N = 200005;
//head



signed main()
{
#ifdef localfreopen
    // freopen("1.in","r",stdin);
#endif
    fastio

    return 0;
}
```

预编译优化命令

```
#pragma GCC optimize("O3,unroll-loops")
//这行告诉GCC编译器使用O3优化级别和循环展开。O3是GCC提供的最高优化级别，它会尝试使用所有的
程序优化策略。"unroll-loops"是一个特定的优化选项，它会尝试将循环展开以减少循环的开销。

#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

```

```
//这行告诉GCC编译器生成的代码应该针对支持AVX2，BMI，BMI2，LZCNT和POPCNT指令集的CPU。这些都是特定的CPU指令集，可以提高代码的性能，但是生成的代码可能无法在不支持这些指令集的CPU上运行。
#pragma GCC optimize("Ofast")
#pragma GCC target("avx", "sse2")
#pragma GCC optimize("inline")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

快读

```
inline int read()
{
    int x=0,f=1;char ch=getchar();
    while (ch<'0'||ch>'9'){if (ch=='-') f=-1;ch=getchar();}
    while (ch>='0'&&ch<='9'){x=x*10+ch-48;ch=getchar();}
    return x*f;
}
```

对拍

```
:loop
 data.exe > 1.in
 my.exe <1.in >my.out
 std.exe <1.in >std.out
 fc my.out std.out
if not errorlevel 1 goto loop
pause
goto loop
```

__int128

```
__int128 read()
{
    __int128 f=1,w=0;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')
        f=-1;
        ch=getchar();
    }
    while(ch<='9'&&ch>='0')
    {
        w=w*10+ch-'0';
        ch=getchar();
    }
    return f*w;
}

void print(__int128 x)
{
    if(x<0)
    {
        putchar('-');
        x=-x;
    }
    if(x>9)print(x/10);
    putchar(x%10+'0');
}
```

对拍(linux)

```
#!/bin/bash
while true; do
./data>1.in
./std<1.in>std.out
./my<1.in>my.out
if diff std.out my.out; then
printf AC
else
echo WA
exit 0
fi
sleep 1
done
```

checker

```
set -e
[ $# == 2 ] || { echo invalid args ; exit 1 ; }
compileg++ $2.cpp || { echo CE ; exit 1 ; }
src=./samples-$1
dir=$1-test
mkdir -p $dir
cp $src/* $dir/
cd $dir
mv ../a.out ./$2
for input in *.in; do
    [ $input == "*.in" ] && exit 0
    cas=${input%.in}
    output=$cas.out
    answer=$cas.ans
    timeout 1 ./$2 < $input > $output 2> $cas.err || { echo Case $cas : TLE
or RE ; continue ; }
    if diff -ZA $output $answer > $cas.dif ; then
        echo Case $cas : AC
    else
        echo Case $cas : WA
        cat $cas.dif $cas.err
    fi
done
```

check(windows)

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

int main (int argc, char *argv[]) {
    std::string s = argv[1];
    for (int i = 1; i <= 100; ++i) {
        string in = std::to_string(i) + ".in";
```

```
        string out = std::to_string(i) + ".out";
        system(("std.exe <" + s + "/" + in + " >my.out").c_str());
        std::cout << "std.exe <" + s + "/" + in + " >my.out" << "\n";
        string a = "fc my.out " + s + "/" + out + "";
        std::cout << a << "\n";
        int ans = system(a.c_str());
        if (ans == 0) {
            cout << "Test case " << i << ": AC" << endl;
        } else {
            cout << "Test case " << i << ": WA" << endl;
            break;
        }
    }
    system("pause");
    return 0;
}

```

check(linux)

```
#include<bits/stdc++.h>

using namespace std;

int main (int argc, char *argv[]) {
    std::string s = argv[1];
    for (int i = 1; i <= 100; ++i) {
        string in = std::to_string(i) + ".in";
        string out = std::to_string(i) + ".out";
        system(("./" + s + " <" + s + "_samples/" + in + "
>my.out").c_str());
        string a = "diff my.out " + s + "_samples/" + out + "";
        int ans = system(a.c_str());
        if (ans != 0) {
            cout << "Test case " << i << ": WA" << endl;
            break;
        } else {
            cout << "Test case " << i << ": AC" << endl;
        }
    }

    return 0;
}

```

builtin函数

```
__builtin_ctz( ) / __buitlin_ctzll( )
返回括号内数的二进制表示数末尾0的个数
__buitlin_clz( ) / __buitlin_clzll( )
用法:返回括号内数的二进制表示数前导0的个数
__builtin_popcount( )
```

```
用法:返回括号内数的二进制表示数1的个数
__builtin_parity( )
用法:判断括号中数的二进制表示数1的个数的奇偶性(偶数返回0 ， 奇数返回1)
__builtin_ffs( )
用法:返回括号中数的二进制表示数的最后一个1在第几位(从后往前算)
__builtin_sqrt( ) 8位
__builtin_sqrtf( ) 4位
用法:快速开平方，需要硬件有浮点支持，能快10倍
__builtin_abs( )
__builtin_fabs( )
__builtin_powi( )
__builtin_memset( )
__builtin_memcpy( )
__builtin_strlen( )
__builtin_sin( )
__builtin_cos( )
__builtin_tan( )
```

二、字符串

kmp

```
vector<int> kmp(string s)
{//string的形式为'#' + t1 + '#' + s
    int n = s.size() - 1;
    vector<int> nxt(s.size());
    int j = 0;
    for(int i = 2 ; i <= n ; i++ ){
        while(j && s[j + 1] != s[i]) j = nxt[j];
        if(s[j + 1] == s[i]) j++;
        nxt[i] = j;
    }
    return nxt;
}//从第lent + 2 位 到 lent + lens + 1位为 s
```

manacher

```
vector<int> manacher(string s)
{//string为#A#B#C#...#Z#
    int n = s.size();
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k])  k++;
        d1[i] = k--;
        if (i + k > r)
        {
            l = i - k;
```

```
            r = i + k;
        }
    }
    return d1;
}
```

最小表示法

```
string minrep(string s)
{//s从s[0]开始存
    int k = 0, i = 0, j = 1, n = s.size();
    while (k < n && i < n && j < n) {
        if (s[(i + k) % n] == s[(j + k) % n]) {
            k++;
        } else {
            s[(i + k) % n] > s[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
            if (i == j) i++;
            k = 0;
        }
    }
    i = min(i, j);
    return s.substr(i, N) + s.substr(0, i);
}
```

Z函数

```
vector<int> exkmp(string s)
{
    vector<int> p(s.size());
    int n = s.size() - 1;
    int L = 1, R = 0;
    p[1] = 0;
    for(int i = 2 ; i <= n ; i++ )
    {
        if(i > R)
        {
            p[i] = 0;
        }else{
            int k = i - L + 1;
            p[i] = min(p[k], R - i + 1);
        }
        while(i + p[i] <= n && s[p[i] + 1] == s[i + p[i]])
        {
            ++p[i];
        }
        if(i + p[i] - 1 > R)
        {
            L = i;
            R = i + p[i] - 1;
        }
    }
    return p;
}//从lent + 2位到lent + lens + 1位为 s
```

```
//******p[1] = 0，但实际从第一位往后能匹配lent的总长
```

AC自动机

```
struct ACautomaton {
    vector<vector<int>> nxt, end;
    vector<int> fail;
    int vtot = 0;
    ACautomaton() : nxt(1, vector<int>(26, 0)), end(1), fail(1){

    }
    ACautomaton(vector<string> ss){
        ACautomaton();
        for (auto s : ss) {
            insert(s);
        }
        buildfail();
    }
    int newnode() {
        int cur = ++vtot;
        nxt.push_back(vector<int>(26, 0));
        end.push_back(vector<int>(0));
        fail.emplace_back(0);
        return cur;
    }
    void insert(string s, int id = 0) {
        int now = 0;
        for (auto c : s) {
            int x = c - 'a';
            if (!nxt[now][x]) {
                nxt[now][x] = newnode();
            }
            now = nxt[now][x];
        }
        end[now].emplace_back(id);
    }
    void buildfail() {
        queue<int> q;
        for (int i = 0; i <= 25; i++) {
            if (nxt[0][i]) {
                fail[nxt[0][i]] = 0;
                q.push(nxt[0][i]);
            }
        }
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            for (int i = 0; i <= 25; i++) {
                if (nxt[now][i]) {
                    fail[nxt[now][i]] = nxt[fail[now]][i];
                    q.push(nxt[now][i]);
                } else {
                    nxt[now][i] = nxt[fail[now]][i];
                }
            }
```

```
        }
    }
    int query(string s) {
        int now = 0, ans = 0;
        for (int i = 0; i < s.size(); i++) {
            char c = s[i];
            int x = c - 'a';
            now = nxt[now][x];
            ///自定义
        }
        return ans;
    }
};// root = 0, ***记得buildfail
```

SA(nlogn)

```
sa[i]：排名为i的后缀的位置
rk[i]：第i个位置开始的后缀的排名，作为基数排序的第一关键字
struct SA{
    vector<int> sa, rk, oldrk, id, key1, cnt, ht;
    vector<vector<int>> st;
    int i, m = 127, p, w;
    bool cmp(int x, int y, int w) {
        return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
    }// key1[i] = rk[id[i]]（作为基数排序的第一关键字数组）
    int n;
    SA(string s)
    {
        n = s.size() - 1;
        oldrk.resize(2 * n + 5);
        sa.resize(n + 2);
        rk.resize(n + 2);
        id.resize(n + 2);
        key1.resize(n + 2);
        cnt.resize(max(n + 5, 1300ll));
        for (i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
        for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
        for (i = n; i >= 1; --i) sa[cnt[rk[i]]--] = i;
        for (w = 1;; w <<= 1, m = p) {  // m=p 就是优化计数排序值域
            for (p = 0, i = n; i > n - w; --i) id[++p] = i;
            for (i = 1; i <= n; ++i)
                if (sa[i] > w) id[++p] = sa[i] - w;
            fill(cnt.begin(), cnt.end(), 0);
            for (i = 1; i <= n; ++i) ++cnt[key1[i] = rk[id[i]]];
            // 注意这里px[i] != i，因为rk没有更新，是上一轮的排名数组

            for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
            for (i = n; i >= 1; --i) sa[cnt[key1[i]]--] = id[i];
            for(int i = 1 ; i <= n ; i++)
            {
                oldrk[i] = rk[i];
            }
            for (p = 0, i = 1; i <= n; ++i)
```

```
                rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
            if (p == n) {
                break;
            }
        }
        // height数组构建
        ht.resize(n + 2);
        int k = 0;
        for(int i = 1 ; i <= n ; i++ )
        {
            k = max(k - 1, 0ll);
            if(rk[i] == 1) continue;
            int j = sa[rk[i] - 1];
            while(s[i + k] == s[j + k]) k++;
            ht[rk[i]] = k;
        }

        // LCPst表构建
        st.resize(24);
        st[0].resize(n + 5);
        for(int i = 1 ; i <= n ; i++ )
        {
            st[0][i] = ht[i];
        }
        for(int j = 1 ; j <= 22 ; j++ )
        {
            st[j].resize(n + 5);
            for(int i = 1 ; i + (1 << j) - 1 <= n ; i++ )
            {
                st[j][i] = min(st[j - 1][i], st[j - 1][i + (1ll << j - 1)]);
            }
        }
    }
    int LCP(int u, int v)
    {
        if(u == v) return n - u + 1;
        if(rk[u] > rk[v]) swap(u, v);
        int l = rk[u] + 1, r = rk[v];
        int len = __lg(r - l + 1);
        return min(st[len][l], st[len][r - (1 << len) + 1]);
    }
};
//字符串存在1~n
//如果要用vector<int>. 记得离散化
// sa[i] 表示字典序第 i 小的后缀起始点在sa[i]
// rk[i] 表示后缀起点在 i 的字符串字典序排 rk[i]
```

SA(offline)

```
// sa[i]：排名为i的后缀的位置
// rk[i]：第i个位置开始的后缀的排名，作为基数排序的第一关键字
// tp[i]：第二关键字中，排名为i的数的位置
// cnt[i]：有多少个元素排名为i
// s[i]：原输入数组

```

```cpp
// init: s[1..n], n = strlen(s + 1), m = SIGMA, makesa()
const int N = 1E6 + 5;
#define rep(i, s, t) for (int i = s; i <= t; ++i)
#define per(i, s, t) for (int i = t; i >= s; --i)

int m = 256;
char s[N];
int n, sa[N], rk[N], tp[N], cnt[N];
void init() {
    rep(i, 1, n) rk[i] = s[i], tp[i] = i;
}
void Qsort() {
    rep(i, 1, m) cnt[i] = 0;
    rep(i, 1, n) ++ cnt[rk[i]];
    rep(i, 1, m) cnt[i] += cnt[i - 1];
    per(i, 1, n) sa[cnt[rk[tp[i]]] --] = tp[i];
}
void get_sort() {
    for(int w = 1, p = 0; w <= n; m = p, p = 0, w <<= 1) {
        rep(i, n - w + 1, n) tp[++ p] = i;
        rep(i, 1, n) if(sa[i] > w) tp[++ p] = sa[i] - w;
        Qsort(), swap(rk, tp), p = rk[sa[1]] = 1;
        rep(i, 2, n) rk[sa[i]] = (tp[sa[i]] == tp[sa[i - 1]]
                                  && tp[sa[i] + w] == tp[sa[i - 1] + w]) ? p
: ++ p;
        if(p == n) return;
    }
}
void makesa() {
    init();Qsort();get_sort();
}
int ht[N];
void makeht() {
    for(int k = 0, i = 1; i <= n; i++) {
        k = max(k - 1, 0);
        if(rk[i] == 1) continue;
        int j = sa[rk[i] - 1];
        while(s[i + k] == s[j + k]) k++;
        ht[rk[i]] = k;
    }
}


int st[25][N];
void makest() {
    rep(i, 1, n) st[0][i] = ht[i];
    rep(j, 1, 22) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            st[j][i] = min(st[j - 1][i], st[j - 1][i + (1ll << j - 1)]);
        }
    }
}

int getLcp(int u, int v) {
    if(u == v) return n - u + 1;
    if(rk[u] > rk[v]) swap(u, v);
```

```
    int l = rk[u] + 1, r = rk[v];
    int len = __lg(r - l + 1);
    return min(st[len][l], st[len][r - (1 << len) + 1]);
}
```

SAIS

```
char str[1000010];
int n, a[2000100], sa[2000100], typ[2000100], c[1000100], p[2000100],
sbuc[1000100], lbuc[1000100], name[1000100];
inline int islms(int *typ, int i)
{
    return !typ[i] && (i == 1 || typ[i - 1]);
}
int cmp(int *s, int *typ, int p, int q)
{
    do {
        if (s[p] != s[q]) return 1;
        p++;  q++;
    } while (!islms(typ, p) && !islms(typ, q));
    return (!islms(typ, p) || !islms(typ, q) || s[p] != s[q]);
}

void isort(int *s, int *sa, int *typ, int *c, int n, int m)
{
    int i;
    for (lbuc[0] = sbuc[0] = c[0], i = 1; i <= m; i++) {
        lbuc[i] = c[i - 1] + 1;
        sbuc[i] = c[i];
    }
    for (i = 1; i <= n; i++)
        if (sa[i]>1 && typ[sa[i] - 1])
            sa[lbuc[s[sa[i] - 1]]++] = sa[i] - 1;
    for (i = n; i >= 1; i--)
        if (sa[i]>1 && !typ[sa[i] - 1])
            sa[sbuc[s[sa[i] - 1]]--] = sa[i] - 1;
}

void build_sa(int *s, int *sa, int *typ, int *c, int *p, int n, int m)
{
    int i;
    for (i = 0; i <= m; i++) c[i] = 0;
    for (i = 1; i <= n; i++) c[s[i]]++;
    for (i = 1; i <= m; i++) c[i] += c[i - 1];
    typ[n] = 0;
    for (i = n - 1; i >= 1; i--)
        if (s[i]<s[i + 1]) typ[i] = 0;
        else if (s[i]>s[i + 1]) typ[i] = 1;
        else typ[i] = typ[i + 1];
    int cnt = 0;
    for (i = 1; i <= n; i++)
        if (!typ[i] && (i == 1 || typ[i - 1])) p[++cnt] = i;
    for (i = 1; i <= n; i++) sa[i] = 0;
```

```
    for (i = 0; i <= m; i++) sbuc[i] = c[i];
    for (i = 1; i <= cnt; i++)
        sa[sbuc[s[p[i]]]--] = p[i];
    isort(s, sa, typ, c, n, m);
    int last = 0, t = -1, x;
    for (i = 1; i <= n; i++)
    {
        x = sa[i];
        if (!typ[x] && (x == 1 || typ[x - 1]))
        {
            if (!last || cmp(s, typ, x, last))
                name[x] = ++t;
            else name[x] = t;
            last = x;
        }
    }
    for (i = 1; i <= cnt; i++)
        s[n + i] = name[p[i]];
    if (t<cnt - 1) build_sa(s + n, sa + n, typ + n, c + m + 1, p + n, cnt,
t);
    else
        for (i = 1; i <= cnt; i++)
            sa[n + s[n + i] + 1] = i;
    for (i = 0; i <= m; i++) sbuc[i] = c[i];
    for (i = 1; i <= n; i++) sa[i] = 0;
    for (i = cnt; i >= 1; i--)
        sa[sbuc[s[p[sa[n + i]]]]--] = p[sa[n + i]];
    isort(s, sa, typ, c, n, m);
}

int main()
{
    scanf("%s", str);
    n = strlen(str);
    int i;
    for (i = 1; i <= n; i++)
        a[i] = str[i - 1];
    a[++n] = 0;
    build_sa(a, sa, typ, c, p, n, 200);
    for (i = 2; i <= n; i++)
        printf("%d%s", sa[i], i<n ? " " : "\n");
    return 0;
}
```

SAIS

```
void induced_sort(const vector<int> &vec, int val_range, vector<int> &SA,
const vector<bool> &sl, const vector<int> &lms_idx) {
    vector<int> l(val_range, 0), r(val_range, 0);
    for (int c : vec) {
        if (c + 1 < val_range) ++l[c + 1];
        ++r[c];
    }
    partial_sum(l.begin(), l.end(), l.begin());
    partial_sum(r.begin(), r.end(), r.begin());
```

```
    fill(SA.begin(), SA.end(), -1);
    for (int i = lms_idx.size() - 1; i >= 0; --i)
        SA[--r[vec[lms_idx[i]]]] = lms_idx[i];
    for (int i : SA)
        if (i >= 1 && sl[i - 1]) {
            SA[l[vec[i - 1]]++] = i - 1;
        }
    fill(r.begin(), r.end(), 0);
    for (int c : vec)
        ++r[c];
    partial_sum(r.begin(), r.end(), r.begin());
    for (int k = SA.size() - 1, i = SA[k]; k >= 1; --k, i = SA[k])
        if (i >= 1 && !sl[i - 1]) {
            SA[--r[vec[i - 1]]] = i - 1;
        }
}
vector<int> SA_IS(const vector<int> &vec, int val_range) {
    const int n = vec.size();
    vector<int> SA(n), lms_idx;
    vector<bool> sl(n);
    sl[n - 1] = false;
    for (int i = n - 2; i >= 0; --i) {
        sl[i] = (vec[i] > vec[i + 1] || (vec[i] == vec[i + 1] && sl[i +
1]));
        if (sl[i] && !sl[i + 1]) lms_idx.push_back(i + 1);
    }
    reverse(lms_idx.begin(), lms_idx.end());
    induced_sort(vec, val_range, SA, sl, lms_idx);
    vector<int> new_lms_idx(lms_idx.size()), lms_vec(lms_idx.size());
    for (int i = 0, k = 0; i < n; ++i)
        if (!sl[SA[i]] && SA[i] >= 1 && sl[SA[i] - 1]) {
            new_lms_idx[k++] = SA[i];
        }
    int cur = 0;
    SA[n - 1] = cur;
    for (size_t k = 1; k < new_lms_idx.size(); ++k) {
        int i = new_lms_idx[k - 1], j = new_lms_idx[k];
        if (vec[i] != vec[j]) {
            SA[j] = ++cur;
            continue;
        }
        bool flag = false;
        for (int a = i + 1, b = j + 1;; ++a, ++b) {
            if (vec[a] != vec[b]) {
                flag = true;
                break;
            }
            if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1])) {
                flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b - 1]));
                break;
            }
        }
        SA[j] = (flag ? ++cur : cur);
    }
    for (size_t i = 0; i < lms_idx.size(); ++i)
        lms_vec[i] = SA[lms_idx[i]];
```

```
    if (cur + 1 < (int)lms_idx.size()) {
        auto lms_SA = SA_IS(lms_vec, cur + 1);
        for (size_t i = 0; i < lms_idx.size(); ++i) {
            new_lms_idx[i] = lms_idx[lms_SA[i]];
        }
    }
    induced_sort(vec, val_range, SA, sl, new_lms_idx);
    return SA;
}
template <class T>
vector<int> suffix_array(const T &s, const int LIM = 128) {
    vector<int> vec(s.size() + 1);
    copy(begin(s), end(s), begin(vec));
    vec.back() = 0 ;
    // vec.back() = '$';
    auto ret = SA_IS(vec, LIM);
    ret.erase(ret.begin());
    return ret;
}
vector<int> getRank(const vector<int> &sa) {
    vector<int> rk(sa.size());
    for (int i = 0 ; i < sa.size(); i++) {
        rk[sa[i]] = i;
    }
    return rk;
}
template <class T>
vector<int> getHeight(const T &s, const vector<int> &sa) {
    int n = s.size(), k = 0;
    vector<int> ht(n), rank(n);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0; i < n; i++, k ? k-- : 0) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = sa[rank[i] + 1];
        while (i + k < n && j + k < n && s[i + k] == s[j + k]) ++ k;
        ht[rank[i] + 1] = k;
    }
    ht[0] = 0;
    return ht;
}
template <class T>
vector<vector<int>> buildLCP(const T &s, const vector<int> ht) {
    vector<vector<int>> st;
    int n = s.size() - 1;
    int LOG = __lg(n) + 1;
    st.resize(LOG);
    st[0].resize(n + 1);
    for(int i = 1 ; i <= n ; i++ )
    {
        st[0][i] = ht[i];
    }
    for(int j = 1 ; j <= LOG ; j++ )
    {
```

```
        st[j].resize(n + 1);
        for(int i = 1 ; i + (1 << j) - 1 <= n ; i++ )
        {
            st[j][i] = min(st[j - 1][i], st[j - 1][i + (1ll << j - 1)]);
        }
    }
    return st;
}
void use() {
    vector<vector<int>> st;
    vector<int> rk;
    int n;
    int u, v;
    function<int(int, int)> lcp = [&](int u, int v)
    {
        if(u == v) return n - u + 1;
        if(rk[u] > rk[v]) swap(u, v);
        int l = rk[u] + 1, r = rk[v];
        int len = __lg(r - l + 1);
        return min(st[len][l], st[len][r - (1 << len) + 1]);
    };
}
```

SAM

```
struct SuffixAutomaton
{
    int tot, last;
    vector<int> len, link, sz;
    vector<vector<int>> nxt;
    //vector<pii> order;
    int n;
    SuffixAutomaton(int _n) :n(_n), sz(2 * _n + 5), len(2 * _n + 5), link(2 * _n + 5), nxt(2 * _n + 5, vector<int>(33, 0))
    {
        len[1] = 0;
        link[1] = -1;
        nxt[1].clear();
        nxt[1].resize(33);
        tot = 2;
        last = 1;
    }
    void extend(int c)
    {
        int cur = tot++, p;
        len[cur] = len[last] + 1;
        nxt[cur].clear();
        nxt[cur].resize(33);
        for (p = last; p != -1 && !nxt[p][c]; p = link[p])
            nxt[p][c] = cur;
        if (p == -1) link[cur] = 1;
        else
        {
```

```
            int q = nxt[p][c];
            if (len[p] + 1 == len[q]) link[cur] = q;
            else
            {
                int clone = tot++;
                len[clone] = len[p] + 1;
                link[clone] = link[q];
                nxt[clone] = nxt[q];
                for (; p != -1 && nxt[p][c] == q; p = link[p])
                    nxt[p][c] = clone;
                link[q] = link[cur] = clone;
            }
        }
        last = cur;
        sz[cur] = 1;
    }
    vector<vector<int>> adj;
    void buildLinkTree()
    {
        adj.resize(tot + 1);
        for (int i = 2; i <= tot; i++ )
        {
            adj[link[i]].push_back(i);
        }
    }
};//sam的root为1
```

ExSAM

```
struct EXSAM
{
    const int CHAR_NUM = 30;    // 字符集个数，注意修改下方的 (-'a')
    int tot;                    // 节点总数：[0, tot)
    int n;
    vector<int> len, link;
    vector<vector<int>> nxt;
    EXSAM (int _n) : n(_n), len(_n * 2 + 5), link(_n * 2 + 5), nxt(n * 2 + 5, vector<int>(CHAR_NUM + 1, 0))
    {
        tot = 2;
        link[1] = -1;
    }
    int insertSAM(int last, int c)    // last 为父 c 为子
    {
        int cur = nxt[last][c];
        if (len[cur]) return cur;
        len[cur] = len[last] + 1;
        int p = link[last];
        while (p != -1)
        {
            if (!nxt[p][c])
                nxt[p][c] = cur;
            else
                break;
            p = link[p];
```

```
        }
        if (p == -1)
        {
            link[cur] = 1;
            return cur;
        }
        int q = nxt[p][c];
        if (len[p] + 1 == len[q])
        {
            link[cur] = q;
            return cur;
        }
        int clone = tot++;
        for (int i = 0; i < CHAR_NUM; ++i)
            nxt[clone][i] = len[nxt[q][i]] != 0 ? nxt[q][i] : 0;
        len[clone] = len[p] + 1;
        while (p != -1 && nxt[p][c] == q)
        {
            nxt[p][c] = clone;
            p = link[p];
        }
        link[clone] = link[q];
        link[cur] = clone;
        link[q] = clone;
        return cur;
    }

    int insertTrie(int cur, int c)
    {
        if (nxt[cur][c]) return nxt[cur][c];  // 已有该节点 直接返回
        return nxt[cur][c] = tot++;             // 无该节点 建立节点
    }

    void insert(const string &s)
    {
        int root = 1;
        for (auto ch : s) root = insertTrie(root, ch - 'a');
    }

    void insert(const char *s, int n)
    {
        int root = 1;
        for (int i = 0; i < n; ++i)
            root =
                insertTrie(root, s[i] - 'a');  // 一边插入一边更改所插入新节点的父节点
    }

    void build()
    {
        queue<pair<int, int>> q;
        for (int i = 0; i < 26; ++i)
            if (nxt[1][i]) q.push({i, 1});
        while (!q.empty())    // 广搜遍历
        {
            auto item = q.front();
```

```
            q.pop();
            auto last = insertSAM(item.second, item.first);
            for (int i = 0; i < 26; ++i)
                if (nxt[last][i]) q.push({i, last});
        }
    }
};
```

PAM

```
const int N = 5e5 + 10, Sigma = 26;
char s[N];
int lastans, n;
struct Palindrome_Automaton {
    int ch[N][Sigma], fail[N], len[N], sum[N], cnt, last;
    Palindrome_Automaton() {
        cnt = 1;
        fail[0] = 1, fail[1] = 1, len[1] = -1;
    }
    int getfail(int x, int i) {
        while(i - len[x] - 1 < 0 || s[i - len[x] - 1] != s[i]) x = fail[x];
        return x;
    }
    void insert(char c, int i) {
        int x = getfail(last, i), w = c - 'a';
        if(!ch[x][w]) {
            len[++cnt] = len[x] + 2;
            int tmp = getfail(fail[x], i);
            fail[cnt] = ch[tmp][w];
            sum[cnt] = sum[fail[cnt]] + 1;
            ch[x][w] = cnt;
        }
        last = ch[x][w];
    }
} PAM;
```

PAM(new)

```
struct PAM {
    int sz, tot, last;
    vector<int> cnt, len, fail;
    vector<vector<int>> ch;
    vector<char> s;
    PAM(int n) : cnt(n + 5), ch(n + 5, vector<int>(30)), len(n + 5), fail(n + 5), s(n + 5) {
        clear();
    }
    int node(int l) {    // 建立一个新节点，长度为 l
        sz++;
        ch[sz].assign(30, 0);
        len[sz] = l;
        fail[sz] = cnt[sz] = 0;
        return sz;
    }
```

```
    void clear() {   // 初始化
        sz = -1;
        last = 0;
        s[tot = 0] = '$';
        node(0);
        node(-1);
        fail[0] = 1;
    }
    int getfail(int x) {   // 找后缀回文
        while (s[tot - len[x] - 1] != s[tot]) x = fail[x];
        return x;
    }
    void insert(char c)    // 建树
    {
        s[++tot] = c;
        int now = getfail(last);
        if (!ch[now][c - 'a'])
        {
            int x = node(len[now] + 2);
            fail[x] = ch[getfail(fail[now])][c - 'a'];
            ch[now][c - 'a'] = x;
        }
        last = ch[now][c - 'a'];
        cnt[last]++;
    }
};
```

#

三、图论

dinic

```
const int V = 1010;
const int E = 101000;
using ll = long long;

template<typename T>
struct MaxFlow
{
    int s, t, vtot;
    int head[V], etot;
    int dis[V], cur[V];
    struct edge
    {
        int v, nxt;
        T f;
    }e[E * 2];
    void addedge(int u, int v, T f)
    {
        e[etot] = {v, head[u], f}; head[u] = etot++;
        e[etot] = {u, head[v], 0}; head[v] = etot++;
    }
    bool bfs()
```

```
    {
        for(int i = 1 ; i <= vtot ; i++ )
        {
            dis[i] = 0;
            cur[i] = head[i];
        }
        queue<int> q;
        q.push(s); dis[s] = 1;
        while(!q.empty())
        {
            int u = q.front(); q.pop();
            for(int i = head[u] ; ~i ; i = e[i].nxt)
            {
                if(e[i].f && !dis[e[i].v])
                {
                    int v = e[i].v;
                    dis[v] = dis[u] + 1;
                    if(v == t) return true;
                    q.push(v);
                }
            }
        }
        return false;
    }
    T dfs(int u, T m)
    {
        if(u == t) return m;
        T flow = 0;
        for(int i = cur[u]; ~i ; cur[u] = i = e[i].nxt)
        {
            if(e[i].f && dis[e[i].v] == dis[u] + 1)
            {
                T f = dfs(e[i].v, min(m, e[i].f));
                e[i].f -= f;
                e[i ^ 1].f += f;
                m -= f;
                flow += f;
                if(!m) break;
            }
        }
        if(!flow) dis[u] = -1;
        return flow;
    }
    T dinic()
    {
        T flow = 0;
        while(bfs()) flow += dfs(s, numeric_limits<T>::max());
        return flow;
    }
    void init(int s_, int t_, int vtot_ )
    {
        s = s_;
        t = t_;
        vtot = vtot_;
        etot = 0;
        for(int i = 1 ; i <= vtot ; i++ )
```

```
        {
            head[i] = -1;
        }
    }
};

MaxFlow<ll> g;
//***记得每次init,

```

费用流

```
const int V = 2010;
const int E = 20100;
// #define int double
using ll = long long;

template<typename T>
struct MaxFlow {
    int s, t, vtot;
    int head[V], etot, cur[V];
    int pre[V];
    bool vis[V];
    T dis[V], cost, flow;

    struct edge {
        int v, nxt;
        T f, c;
    }e[E * 2];

    void addedge(int u, int v, T f, T c, T f2 = 0)
    {
        e[etot] = {v, head[u], f, c}; head[u] = etot++;
        e[etot] = {u, head[v], f2, -c}; head[v] = etot++;
    }

    bool spfa() {
        T inf = numeric_limits<T>::max() / 2;
        for(int i = 1; i <= vtot; i++) {
            dis[i] = inf;
            vis[i] = false;
            pre[i] = -1;
            cur[i] = head[i];
        }
        dis[s] = 0;
        vis[s] = true;
        queue<int> q;
        q.push(s);
        while(!q.empty()) {
            int u = q.front();
            for(int i = head[u]; ~i; i = e[i].nxt) {
                int v = e[i].v;
                if(e[i].f && dis[v] > dis[u] + e[i].c) {
                    dis[v] = dis[u] + e[i].c;
                    pre[v] = i;
```

```
                    if(!vis[v]) {
                        vis[v] = 1;
                        q.push(v);
                    }
                }
            }
            q.pop();
            vis[u] = false;
        }
        return dis[t] < inf;
    }

    void augment() {
        int u = t;
        T f = numeric_limits<T>::max();
        while(~pre[u]) {
            f = min(f, e[pre[u]].f);
            u = e[pre[u] ^ 1].v;
        }
        flow += f;
        cost += f * dis[t];
        u = t;
        while(~pre[u]) {
            e[pre[u]].f -= f;
            e[pre[u] ^ 1].f += f;
            u = e[pre[u] ^ 1].v;
        }
    }

    pair<T, T> sol() {
        flow = cost = 0;
        while(spfa()) {
            augment();
        }
        return {flow, cost};
    }

    void init(int s_, int t_, int vtot_ )
    {
        s = s_;
        t = t_;
        vtot = vtot_;
        etot = 0;
        for(int i = 1 ; i <= vtot ; i++ )
        {
            head[i] = -1;
        }
    }
};

//***记得每次init,
```

二分图最大匹配

```cpp
int a[N];
int v[N], n1, n2;
int to[N], b[N];
int n;
vector<int> e[N];
//n1为左边点数量，n2为右边点数量，v为右边的点连向左边哪条边
bool find(int x)
{
    b[x] = true;
    for(auto y : e[x])
    {
        if(!v[y] || (!b[v[y]] && find(v[y])))
        {
            v[y] = x;
            return true;
        }
    }
    return false;
}

int match()
{
    int ans = 0;
    memset(v, 0 ,sizeof(v));
    for(int i = 1 ; i <= n1 ; i++ )
    {
        memset(b, 0, sizeof(b));
        if(find(i))
        {
            ++ans;
        }
    }
    return ans;
}
```

2—SAT—Tarjan

```cpp
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + f].push_back(2 * v + g);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
```

```
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = std::min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    std::vector<bool> answer() { return ans; }
};
```

SCC hosoraju

```
int vis[N], n, m;
vector<int> out, c, e[N], erev[N];
int sz[N];
int bel[N], cnt;
vector<vector<int> >scc;

void dfs1(int u)
{
    vis[u] = 1;
    for(auto v : e[u])
    {
        if(!vis[v]) dfs1(v);
    }
    out.push_back(u);
}

void dfs2(int u, int cnt)
{

    vis[u] = 1;
    for(auto v : erev[u])
    {
        if(!vis[v]) dfs2(v, cnt);
    }
    bel[u] = cnt;
```

```
    sz[cnt]++;
    c.push_back(u);
}

int main()
{
    fastio
    //freopen("1.in","r",stdin);
    int n, m, x, y;
    cin >> n >> m;
    for(int i = 1 ; i <= m ; i++ )
    {
        cin >> x >> y;
        e[x].push_back(y);
        erev[y].push_back(x);
    }
    memset(vis, 0, sizeof(vis));
    for(int i = 1 ; i <= n ; i++ )
    {
        if(!vis[i])
        {
            dfs1(i);
        }
    }
    reverse(out.begin(), out.end());
    memset(vis, 0, sizeof(vis));
    for(auto u : out)
    {
        if(!vis[u])
        {
            c.clear();
            dfs2(u, ++cnt);
            sort(c.begin(), c.end());
            scc.push_back(c);
        }

    }
    sort(scc.begin(), scc.end());
    for(auto c : scc)
    {
        for(auto x : c)
        {
            cout << x << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

SCC Tarjan

```
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
```

```
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, -1);
        low.resize(n + 1);
        bel.assign(n + 1, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            ++cnt;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
        }
    }

    std::vector<int> work() {
        for (int i = 1; i <= n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

边双连通分量

```cpp
int head[N], e[N], nxt[N], idx = 1, n, m;
int dfn[M], low[M], cnt, b[N], bel[N], anscnt[M];
vector<vector<int> > dcc;
void add(int x, int y)
{
    nxt[++idx] = head[x];
    head[x] = idx;
    e[idx] = y;
}
void tarjan(int x, int e_in)
{
    dfn[x] = low[x] = ++cnt;
    for(int i = head[x] ; i ; i = nxt[i])
    {
        int y = e[i];
        if(!dfn[y])
        {
            tarjan(y, i);
            if(dfn[x] < low[y])
            {
                b[i] = b[i ^ 1] = 1;
            }
            low[x] = min(low[x], low[y]);
        }else if (i != (e_in ^ 1))
        {
            low[x] = min(low[x], dfn[y]);
        }
    }
}

vector<int> v;

void dfs(int x, int cnt)
{
    bel[x] = cnt;
    v.push_back(x);
    anscnt[cnt]++;
    for(int i = head[x] ; i ; i = nxt[i])
    {
        int y = e[i];
        if(bel[y] || b[i]) continue;
        dfs(y, cnt);
    }

}
signed main()
{
    fastio
    //freopen("1.in","r",stdin);
    cin >> n >> m;
    int x, y;
    for(int i = 1 ; i <= m ; i++ )
    {
```

```
        cin >> x >> y;
        if(x == y) continue;
        add(x, y);
        add(y, x);
    }
    for(int i = 1 ; i <= n ; i++ )
    {
        if(!dfn[i]) tarjan(i, 0);
    }
    int ans = 0;
    for(int i = 1 ; i <= n ; i++ )
    {
        if(!bel[i])
        {
            v.clear();
            dfs(i, ++ans);
            dcc.push_back(v);
        }

    }
    int sz = dcc.size();
    cout << dcc.size() << "\n";
    for(int i = 0 ; i < sz ; i++ )
    {
        auto v = dcc[i];
        cout << anscnt[i + 1] << " ";
        for(auto x : v)
        {
            cout << x << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

割点

```
struct CutPoint {
    int n, m, idx;
    std::vector<int> dfn, low, vis, cut;
    std::vector<std::vector<int>> adj;
    CutPoint(int _n, int _m) : n(_n), m(_m), dfn(_n + 1),
    low(_n + 1), vis(_n + 1), cut(_n + 1), adj(_n + 1) {

    }

    void dfs(int x, int root) {
        vis[x] = 1;
        dfn[x] = ++idx;
        low[x] = idx;
        int child = 0;
        for (auto y : adj[x]) {
            if (!vis[y]) {
                dfs(y, root);
                low[x] = std::min(low[x], low[y]);
```

```
                if (low[y] >= dfn[x] && x != root) {
                    cut[x] = 1;
                }
                if (x == root) {
                    child++;
                }
            }
            low[x] = std::min(low[x], dfn[y]);
        }
        if (child >= 2 && x == root) {
            cut[x] = 1;
        }
    }

    std::vector<int> work() {
        std::vector<int> q;
        for (int i = 1; i <= n; i++) {
            if (!vis[i]) {
                dfs(i, i);
            }
        }
        for (int i = 1; i <= n; i++) {
            if (cut[i]) {
                q.push_back(i);
            }
        }
        return q;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }
};
```

割边

```
struct CutEdges {
    int n;
    int idx = 0;
    vector<int> low, dfn, fa;
    vector<int> head, nxt, to;
    vector<int> b;
    int iddx = 1;
    vector<pair<int,int>> bridge;
    CutEdges(int n, int m) : low(n + 1), dfn(n + 1), fa(n + 1),
    head(n + 1), to(2 * m + 4), nxt(2 * m + 4), b(2 * m + 4) {
        this->n = n;
    }
    void addEdge(int x, int y) {
        nxt[++iddx] = head[x];
        head[x] = iddx;
        to[iddx] = y;
    }
    vector<pair<int, int>> work() {
        for (int i = 1; i <= n; i++) {
```

```
            if (!dfn[i]) tarjan(i, 0);
        }
        return bridge;
    }
    void tarjan(int x, int e_in) {;
        dfn[x] = low[x] = ++idx;
        for(int i = head[x]; i; i = nxt[i]) {
            int y = to[i];
            if(!dfn[y]) {
                tarjan(y, i);
                if(dfn[x] < low[y]) {
                    bridge.push_back({x, y});
                    b[i] = b[i ^ 1] = 1;
                }
                low[x] = min(low[x], low[y]);
            } else if (i != (e_in ^ 1)) {
                low[x] = min(low[x], dfn[y]);
            }
        }
    }

};
CutEdges g(n, m);
```

无向图欧拉图

```
vector<pair<int ,int > > e[N];
int d[N], n, m;
int f[N], b[N], sz[N], ans[N], idxans;

void dfs(int x)
{
    //cout << "dfs = " << x << endl;
    for(; f[x] < sz[x] ; )
    {
        int y = e[x][f[x]].first, id = e[x][f[x]].second;
        if(!b[id])
        {
            b[id] = 1;
            f[x]++;
            dfs(y);
            ans[++idxans] = y;
        }else{
            f[x]++;
        }
    }
}

void Euler()
{
    memset(f, 0, sizeof(f));
    memset(b, 0 ,sizeof(b));
    int cnt = 0, x = 0;
```

```
    for(int i = 1 ; i <= n ; i++ )
    {
        if(d[i] & 1)
        {
            cnt++;
            x = i;
        }
    }
    if(!(cnt == 0 || cnt == 2))
    {
        cout << "No\n";
        return;
    }
    for(int i = 1 ; i <= n ; i++ )
    {
        sz[i] = e[i].size();
        if(!x)
            if(d[i])
            {
                x = i;
            }
    }
    dfs(x);
    ans[++idxans] = x;
    if(idxans == m + 1)
    {
        cout << "Yes\n";
    }else{
        cout << "No\n";
    }
}
int main()
{
    fastio
    //freopen("1.in","r",stdin);
    cin >> n >> m;
    int idx = 0;
    for(int i = 1 ; i <= m ; i++ )
    {
        int x, y;
        cin >> x >> y;
        ++idx;
        ++d[x];
        ++d[y];
        e[x].push_back({y, idx});
        e[y].push_back({x, idx});

    }
    Euler();
    return 0;
}
```

有向图欧拉图

```
int n;
```

```cpp
vector<int> e[N];
int ind[N], outd[N], f[N], sz[N], ans[N], idx = 0;

void dfs(int x)
{
    for(; f[x] < sz[x] ;)
    {
        int y = e[x][f[x]];
        f[x]++;
        dfs(y);
        ans[++idx] = y;
    }
}
void Euler()
{
    memset(f, 0, sizeof(f));
    int cntdiff = 0;
    int cntin = 0;
    int x = 0;
    for(int i = 1 ; i <= n ; i++ )
    {
        if(ind[i] != outd[i])
        {
            cntdiff++;
        }
        if(ind[i] + 1 == outd[i])
        {
            cntin++;
            x = i;
        }
    }
    if(!(cntdiff == 2 && cntin == 1 || cntdiff == 0))
    {
        cout << "No\n";
        return;
    }
    for(int i = 1 ; i <= n ; i++ )
    {
        sz[i] = e[i].size();
        //cout << e[i].size();
        if(!x)
        {
            if(ind[i])
            {
                x = i;
            }
        }
    }
    dfs(x);
    ans[++idx]= x;
    if(idx == n + 1)
    {
        cout << "Yes\n";
    }else{
        cout << "No\n";
    }
```

```
    for(int i = idx ; i > 0 ; i--)
    {
        cout << ans[i] << " ";
    }
}
```

笛卡尔树

```
//每个父节点都小于其所有子节点

int a[N], n, l[N], r[N];
int root = 0;

void build()
{
    stack<int> st;
    for(int i = 1 ; i <= n ; i++ )
    {
        int last = 0;
        while(!st.empty() && a[st.top()] > a[i])
        {
            last = st.top();
            st.pop();
        }
        if(!st.empty())
        {
            r[st.top()] = i;
        }else{
            root = i;
        }
        l[i] = last;
        st.push(i);
    }
}
```

dfs序求lca

```
int main()
{
    int idx = 0;
    vector<int> dfn(n + 5);
    vector st(__lg(n) + 2, vector<int> (n + 5));//****不能改成23****
    function<int(int,int)> get = [&](int x, int y)
    {
        return dfn[x] < dfn[y] ? x : y;
    };
    function<void(int,int)> dfs = [&](int x, int fa)
    {
        st[0][dfn[x] = ++idx] = fa;
        for(int y : adj[x]) if(y != fa) dfs(y, x);
    };
    function<int(int,int)> lca = [&](int u, int v)
```

```
    {
        if(u == v) return u;
        if((u = dfn[u]) > (v = dfn[v])) swap(u, v);
        int d = __lg(v - u++);
        return get(st[d][u], st[d][v - (1 << d) + 1]);
    };
    dfs(s, 0);
    for(int i = 1 ; i <= __lg(n) ; i++ )//****不能改成23****
    {
        for(int j = 1 ; j + (1 << i - 1) <= n  ; j++ ) // ****注意边界****
        {
            st[i][j] = get(st[i - 1][j], st[i - 1][j + (1 << i - 1)]);
        }
    }
    /// lca(u, v);
}
```

HLD

```
using i64 = long long;
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n + 1);
        top.resize(n + 1);
        dep.resize(n + 1);
        parent.resize(n + 1);
        in.resize(n + 1);
        out.resize(n + 1);
        seq.resize(n + 1);
        cur = 1;
        adj.assign(n + 1, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 1) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
```

```
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(),
parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    bool isAncester(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }
```

```
92
93      int rootedParent(int u, int v) {
94          std::swap(u, v);
95          if (u == v) {
96              return u;
97          }
98          if (!isAncester(u, v)) {
99              return parent[u];
100         }
101         auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int
    x, int y) {
102             return in[x] < in[y];
103         }) - 1;
104         return *it;
105     }
106
107     int rootedSize(int u, int v) {
108         if (u == v) {
109             return n;
110         }
111         if (!isAncester(v, u)) {
112             return siz[v];
113         }
114         return n - siz[rootedParent(u, v)];
115     }
116
117     int rootedLca(int a, int b, int c) {
118         return lca(a, b) ^ lca(b, c) ^ lca(c, a);
119     }
120 };
```

点分治

```
1  signed main()
2  {
3      fastio
4      int n, k, ans = 0;
5      cin >> n >> k;
6      ans = n + 1;
7      vector<vector<pair<int,int>>> adj(n + 1);
8      vector<int> sz(n + 1, 0), maxsz(n + 1, 0), del(n + 1, 0);
9      vector<int> mark(k + 1, 0), c(k + 1, 0);
10     int T = 1;
11     int u, v, w;
12     for(int i = 1 ; i < n ; i++ )
13     {
14         cin >> u >> v >> w;
15         u++;
16         v++;
17         adj[u].emplace_back(v, w);
18         adj[v].emplace_back(u, w);
19     }
20     function<void(int, int)> solve = [&](int x, int s)
```

```
    {
        T++;
        int mxs = s + 1, root = -1;
        function<void(int, int)> dfs1 = [&](int x, int fx)
        {
            sz[x] = 1;
            maxsz[x] = 0;
            for(auto [y, w] : adj[x])
            {
                if(del[y] || y == fx) continue;
                dfs1(y, x);
                sz[x] += sz[y];
                maxsz[x] = max(maxsz[x], sz[y]);
            }
            maxsz[x] = max(maxsz[x], s - sz[x]);
            if(maxsz[x] < mxs)
            {
                mxs = maxsz[x], root = x;
            }
        };
        dfs1(x, -1);
        ////////////////////////////////
        mark[0] = T;
        c[0] = 0;
        for(auto [y, w] : adj[root])
        {
            if(del[y]) continue;
            vector<pair<int, int>> self;
            function<void(int, int, int, int)> dfs2 = [&](int x, int fx, int
dis, int dep)
            {
                self.emplace_back(dis, dep);
                for(auto [y, w] : adj[x])
                {
                    if(del[y] || y == fx) continue;
                    dfs2(y, x, dis + w, dep + 1);
                }
            };
            dfs2(y, root, w, 1);
            for(auto [dis, dep] : self)
            {
                if(k - dis >= 0 && mark[k - dis] == T)
                {
                    ans = min(ans, c[k - dis] + dep);
                }
            }
            for(auto [dis, dep] : self)
            {
                if(dis > k) continue;
                if(mark[dis] == T)
                {
                    c[dis] = min(c[dis], dep);
                }else{
                    c[dis] = dep;
                    mark[dis] = T;
                }
```

```
            }
        }
        ////////////////////////////////
        del[root] = 1;
        for(auto [y, w] : adj[root])
        {
            if(del[y]) continue;
            solve(y, sz[y]);
        }
    };
    solve(1, n);
    cout << (ans > n ? -1 : ans) << "\n";
    return 0;
}
```

四、数论

exgcd

```
int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return d;
}
```

整除分块

```
for(ll l = 1 ; l <= n ; l++ )
    {
        ll d = n / l, r = n / d;
        cout << l << " : " << r << " = " << d << endl;
        l = r;
    }
```

sieve

```
int tot;
int p[N], pr[N], pe[N];
// p[x]为x最小的质因子, pr[x]为第x个质数, pe[x]为x的最小质因子个数幂

```

```cpp
void sieve(int n) {
    for (int i = 2; i <= n; i++) {
        if (pe[i] == 0) p[i] = i, pe[i] = i, pr[++tot] = i;
        for (int j = 1; j <= tot && i * pr[j] <= n; j++) {
            p[i * pr[j]] = pr[j];
            if (p[i] == pr[j]) {
                pe[i * pr[j]] = pe[i] * pr[j];
                break;
            } else {
                pe[i * pr[j]] = pr[j];
            }
        }
    }
}

void compute(int f[], int n, std::function<int(int)> calcpe) {
    f[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (i == pe[i]) f[i] = calcpe(i);
        else f[i] = f[pe[i]] * f[i / pe[i]];
    }
}
compute(d, n, [&](int x) {
    return d[x / p[x]] + 1;
}); // d

compute(sigma, n, [&](int x) {
    return sigma[x / p[x]] + x;
}); // sigma

compute(phi, [&](int x) {
    phi[x] = x - x / p[x]; // f[x] = x / p[x] * (p[x] - 1);
}); // phi

compute(mu, n, [&](int x) {
    mu[x] = (x == p[x] ? -1 : 0);
}); // mu
```

积性函数

$id(x) = x$

$I(x) = 1(x) = 1$

$e(x) = \begin{cases} 1, x = 1 \\ 0, x \neq 1 \end{cases}$

$d(n) =$ 因子个数

$\sigma(n) =$ 因子和

$\mu(n) = \begin{cases} 1 & , x = 1 \\ (-1)^k & , n\text{不含平方因子，}k\text{为质因子个数} \\ 0 & , n\text{含有平方因子} \end{cases}$

Dirchlet卷积

$h(n)=f*g=\sum_{d|n}f(d)g(\frac{n}{d})=\sum_{d_1d_2=n}f(d_1)g(d_2)$

$d=1*1$

$d(n)=\sum_{d|n}I(d)=\sum_{d|n}I(d)I(\frac{n}{d})$

$\sigma=1*id$

$\sigma(n)=\sum_{d|n}I(d)id(\frac{n}{d})$

$f=f*e$

$f*g=g*f$

$f*(g*h)=(f*g)*h$

f,g是积性函数 $\rightarrow$ $f*g$是积性函数

莫比乌斯反演

$f(n)=\sum_{d|n}g(d)\Leftrightarrow g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})=\sum_{d|n}\mu(\frac{n}{d})f(d)$

$f=g*1\Leftrightarrow g=f*\mu$

$1*\mu=e$

$id*\mu=\phi$

$\phi*1=\mu$

$id=\mu*\sigma$

$n=\sum_{d|n}\phi(d)$

ax-by=1的解

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return d;
}

void solve()
{
    ll a, b;
```

```
    cin >> a >> b;
    ll x, y;
    ll d = exgcd(a, b, x, y);
    y = -y;
    while(x < 0 || y < 0)
    {
        x += b/d;
        y += a/d;
    }
    while(x >= b/d && y >= a/d)
    {
        x -= b/d;
        y -= a/d;
    }
    cout << x << " " << y << "\n";
}
```

pollard_rho

```
using i64 = long long;
using i128 = __int128;
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1;
    for (; b; b >>= 1, a = i128(a) * a % m) {
        if (b & 1) {
            res = i128(res) * a % m;
        }
    }
    return res;
}

bool isprime(i64 p) {
    if (p < 2) {
        return 0;
    }
    i64 d = p - 1, r = 0;
    while (!(d & 1)) {
        r++;
        d >>= 1;
    }
    int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (auto a : prime) {
        if (p == a) {
            return true;
        }
        i64 x = power(a, d, p);
        if (x == 1 || x == p - 1) {
            continue;
        }
        for (int i = 0; i < r - 1; i++) {
            x = i128(x) * x % p;
            if (x == p - 1) {
                break;
            }
        }
```

```
        if (x != p - 1) {
            return false;
        }
    }
    return true;
}

mt19937 rng((unsigned int)
chrono::steady_clock::now().time_since_epoch().count());

i64 pollard_rho(i64 x) {
    i64 s = 0, t = 0;
    i64 c = i64(rng()) % (x - 1) + 1;
    i64 val = 1;
    for (int goal = 1; ; goal <<= 1, s = t, val = 1) {
        for (int step = 1; step <= goal; step++) {
            t = (i128(t) * t + c) % x;
            val = i128(val) * abs(t - s) % x;
            if (step % 127 == 0) {
                i64 g = gcd(val, x);
                if (g > 1) {
                    return g;
                }
            }
        }
        i64 g = gcd(val, x);
        if (g > 1) {
            return g;
        }
    }
}

unordered_map<i64, int> getprimes(i64 x) {
    unordered_map<i64, int> p;
    function<void(i64)> get = [&](i64 x) {
        if (x < 2) {
            return;
        }
        if (isprime(x)) {
            p[x]++;
            return;
        }
        i64 mx = pollard_rho(x);
        get(x / mx);
        get(mx);
    };
    get(x);
    return p;
}

```

fft

```cpp
void fft(vector<complex<double>>&a){
    int n=a.size(),L=31-__builtin_clz(n);
    vector<complex<long double>>R(2,1);
    vector<complex<double>>rt(2,1);
    for(int k=2;k<n;k*=2){
        R.resize(n);
        rt.resize(n);
        auto x=polar(1.0L, acos(-1.0L)/k);
        for(int i=k;i<2*k;++i) rt[i]=R[i]=i&1?R[i/2]*x:R[i/2];
    }
    vector<int>rev(n);
    for(int i=0;i<n;++i) rev[i]=(rev[i/2]|(i&1)<<L)/2;
    for(int i=0;i<n;++i) if(i<rev[i]) swap(a[i],a[rev[i]]);
    for (int k=1;k<n;k*=2)
        for (int i=0;i<n;i+=2*k)
            for(int j=0;j<k;++j){
                complex<double>z=rt[j+k]*a[i+j+k];
                a[i+j+k]=a[i+j]-z;
                a[i+j]+=z;
            }
}

vector<double>mul(const vector<double>&a,const vector<double>&b){
    if(a.empty() || b.empty()) return {};
    vector<double>res(a.size()+b.size()-1);
    int L=32-__builtin_clz(res.size()),n=1<<L;
    vector<complex<double>>in(n),out(n);
    copy(a.begin(),a.end(),in.begin());
    for(int i=0;i<b.size();++i) in[i].imag(b[i]);
    fft(in);
    for(auto &x:in) x*=x;
    for(int i=0;i<n;++i) out[i]=in[-i&(n-1)]-conj(in[i]);
    fft(out);
    for(int i=0;i<res.size();++i) res[i]=imag(out[i])/(4 * n);
    return res;
}
```

ntt

```cpp
int mod=998244353;

int qpow(int a,int b){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
    }
    return ans;
}

vector<int>roots{0,1};
vector<int>rev;

void dft(vector<int>&a){
    int n=a.size();
```

```
    if(rev.size()!=n){
        rev.resize(n);
        int k=__builtin_ctzll(n)-1;
        for(int i=0;i<n;++i) rev[i]=rev[i>>1]>>1|(i&1)<<k;
    }
    for(int i=0;i<n;++i) if(i<rev[i]) swap(a[i],a[rev[i]]);
    if(roots.size()<n){
        int k=__builtin_ctzll(roots.size());
        roots.resize(n);
        while((1<<k)<n){
            int e=qpow(3,(mod-1)>>(k+1));
            for(int i=(1<<(k-1));i<(1<<k);++i){
                roots[2*i]=roots[i];
                roots[2*i+1]=roots[i]*e%mod;
            }
            k++;
        }
    }
    for(int k=1;k<n;k<<=1){
        for(int i=0;i<n;i+=2*k){
            for(int j=0;j<k;++j){
                int u=a[i+j],v=a[i+j+k]*roots[k+j]%mod;
                a[i+j]=(u+v)%mod;
                a[i+j+k]=(u-v+mod)%mod;
            }
        }
    }
}
void idft(vector<int>&a){
    reverse(a.begin()+1,a.end());
    dft(a);
    int n=a.size(),inv=(1-mod)/n+mod;
    for(int i=0;i<n;++i) a[i]=a[i]*inv%mod;
}

struct Poly{
    vector<int>a;
    friend Poly operator*(Poly a,Poly b){
        int sz=1,tot=a.a.size()+b.a.size()-1;
        while(sz<tot) sz<<=1;
        a.a.resize(sz);b.a.resize(sz);
        dft(a.a);dft(b.a);
        for(int i=0;i<sz;++i) a.a[i]=a.a[i]*b.a[i]%mod;
        idft(a.a);
        a.a.resize(tot);
        return a;
    }
};
```

拉格朗日单点求值

```
int Lagrange(vector<int>x,vector<int>y,int n,int k){
    for(int i=0;i<n;++i) if(x[i]==k) return y[i];
    vector<int>inv(n,1ll);
    for(int i=0;i<n;++i){
```

```
        for(int j=i+1;j<n;++j){
            inv[i]=inv[i]*(x[i]-x[j]+MOD)%MOD;
            inv[j]=inv[j]*(x[j]-x[i]+MOD)%MOD;
        }
    }
    int sum=1,ans=0;
    for(int i=0;i<n;++i) sum=sum*(k-x[i]+MOD)%MOD;
    for(int i=0;i<n;++i){
        int tmp=inv[i]*(k-x[i]+MOD)%MOD;
        ans=(ans+y[i]*sum%MOD*ksm(tmp,MOD-2)%MOD)%MOD;
    }
    return ans;
}
```

拉格朗日多点插值

```
vector<Poly>Q;

Poly MulT(Poly a,Poly b){
    int n=a.size(),m=b.size();
    reverse(b.a.begin(),b.a.end());
    b=a*b;
    for(int i=0;i<n;++i) a[i]=b[i+m-1];
    return a;
}

void MPinit(Poly &a,int u,int cl,int cr){
    if(cl==cr){
        Q[u].resize(2);
        Q[u][0]=1,Q[u][1]=mod-a[cl];
        return;
    }
    int mid=cl+cr>>1;
    MPinit(a,u<<1,cl,mid);MPinit(a,u<<1|1,mid+1,cr);
    Q[u]=Q[u<<1]*Q[u<<1|1];
}

void MPcal(int u,int cl,int cr,Poly f,Poly &g){
    f.resize(cr-cl+1);
    if(cl==cr){
        g[cl]=f[0];
        return;
    }
    int mid=cl+cr>>1;
    MPcal(u<<1,cl,mid,MulT(f,Q[u<<1|1]),g);
    MPcal(u<<1|1,mid+1,cr,MulT(f,Q[u<<1]),g);
}

Poly Multipoints(Poly f,Poly a,int n){                  //n为f和a的最大长度
    f.resize(n+1),a.resize(n);
    Poly v(n);
    Q.resize(n<<2);
    MPinit(a,1,0,n-1);
    MPcal(1,0,n-1,MulT(f,Q[1].inv(n+1)),v);
    return v;
```

```
}
```

五、数据结构

ST表

```
for(int i = 1 ; i <= n ; i++ )
{
    a[i] = read();
    f[0][i] = a[i];
}
for(int i = 1 ; i <= 22 ; i++ )
{
    for(int j = 1 ; j + (1 << i) - 1 <= n ; j++ )
    {
        f[i][j] = max(f[i-1][j], f[i-1][j + (1 << i - 1)]);
    }
}
for(int i = 1 ; i <= m ; i++ )
{
    int l = read(), r = read();
    int len = __lg(r - l + 1);
    printf("%d\n", max(f[len][l], f[len][r - (1 << len) + 1]));
}
```

树状数组

```
template<class T>
struct BIT {
    int size = 1;
    std::vector<T> c;
    BIT (int x) {
        size = x + 5;
        c.resize(x + 5);
    }
    void init(int x) {
        size = x + 5;
        c.resize(x + 5);
    }
    void clear() {
        for(int i = 0; i < size; i++) {
            c[i] = 0;
        }
    }
    void change(int x, T y) {
        for(; x < size ; x += x & (-x)) {
            c[x] += y;
        }
    }
    T query(int x) {
        T s = 0;
```

```
        for(;x ;x -= x & (-x)) {
            s += c[x];
        }
        return s;
    }
    T query(int l, int r) {
        if (l == 0) return query(r);
        return query(r) - query(l - 1);
    }
    int kth(int k) {
        int sum = 0, x = 0;
        for (int i = log2(size); ~i; --i) {
            x += 1 << i;
            if (x >= size || sum + c[x] >= k)
                x -= 1 << i;
            else
                sum += c[x];
        }
        return x + 1;
    }
};
```

并查集

```
struct DSU {
    std::vector<int> f, siz;
    DSU(int n) : f(n), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }
    int leader(int x) {
        while (x != f[x]) x = f[x] = f[f[x]];
        return x;
    }
    bool same(int x, int y) { return leader(x) == leader(y); }
    bool merge(int x, int y) {
        x = leader(x);
        y = leader(y);
        if (x == y) return false;
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) { return siz[leader(x)]; }
};

struct DSU {
    std::vector<int> parent, siz;
    std::vector<std::array<int, 5> > stk;
    DSU(int n) : parent(n + 1), siz(n + 1, 1) {
        std::iota(parent.begin(), parent.end(), 0);
    }

    int leader(int x) {
        while (x != parent[x]) {
            x = parent[x];
        }
        return x;
```

```
    }

    bool merge(int x, int y, int t) {
        x = leader(x), y = leader(y);
        if (x == y) return false;
        if (siz[x] < siz[y]) {
            std::swap(x, y);
        }

        stk.push_back({t, x, siz[x], y, siz[y]});
        siz[x] += siz[y];
        parent[y] = x;
        return true;
    }

    void undo(int t) {
        while (stk.size() && stk.back()[0] > t) {
            auto &[_, x, sx, y, sy] = stk.back();
            siz[x] = sx;
            parent[x] = x;
            siz[y] = sy;
            parent[y] = y;
            stk.pop_back();
        }
    }

};
```

二维树状数组维护区间查询，修改

```
ll c1[N][N], c2[N][N], c3[N][N], c4[N][N];

int n, m, k, q;

int lowbit(int x)
{
    return x & (-x);
}

void add(ll x, ll y, ll d)
{
    for(int i = x ; i <= n ; i += lowbit(i))
    {
        for(int j = y ; j <= m ; j += lowbit(j))
        {
            //cout << "test" << endl;
            c1[i][j] += d;
            c2[i][j] += d * x;
            c3[i][j] += d * y;
            c4[i][j] += d * x * y;
        }
    }
}

void modify(int x1, int y1, int x2, int y2, int d)
```

```
{
    add(x1, y1, d);
    add(x1, y2 + 1, -d);
    add(x2 + 1, y1, -d);
    add(x2 + 1, y2 + 1, d);
}

ll sum(ll x, ll y)
{
    ll ans = 0;
    for(int i = x ; i ; i -= lowbit(i))
    {
        for(int j = y ; j ; j -= lowbit(j))
        {
            ans += (x + 1) * (y + 1) * c1[i][j];
            ans -= (y + 1) * c2[i][j];
            ans -= (x + 1) * c3[i][j];
            ans += c4[i][j];
        }
    }
    return ans;
}
ll query(int x1, int y1, int x2, int y2)
{
    return (sum(x2, y2) - sum(x1 - 1, y2) - sum(x2, y1 - 1) + sum(x1 - 1, y1 - 1));
}
int h[100005];
int main()
{
    fastio
    //freopen("1.in","r",stdin);
    cin >> n >> m >> k >> q;
    for(int i = 1 ; i <= k ; i++ )
    {
        cin >> h[i];
    }
    for(int i = 1 ; i <= q ; i++ )
    {
        int op;
        cin >> op;
        if(op == 1)
        {
            int a, b, c, d, id;
            cin >> a >> b >> c >> d >> id;
            modify(a, b, c, d, h[id]);
        }else{
            int a, b, c, d;
            cin >> a >> b >> c >> d;
            cout << query(a, b, c, d) << "\n";
        }
    }
    return 0;
}

```

SegmentTree

```cpp
struct Info {

};

Info operator+(const Info &a, const Info &b){

}

template<class Info>
struct SegmentTree{
    int n;
    vector<Info> info;

    SegmentTree() {}

    SegmentTree(int n, Info _init = Info()){
        init(vector<Info>(n, _init));
    }

    SegmentTree(const vector<Info> &_init){
        init(_init);
    }

    void init(const vector<Info> &_init){
        n = (int)_init.size();
        info.assign((n << 2) + 1, Info());
        function<void(int, int, int)> build = [&](int p, int l, int r){
            if (l == r){
                info[p] = _init[l - 1];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m + 1, r);
            pull(p);
        };
        build(1, 1, n);
    }

    void pull(int p){
        info[p] = info[2 * p] + info[2 * p + 1];
    }

    void modify(int p, int l, int r, int x, Info v){
        if (l == r){
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x <= m){
            modify(2 * p, l, m, x, v);
        }
        else{
```

```
            modify(2 * p + 1, m + 1, r, x, v);
        }
        pull(p);
    }

    void modify(int p, Info v){
        modify(1, 1, n, p, v);
    }

    Info query(int p, int l, int r, int x, int y){
        if (l > y || r < x){
            return Info();
        }
        if (l >= x && r <= y){
            return info[p];
        }
        int m = (l + r) / 2;
        return query(2 * p, l, m, x, y) + query(2 * p + 1, m + 1, r, x, y);
    }

    Info query(int l, int r){
        return query(1, 1, n, l, r);
    }
};
```

LazySegmentTree

```
struct Info {
    ll sum = 0, len = 0;
};

struct Tag {
    ll add = 0;
};

Info operator+(const Info &a, const Info &b){
    return {a.sum + b.sum, a.len + b.len};
}

void apply(Info &x, Tag &a, Tag f){
    x.sum += x.len * f.add;
    a.add += f.add;
}

template<class Info, class Tag>
struct LazySegmentTree{
    int n;
    vector<Info> info;
    vector<Tag> tag;

    LazySegmentTree() {}

    LazySegmentTree(int n, Info _init = Info()){
```

```
        init(vector<Info>(n, _init));
    }

    LazySegmentTree(const vector<Info> &_init){
        init(_init);
    }

    void init(const vector<Info> &_init){
        n = (int)_init.size() - 1;
        info.assign((n << 2) + 1, Info());
        tag.assign((n << 2) + 1, Tag());
        function<void(int, int, int)> build = [&](int p, int l, int r){
            if (l == r){
                info[p] = _init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m + 1, r);
            pull(p);
        };
        build(1, 1, n);
    }

    void pull(int p){
        info[p] = info[2 * p] + info[2 * p + 1];
    }

    void apply(int p, const Tag &v){
        ::apply(info[p], tag[p], v);
    }

    void push(int p){
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }

    void modify(int p, int l, int r, int x, const Info &v){
        if (l == r){
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x <= m){
            modify(2 * p, l, m, x, v);
        }
        else{
            modify(2 * p + 1, m + 1, r, x, v);
        }
        pull(p);
    }

    void modify(int p, const Info &v){
        modify(1, 1, n, p, v);
```

```
    }

    Info query(int p, int l, int r, int x, int y){
        if (l > y || r < x){
            return Info();
        }
        if (l >= x && r <= y){
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return query(2 * p, l, m, x, y) + query(2 * p + 1, m + 1, r, x, y);
    }

    Info query(int l, int r){
        return query(1, 1, n, l, r);
    }

    void modify(int p, int l, int r, int x, int y, const Tag &v){
        if (l > y || r < x){
            return;
        }
        if (l >= x && r <= y){
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        modify(2 * p, l, m, x, y, v);
        modify(2 * p + 1, m + 1, r, x, y, v);
        pull(p);
    }

    void modify(int l, int r, const Tag &v){
        return modify(1, 1, n, l, r, v);
    }
};

```

DynamicSegmentTree

```
class SegTree {
private:
    struct Node {
        Node () : left_(nullptr), right_(nullptr), val_(0), lazy_(0) {}
        int val_;
        int lazy_;
        Node* left_;
        Node* right_;
    };

public:
    Node* root_;
```

```
    SegTree() { root_ = new Node(); }
    ~SegTree() {}

    // 更新区间值
    void upDate(Node* curNode, int curLeft, int curRight, int upDateLeft, int upDateRight, int addVal) {
        if (upDateLeft <= curLeft && upDateRight >= curRight) {
            // 如果需要更新的区间[upDateLeft, upDateRight] 包含了 当前这个区间[curLeft, curRight]
            // 那么暂存一下更新的值
            // 等到什么时候用到孩子结点了，再把更新的值发放给孩子
            curNode->val_ += addVal * (curRight - curLeft + 1);
            curNode->lazy_ += addVal;
            return;
        }

        // 到这里说明要用到左右孩子了
        // 因此，要用pushDown函数把懒标签的值传递下去
        int mid = (curLeft + curRight) / 2;
        pushDown(curNode, mid - curLeft + 1, curRight - mid);

        // 说明在[curLeft, curRight]中，
        if (upDateLeft <= mid) {
            upDate(curNode->left_, curLeft, mid, upDateLeft, upDateRight, addVal);
        }
        if (upDateRight > mid) {
            upDate(curNode->right_, mid + 1, curRight, upDateLeft, upDateRight, addVal);
        }

        // 更新了子节点还需要更新现在的结点
        pushUp(curNode);
    }


    // 把结点curNode的懒标记分发给左右孩子 然后自己的懒标记清零
    void pushDown(Node* curNode, int leftChildNum, int rightChildNum) {
        if (curNode->left_ == nullptr) curNode->left_ = new Node;
        if (curNode->right_ == nullptr) curNode->right_ = new Node;

        if (curNode->lazy_ == 0) return;

        curNode->left_->val_ += curNode->lazy_ * leftChildNum;
        curNode->left_->lazy_ += curNode->lazy_;

        curNode->right_->val_ += curNode->lazy_ * rightChildNum;
        curNode->right_->lazy_ += curNode->lazy_;

        curNode->lazy_ = 0;

        // 注意不需要递归再继续下推懒标签
        // 每次只需要推一层即可
    }

    // 一般是子节点因为要被用到了，所以需要更新值 因此也要同时更新父节点的值
```

```
    void pushUp(Node* curNode) {
        curNode->val_ = curNode->left_->val_ + curNode->right_->val_;
    }

    // 查询
    int query(Node* curNode, int curLeft, int curRight, int queryLeft, int queryRight) {
        if (queryLeft <= curLeft && queryRight >= curRight) {
            return curNode->val_;
        }
        // 用到左右结点力 先下推!
        int mid = (curLeft + curRight) / 2;
        pushDown(curNode, mid - curLeft + 1, curRight - mid);

        int curSum = 0;
        if (queryLeft <= mid) curSum += query(curNode->left_, curLeft, mid, queryLeft, queryRight);
        if (queryRight > mid) curSum += query(curNode->right_, mid + 1, curRight, queryLeft, queryRight);

        return curSum;
    }
};
```

PersistentSegmentTree

```
struct Info {
    int sum = 0;
};

Info operator+(const Info &a, const Info &b) {
    return {a.sum + b.sum};
}

struct PersistentSegmentTree {
    vector<Info> tr;
    vector<Info> a;
    vector<int> ls, rs;
    int n, idx = 1;
    PersistentSegmentTree(int _n) {
        this->n = _n;
        this->a = a;
        ls.resize(_n << 5);
        rs.resize(_n << 5);
        tr.resize(_n << 5);
        a.assign(_n + 1, {0});
        build(1, 1, _n);
    }
    void build(int u, int L, int R) {
        // test(u, L, R);
        if (L == R) {
            tr[u] = a[L];
            return;
        }
        int mid = L + R >> 1;
```

```
        if (!ls[u]) {
            ls[u] = ++idx;
        }
        if (!rs[u]) {
            rs[u] = ++idx;
        }
        build(ls[u], L, mid);
        build(rs[u], mid + 1, R);
    }
    int modify(int u, int L, int R, int p, int x) {
        if (L == R && p == L) {
            tr[++idx] = {tr[u].sum + 1};
            return idx;
        }
        int mid = L + R >> 1;
        if (p <= mid) {
            int id = modify(ls[u], L, mid, p, x);
            ls[++idx] = id;
            rs[idx] = rs[u];
            tr[idx] = tr[ls[idx]] + tr[rs[idx]];
            return idx;
        } else {
            int id = modify(rs[u], mid + 1, R, p, x);
            rs[++idx] = id;
            ls[idx] = ls[u];
            tr[idx] = tr[ls[idx]] + tr[rs[idx]];
            return idx;
        }
    }
    int query(int u, int v, int L, int R, int k) {
        if (L == R) return L;
        int x = tr[ls[v]].sum - tr[ls[u]].sum;
        int mid = L + R >> 1;
        if (x >= k) {
            return query(ls[u], ls[v], L, mid, k);
        } else {
            return query(rs[u], rs[v], mid + 1, R, k - x);
        }
    }
};
```

pbds

```
#include<ext/pb_ds/tree_policy.hpp>
#include<ext/pb_ds/assoc_container.hpp>

using namespace __gnu_pbds;
__gnu_pbds::tree<pair<ll,ll>, null_type, less<pair<ll,ll>>, rb_tree_tag,
tree_order_statistics_node_update> T;

if(op == 1)
{
    T.insert({x, i});
```

```
}else if (op == 2)
{
    T.erase(T.lower_bound({x, 0}));
}else if (op == 3)
{
    cout << T.order_of_key({x, 0}) + 1 << "\n";
}else if (op == 4)
{
    cout << T.find_by_order(x - 1)->first << "\n";
}else if (op == 5)
{
    cout << prev(T.lower_bound({x, 0}))->first << "\n";
}else if (op == 6)
{
    cout << T.lower_bound({x + 1, 0})->first << "\n";
}


```

bitset

```
#include<tr2/dynamic_bitset>
using std::tr2::dynamic_bitset;
dynamic_bitset< >bt(n+1);
_Find_fisrt就是找到从低位到高位第一个1的位置
_Find_next就是找到当前位置的下一个1的位置
all/any/none checks if all, any or none of the bits are set to true
count   returns the number of bits set to true
set 1
reset 0
filp
```

六、简单计算几何

点

```
using i64 = long long;

using T = double;
struct Point {
    T x;
    T y;
    Point(T x = 0, T y = 0) : x(x), y(y) {}

    Point &operator+=(const Point &p) {
        x += p.x, y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) {
        x -= p.x, y -= p.y;
        return *this;
```

```
    }
    Point &operator*=(const T &v) {
        x *= v, y *= v;
        return *this;
    }
    friend Point operator-(const Point &p) {
        return Point(-p.x, -p.y);
    }
    friend Point operator+(Point lhs, const Point &rhs) {
        return lhs += rhs;
    }
    friend Point operator-(Point lhs, const Point &rhs) {
        return lhs -= rhs;
    }
    friend Point operator*(Point lhs, const T &rhs) {
        return lhs *= rhs;
    }
};

T dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}

T cross(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}
```

七、杂项

矩阵快速幂

```
struct Matrix{
    int n , m ;
    vector<vector<ll>> s;

    Matrix(int n , int m):n(n) ,m(m) , s(n , vector<ll>(m ,0)){}

    friend Matrix operator * (Matrix a , Matrix b){
        assert(a.m == b.n);
        Matrix res(a.n , b.m);
        for(int k = 0 ; k < a.m ; k ++ )
            for(int i = 0 ; i < a.n ; i ++ )
                for(int j = 0 ; j < b.m ; j ++ )
                    res.s[i][j] = (res.s[i][j] + a.s[i][k] * b.s[k][j] %
mod) % mod;
        return res;
    }

    Matrix qmi(ll b){
        assert(n == m);
        Matrix res(n , n);
        for(int i = 0 ; i < n ; i ++ )
            res.s[i][i] = 1;
        while(b){
```

```
            if(b & 1)res = ((*this) * res );
            b >>= 1;
            *this = (*this) * (*this);
        }
        return (*this) = res;
    };

};
```

组合数

```
ll fact[N] = {1}, inv[N] = {1};
ll C(ll x, ll y)
{
    return(((fact[x] * inv[y])% MOD * inv[x-y]) % MOD);
}

ll P(ll x, ll y)
{
    return fact[x] * inv[x - y] % MOD;
}

ll ksm(ll x, ll y)
{
    ll ans = 1;
    x %= MOD;
    while(y)
    {
        if(y&1)
        {
            ans = ans * x % MOD;
        }
        x = x * x % MOD;
        y /= 2;
    }
    return ans;
}

void build()
{
    for(int i = 1 ; i < N ; i++ )
    {
        fact[i] = fact[i-1] * i % MOD;
    }
    for(int i = 1 ; i < N ; i++ )
    {
        inv[i] = inv[i-1] * ksm(i, MOD-2) % MOD;
    }
}
```

八、python

```
'''
```

```
def main():
    Do somthing
if __name__ == '__main__':
    t = int(input())
    for i in range(t):
        main()
'''
for T in range(0,int(input())): #T组数据
    N=int(input())
    n,m=map(int,input().split())
    s=input()
    s=[int(x) for x in input().split()] #一行输入的数组
    a[1:]=[int(x) for x in input().split()] #从下标1开始读入一行
    for i in range(0,len(s)):
        a,b=map(int,input().split())

while True: #未知多组数据
    try:
        #n,m=map(int,input().split())
        #print(n+m,end="\n")
    except EOFError: #捕获到异常
        break
//////////////////////
'''多行输入，指定行数'''

n, m = map(int, input().strip().split())#获取第一行，获取第二行可以再写一句同样的语句
#需要矩阵承接数据时
data = []
for i in range(n):
    tmp = list(map(int, input().split()))
    data.append(tmp)

'''多行输入，不指定行数'''
try:
    data = []
    while True:
        line = input().strip() #strip去除左右两边的空白符
        if line == ' ':
            break
        tmp = list(map(int, line.split())) #split按空白符拆开
        data.append(tmp)
expect:
    pass

```

一些基本数据结构

python中的栈和队列可以使用列表来模拟，或者`import deque`

匿名函数使用lambda关键字来定义`lambda 参数: 表达式`

```
#使用中括号[]定义一个列表
# l=[23,'wtf',3.14]
list.append(obj)#将obj添加到list末尾,O(1)
list.insert(index,obj)#将obj插入列表index位置,O(n)
```

```
list.pop([index=-1])#移除元素并返回该元素
list.sort(key=None,reverse=False)#默认升序排序,O(nlogn)
list.reverse()#反转列表元素
list.clear()
len(list)#列表元素个数,O(1)
max(list)#返回列表元素最大值,O(n)
del list[2]#删除list中第三个元素

#用小括号定义一个元组,可以当作不能修改的list
# t=(23,'wtf',3.14)

#用花括号{}定义一个字典
d={key1:value1,key2:value2}#通过key访问value
print(d[key1])#输出value1
if key in dict : #key不存在会报错,要先询问
    do somthing #或者使用
d.get(key)
for key in d: #遍历字典d
    print(key,':',d.get(key))
dMerge=dict(d1,**d2)#将d1和d2合并为dMerge

#调用set()方法创建集合
s=set([1,2,3])#定义
s.add(4)#添加
s.remove(4)#删除
```

math库

```
import math
math.e #常量e,2.718281828459045
math.pi #常量pi,3.141592653589793
math.factorial(x) #x的阶乘
math.gcd(x,y) #x,y的gcd
math.sqrt(x) #x的平方根
x=math.log(n,a) #以a为底n的对数x,a^x=n,默认底数为e
math.log(32,2) #5.0
math.degrees(math.pi/4) #将π/4转为角度
math.radians(45) #将45度转为弧度
math.cos(math.pi/4) #参数都为弧度
```

快速幂

```
def qmod(a,b,mod):
    a=a%mod
    ans=1
    while b!=0:
        if b&1:
            ans=(ans*a)%mod
        b>>=1
        a=(a*a)%mod
    return ans
```

并查集

```python
N,m=map(int,input().split())
fa=[int(i) for i in range(N+1)]
siz=[1]*(N+1)
def findfa(x):
    if fa[x]!=x:
        fa[x]=findfa(fa[x])
    return fa[x]
def Merge(x,y):
    xx,yy=findfa(x),findfa(y)
    if xx == yy:
        return False
    if siz[xx] > siz[yy]: #按秩合并
        fa[yy]=xx
        siz[xx]+=siz[yy]
    else:
        fa[xx]=yy
        siz[yy]+=siz[xx]
    return True
for i in range(m):
    z,x,y=map(int,input().split())
    if z==1:
        Merge(x,y)
    else:
        print('Y' if findfa(x)==findfa(y)else 'N')
```

线段树区间加区间和

```python
class SegTreeNode(): #python3中所有类默认都是新式类
    def __init__(self): #类似构造函数,类方法必须包含参数self
        self.value=0
        self.lazytag=0

Data=[0 for i in range(0,100010)]

class SegTree():
    def __init__(self):
        self.SegTree=[SegTreeNode() for i in range(0,400010)]

    def Build_SegTree(self,Root,L,R):
        if L==R:
            self.SegTree[Root].value=Data[L]
            return
        mid=(L+R)>>1
        self.Build_SegTree(Root<<1,L,mid)
        self.Build_SegTree(Root<<1|1,mid+1,R)

 self.SegTree[Root].value=self.SegTree[Root<<1].value+self.SegTree[Root<<1|1
].value
        return

    def Push_Down(self,Root,L,R):
        if self.SegTree[Root].lazytag==0:
```

```
            return
        Add=self.SegTree[Root].lazytag
        self.SegTree[Root].lazytag=0
        mid=(L+R)>>1
        self.SegTree[Root<<1].value+=(mid-L+1)*Add
        self.SegTree[Root<<1|1].value+=(R-mid)*Add
        self.SegTree[Root<<1].lazytag+=Add
        self.SegTree[Root<<1|1].lazytag+=Add
        return

    def Update(self,Root,L,R,QL,QR,Add):
        if R<QL or QR<L:
            return
        if QL<=L and R<=QR:
            self.SegTree[Root].value+=(R-L+1)*Add
            self.SegTree[Root].lazytag+=Add
            return
        mid=(L+R)>>1
        self.Push_Down(Root,L,R)
        self.Update(Root<<1,L,mid,QL,QR,Add)
        self.Update(Root<<1|1,mid+1,R,QL,QR,Add)

 self.SegTree[Root].value=self.SegTree[Root<<1].value+self.SegTree[Root<<1|1
].value
        return

    def Query(self,Root,L,R,QL,QR):
        if R<QL or QR<L:
            return 0
        if QL<=L and R<=QR:
            return self.SegTree[Root].value
        mid=(L+R)>>1
        self.Push_Down(Root,L,R)
        return
self.Query(Root<<1,L,mid,QL,QR)+self.Query(Root<<1|1,mid+1,R,QL,QR)

Tree=SegTree()
N,M=map(int,input().split())
a=input().split() #初始值

for i in range(1,N+1):
    Data[i]=int(a[i-1])

Tree.Build_SegTree(1,1,N)

while M:
    opt,L,R=map(int,input().split())
    if opt==1:
        Tree.Update(1,1,N,L,R,int(a[3]))
    else:
        print(str(Tree.Query(1,1,N,L,R)))
    M-=1
```

字符串

```python
ord('a')# 返回单个字符的 unicode:
chr(100)# 返回'd'

#strip和split
'   spacious   '.strip()#strip()移除 string 前后的字符串，默认来移除空格
'1,2,3'.split(',') #['1', '2', '3'],按照某个字符串来切分，返回一个 list,
'1,2,3'.split(',', maxsplit=1)#['1', '2,3'],传入一个参数maxsplit来限定分离数

#将字符串和列表相互转换
字符串转换成列表，注意交换字符需要先转换成列表
#1.list
str1 = '12345'
list1 = list(str1)
print(list1) #['1', '2', '3', '4', '5']
#2.str.split()通过指定分隔符对字符串进行切片
str3 = 'this is string example'
list3 = str3.split('i', 1)
print(list3) #['th', 's is string example']

列表转换成字符串，join里面的可以是list、set
#1.split.join(str),split是指定的分隔符，str是要转换的字符串
list1 = ['1', '2', '3', '4', '5']
str1 = "".join(list1)#12345

list3 = ['www', 'baidu', 'com']
str3 = ".".join(list3)#www.baidu.com

#是元音
def isVowel(ch:str) -> bool:
            return ch in "aeiouAEIOU"
isVowel(s[i])

```

二维列表

```python
ls = [] #二维列表新建可以直接建一个一维列表，后面直接append列表数据就可以了
ls_T = list(map(list, zip(*ls)))# 转置，用于取列元素
if 元素 in ls_T[0]: #判断是不是在0列里面
    j = ls_T[0].index(元素) #第0列中该元素的位置，即多少行
```

list

```python
#初始化
l = [0] * len(array)
l=[]

#从后往前访问
l[-1]表示最后一个数
for i in range(0, -10, -1)     #0, -1, -2, -3, -4, -5, -6, -7, -8, -9
for j in reversed(range(len(nums)-1)) #加一个reverse可以直接颠倒

#enumerate 枚举
l = ["a", "b", "c"]
for i, v in enumerate(l):
```

```
    print(i, v)
#0 a
#1 b
#2 c

#map
#可以将参数一一映射来计算， 比如
date = "2019-8-15"
Y, M, D = map(int, date.split('-'))    #Y = 2019, M = 8, D = 15
#map返回的是迭代对象而不是一个列表，要转成列表要加list


#sort
1.调用sort()排序，不会产生新的列表。lst.sort()升序排序
降序排序lst.sort(reverse=True)  升序排序lst.sort()
2.使用内置函数sorted()排序，会产生新的列表对象
lst1=sorted(lst)升序排序   lst2=sorted(lst,reverse=True)降序排序
l1 = [(1,2), (0,1), (3,10) ]
l2 = sorted(l1, key=lambda x: x[0])#按照 tuple 的第一个元素进行排序key允许传入一个自定义参数
# l2 = [(0, 1), (1, 2), (3, 10)]
#排序默认从小到大。可以用reverse=True倒序

#列表生成式
lst = [i*j for i in range(1,10)]
#ZIP
x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)
list(zipped)#[(1, 4), (2, 5), (3, 6)]
```keys(), values(), items()
这三个方法可以分别获得key, value, {key: value}的数组。

#max可以代替if来更新更大的数
maxnums=max(maxnums,tmp)

#多维数组
res = [[], []]
res[0].append()

#extend一次性添加多个元素
lst1.extend(lst2)
#insert在i位置添加x
lst.insert(i, x)

```
```

常用函数

```
round(x)：四舍五入
abs(x)/max()/min()：绝对值/最大值/最小值
range(start=0, stop, step=1])：返回一个可迭代对象，常用于for循环
pow(x, y, [z])：求幂函数x^y，运算完毕可以顺带对z取模
sorted(iterable, key, reverse)：采用Timsort的稳定排序算法，默认升序
int(x, base=10))/float()/str()：转整数(可自定义进制)/转浮点数/转字符串
bin()/oct()/hex()：10进制转二进制(返回0b开头的字符串)/10进制转八进制(返回0开头的字符串)/10进制转十六进制(返回0x开头的字符串)
ord()/chr()：字符转ASCII或ASCII转字符
math.gcd(x,y)：返回x和y的最大公约数

if ……elif……else注意不要用else if
```

验证数据

最大流(dinic)

```
signed main()
{
    mt19937 rand(0);
    for (int i = 1; i <= 20; i++) {
        int n = rand() % 100, m = rand() % (n * n);
        int s = n + 1, t = n + 2;
        g.init(s, t, t);
        for (int i = 1; i <= m; i++) {
            int u, v, w;
            u = rand() % (n + 2) + 1;
            v = rand() % (n + 2) + 1;
            w = rand() % n;
            g.addedge(u, v, w);//u -> v单向边
        }
        cout << g.dinic() << " ";
    }
}
```

```
722 12 377 500 1240 412 460 550 95 2039 523 0 40 1655 877 221 3562 100 0 2528
```

最小费用最大流

```
mt19937 rand(0);
for (int i = 1; i <= 20; i++) {
    int n = rand() % 100 + 2;
    int m = rand() % (n * n) + 1;
    int s = n - 1, t = n;
    g.init(s, t, t);
    for (int i = 1; i <= m; i++) {
        int u, v, w, c;
        u = rand() % n + 1;
        v = rand() % n + 1;
```

```
        w = rand() % 1000;
        c = rand() % 1000;
        g.addedge(u, v, w, c);
    }
    auto [ans1, ans2] = g.sol();
    cout << ans1 << " " << ans2 << "\n";

}
```

```
15932 14704703
2209 2617444
21746 22827734
13113 14083600
21734 21946209
28796 27196768
3776 4579568
28384 29502294
23196 24861190
0 0
1288 1898029
1025 1127660
2807 1738067
36782 38352187
624 1922442
9168 10702007
10849 10835609
3154 4430069
8088 8840656
32961 31591050
```

Splay

```
mt19937 rand(0);
int n = 1000;
insert(1e18);
insert(-1e18);
int ans1 = 0, ans2 = 0;
while(n--) {
    int x = rand() % 6 + 1;
    int y = rand() % 10000;
    if(x == 1) {
        insert(y);
    }
    if(x == 2) {
        del(y);
    }
    if(x == 3) {
        int tmp = get_rank(y);
        ans1 += tmp;
        ans2 ^= tmp;
    }
    if(x == 4) {
        int tmp = get_val(y);
        ans1 += tmp;
```

```
        ans2 ^= tmp;
    }
    if(x == 5) {
        int tmp = tr[get_pre(y)].v;
        ans1 += tmp;
        ans2 ^= tmp;
    }
    if(x == 6) {
        int tmp = tr[get_suc(y)].v;
        ans1 += tmp;
        ans2 ^= tmp;
    }
}
cout << ans1 << " " << ans2 << "\n";
```

```
1000000000002329961 1000000000000001667
```

fft

```
signed main {
    mt19937 rand(0);
    for (int i = 1; i <= 20; i++) {
        int n = rand() % 100 + 1, m = rand() % 100 + 1;
        vector<double> a(n), b(m);
        for (int i = 0 ; i < n; i++) {
            a[i] = rand() % 100 + 1;
        }
        for (int i = 0; i < m; i++) {
            b[i] = rand() % 100 + 1;
        }
        auto ans = mul(a, b);
        double sum = 0;
        for (auto x : ans) {
            sum += x;
        }
        cout << fixed << setprecision(0) << sum << " ";
    }
}
```

```
5507964 1764445 1323685 8355072 22732435 4250782 3275356 1602420 4754812
6657250 4982142 2173858 109809 2031180 12458752 7225646 11931738 15165549
595010 47796
```

ntt

```
signed main() {
    mt19937 rand(0);
    for (int i = 1; i <= 20; i++) {
        int n = rand() % 100 + 1, m = rand() % 100 + 1;
        Poly a, b;
```

```
        a.a.resize(n);
        b.a.resize(m);
        for (int i = 0 ; i < n; i++) {
            a.a[i] = rand() % 100 + 1;
        }
        for (int i = 0; i < m; i++) {
            b.a[i] = rand() % 100 + 1;
        }
        Poly ans = a * b;
        int sum = 0;
        for (auto x : ans.a) {
            sum += x;
        }
        cout << sum << " ";
    }
}
```

```
5507964 1764445 1323685 8355072 22732435 4250782 3275356 1602420 4754812
6657250 4982142 2173858 109809 2031180 12458752 7225646 11931738 15165549
595010 47796
```

Prime

```
2 2 3 3 5 5 7 7
11 11 11 13 17 19 23 29 41 79 83 83 89 97
101 103 107 109 113 331 547 587 797 977 983 991 997
1009 1013 1019 1021 1031 2693 8039 8467 9547 9941 9949 9967 9973
10007 10009 10037 10039 10061 46381 57077 62851 98213 99961 99971 99989
99991
100003 100019 100043 100049 100057 107183 234383 573509 984007 999959 999961
999979 999983
1000003 1000033 1000037 1000039 1000081 1016927 1055189 6900961 7922111
9999943 9999971 9999973 9999991
10000019 10000079 10000103 10000121 10000139 10917271 68353301 75707057
88814903 99999941 99999959 99999971 99999989
100000007 100000037 100000039 100000049 100000073 166082023 574322789
654228647 676053907 999999883 999999893 999999929 999999937
1000000007 1000000009 1000000021 1000000033 1000000087 3397148761 5440806487
7154354923 9380086069 9999999881 9999999929 9999999943 9999999967
10000000019 10000000033 10000000061 10000000069 10000000097 12482387257
37315188863 50976305209 54383534603 99999999907 99999999943 99999999947
99999999977
100000000003 100000000019 100000000057 100000000063 100000000069
286708136027 452216566141 733218692447 772825281731 999999999937
999999999959 999999999961 999999999989
1000000000039 1000000000061 1000000000063 1000000000091 1000000000121
3032586593209 3836572107527 5416485186193 7173322551641 9999999999763
9999999999799 9999999999863 9999999999971
10000000000037 10000000000051 10000000000099 10000000000129 10000000000183
36885590072783 37541934267829 48213030604087 50498419100719 99999999999931
99999999999959 99999999999971 99999999999973
```

```
15 100000000000031 100000000000067 100000000000097 100000000000099
   100000000000133 403474996735879 421689567204827 779233068792001
   896525624400839 999999999999877 999999999999883 999999999999947
   999999999999989
16 1000000000000037 1000000000000091 1000000000000159 1000000000000187
   1000000000000223 4520969764685597 4892903438650489 5545765365226421
   6511840361739199 9999999999999851 9999999999999887 9999999999999917
   9999999999999937
17 10000000000000061 10000000000000069 10000000000000079 10000000000000099
   10000000000000453 15275914421071933 20593827648809693 58624671566507821
   71885865416282849 99999999999999943 99999999999999961 99999999999999977
   99999999999999997
18 100000000000000003 100000000000000013 100000000000000019 100000000000000021
   100000000000000049 407170308628310287 798161809347356543 912597791950842211
   971601792743437627 999999999999999863 999999999999999877 999999999999999967
   999999999999999989
19 1000000000000000003 1000000000000000009 1000000000000000031
   1000000000000000079 1000000000000000177 1123531975834070419
   2245710371805018757 3108977463994466341 4717942754514214051
```