# 南昌大学 **ACM** 校队模板

2021 年 11 月 28 日

*Author:* 刘浩然
*Email: haoran.mc@outlook.com*

# 目录

# 1 语言基础

## 1.1 封装

### 1.1.1 头文件.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> PII;
typedef pair<int, PII> PIII;
#define bug printf("<-->\n");
#define lowbit(x) ((x) & -(x))   // lowbit(ob0100) = 4
#define _max(a, b) (a > b ? a : b)
#define _min(a, b) (a < b ? a : b)
#define NEXTLINE puts("");
#define pb push_back
#define LINF LLONG_MAX
#define IOS ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define p(x) printf("-- %d\n", (x));
#define x first
#define y second
// #define int long long
// const int maxp = 1010;    //点的数量
// const int maxn = <++>;
const int INF    = 0x3f3f3f3f;
const ll  INF_LL = 0x3f3f3f3f3f3f3f3fLL;
// 上 右 下 左 上左 上右 下右 下左
const int dx[] = {-1, 0, 1, 0, -1, -1, 1, 1};
const int dy[] = {0, 1, 0, -1, -1, 1, 1, -1};
const double PI = acos(-1.0);
const double eps = 1e-6;
const double gold = (1 + sqrt(5)) / 2; //黄金分割 = 1.61803398...
const int FACT[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};
priority_queue<int, vector<int>, less<int>> he; /*默认是less，即数字大的优先级高*/
priority_queue<int, vector<int>, greater<int>> hE; /*默认是less，即数字大的优先级高*/

void clear(queue<int>& q) {
    queue<int> empty;
    swap(empty, q);
}

inline int sigma(char c) {return c - 'a';};

inline int sgn(double x) {   // 判断x是否等于0
    if (fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}

inline int dcmp(double x, double y) { //比较两个浮点数：0 相等；-1 小于；1 大于
    if (fabs(x - y) < eps) return 0;
    else return x < y ? -1 : 1;
}

struct Point {
    double x, y;
    Point() {}
    Point(double _x, double _y): x(_x), y(_y) {}
    bool operator == (const Point _point) {return sgn(x - _point.x) == 0 && sgn(y - _point.y) == 0;}
};

inline double dis(Point p1, Point p2) {return hypot(p1.x-p2.x, p1.y-p2.y);}

template <typename T>
T Smax(T x) {return x;}
template<typename T, typename... Args>
T Smax(T a, Args... args) {
    return _max(a, Smax(args...));
}
```

```cpp
template <typename T>
T Smin(T x) {return x;}
template<typename T, typename... Args>
T Smin(T a, Args... args) {
    return _min(a, Smin(args...));
}

void solve() {
    ;//<++>
}

int main() {
    // signed main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    // freopen("out.txt", "w", stdout);
#endif
    // ios::sync_with_stdio(false);
    // cin.tie(NULL), cout.tie(NULL);
    solve();
    return 0;
}
```

### 1.1.2 复数.cpp

```cpp
#include <cstdio>
struct Complex {
    double r, i;   // 一定注意，使用%f输出
    Complex() {}
    Complex(double _r, double _i) : r (_r), i (_i) {}
    inline void real(const double& x) {r = x;}
    inline double real() {return r;}
    inline Complex operator + (const Complex& rhs) const {
        return Complex (r + rhs.r, i + rhs.i) ;
    }
    inline Complex operator - (const Complex& rhs) const {
        return Complex (r - rhs.r, i - rhs.i);
    }
    inline Complex operator * (const Complex& rhs) const {
        return Complex (r*rhs.r - i*rhs.i, r*rhs.i + i*rhs.r);
    }
    inline void operator /= (const double& x) {
        r /= x, i /= x ;
    }
    inline void operator *= (const Complex& rhs) {
        *this = Complex (r*rhs.r - i*rhs.i, r*rhs.i + i*rhs.r);
    }
    inline void operator += (const Complex& rhs) {
        r += rhs.r, i += rhs.i;
    }
    inline Complex conj() {    // 共轭复数
        return Complex (r, -i) ;
    }
};

int main() {
    Complex c;
    c.real(10);
    printf("%f %f\n", c.r, c.i);
    printf("%f\n", c.real());
    Complex c1(5, 4);
    Complex c2(3, 2);

    c1 += c2;
    printf("%f %f\n", c1.r, c1.i);
    return 0;
}
```

## 1.2 STL

### 1.2.1 STL.md

```
# VECTOR
## vector的定义
vector翻译为映射，可以将任何基本类型(包括STL容器)映射到任何基本类型(包括STL容器)

```cpp
vector<T> vec1;                 /+创建了vec1+/
vector<T> vec2(vec1);           /+创建的vec2中包含vec1中所有副本+/
vector<T> vec2 = vec1;          /+创建的vec2中包含vec1中所有副本+/
vector<T> vec3(n, val);         /+创建的vec3中包含了n个重复的元素，每个元素的值都是val+/
vector<T> vec4(n);              /+创建的vec4中包含了n个重复地执行了值初始化的对象+/
vector<T> vec5{a, b, c ...};    /+创建的vec5中包含了初始化个树的对象，每个元素被赋予相应的初始值+/
vector<T> vec5 = {a, b, c ...}; /+创建的vec5中包含了初始化个树的对象，每个元素被赋予相应的初始值+/
```
## 常用函数
- begin
- end
- push_back O(1)
- pop_back O(1)
- size O(1)
- clear O(N)
- insert(it, x) O(N)
- erase O(N)
- rbegin
- rend
- front
- back
- vec2 = vec1
## 去重
```cpp
sort(ys.begin(), ys.end());
ys.erase(unique(ys.begin(), ye.end()), ys.end());
```
# STRING
## 字符串的输入输出
- string的输入输出
    + cin遇到空格或换行停止输入
    + getline(cin, str)是iostream中一个独立的函数，遇到换行结束输入
    + str.c_str将string类型转换为字符数组进行输出
    + cin.get与cin.getline是cin的成员，不能完成string类型变量的输入
## 常用函数
- operator string字符串可以与ch字符之间进行operator运算
- size O(1)
- length O(1)
- insert O(1)
    + insert(pos, str)
    + insert(str1_it, str2_begin, str2_end)
- erase O(N)
    + erase(it)
    + erase(first, last)
    + erase(pos, length)
- clear O(1)
- substr(pos, len) O(len)
- string::npos 是一个常数，其本身的值为-1，但由于是unsigned_int类型，因此实际上也可以认为是unsigned_int类型的最大值
    + string::npos用以作为find函数失配时的返回值
- find O(nm)
    + str1.find(str2)
    + str1.find(str2, pos)
    + 查询不到返回string::npos
- replace O(str1.length())
    + str1.replace(pos, len, str2)，把str从pos位开始，长度为len的子串替换为str2
    + str1.replace(it_begin, it_end, str2)，把str的迭代器[it_begin, it_end)范围内的子串替换为str2
# SET
## 常用函数
- count 判断元素是否在集合set中，存在返回1个，不存在返回0个
- insert O(logN)
- find O(logN) 没找到返回st.end()
```

- erase
    + st.erase(it) O(1)
    + st.erase(val) O(logN)
    + st.erase(first, last) O(last - first)
- size O(1)
- clear O(N)
- lower_bound :: st.lower_bound(key)，返回set中第一个大于等于key的迭代器指针
- upper_bound :: st.upper_bound(key)，返回set中第一个大于key的迭代器指针

## 注意!

- set容器只能通过迭代器访问

```cpp
for (std::set<int>::iterator it = st.begin(); it != st.end(); ++ it)
    printf("%d\n", *it);
```

# MAP

## 常用函数

- begin
- end
- find O(logN) 没有找到返回mp.end()
- mp.insert(make_pair(1, 2))
- erase
    + mp.erase(it) O(1)
    + mp.erase(key) O(logN)
    + mp.erase(first, last) O(last - first)
- size O(1)
- clear O(N)
- empty
- lower_bound map :: mp.lower_bound(key)，返回map中第一个大于等于key的迭代器指针
- upper_bound map :: mp.upper_bound(key)，返回map中第一个大于key的迭代器指针

## 注意!

- 字符串到整型的映射，必须使用string而不能使用char数组
- map使用erase函数时需要预先将迭代器后移
- map新建一个的初始值为0

# QUEUE

## 常用函数

- quu.count(1) 在队列中找1的个数
- push O(1)
- front O(1)
- back O(1)
- pop O(1)
- empty O(1)
- size O(1)

## queue清空

### 直接用空的队列对象赋值

```
queue<int> q1;
q1 = queue<int>();
```

### 遍历出队列

```cpp
while (!Q.empty()) Q.pop();
```

### 使用swap

这种是最高效的，定义clear，保持STL容器的标准

```cpp
void clear(queue<int>& q) {
    queue<int> empty;
    swap(empty, q);
}
```

## 注意!

- 使用front和pop函数之前必须使用empty函数判断队列是否为空，否则可能因为队空出现错误

# PRIORITY_QUEUE

## 默认设置

- 默认设置数字的大的是优先级高
- 与queue不同的是，priority_queue没有front与back函数，只能使用top函数取队顶
- 使用top函数前必须要用empty函数判断优先队列是否为空

## 常用函数

- push
- top
- pop

```
- empty
- size
## 优先级的设置
```cpp
priority_queue<int> pqu;
priority_queue<int, vector<int>, less<int>> heap; /*默认是less，大根堆*/
priority_queue<int, vector<int>, greater<int>> heap;
```
# STACK
## 常用函数
- push
- top
- pop
- empty
- size
# PAIR
## pair容器内元素的访问
- pr.first
- pr.second
## pair常用函数
- mp.insert(make_pair("ABCD", 5));
# ALGORITHM
- sort
- max & min min(x, y) 参数必须是两个，比较三个数，min(x, min(y, z))
- reverse(it, it + i) 将数组指针或容器的迭代器在[it, it + i)范围内的元素进行反转
- swap(x, y) 交换x, y
- abs(x) 返回x的绝对值，x必须是整数，浮点型的绝对值应使用math头文件下的fabs
- next_permutation() 给出一个序列在全排列中的下一个序列，在已经到达全排列的最后一个时会返回false
- fill 和memset不同，fill()可以把数组或容器中的某一段区间赋为某个相同的值，可以是数组类型对应范围中的任意值
- lower_bound :: lower_bound(vec.begin(), vec.end(), val)，第一个大于等于val的值
- upper_bound :: upper_bound(vec.begin(), vec.end(), val)，第一个大于val的值
```

## 1.3 标准模板库

### 1.3.1 string.h.md

```
# memset
# memcpy
# strcat
# strcmp
# strlen
len = strlen(str);

# strupr
# strlwr
# strcpy
```

## 1.4 快读

### 1.4.1 快读快输.cpp

```cpp
#include <cstdio>
#include <ctime>
inline int read() {
    int x = 0; bool flag = 1; char ch = getchar();
    while ((ch<'0' || ch>'9') && ch == '-')
        flag = 0, ch = getchar();
    while (ch >= '0' && ch <= '9')
        x = (x << 1) + (x << 3) + ch - '0', ch = getchar();
    return flag ? x : ~(x-1);
}

inline void write(int x) {
    if (x<0)
        x = ~(x-1), putchar('-');
    else if
        (x > 9) write(x/10);
    putchar(x%10+'0');
```

```
}

/*
 *inline int rwrite(int x)
 *{
 *    if(x<0) {putchar('-'); x=~(x-1);}
 *    int s[20],top=0;
 *    while(x) {s[++top]=x%10; x/=10;}
 *    if(!top) s[++top]=0;
 *    while(top) putchar(s[top--]+'0');
 *}
 */

int main() {
    clock_t start;
    clock_t end;
    start = clock();

    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    int num;
    for (int i = 0; i < 1e7; ++i) {
        num = read();
        write(num);
        putchar('\n');
    }

    end   = clock();
    printf("time = %f\n", (double)(end - start) / CLOCKS_PER_SEC);
    return 0;
}
```

### 1.4.2 普通读取.cpp

```
#include <cstdio>
#include <ctime>
int main() {
    clock_t start;
    clock_t end;
    start = clock();

    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    int num;
    for (int i = 0; i < 1e7; ++i) {
        scanf("%d", &num);
        printf("%d\n", num);
    }

    end   = clock();
    printf("time = %f\n", (double)(end - start) / CLOCKS_PER_SEC);
    return 0;
}
```

## 1.5 重载运算符

### 1.5.1 重载运算符.cpp

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
typedef long long ll;
int n;
ll cnt;

struct Node {
```

```
    int k, l, r;
    bool operator < (const Node& w) const {
        if (k != w.k)
            return k < w.k;
        else if (l != w.l)
            return l < w.l;
        else
            return r < w.r;
    }
};

vector<Node> cols, rows;

void merge(vector<Node>& segs) {
    sort(segs.begin(), segs.end());
    vector<Node> res;
    int st = -2e9, ed = st, k = -2e9;
    for (auto seg : segs) {
        if (seg.k == k) {
            if (ed < seg.l) {
                if (st != -2e9) {
                    res.push_back({k, st, ed});
                    cnt += ed - st + 1;
                }
                st = seg.l, ed = seg.r;
            }
            else
                ed = max(ed, seg.r);
        }
        else {
            if (st != -2e9) {
                res.push_back({k, st, ed});
                cnt += ed - st + 1;;
            }
            k = seg.k, st = seg.l, ed = seg.r;
        }
    }

    if (st != -2e9) {
        res.push_back({k, st, ed});
        cnt += ed - st + 1;;
    }
    segs = res;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        if (x1 == x2)
            cols.push_back({x1, min(y1, y2), max(y1, y2)});
        else
            rows.push_back({y1, min(x1, x2), max(x1, x2)});
    }

    merge(rows), merge(cols);

    for (auto row : rows)
        for (auto col : cols)
            if (row.k >= col.l && row.k <= col.r && row.r >= col.k  && row.l <= col.k)
                --cnt;

    printf("%lld\n", cnt);
    return 0;
}
```

## 1.6 已备函数

### 1.6.1 二进制-拆为 2 的幂相加.cpp

```cpp
// 创建日期: 2021年11月13日 星期六 19时31分00秒
#include <cstdio>
#include <vector>
#define NEXTLINE puts("");
using namespace std;

// 15 拆成 1 + 2 + 4 + 8
vector<int> dividenumtopower2(int num) {
    vector<int> vec;
    int b = 1;
    while (num) {
        if (num & b) {
            vec.push_back(b);
            num -= b;
        }
        b <<= 1;
    }
    return vec;
}

int main() {
    for (int i = 1; i < 20; ++ i) {
        vector<int> vec = dividenumtopower2(i);
        printf("%d:", i);
        for (auto i : vec)
            printf(" %d", i);
        NEXTLINE;
    }
    return 0;
}
```

### 1.6.2 二进制-获取二进制表示中最高比重.cpp

```cpp
// 创建日期: 2021年11月13日 星期六 18时35分45秒
#include <cstdio>
#include <vector>
#define lowbit(x) ((x) & -(x))   // lowbit(ob0100) = 4
#define NEXTLINE puts("");

int getmaxproportion(int num) {
    int b = 1, last = 0, pos = 0;
    while (num) {
        if (num & b) {
            last = b;
            num -= b;
        }
        b <<= 1;
        pos ++ ;
    }
    return pos;
}

void outputbinary(int num) {
    std::vector<int> vec;
    while (num) {
        vec.push_back(num % 2);
        num /= 2;
    }
    for (int i = (int)vec.size() - 1; i >= 0; -- i)
        printf("%d", vec[i]);
    NEXTLINE;
}

int main() {
    for (int i = 1; i < 20; ++ i) {
```

```
        printf("%2d %d ", i, getmaxproportion(i));
        outputbinary(i);
    }
    return 0;
}
```

### 1.6.3 二进制-输出二进制表示.cpp

```cpp
// 创建日期: 2021年11月01日 星期一 09时11分42秒
#include <cstdio>
#include <vector>
#define NEXTLINE puts("");

void outputbinary(int num) {
    std::vector<int> vec;
    while (num) {
        vec.push_back(num % 2);
        num /= 2;
    }
    for (int i = (int)vec.size() - 1; i >= 0; -- i)
        printf("%d", vec[i]);
    NEXTLINE;
}

int main() {
    for (int i = 1; i < 20; ++ i)
        outputbinary(i);
    return 0;
}
```

## 1.7 PY

### 1.7.1 初始化列表.md

```
# 一维

```python
list1 = range(10)
# list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

list2 = [0] * 5
# list2 = [0, 0, 0, 0, 0]

list3 = [1 for i in range(5)]
# list3 = [1, 1, 1, 1, 1]
```

# 二维

```python
multilist1 = [[0 for col in range(5)] for row in range(6)]
# multilist1 =  [
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0]
# ]

multilist2 = [[0] * 5 for row in range(6)]
# multilist2 =  [
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
#     [0, 0, 0, 0, 0],
```
```

```
#     [0, 0, 0, 0, 0]
# ]
```

## 1.8 数学

### 1.8.1 math.md

```
1. 坐标旋转
x' = x * cos(t) - y * sin(t);
y' = x * sin(t) + y * cos(t);
```

## 1.9 语法基础

### 1.9.1 函数返回数组.cpp

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* 要生成和返回随机数的函数 */
int * getRandom( ) {
    static int r[10];

    for (int i = 0; i < 10; ++ i) {
        r[i] = rand();
        printf( "r[%d] = %d\n", i, r[i]);

    }
    return r;
}

int main () {
    int *p;
    p = getRandom();

    for (int i = 0; i < 10; i++ )
        printf( "*(p + %d) : %d\n", i, *(p + i));
    return 0;
}
```

# 2 基本算法

## 2.1 递归 and 分治

### 2.1.1 递归实现指数型枚举

#### 2.1.1.1 递归-vector.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年05月11日 ---- 15时10分
 *   题     目：AcWing 92 递归实现指数型枚举
 *   算     法：递归
 *   描     述：从 [1, n]这n（n < 20）个整数中随机选取任意多个
 *   输出所有可能的选择方案，每个方案的数按升序输出
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
using namespace std;
int n;
vector<int> vec;

void DFS(int x) {
```

```
    if (x == n + 1) {
        for (auto i : vec)
            printf("%d ", i);
        puts("");
        return;
    }
    DFS(x + 1);          // 不选择 x
    vec.push_back(x);    // 选择 x
    DFS(x + 1);          // 选择 x
    vec.pop_back();      // 回溯
}

int main() {
    scanf("%d", &n);
    DFS(1);   // 决策数字1
    return 0;
}
```

#### 2.1.1.2 递归-used 数组.cpp

```
/*------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年05月11日 ---- 15时10分
 *   题    目：AcWing 92 递归实现指数型枚举
 *   算    法：递归
 *   描    述：从 [1, n]这n（n < 20）个整数中随机选取任意多个
 *   输出所有可能的选择方案，每个方案的数按升序输出
 *
------------------------------------------------------------*/

#include <cstdio>
const int maxn = 20;
int n;
bool used[maxn];

void DFS(int x) {
    if (x == n + 1) {
        for (int i = 1; i <= n; ++ i)
            if (used[i])
                printf("%d ", i);
        puts("");
        return;
    }
    DFS(x + 1);
    used[x] = true;
    DFS(x + 1);
    used[x] = false;
}

int main() {
    scanf("%d", &n);
    DFS(1);
    return 0;
}
```

#### 2.1.1.3 递归-状压.cpp

```
/*------------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年05月11日 ---- 15时10分
 *   题    目：AcWing 92 递归实现指数型枚举
 *   算    法：递归
 *   描    述：从1~n这n(n < 20)个整数中随机选取任意多个，输出所有可能的选择方案
 *
 ------------------------------------------------------------------*/

```

```cpp
#include <cstdio>
int n;

void DFS(int u, int state) {
    if (u == n) {
        for (int i = 0; i < n; ++ i)
            if (state >> i & 1)
                printf("%d ", i + 1);
        puts("");
        return;
    }
    DFS(u + 1, state);   // 不选
    DFS(u + 1, state | 1 << u);   // 选
}

int main() {
    scanf("%d", &n);
    DFS(0, 0);
    return 0;
}
```

### 2.1.2 递归实现组合型枚举

#### 2.1.2.1 简单递归-path 数组.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年05月11日 星期二 21时00分48秒
 *   题    目：AcWing 93 递归组合型枚举
 *   算    法：递归，二进制状态压缩
 *   描    述：从 [1, n] 这 n 个整数中选择 m 个，输出所有可能的方案数
 *   对于不同的方案，字典序小的方案先输出
 *   对于每个方案，升序输出
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 30;
int n, m;
int path[maxn];

// start 表示接下来从哪里开始搜索
// cnt 表示已经选择了几个数
void DFS(int start, int cnt) {
    if (cnt == m + 1) {
        for (int i = 1; i <= m; ++ i)
            printf("%d ", path[i]);
        puts("");
    }
    else {
        for (int i = start; i <= n; ++ i) {
            path[cnt] = i;
            DFS(i + 1, cnt + 1);
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    DFS(1, 1);
    return 0;
}
```

#### 2.1.2.2 递归 + 状压.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
```

```
 *    创建日期：2021年05月11日 星期二 21时00分48秒
 *    题      目：AcWing 93 递归组合型枚举
 *    算      法：递归，二进制状态压缩
 *    描      述：从 [1, n] 这 n 个整数中选择 m 个，输出所有可能的方案数
 *    对于不同的方案，字典序小的方案先输出
 *    对于每个方案，升序输出
 *
 ----------------------------------------------------------------*/

#include <cstdio>
int n, m;

void DFS(int start, int cnt, int state) {
    if (n - start + cnt < m)   // 剪枝
        return;
    if (cnt == m) {
        for (int i = 0; i < n; ++ i)
            if (state >> i & 1)
                printf("%d ", i + 1);
        puts("");
        return;
    }

    // 下面两句反过来试试?
    DFS(start + 1, cnt + 1, state | 1 << start);
    DFS(start + 1, cnt, state);
}

int main() {
    scanf("%d %d", &n, &m);
    DFS(0, 0, 0);
    return 0;
}
```

#### 2.1.2.3 非递归 + 手动写栈.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：02.cpp
 *    创建日期：2021年05月11日 星期二 21时42分25秒
 *    题      目：AcWing 93 递归组合型枚举
 *    算      法：非递归，二进制状态压缩
 *    描      述：不使用递归
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <stack>
using namespace std;
int n, m;

struct State {
    int pos;
    int start, cnt, state;
};

void DFS(int start, int cnt, int state) {
    // 0:
    if (n - start + cnt  < m)
        return;
    if (cnt == m) {
        for (int i = 0; i < n; ++ i)
            if (state >> i & 1)
                printf("%d ", i + 1);
        puts("");
        return;
    }
    DFS(start + 1, cnt + 1, state | 1 << start);
    // 1:
```

```
    DFS(start + 1, cnt, state);
    // 2:
}

int main() {
    scanf("%d %d", &n, &m);
    stack<State> stk;
    stk.push({0, 0, 0, 0});
    while (stk.size()) {
        auto sta = stk.top(); stk.pop();
        if (sta.pos == 0) {
            if (n - sta.start + sta.cnt < m)
                continue;
            if (sta.cnt == m) {
                for (int i = 0; i < n; ++ i)
                    if (sta.state >> i & 1)
                        printf("%d ", i + 1);
                puts("");
                continue;
            }
            sta.pos = 1; stk.push(sta);
            stk.push({0, sta.start + 1, sta.cnt + 1, sta.state | 1 << sta.start});
        }
        else if (sta.pos == 1) {
            sta.pos = 2; stk.push(sta);
            stk.push({0, sta.start + 1, sta.cnt, sta.state});
        }
        else if (sta.pos == 2) {
            continue;
        }
    }
    return 0;
}
```

### 2.1.3 递归实现排列型枚举

#### 2.1.3.1 经典全排列.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-全排列.cpp
 *   创建日期：2021年05月12日 星期三 13时44分31秒
 *   题    目：AcWing 0094 递归实现排列型枚举
 *   算    法：递归
 *   描    述：输出 [1, n] 这 n 个数所有可能的排列，不同排列按字典序
 *   较小的先输出
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 10;
int n, site[maxn], ha[maxn];

void DFS(int idx) {
    if (idx == n + 1) {
        for (int i = 1; i <= n; ++ i)
            printf("%d ", site[i]);
        puts("");
        return;
    }
    for (int i = 1; i <= n; ++ i) {
        if (!ha[i]) {
            site[idx] = i;
            ha[i] = true;
            DFS(idx + 1);
            ha[i] = false;
        }
    }
}

```

```
int main() {
    scanf("%d", &n);
    DFS(1);
    return 0;
}
```

#### 2.1.3.2 swap.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-全排列.cpp
 *   创建日期：2021年05月12日 星期三 13时44分31秒
 *   题    目：AcWing 0094 递归实现排列型枚举
 *   算    法：递归
 *   描    述：输出 [1, n] 这 n 个数所有可能的排列，不同排列按字典序
 *   较小的先输出
 *
 *   不是很理解
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 10;
int n, cnt, arr[maxn];

void DFS(int idx) {
    if (idx == n) {
        for (int i = 0; i < n; ++ i)
            printf("%d ", arr[i]);
        puts("");
        return;
    }
    for (int i = idx; i < n; ++ i) {
        swap(arr[i], arr[idx]);
        DFS(idx + 1);
        swap(arr[i], arr[idx]);
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        arr[i] = i + 1;
    DFS(0);
    return 0;
}
```

#### 2.1.3.3 递归全排列 + 状压.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-全排列.cpp
 *   创建日期：2021年05月12日 星期三 13时44分31秒
 *   题    目：AcWing 0094 递归实现排列型枚举
 *   算    法：递归
 *   描    述：输出 [1, n] 这 n 个数所有可能的排列，不同排列按字典序
 *   较小的先输出
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
using namespace std;
int n;
vector<int> vec;

```

```cpp
void DFS(int idx, int state) {
    if (idx == n) {
        for (auto i : vec)
            printf("%d ", i);
        puts("");
        return;
    }
    for (int i = 0; i < n; ++ i)
        if (!(state >> i & 1)) {
            vec.push_back(i + 1);
            DFS(idx + 1, state | 1 << i);
            vec.pop_back();
        }
}

int main() {
    scanf("%d", &n);
    DFS(0, 0);
    return 0;
}
```

## 2.2 贪心

### 2.2.1 区间合并

#### 2.2.1.1 区间合并.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称: 02-lhr.cpp
 *   创建日期: 2021年10月27日 星期三 22时32分21秒
 *   题    目: AcWing 0803 区间合并
 *   算    法: 区间合并
 *   描    述: <++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <utility>
#include <algorithm>
using namespace std;
typedef pair<int, int> PII;
#define bug printf("<-->\n");
const int INF = 0x3f3f3f3f;
vector<PII> segs;  // pair在C++中优先以左端点排序
// segment, section, sequence
// const int maxn = 1e5 + 5;

// 将含有交集的区间合并[1, 5] [4, 8] --> [1, 8]
void merge(vector<PII> &segs) {
    // for循环无法将最后一个区间添加到结果容器中
    // 加上这一句，就可以使最后一个区间添加到结果容器中
    segs.push_back({INF, INF});
    vector<PII> res;
    sort(segs.begin(), segs.end());
    int st = segs[0].first,
        ed = segs[0].second;
    for (auto seg : segs) {
        // printf("%d %d\n", seg.first, seg.second);
        if (ed >= seg.first)
            ed = max(ed, seg.second);
        else {
            res.push_back({st, ed});
            st = seg.first,
               ed = seg.second;
        }
    }
    segs = res;
}
```

```

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
#endif
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        segs.push_back({l, r});
    }
    merge(segs);
    printf("%d\n", (int)segs.size());
    /*
     * for (auto seg : segs)
     *     printf("%d %d\n", seg.first, seg.second);
     */
    return 0;
}
```

### 2.2.2 均分纸牌

#### 2.2.2.1 均分纸牌.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年06月02日 星期三 10时16分59秒
 *   题    目：AcWing 1536 均分纸牌
 *   算    法：<++>
 *   描    述：人都给我看傻了，太顶了
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 100 + 5;
int card[maxn];
int n, sum, res;

int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &card[i]);
        sum += card[i];
    }

    int avg = sum / n;

    for (int i = 0; i < n; i++)
        if (card[i] != avg)
            card[i + 1] += card[i] - avg,
                res++;

    printf("%d\n", res);
    return 0;
}
```

### 2.2.3 糖果传递

#### 2.2.3.1 七夕祭.cpp

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
typedef long long ll;
int x[maxn], y[maxn];

```

```
//下标从1开始
ll solve(int I[], int n) {
    ll avg = 0;
    for (int i = 1; i <= n; ++i)
        avg += I[i];
    avg /= n;

    int x[maxn];
    x[1] = 0;
    for (int i = 2; i <= n; ++i)
        x[i] = x[i-1] + avg - I[i];

    sort(x + 1, x + n + 1);
    ll x1 = x[n / 2];  //中位数
    ll res = 0;
    for (int i = 1; i <= n; ++i)
        res += abs(x[i] - x1);
    return res;
}

int main() {
    int n, m, t;
    scanf("%d %d %d", &n, &m, &t);
    ll sumx = 0, sumy = 0;
    while (t--) {
        int ix, iy;
        scanf("%d %d", &ix, &iy);
        x[ix]++, y[iy]++;
        sumx++, sumy++;
    }
    if (sumx % n && !(sumy % m))
        printf("column %lld\n", solve(y, m));
    else if (!(sumx % n) && sumy % m)
        printf("row %lld\n", solve(x, n));
    else if (!(sumx % n) && !(sumy % m))
        printf("both %lld\n", solve(x, n) + solve(y, m));
    else
        printf("impossible\n");
    return 0;
}
```

#### 2.2.3.2 糖果传递.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：02.cpp
 *   创建日期：2021年06月02日 星期三 17时14分12秒
 *   题    目：AcWing 0122 糖果传递
 *   算    法：推公式
 *   描    述：自己推了一遍，这个题目很好的，建议搞懂
 *      a1 --x1-> a2 --x2-> a3 --x3-> a4  ...  an
 *      ^                                      |
 *      |------------------xn-------------------
 *
 *      公式：x1 = x1
 *            x2 = abs(x1 - (avg - a2))
 *            x3 = abs(x1 - (2* avg - (a2 + a3))
 *            x4 = abs(x1 - (3* avg - (a2 + a3 + a4))
 *            x5 = abs(x1 - (4* avg - (a2 + a3 + a4 + a5))
 *            ......
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e6 + 5;
typedef long long ll;
int n;
```

```cpp
ll I[maxn], x[maxn];

int main() {
    scanf("%d", &n);
    ll sum = 0;
    for (int i = 1; i <= n; ++i) {
        scanf("%lld", &I[i]);
        sum += I[i];
    }
    int avg = sum / n;
    x[1] = 0;
    for (int i = 2; i <= n; ++i)
        x[i] = x[i-1] + avg - I[i];

    sort(x + 1, x + n + 1);
    ll x1 = x[n / 2];
    ll res = 0;
    for (int i = 1; i <= n; ++i)
        res += abs(x[i] - x1);
    printf("%lld\n", res);
    return 0;
}
```

### 2.2.4 区间贪心

#### 2.2.4.1 区间分组.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：区间分组.cpp
 *   创建日期：2021年11月02日 星期二 22时09分42秒
 *   题    目：AcWing 0906 区间分组
 *   算    法：区间贪心
 *   描    述：给定 n 个区间，请你将这些区间分成若干组，使得每组内部
 *   的区间两两之间（包括端点）没有交集，并使得组数尽可能小
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <queue>
using namespace std;
const int maxn = 1e5 + 5;

struct Range {
    int l, r;
    bool operator < (const Range R) const {
        return l < R.l;
    }
} range[maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d %d", &range[i].l, &range[i].r);
    sort(range, range + n);
    // 小根堆，存放的是每组里最右端点
    priority_queue<int, vector<int>, greater<int>> heap;
    for (int i = 0; i < n; ++ i) {
        if (heap.empty() || heap.top() >= range[i].l)
            heap.push(range[i].r);
        else {
            // int t = heap.top();
            heap.pop();
            heap.push(range[i].r);
        }
    }
    printf("%d\n", heap.size());
    return 0;
}
```

#### 2.2.4.2 区间覆盖.cpp

```cpp
/*--------------------------------------------------------------------
 *
 *    文件名称：区间覆盖.cpp
 *    创建日期：2021年11月02日 星期二 20时32分26秒
 *    题      目：AcWing 0907 区间覆盖
 *    算      法：区间贪心
 *    描      述：给定 n 个区间 [li, ri]，以及一个线段区间 [st, ed]，请你
 *    选择尽量少的区间，将指定线段区间完全覆盖，输出最少区间数，如果
 *    无法完全覆盖则输出 -1
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct Range {
    int l, r;
    bool operator < (const Range &R) const {
        return l < R.l;
    }
} range[maxn];

int main() {
    int st, ed;
    scanf("%d %d", &st, &ed);
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d %d", &range[i].l, &range[i].r);
    sort(range, range + n);
    int res = 0;
    bool success = false;
    for (int i = 0; i < n; ++ i) {
        int j = i, r = -INF;
        // 左端点小于 st，右端点最大是 r
        while (j < n && range[j].l <= st) {
            r = max(r, range[j].r);
            j ++ ;
        }
        if (r < st)
            break;
        res ++ ;
        if (r >= ed) {
            success = true;
            break;
        }
        st = r;
        i = j - 1;
    }
    if (!success)
        res = -1;
    printf("%d\n", res);
    return 0;
}
```

#### 2.2.4.3 区间选点.cpp

```cpp
/*--------------------------------------------------------------------
 *
 *    文件名称：区间选点.cpp
 *    创建日期：2021年11月02日 星期二 14时16分53秒
 *    题      目：AcWing 0905 区间选点
```

```
 *   算     法：贪心 区间问题
 *   描     述：给定 n 个区间 li, ri，请你在数轴上选择尽量少的点，使得
 *   每个区间内至少包含一个选出的点，输出选择的点的最小数量。
 *   位于区间端点上的点也算作区间内。
 *
 ----------------------------------------------------------------*/

/**
 *       |-----------|                |------------------|
 *   ---------------x--------------------x---------------------------->
 *             |----------------|  |--------|
 *                 |------|
 */

#include <cstdio>
#include <algorithm>
using namespace std;
int n;
const int maxn = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct Range {
    int l, r;
    bool operator < (const Range &R) const {
        return r < R.r;
    }
} range[maxn];

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++ i) {
        int l, r;
        scanf("%d %d", &l, &r);
        range[i] = {l, r};
    }
    sort(range, range + n);
    int res = 0, ed = -INF;
    for (int i = 0; i < n; ++ i)
        if (range[i].l > ed) {
            res ++ ;
            ed = range[i].r;
        }
    printf("%d\n", res);
    return 0;
}
```

#### 2.2.4.4 最大不相交区间数量.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：最大不相交区间数量.cpp
 *   创建日期：2021年11月02日 星期二 19时46分23秒
 *   题     目：AcWing 0908 最大不相交区间数量
 *   算     法：区间贪心
 *   描     述：给定 n 个区间 li, ri，请你在数轴上选择若干个区间，
 *   使得选中的区间之间互不相交（包括端点）。输出可选区间的最大数量。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct Range {
    int l, r;
    bool operator < (const Range & R) const {
        return r < R.r;
```

```
    }
} range[maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d %d", &range[i].l, &range[i].r);
    sort(range, range + n);
    int res = 0, ed = -INF;
    for (int i = 0; i < n; ++ i)
        if (range[i].l > ed) {
            res ++ ;
            ed = range[i].r;
        }
    printf("%d\n", res);
    return 0;
}
```

### 2.2.5 哈夫曼编码

#### 2.2.5.1 合并果子.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：合并果子.cpp
 *   创建日期：2021年11月02日 星期二 22时37分15秒
 *   题    目：AcWing 00148 合并果子
 *   算    法：哈夫曼编码
 *   描    述：把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和
 *   输出最小消耗的体力。
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <queue>
using namespace std;

int main() {
    int n; scanf("%d", &n);
    priority_queue<int, vector<int>, greater<int>> heap;
    for (int i = 0; i < n; ++ i) {
        int weight; scanf("%d", &weight);
        heap.push(weight);
    }
    int res = 0;
    while (heap.size() > 1) {
        int a = heap.top(); heap.pop();
        int b = heap.top(); heap.pop();
        res += a + b;
        heap.push(a + b);
    }
    printf("%d\n", res);
    return 0;
}
```

### 2.2.6 排序不等式

#### 2.2.6.1 排队打水.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：排队打水.cpp
 *   创建日期：2021年11月02日 星期二 23时49分52秒
 *   题    目：AcWing 0913 排队打水
 *   算    法：排序不等式
 *   描    述：有 n 个人排队到 1 个水龙头处打水，第 i 个人装满水桶
 *   所需时间是 ti，请问如何安排他们的打水顺序才能使所有人的等待时间
 *   之和最小
 *
```

```
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long ll;
const int maxn = 1e5 + 5;
int t[maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d", &t[i]);
    sort(t, t + n);
    ll res = 0;
    for (int i = 0; i < n; ++ i)
        res += t[i] * (n - i - 1);
    printf("%lld\n", res);
    return 0;
}
```

### 2.2.7 绝对值不等式

#### 2.2.7.1 货仓选址.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：货仓选址.cpp
 *   创建日期：2021年11月03日 星期三 20时29分32秒
 *   题    目：AcWing 0104 货仓选址
 *   算    法：贪心
 *   描    述：在一条数轴上有 n 家商店，它们的坐标是 ai
 *   现在需要在数轴上建立一家货仓，货仓建在哪里可以使得货仓到每家
 *   商店的距离之和最小
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int a[maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d", &a[i]);
    sort(a, a + n);
    int res = 0;
    for (int i = 0; i < n; ++ i)
        res += abs(a[i] - a[n / 2]);
    printf("%d\n", res);
    return 0;
}
```

### 2.2.8 推公式

#### 2.2.8.1 刷杂技的牛.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：刷杂技的牛.cpp
 *   创建日期：2021年11月03日 星期三 20时33分53秒
 *   题    目：AcWing 0125 刷杂技的牛
 *   算    法：贪心
 *   描    述：奶牛学习叠罗汉
 *   这 n 头奶牛中的每一头都有着自己的重量 wi 以及自己的强壮程度 si。
 *   这头牛的头上所有牛的总重量（不包括它自己）减去它的身体强壮程度
 *   的值称为该牛的风险值，风险值越大，这只牛撑不住的可能性越高。
```

```
 *    你的任务是确定奶牛的排序，使得所有奶牛的风险值中的最大值尽可能的小。
 *
 *    按 wi + si 从小到大排序
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <utility>
using namespace std;
typedef pair<int, int> PII;
const int maxn = 5e4 + 5;
const int INF = 0x3f3f3f3f;
PII cows[maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i) {
        int w, s;
        scanf("%d %d", &w, &s);
        cows[i] = {w + s, w};
    }
    sort(cows, cows + n);
    int sum = 0, res = -INF;
    for (int i = 0; i < n; ++ i) {
        int w = cows[i].second,
            s = cows[i].first - cows[i].second;
        res = max(res, sum - s);
        sum += w;
    }
    printf("%d\n", res);
    return 0;
}
```

## 2.3 排序

### 2.3.1 选择排序

#### 2.3.1.1 选择排序.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：selectSort.cpp
 *    创建日期：2021年08月08日 星期日 23时51分34秒
 *    题    目：<++>
 *    算    法：<++>
 *    描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <stdio.h>
int num[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
int n = 10;

void selectSort() {
    int mini, idx;
    bool flag;

    for (int i = 0; i < n; i++) {
        flag = false;
        mini = num[i];
        idx  = i;
        for (int j = i; j < n; j++)
            if (num[j] < mini) {
                flag = true;
                mini = num[j];
                idx  = j;
            }
        if (flag) {
            num[i]  += num[idx];
```

```
            num[idx] = num[i] - num[idx];
            num[i]  -= num[idx];
        }
    }
}

int main() {
    selectSort();
    for (int i = 0; i < 10; i++)
        printf("%d ", num[i]);

    printf("\n");
    return 0;
}
```

### 2.3.2 冒泡排序

#### 2.3.2.1 冒泡排序.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：bubbleSort.cpp
 *   创建日期：2021年08月08日 星期日 23时51分18秒
 *   题    目：<++>
 *   算    法：<++>
 *   描    述：<++>
 *
 -----------------------------------------------------------------*/

#include <stdio.h>
int n = 10;
int num[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

void bubbleSort() {
    for (int i = 0; i < n-1; ++i)
        for (int j = n-1; j > i; --j)
            if (num[j] < num[j-1]) {
                num[j] += num[j-1];
                num[j-1] = num[j] - num[j-1];
                num[j] -= num[j-1];
            }
}

int main() {
    bubbleSort();
    for (int i = 0; i < 10; ++i)
        printf("%d ", num[i]);
    printf("\n");
    return 0;
}
```

### 2.3.3 插入排序

#### 2.3.3.1 插入排序.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：insertSort.cpp
 *   创建日期：2021年08月08日 星期日 23时51分03秒
 *   题    目：<++>
 *   算    法：<++>
 *   描    述：第i个数往前比较，如果num[i]比num[j]小，就num[j] = num[j - 1]
 *   因为要给num[i]腾出一个空
 *
 *          |------|
 *          V      |
 *   1 2 3 4 6 7 8 5 9
 *
 -----------------------------------------------------------------*/
```

```

#include <stdio.h>

void insertSort(int num[], int n) {
    // 前i - 1个数已经排好(不是1234这种，是1357这种)，看第i个数插在哪
    for (int i = 1; i < n; ++ i) {
        int insertNum = num[i];
        int j;
        for (j = i; j >= 1 && insertNum < num[j - 1]; -- j)
            num[j] = num[j - 1];
        num[j] = insertNum;
    }
}

int main() {
    int num[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
    int n = 10;
    insertSort(num, n);
    for (int i = 0; i < n; ++ i)
        printf("%d ", num[i]);
    printf("\n");
    return 0;
}
```

### 2.3.4 快速排序

#### 2.3.4.1 双指针快排.cpp

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <algorithm>
using namespace std;
int num[] = {5, 4, 7, 9, 0, 2, 1, 3, 8, 6};

int partition(int left, int right) {
    int random = (int)round(1.0 * rand() / RAND_MAX * (right - left) + left);
    swap(num[random], num[left]);

    int temp = num[left];

    while (left < right) {
        while (left < right && num[right] > temp)
            right--;

        num[left] = num[right];

        while (left < right && num[left] <= temp)
            left++;

        num[right] = num[left];
    }
    num[left] = temp;
    return left;
}

void quickSort(int left, int right) {
    if (left < right) {
        int mark = partition(left, right);
        quickSort(left, mark - 1);
        quickSort(mark + 1, right);
    }
}

int main() {
    quickSort(0, 9);

    for (int i = 0; i < 10; i++)
        printf("%d ", num[i]);
```

```
    printf("\n");

    return 0;
}
```

#### 2.3.4.2 快速排序.cpp

```
/*--------------------------------------------------------------
 *
 *   文件名称：quickSort
 *   创建日期：2021年05月30日 星期日 00时53分58秒
 *   题    目：AcWing 0785 快速排序
 *   算    法：快速排序
 *   描    述：去看笔记-首元素做基准
 *
 --------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
#define NEXTLINE puts("");
int n = 10;
int num[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

void quickSort(int l, int r) {
    if (l >= r) return;

    int i = l - 1, j = r + 1, x = num[(l + r) >> 1];
    while (i < j) {
        do i ++ ; while (num[i] < x);
        do j -- ; while (num[j] > x);
        if (i < j) swap(num[i], num[j]);
    }
    quickSort(l, j);
    quickSort(j + 1, r);
}

int main() {
    quickSort(0, 9);
    for (int i = 0; i < n; ++i)
        printf("%d ", num[i]);
    NEXTLINE;
    return 0;
}
```

#### 2.3.4.3 第 k 个数.cpp

```
/*--------------------------------------------------------------
 *
 *   文件名称：05-第k个数.cpp
 *   创建日期：2021年08月08日 星期日 23时35分55秒
 *   题    目：AcWing 0786 第k个数
 *   算    法：<++>
 *   描    述：<++>
 *
 --------------------------------------------------------------*/

#include <iostream>
using namespace std;
const int maxn = 1000010;
int num[maxn];

// 从0开始，左闭右闭
int quickSort(int l, int r, int k) {
    if (l >= r) return num[l];

    int i = l - 1, j = r + 1, x = num[l + r >> 1];
    while (i < j) {
```

```
        do i ++ ; while (num[i] < x);
        do j -- ; while (num[j] > x);
        if (i < j) swap(num[i], num[j]);
    }

    if (j - l + 1 >= k)
        return quickSort(l, j, k);
    else
        return quickSort(j + 1, r, k - (j - l + 1));
}

int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i ++ )
        scanf("%d", &num[i]);
    printf("%d\n", quickSort(0, n-1, k));
    return 0;
}
```

### 2.3.5 归并排序

#### 2.3.5.1 归并排序.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称: mergeSort.cpp
 *   创建日期: 2021年08月08日 星期日 23时51分53秒
 *   题    目: <++>
 *   算    法: <++>
 *   描    述: <++>
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#define NEXTLINE puts("");
int n = 10;
int num[] = {6, 3, 8, 9, 4, 1, 5, 0, 2, 7};
int tmp[10];

void mergeSort(int l, int r) {
    if (l >= r) return;

    int mid = l + r >> 1;
    mergeSort(l, mid);
    mergeSort(mid + 1, r);

    // v         v
    // -------------------
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (num[i] <= num[j])
            tmp[k++] = num[i++];
        else
            tmp[k++] = num[j++];

    while (i <= mid)
        tmp[k++] = num[i++];
    while (j <= r)
        tmp[k++] = num[j++];

    for (i = l, j = 0; i <= r; i++, j++)
        num[i] = tmp[j];
}

int main() {
    mergeSort(0, n-1);
    for (int i = 0; i < n; ++i)
        printf("%d ", num[i]);
    NEXTLINE;
```

```
    return 0;
}
```

#### 2.3.5.2 逆序数.cpp

```cpp
#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 1e5 + 5;
int n;
int num[maxn];
int tmp[maxn];
long long inverse;

/*
 * 左半边内部的逆序对数量: merge_sort(L, mid)
 * 左半边内部的逆序对数量: merge_sort(mid + 1, R)
 */
void mergeSort(int l, int r) {
    if (l >= r) return;

    int mid = (l + r) >> 1;
    mergeSort(l, mid);
    mergeSort(mid + 1, r);

    // v         v
    // -------------------
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (num[i] <= num[j])
            tmp[k++] = num[i++];
        else {
            tmp[k++] = num[j++];
            /*
             * --------i---------------------
             * ----------------------j--------
             *  num[i] > num[j]
             *  必有num[i+1], num[i+2], ... num[mid] > num[j]
             *  这些对都是逆序对
             */
            inverse += (long long)(mid - i + 1);  //只是在归并排序上添加这句
        }

    while (i <= mid)
        tmp[k++] = num[i++];
    while (j <= r)
        tmp[k++] = num[j++];

    for (i = l, j = 0; i <= r; i++, j++)
        num[i] = tmp[j];
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &num[i]);
    mergeSort(0, n-1);
    /*
     * for (int i = 0; i < n; ++i)
     *     printf("%d ", num[i]);
     * NEXTLINE;
     */
    printf("%lld\n", inverse);
    return 0;
}
```

### 2.3.6 堆排序

#### 2.3.6.1 堆排序.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *    文件名称: heapSort.cpp
 *    创建日期: 2021年08月08日 星期日 23时42分11秒
 *    题      目: AcWing 0838 堆排序
 *    算      法: 堆
 *    描      述: <++>
 *
 -----------------------------------------------------------------*/

#include <cstdio>
int main() {
    return 0;
}
```

## 2.4 前缀和 and 差分

### 2.4.1 一维前缀和.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *    文件名称: 一维前缀和.cpp
 *    创建日期: 2021年05月31日 星期一 01时14分51秒
 *    题      目: AcWing 0795 前缀和
 *    算      法: 前缀和
 *    描      述: <++>
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int sequ[maxn];
int preS[maxn];

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    // 前缀和的题目从下标1开始读
    for (int i = 1; i <= n; ++i)
        scanf("%d", &sequ[i]);
    for (int i = 1; i <= n; ++i)
        preS[i] = preS[i-1] + sequ[i];
    while (m--) {
        int l, r;
        scanf("%d %d", &l, &r);
        printf("%d\n", preS[r] - preS[l - 1]);
    }
    return 0;
}
```

### 2.4.2 一维差分.cpp

```cpp
#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 1e5 + 5;
int sequ[maxn];
int diff[maxn];

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &sequ[i]);
        diff[i] = sequ[i] - sequ[i-1];
    }
    while (m--) {
        int l, r, c;
```

```
        scanf("%d %d %d", &l, &r, &c);
        diff[l] += c;
        diff[r+1] -= c;
    }
    for (int i = 1; i <= n; ++i)
        diff[i] += diff[i-1];
    for (int i = 1; i <= n; ++i)
        printf("%d ", diff[i]);
    NEXTLINE;
    return 0;
}
```

### 2.4.3 二维前缀和.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：二维前缀和.cpp
 *   创建日期：2021年05月31日 星期一 11时08分42秒
 *   题    目：AcWing 0796 子矩阵的和
 *   算    法：<++>
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1005;
typedef long long ll;
int n, m, q;
int g[maxn][maxn];
ll preS[maxn][maxn];

int main() {
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            scanf("%d", &g[i][j]);
            preS[i][j] = g[i][j] + preS[i-1][j] + preS[i][j-1] - preS[i-1][j-1];
            if (len(str) > 6):
                continue
        }
    while (q--) {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        int minx = min(x1, x2);
        int miny = min(y1, y2);
        int maxx = max(x1, x2);
        int maxy = max(y1, y2);
        ll res = preS[maxx][maxy] - preS[maxx][miny-1] - preS[minx-1][maxy] + preS[minx-1][miny-1];
        printf("%lld\n", res);
    }
    return 0;
}


/*
 * 1 7 2 4
 * 1 4 2 7
 * 2 7 1 4
 * 2 4 1 7
 * tmpx = min(x1, x2);
 * tmpy = min(y1, y2);
 *
 * ------o--------------------
 * ---o------------------------
 * ----------------------------
 * ----------------------------
 * ----------------------------
```

```
 * --------------------------
 */
```

### 2.4.4 二维差分.cpp

```cpp
#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 1e3 + 5;
int g[maxn][maxn];
int diff[maxn][maxn];
int n, m, q;

void insert(int x1, int y1, int x2, int y2, int c) {
        diff[x1][y1] += c;
        diff[x2+1][y1] -= c;
        diff[x1][y2+1] -= c;
        diff[x2+1][y2+1] += c;
}

int main() {
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            scanf("%d", &g[i][j]);
            insert(i, j, i, j, g[i][j]);
        }
    while (q--) {
        int x1, y1, x2, y2, c;
        scanf("%d %d %d %d %d", &x1, &y1, &x2, &y2, &c);
        insert(x1, y1, x2, y2, c);
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            diff[i][j] = diff[i][j] + diff[i-1][j] + diff[i][j-1] - diff[i-1][j-1];
            printf("%d ", diff[i][j]);
        }
        NEXTLINE;
    }
    return 0;
}
```

### 2.4.5 异或前缀和.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：Color-Sequence.cpp
 *   创建日期：2021年10月21日 星期四 09时52分03秒
 *   题    目：2020 ICPC 江西省大学生程序设计竞赛
 *   算    法：异或前缀和
 *   描    述：给定一个颜色序列
 *   求它有多少个颜色出现次数都是偶数的连续子序列。
 *   颜色数量不超过 20 种
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <unordered_map>
using namespace std;
typedef long long ll;
const int maxn = 1e6 + 5;
int color[maxn], preS[maxn];
unordered_map<int, int> ha;

int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; ++ i) {
        scanf("%d", &color[i]);
```

```
        preS[i] = preS[i - 1] ^ (1 << color[i]);   // 异或前缀和
    }
    // 现在得到的 preS 数组中存储了一个数
    // 如果这个数的第 i 位是 0，则表明颜色 i 出现了偶数次
    ll res = 0;
    ha[0] = 1;   // 这种情况一定是偶数的
    // 找同值前缀和
    // 如果 preS[a] == preS[b]，则 str[a ~ b] 是偶数的
    for (int i = 1; i <= n; ++ i) {
        res += ha[preS[i]];
        ++ ha[preS[i]];
    }
    printf("%lld\n", res);
    return 0;
}
```

## 2.5 二分

### 2.5.1 整数集合上的二分

#### 2.5.1.1 数的范围.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：lower-upper-bound.cpp
 *   创建日期：2021年08月17日 星期二 13时44分44秒
 *   题    目：AcWing 0789 数的范围
 *   算    法：二分
 *   描    述：对于每个查询，返回一个元素 k 的起始位置和终止位置(从0开始)
 *      如果数组中不存在该元素，则返回 -1 -1。
 *      也就是lowerBound和upperBound
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#define bug printf("<-->\n");
const int maxn = 1e5 + 5;
int n, q;
int num[maxn];

/*
 * 找到第一个大于等于val的数：
 * [l, r] --> [l, mid], [mid + 1, r]
 * 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用：
 */
int lowerBound(int l, int r, int val) {
    while (l < r) {
        int mid = (l + r) >> 1;
        if (num[mid] < val)
            l = mid + 1;
        else
            r = mid;
    }
    if (num[l] != val)
        return -1;
    else
        return l;
}

/*
 * 找到第一个小于等于val的数：
 * [l, r] --> [l, mid - 1], [mid, r]
 * 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用：
 */
int upperBound(int l, int r, int val) {
    while (l < r) {
        //这里要加一
        int mid = (l + r + 1) >> 1;
        if (num[mid] <= val)
            l = mid;
```

```
        else
            r = mid - 1;
    }
    if (num[l] != val)
        return -1;
    else
        return l;
}

int main() {
    scanf("%d %d", &n, &Q);
    for (int i = 0; i < n; ++i)
        scanf("%d", &num[i]);
    while (Q--) {
        int val;
        scanf("%d", &val);
        int lower = lowerBound(0, n-1, val);
        int upper = upperBound(0, n-1, val);
        printf("%d %d\n", lower, upper);
    }
    return 0;
}
```

### 2.5.2 实数域上的二分

#### 2.5.2.1 数的三次方根.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01-三次方根.cpp
 *   创建日期：2021年08月17日 星期二 13时50分14秒
 *   题    目：AcWing 0790 数的三次方根
 *   算    法：二分
 *   描    述：给定一个浮点数 n，求它的三次方根，保留6位小数
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const double eps = 1e-8;

bool judge(double mid, double n) {
    if (mid * mid * mid < n)
        return true;
    else
        return false;
}

// 0.001
double bsearch_3(double n) {
    double l = -10000, r = 10000;
    while (r - l > eps) {
        double mid = (l + r) / 2;
        if (judge(mid, n))
            l = mid;
        else
            r = mid;
    }
    return r;
}

int main() {
    double n;
    scanf("%lf", &n);
    double res = bsearch_3(n);
    printf("%.6f\n", res);
    return 0;
}
```

### 2.5.3 二分答案转化为判定

#### 2.5.3.1 根据厚度将书分组.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01-根据厚度将书分组.cpp
 *   创建日期：2021年05月28日 星期五 22时43分04秒
 *   题    目：<++>
 *   算    法：二分
 *   描    述：N本书排成一行，第i本厚度是book[i]，把它们分成连续的M组
 *          使T(厚度最大的一组的厚度)最小，最小值是多少
 *
 -----------------------------------------------------------------*/

#include <cstdio>
int n = 9, m = 3;
int book[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

//将n本书分成若干组，每组最大厚度不超过size，分成的组数是否小于m
bool valid(int size) {
    int thick = 0;
    int group = 1; //初始值要为1
    for (int i = 0; i < n; ++i) {
        if (size - thick >= book[i])
            thick += book[i];
        else {
            group++;
            thick = book[i];
        }
    }
    return group <= m;
}

int main() {
    int sum_thick = 0;
    for (int i = 0; i < n; ++i)
        sum_thick += book[i];

    int l = 0, r = sum_thick;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (valid(mid))
            r = mid;
        else
            l = mid + 1;
    }
    printf("%d\n", l);
    return 0;
}
```

## 2.6 双指针

### 2.6.1 双指针.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01-two-pointers.cpp
 *   创建日期：2021年06月05日 星期六 08时00分00秒
 *   题    目：<++>
 *   算    法：<++>
 *   描    述：有双指针这个算法
 *
 -----------------------------------------------------------------*/

#include <stdio.h>
int main() {
    int a[] = {0, 1, 2, 3, 4, 5, 6};
    int M = 8;
    int i = 1;
```

```
    int j = 6;
    while (i < j) {
        if (a[i] + a[j] == M)
            printf("%d %d\n", i++, j--);
        else if (a[i] + a[j] < M)
            i++;
        else
            j--;
    }
    return 0;
}
```

# 3 搜索

## 3.1 深度优先搜索 (DFS)

### 3.1.1 n-皇后问题 (1).cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年08月15日 星期日 12时03分56秒
 *   题    目：AcWing 0843 n-皇后问题
 *   算    法：深度优先搜索
 *   描    述：使用排列组合的方法解决
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 10;
#define NEXTLINE puts("");
int n, path[maxn];
char g[maxn][maxn];
bool col[maxn], dg[2 * maxn], udg[2 * maxn];

void DFS(int u) {
    if (u == n) {
        for (int i = 0; i < n; ++i)
            printf("%s\n", g[i]);
        NEXTLINE;
        return ;
    }
    // 第u行第i列
    for (int i = 0; i < n; ++i)
        /**
         * 可以看成一个坐标轴 y = u + i, y = n - u + i;
         * ------------------>
         * |
         * |
         * |
         * |
         * |
         * |
         * |
         * V
         *
         */
        if (!col[i] && !dg[u + i] && !udg[n - u + i]) {
            g[u][i] = 'Q';
            col[i] = dg[u + i] = udg[n - u + i] = true;  // 列，正对角线，反对角线标记
            DFS(u + 1);
            col[i] = dg[u + i] = udg[n - u + i] = false;
            g[u][i] = '.';
        }
}

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

```
        for (int j = 0; j < n; ++j)
            g[i][j] = '.';
    DFS(0);
    return 0;
}
```

### 3.1.2 n-皇后问题 (2).cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：n-皇后问题(2).cpp
 *   创建日期：2021年08月15日 星期日 13时18分42秒
 *   题    目：AcWing 0843 n-皇后问题
 *   算    法：深度优先搜索
 *   描    述：这一种更加的暴力
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 10;
#define NEXTLINE puts("");
int n;
char g[maxn][maxn];
bool row[maxn], col[maxn], dg[2 * maxn], udg[2 * maxn];

void DFS(int x, int y, int s) {
    if (y == n)
        y = 0, x ++ ;
    if (x == n) {
        if (s == n) {
            for (int i = 0; i < n; ++i)
                printf("%s\n", g[i]);
            NEXTLINE;
        }
        return ;
    }
    // 不放皇后
    DFS(x, y + 1, s);

    // 放皇后
    if (!row[x] && !col[y] && !dg[x + y] && !udg[n - x + y]) {
        g[x][y] = 'Q';
        row[x] = col[y] = dg[x + y] = udg[n - x + y] = true;
        DFS(x, y + 1, s + 1);
        row[x] = col[y] = dg[x + y] = udg[n - x + y] = false;
        g[x][y] = '.';
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            g[i][j] = '.';
    DFS(0, 0, 0);   // 从左上角开始搜，记录当前有多少个皇后
    return 0;
}
```

### 3.1.3 排列数字.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年08月11日 星期三 11时14分32秒
 *   题    目：AcWing 0842 排列数字
 *   算    法：深度优先搜索
 *   描    述：排列数字，将[1, n]的数字排列
```

```
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 10;
#define NEXTLINE puts("");
int path[maxn];
bool used[maxn];
int n;

void DFS(int u) {
    if (u == n) {
        for (int i = 0; i < n; ++i)
            printf("%d ", path[i]);
        NEXTLINE
        return;
    }
    for (int i = 1; i <= n; ++i)
        if (!used[i]) {
            path[u] = i;
            used[i] = true;
            DFS(u + 1);
            used[i] = false;
        }
}

int main() {
    scanf("%d", &n);
    DFS(0);
    return 0;
}
```

### 3.1.4 搜索顺序

#### 3.1.4.1 马走日.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：马走日.cpp
 *   创建日期：2021年11月26日 星期五 19时55分53秒
 *   题    目：AcWing 1116 马走日
 *   算    法：深度优先搜索
 *   描    述：在一个棋盘上，给出马的坐标，马可以有多少种途径遍历
 *   棋盘上的 *所有点*
 *
 ----------------------------------------------------------------*/

#include <cstdio>
int n, m;
const int maxn = 10;
int g[maxn][maxn];
bool used[maxn][maxn];
int res;
const int dx[] = {-1, -2, -2, -1, 1, 2, 2, 1};
const int dy[] = {-2, -1, 1, 2, 2, 1, -1, -2};

void DFS(int x, int y, int cnt) {
    if (cnt == n * m) {
        res ++ ;
        return;
    }
    used[x][y] = true;
    for (int i = 0; i < 8; ++ i) {
        int nx = x + dx[i],
            ny = y + dy[i];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (used[nx][ny]) continue;
        DFS(nx, ny, cnt + 1);
    }
    used[x][y] = false;
```

```
}

int main() {
    int t; scanf("%d", &t);
    while (t -- ) {
        int x, y;
        scanf("%d %d %d %d", &n, &m, &x, &y);
        res = 0;
        DFS(x, y, 1);
        printf("%d\n", res);
    }
    return 0;
}
```

### 3.1.5 连通性模型

#### 3.1.5.1 红与黑.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：红与黑.cpp
 *   创建日期：2021年11月26日 星期五 19时35分22秒
 *   题    目：AcWing 1113 红与黑
 *   算    法：深度优先搜索 洪泛法
 *   描    述：把 @ 所在的 . 的区域大小输出
 *
 *   ....#.
 *   .....#
 *   ......
 *   ......
 *   ......
 *   ......
 *   ......
 *   #@...#
 *   .#..#.
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 25;
int n, m;
char g[maxn][maxn];
int dx[] = {0, -1, 0, 1};
int dy[] = {-1, 0, 1, 0};

int DFS(int x, int y) {
    int res = 1;
    g[x][y] = '#';
    for (int i = 0; i < 4; ++ i) {
        int nx = x + dx[i],
            ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && g[nx][ny] == '.')
            res += DFS(nx, ny);
    }
    return res;
}

int main() {
    while (scanf("%d %d", &m, &n) && (n || m)) {
        for (int i = 0; i < n; ++ i)
            scanf("%s", g[i]);
        int x, y;
        for (int i = 0; i < n; ++ i)
            for (int j = 0; j < m; ++ j)
                if (g[i][j] == '@') {
                    x = i, y = j;
                    break;
                }
        printf("%d\n", DFS(x, y));
    }
```

```
    return 0;
}
```

#### 3.1.5.2 迷宫.cpp

```
/*--------------------------------------------------------------------
 *
 *   文件名称：迷宫.cpp
 *   创建日期：2021年11月26日 星期五 18时38分26秒
 *   题    目：AcWing 1112 迷宫
 *   算    法：深度优先搜索
 *   描    述：
 *   .....
 *   ###.#
 *   ..#..
 *   ###..
 *   ...#.
 *
 *   会给出 A 和 B 的坐标，问是否可以从 A 到 B
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 110;
int n, ax, ay, bx, by;
char g[maxn][maxn];
bool used[maxn][maxn];
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};

bool DFS(int x, int y) {
    if (g[x][y] == '#')
        return false;
    if (x == bx && y == by)
        return true;
    used[x][y] = true;
    for (int i = 0; i < 4; ++ i) {
        int nx = x + dx[i],
            ny = y + dy[i];
        if (nx < 0 || nx >= n || ny < 0 || ny >= n) continue;
        if (used[nx][ny]) continue;
        if (g[nx][ny] == '#') continue;
        if (DFS(nx, ny))
            return true;
    }
    return false;
}

int main() {
    int t; scanf("%d", &t);
    while (t -- ) {
        scanf("%d", &n);
        for (int i = 0; i < n; ++ i)
            scanf("%s", g[i]);
        memset(used, 0, sizeof used);
        scanf("%d %d %d %d", &ax, &ay, &bx, &by);
        if (DFS(ax, ay))
            puts("YES");
        else
            puts("NO");
    }
    return 0;
}
```

## 3.2 宽度优先搜索 (BFS)

### 3.2.1 八数码.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：八数码.cpp
 *   创建日期：2021年11月26日 星期五 10时35分46秒
 *   题    目：AcWing 0179 八数码问题
 *   算    法：宽度优先搜索
 *   描    述：<++>
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <string>
#include <queue>
#include <unordered_map>
#include <iostream>
using namespace std;
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};
const char op[] = "lurd";

string BFS(string start) {
    string over = "12345678x";
    queue<string> q;
    unordered_map<string, pair<string, char>> pre;
    unordered_map<string, bool> used;
    q.push(start);
    used[start] = true;
    while (q.size()) {
        string t = q.front(); q.pop();
        if (t == over)
            break;
        int x, y;
        for (int i = 0; i < t.size(); ++ i)
            if (t[i] == 'x')
                x = i / 3, y = i % 3;
        string source = t;
        for (int i = 0; i < 4; ++ i) {
            int nx = x + dx[i],
                ny = y + dy[i];
            if (nx < 0 || nx >= 3 || ny < 0 || ny >= 3)
                continue;
            swap(t[x * 3 + y], t[nx * 3 + ny]);
            if (!used[t]) {
                used[t] = true;
                q.push(t);
                pre[t] = {source, op[i]};
            }
            swap(t[x * 3 + y], t[nx * 3 + ny]);
        }
    }
    string res;
    while (over != start) {
        res += pre[over].second;
        over = pre[over].first;
    }
    reverse(res.begin(), res.end());
    return res;
}

int main() {
    string g, seq, c;
    while (cin >> c) {
        g += c;
        if (c != "x")
            seq += c;
    }
    int cnt = 0;
    for (int i = 0; i < seq.size(); ++ i)
```

```
        for (int j = i + 1; j < seq.size(); ++ j)
            if (seq[i] > seq[j])
                cnt ++ ;
    if (cnt & 1)
        cout << "unsolvable" << endl;
    else
        cout << BFS(g) << endl;
    return 0;
}
```

### 3.2.2 双端队列宽搜

#### 3.2.2.1 电路维修.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：电路维修.cpp
 *   创建日期：2021年11月24日 星期三 21时23分48秒
 *   题    目：AcWing 0175 电路维修
 *   算    法：双端队列宽度优先搜索
 *   描    述：
 *   \\/\\
 *   \\///
 *   /\\\\
 *
 *   对于上面这个电路，电从左上角进入，转动最少的电线能使电从右下角
 *   离开
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <deque>
using namespace std;
const int maxn = 505;
int n, m;
char g[maxn][maxn];
typedef pair<int, int> PII;
int dist[maxn][maxn];
bool used[maxn][maxn];
const int dx[] = {-1, -1, 1, 1}, dy[] = {-1, 1, 1, -1};
const int ix[] = {-1, -1, 0, 0}, iy[] = {-1, 0, 0, -1};
#define x first
#define y second

int BFS() {
    deque<PII> q;
    memset(used, 0, sizeof used);
    memset(dist, 0x3f, sizeof dist);
    char cs[5] = "\\/\\/";
    dist[0][0] = 0;
    q.push_back({0, 0});
    while (q.size()) {
        auto t = q.front(); q.pop_front();
        if (t.x == n && t.y == m)
            return dist[t.x][t.y];
        if (used[t.x][t.y])
            continue;
        used[t.x][t.y] = true;
        for (int i = 0; i < 4; ++ i) {
            int x = t.x + dx[i],
            y = t.y + dy[i];
            if (x < 0 || x > n || y < 0 || y > m) continue;
            int ga = t.x + ix[i], gb = t.y + iy[i];
            int w = (g[ga][gb] != cs[i]);
            int d = dist[t.x][t.y] + w;
            if (d <= dist[x][y]) {
                dist[x][y] = d;
                if (!w)
                    q.push_front({x, y});
```

```
                else
                    q.push_back({x, y});
            }
        }
    }
    return -1;
}

int main() {
    int t; scanf("%d", &t);
    while (t -- ) {
        scanf("%d %d", &n, &m);
        for (int i = 0; i < n; ++ i)
            scanf("%s", g[i]);
        if ((n + m) & 1)
            puts("NO SOLUTION");
        else
            printf("%d\n", BFS());
    }
    return 0;
}
```

### 3.2.3 多源 BFS

#### 3.2.3.1 矩阵距离.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：矩阵距离.cpp
 *   创建日期：2021年11月24日 星期三 13时03分49秒
 *   题    目：AcWing 0173 矩阵距离
 *   算    法：多源BFS
 *   描    述：输入一个01矩阵，输出每个0到达最近的1的曼哈顿距离
 *
 *   3 4
 *   0001
 *   0011
 *   0110
 *
 *   3 2 1 0
 *   2 1 0 0
 *   1 0 0 1
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int maxn = 1e3 + 5;
int n, m;
char g[maxn][maxn];
int dist[maxn][maxn];
typedef pair<int, int> PII;
#define NEXTLINE puts("");
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};

void BFS() {
    memset(dist, -1, sizeof dist);
    queue<PII> q;
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j)
            if (g[i][j] == '1') {
                q.push({i, j});
                dist[i][j] = 0;
            }
    while (q.size()) {
        auto t = q.front();
        q.pop();
```

```
        for (int i = 0; i < 4; ++ i) {
            int x = t.first + dx[i],
                y = t.second + dy[i];
            if (0 <= x && x < n && 0 <= y && y < m && dist[x][y] == -1) {
                dist[x][y] = dist[t.first][t.second] + 1;
                q.push({x, y});
            }
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i)
        scanf("%s", g[i]);
    BFS();
    for (int i = 0; i < n; ++ i) {
        for (int j = 0; j < m; ++ j)
            printf("%d ", dist[i][j]);
        NEXTLINE;
    }
    return 0;
}
```

### 3.2.4 最小步数模型

#### 3.2.4.1 魔板.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：魔板.cpp
 *   创建日期：2021年11月24日 星期三 18时03分34秒
 *   题    目：AcWing 1107 魔板
 *   算    法：BFS最小步数模型
 *   描    述：输入 2 * 4 的矩阵，求出经过最少的哪些操作可以使
 *   初态矩阵 1 2 3 4 转化为输入的矩阵
 *            8 7 6 5
 *   矩阵的排列就是按上面初态矩阵的排列
 *   比如输入 8 7 6 5 4 3 2 1
 *
 *   那么输入的矩阵就是：
 *   8 7 6 5
 *   1 2 3 4
 *
 *   可以的操作有：
 *   A：交换上下两行；
 *   B：将最右边的一列插入到最左边；
 *   C：魔板中央对的4个数作顺时针旋转。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <unordered_map>
#include <string>
#include <queue>
#include <iostream>
#include <algorithm>
using namespace std;
int g[2][4];
unordered_map<string, int> dist;
unordered_map<string, pair<char, string>> pre;
queue<string> q;

void set(string str) {
    for (int i = 0; i < 4; ++ i)
        g[0][i] = str[i];
    for (int i = 3, j = 4; i >= 0; -- i, ++ j)
        g[1][i] = str[j];
}

```

```
string get() {
    string res;
    for (int i = 0; i < 4; ++ i)
        res += g[0][i];
    for (int i = 3; i >= 0; -- i)
        res += g[1][i];
    return res;
}

string move0(string str) {
    set(str);
    for (int i = 0; i < 4; ++ i)
        swap(g[0][i], g[1][i]);
    return get();
}

string move1(string str) {
    set(str);
    char g03 = g[0][3], g13 = g[1][3];
    for (int i = 3; i > 0; -- i) {
        g[0][i] = g[0][i - 1];
        g[1][i] = g[1][i - 1];
    }
    g[0][0] = g03;
    g[1][0] = g13;
    return get();
}

string move2(string str) {
    set(str);
    char g01 = g[0][1];
    g[0][1] = g[1][1];
    g[1][1] = g[1][2];
    g[1][2] = g[0][2];
    g[0][2] = g01;
    return get();
}

void BFS(string start, string end) {
    if (start == end)
        return;
    q.push(start);
    dist[start] = 0;
    while (q.size()) {
        auto t = q.front(); q.pop();
        string m[3];
        m[0] = move0(t);
        m[1] = move1(t);
        m[2] = move2(t);

        for (int i = 0; i < 3; ++ i) {
            string str = m[i];
            if (dist.count(str) == 0) {
                dist[str] = dist[t] + 1;
                pre[str] = {char(i + 'A'), t};
                if (str == end)
                    break;
                q.push(str);
            }
        }
    }
}

int main() {
    int x;
    string start, end;
    for (int i = 0; i < 8; ++ i) {
        cin >> x;
        end += char(x + '0');
    }
```

```
    for (int i = 0; i < 8; ++ i)
        start += char(i + '1');
    BFS(start, end);
    cout << dist[end] << endl;
    string res;
    while (end != start) {
        res += pre[end].first;
        end = pre[end].second;
    }
    reverse(res.begin(), res.end());
    if (res.length() > 0)
        cout << res << endl;
    return 0;
}
```

### 3.2.5 最短路模型

#### 3.2.5.1 抓住那头牛.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：抓住那头牛.cpp
 *   创建日期：2021年11月23日 星期二 23时43分17秒
 *   题    目：AcWing 1100 抓住那头牛
 *   算    法：最短路模型
 *   描    述：农夫位于坐标轴的点 N，牛位于 K
 *   农夫有两种移动方式：
 *       1. x - 1，x + 1
 *       2. 2 * x
 *   每种移动都花费一分种
 *
 *   农夫最少要花多少时间抓住牛,牛不动
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5;
int n, k, q[maxn];
bool used[maxn];
int dist[maxn];

int BFS() {
    memset(dist, -1, sizeof dist);
    dist[n] = 0;
    int hh = 0, tt = -1;
    q[ ++ tt] = n;
    while (hh <= tt) {
        int t = q[hh ++ ];
        if (t == k)
            return dist[k];
        if (t + 1 < maxn && dist[t + 1] == -1) {
            dist[t + 1] = dist[t] + 1;
            q[ ++ tt] = t + 1;
        }
        if (t - 1 >= 0 && dist[t - 1] == -1) {
            dist[t - 1] = dist[t] + 1;
            q[ ++ tt] = t - 1;
        }
        if (t * 2 <= maxn && dist[t * 2] == -1) {
            dist[t * 2] = dist[t] + 1;
            q[ ++ tt] = t * 2;
        }
    }
    return -1;
}

int main() {
    scanf("%d %d", &n, &k);
    printf("%d\n", BFS());
```

```
    return 0;
}
```

#### 3.2.5.2 武士风度的牛.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：武士风度的牛.cpp
 *   创建日期：2021年11月23日 星期二 22时19分20秒
 *   题    目：AcWing 0188 武士风度的牛
 *   算    法：最短路模型
 *   描    述：马走日
 *   . . . . . . . . . .
 *   . . . . * . . . . .
 *   . . . . . . . . . .
 *   . . . * . * . . . .
 *   . . . . . . . * . .
 *   . . * . . * . . . H
 *   * . . . . . . . . .
 *   . . . * . . . * . .
 *   . K . . . . . . . .
 *   . . . * . . . . . *
 *   . . * . . . . * . .
 *
 *   从 K 到 H，* 是障碍，没有别马腿
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <utility>
#include <queue>
using namespace std;
typedef pair<int, int> PII;
const int maxn = 155;
int n, m;
char g[maxn][maxn];
int dist[maxn][maxn];
const int dx[] = {-1, -2, -2, -1, 1, 2, 2, 1};
const int dy[] = {-2, -1, 1, 2, 2, 1, -1, -2};
#define x first
#define y second

int BFS(PII start, PII end) {
    queue<PII> q;
    memset(dist, -1, sizeof dist);
    dist[start.x][start.y] = 0;
    q.push(start);
    while (q.size()) {
        auto t = q.front();
        q.pop();
        for (int i = 0; i < 8; ++ i)
            for (int j = 0; j < 8; ++ j) {
                int x = t.x + dx[i],
                    y = t.y + dy[i];
                if (x < 0 || x >= n || y < 0 || y >= m) continue;
                if (g[x][y] == '*') continue;
                if (dist[x][y] != -1) continue;
                dist[x][y] = dist[t.x][t.y] + 1;
                if (make_pair(x, y) == end)
                    return dist[x][y];
                q.push({x, y});
            }
    }
    return -1;   // 本题保证有解
}

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

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

    PII start, end;
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j) {
            if (g[i][j] == 'K')
                start = {i, j};
            if (g[i][j] == 'H')
                end = {i, j};
        }
    printf("%d\n", BFS(start, end));
    return 0;
}
```

#### 3.2.5.3 迷宫问题.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：迷宫问题.cpp
 *   创建日期：2021年11月23日 星期二 19时50分36秒
 *   题    目：AcWing 1076 迷宫问题
 *   算    法：最短路模型
 *   描    述：输出从左上角到右下角的最短路线，0表示可以走，1表示
 *   不可以走
 *   0 1 0 0 0
 *   0 1 0 1 0
 *   0 0 0 0 0
 *   0 1 1 1 0
 *   0 0 0 1 0
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <utility>
#include <cstring>
using namespace std;
const int maxn = 1e3 + 5;
const int maxm = maxn * maxn;
typedef pair<int, int> PII;
int n, g[maxn][maxn];
PII q[maxm];
PII pre[maxn][maxn];
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};
#define x first
#define y second

void BFS(int x, int y) {
    int hh = 0, tt = -1;
    q[ ++ tt] = {x, y};
    memset(pre, -1, sizeof pre);
    pre[x][y] = {0, 0};
    while (hh <= tt) {
        PII t = q[hh ++ ];
        for (int i = 0; i < 4; ++ i) {
            int nx = t.x + dx[i],
                ny = t.y + dy[i];
            if (nx < 0 || nx >= n || ny < 0 || ny >= n) continue;
            if (g[nx][ny]) continue;
            if (pre[nx][ny].x != -1) continue;
            q[ ++ tt] = {nx, ny};
            pre[nx][ny] = t;
        }
    }
}

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

```
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < n; ++ j)
            scanf("%d", &g[i][j]);
    BFS(n - 1, n - 1);
    // pre 中存储的是结果路线中每个位置的前一个位置
    PII end(0, 0);
    while (true) {
        printf("%d %d\n", end.x, end.y);
        if (end.x == n - 1 && end.y == n - 1)
            break;
        end = pre[end.x][end.y];
    }
    return 0;
}
```

### 3.2.6 洪泛法

#### 3.2.6.1 山峰和山谷.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：山峰和山谷.cpp
 *   创建日期：2021年11月23日 星期二 09时14分20秒
 *   题    目：AcWing 1106 山峰和山谷
 *   算    法：洪泛法
 *   描    述：就是找山峰山谷
 *
 *   8 8 8 7 7
 *   7 7 8 8 7
 *   7 7 7 7 7
 *   7 8 8 7 8
 *   7 8 8 8 8
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <utility>
#include <algorithm>
using namespace std;
typedef pair<int, int> PII;
#define x first
#define y second
const int maxn = 1e3 + 5, maxm = maxn * maxn;
int n;
int h[maxn][maxn];
PII q[maxm];
bool used[maxn][maxn];

void BFS(int x, int y, bool &has_lower, bool &has_higher) {
    int hh = 0, tt = -1;
    q[ ++ tt] = {x, y};
    used[x][y] = true;
    while (hh <= tt) {
        PII t = q[hh ++ ];
        for (int i = t.x - 1; i <= t.x + 1; ++ i)
            for (int j = t.y - 1; j <= t.y + 1; ++ j) {
                if (i == t.x && j == t.y) continue;
                if (i < 0 || i >= n || j < 0 || j >= n) continue;
                if (h[i][j] != h[t.x][t.y]) {
                    if (h[i][j] > h[t.x][t.y])
                        has_higher = true;
                    else
                        has_lower = true;
                }
                else if (!used[i][j]) {
                    q[ ++ tt] = {i, j};
                    used[i][j] = true;
                }
            }
    }
```

```
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < n; ++ j)
            scanf("%d", &h[i][j]);
    // 山峰，山谷
    int peak = 0, valley = 0;
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < n; ++ j)
            if (!used[i][j]) {
                bool has_higher = false, has_lower = false;
                BFS(i, j, has_lower, has_higher);
                if (!has_lower)
                    valley ++ ;
                if (!has_higher)
                    peak ++ ;
            }
    printf("%d %d\n", peak, valley);
    return 0;
}
```

#### 3.2.6.2 池塘计数.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：池塘计数.cpp
 *   创建日期：2021年11月22日 星期一 21时47分37秒
 *   题    目：AcWing 1097 池塘计数
 *   算    法：洪泛法
 *   描    述：雨水是 'W'，'.' 是土地，每个单元格与相邻的八个格子相连
 *   问共有多少片相连的 'W' 块。
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <utility>
using namespace std;
int n, m;
const int maxn = 1e3 + 5;
const int maxm = maxn * maxn;
const int dx[] = {-1, -1, 0, 1, 1, 1, 0, -1};
const int dy[] = {0, -1, -1, -1, 0, 1, 1, 1};
bool used[maxn][maxn];
#define x first
#define y second
typedef pair<int, int> PII;

PII q[maxm];
char g[maxn][maxn];

void BFS(int x, int y) {
    int hh = 0, tt = -1;
    q[ ++ tt] = {x, y};
    used[x][y] = true;

    while (hh <= tt) {
        PII t = q[hh ++ ];
        for (int i = t.x - 1; i <= t.x + 1; ++ i)
            for (int j = t.y - 1; j <= t.y + 1; ++ j) {
                if (i == t.x && j == t.y)   // 把一个 3 * 3 的矩阵中间挖掉
                    continue;
                if (i < 0 || i >= n || j < 0 || j >= m)
                    continue;
                if (g[i][j] == '.' || used[i][j])
                    continue;
                q[ ++ tt] = {i, j};
                used[i][j] = true;
```

```
            }
        }
    }

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i)
        scanf("%s", g[i]);
    int cnt = 0;
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j)
            if (g[i][j] == 'W' && !used[i][j]) {
                BFS(i, j);
                cnt ++ ;
            }
    printf("%d\n", cnt);
    return 0;
}
```

### 3.2.7 走迷宫.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：走迷宫.cpp
 *   创建日期：2021年10月07日 星期四 10时44分54秒
 *   题    目：AcWing 0844 走迷宫
 *   算    法：宽度优先搜索，泛洪法
 *   描    述：
 *   起点：(1, 1)   终点：(n, m)   1 为障碍物   求到终点最少移动距离
 *   题目保证有解
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <utility>
#include <cstring>
using namespace std;
const int maxn = 100 + 5;
// 上 右 下 左 上左 上右 下右 下左
const int dirx[] = {-1, 0, 1, 0, -1, -1, 1, 1};
const int diry[] = {0, 1, 0, -1, -1, 1, 1, -1};
typedef pair<int, int> PII;
int n, m;
int g[maxn][maxn];
int d[maxn][maxn];
PII quu[maxn * maxn];   // 考试时使用 q 就好
int hh = 0, tt = -1;

// 插入 quu[++ tt] = x;
// 弹出 hh ++ ;

void BFS() {
    quu[++ tt] = {0, 0};   // 起点
    memset(d, -1, sizeof d);   // 有 used 的作用
    d[0][0] = 0;
    while (hh <= tt) {
        auto t = quu[hh ++ ];
        // printf("%d %d\n", t.first, t.second);
        for (int i = 0; i < 4; ++ i) {
            int newx = t.first + dirx[i];   // 考试时使用 x, y 就好
            int newy = t.second + diry[i];
            if (newx >= 0 && newx < n && newy >= 0 && newy < m && g[newx][newy] == 0 && d[newx][newy] == -1)
    {
                d[newx][newy] = d[t.first][t.second] + 1;
                quu[ ++ tt] = {newx, newy};
            }
        }
    }
}
```

```

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j)
            scanf("%d", &g[i][j]);
    BFS();
    printf("%d\n", d[n - 1][m - 1]);
    return 0;
}
```

## 3.3 双向搜索

### 3.3.1 字串变换.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：字串变换.cpp
 *   创建日期：2021年11月24日 星期三 22时37分50秒
 *   题    目：AcWing 0190 字串变换
 *   算    法：双向搜索
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <queue>
#include <unordered_map>
#include <string>
#include <iostream>
using namespace std;
const int maxn = 6;
string A, B;
string a[maxn], b[maxn];
int n;

int extend(queue<string>& q, unordered_map<string, int>& da, unordered_map<string, int>& db,
        string a[maxn], string b[maxn]) {
    int d = da[q.front()];
    while (q.size() && da[q.front()] == d) {
        auto t = q.front(); q.pop();
        for (int i = 0; i < n; ++ i)
            for (int j = 0; j < t.size(); ++ j)
                if (t.substr(j, a[i].size()) == a[i]) {
                    string str = t.substr(0, j) + b[i] + t.substr(j + a[i].size());
                    if (db.count(str))
                        return da[t] + db[str] + 1;
                    if (da.count(str))
                        continue;
                    da[str] = da[t] + 1;
                    q.push(str);
                }
    }
    return 11;
}

int BFS() {
    if (A == B)
        return 0;
    queue<string> qa, qb;
    unordered_map<string, int> da, db;

    qa.push(A), qb.push(B);
    da[A] = db[B] = 0;

    int step = 0;
    while (qa.size() && qb.size()) {
        int t;
        if (qa.size() < qb.size())
```

```
            t = extend(qa, da, db, a, b);
        else
            t = extend(qb, db, da, b, a);
        if (t <= 10)
            return t;
        if ( ++ step == 10)
            return -1;
    }
    return -1;
}

int main() {
    cin >> A >> B;
    while (cin >> a[n] >> b[n])
        n ++ ;
    int t = BFS();
    if (t == -1)
        puts("NO ANSWER!");
    else
        cout << t << endl;
    return 0;
}
```

## 3.4 A*

### 3.4.1 八数码.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：八数码.cpp
 *   创建日期：2021年11月25日 星期四 11时27分52秒
 *   题    目：AcWing 0179 八数码
 *   算    法：astar 直接使用 BFS + cantor 也可以，代码在
 *           数学 -> 组合数学 -> 康托展开
 *   描    述：将输入的一个八数码通过移动 x 来转换为初始八数码
 *
 *   1 2 3            1 2 3
 *   x 4 6     ->     4 5 6
 *   7 5 8            7 8 x
 *
 *   u : 向上
 *   d : 向下
 *   l : 向左
 *   r : 向右
 *
 *   八数码问题无解当且仅当逆序对数量是奇数。
 *
 ----------------------------------------------------------------*/
#include <cstdio>
#include <string>
#include <iostream>
#include <queue>
#include <unordered_map>
#include <algorithm>
using namespace std;
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};
const char op[] = "lurd";

int f(string state) {
    int res = 0;
    for (int i = 0; i < state.size(); ++ i)
        if (state[i] != 'x') {
            int t = state[i] - '1';
            res += abs(i / 3 - t / 3) + abs(i % 3 - t % 3);
        }
    return res;
}

```

```
string BFS(string start) {
    string over = "12345678x";
    unordered_map<string, int> dist;
    unordered_map<string, bool> used;
    unordered_map<string, pair<string, char>> prev;
    priority_queue<pair<int, string>, vector<pair<int, string>>, greater<pair<int, string>>> he;
    he.push({f(start), start});
    dist[start] = 0;
    while (he.size()) {
        auto t = he.top(); he.pop();
        string state = t.second;
        if (state == over)
            break;
        if (used[state])
            continue;
        used[state] = true;
        int step = dist[state];
        int x, y;
        for (int i = 0; i < state.size(); ++ i)
            if (state[i] == 'x')
                x = i / 3, y = i % 3;
        string source = state;
        for (int i = 0; i < 4; ++ i) {
            int nx = x + dx[i],
                ny = y + dy[i];
            if (nx < 0 || nx >= 3 || ny < 0 || ny >= 3)
                continue;
            swap(state[x * 3 + y], state[nx * 3 + ny]);
            if (!dist.count(state) || dist[state] > step + 1) {
                dist[state] = step + 1;
                prev[state] = {source, op[i]};
                he.push({dist[state] + f(state), state});
            }
            swap(state[x * 3 + y], state[nx * 3 + ny]);
        }
    }
    string res;
    while (over != start) {
        res += prev[over].second;
        over = prev[over].first;
    }
    reverse(res.begin(), res.end());
    return res;
}

int main() {
    string g, c, seq;
    while (cin >> c) {
        g += c;
        if (c != "x")
            seq += c;
    }
    int t = 0;
    for (int i = 0; i < seq.size(); ++ i)
        for (int j = i + 1; j < seq.size(); ++ j)
            if (seq[i] > seq[j])
                t ++ ;
    if (t & 1)
        cout << "unsolvable" << endl;
    else
        cout << BFS(g) << endl;
    return 0;
}
```

### 3.4.2 第 k 短路.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：第k短路.cpp
```

```
/*
 *    创建日期：2021年11月25日 星期四 10时41分54秒
 *    题    目：AcWing 0178 第k短路
 *    算    法：A*
 *    描    述：n 个点，m 条边的有向图，求从起点 s 到终点 t 的第 k
 *    短路的长度，路径允许重复经过点和边
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <queue>
#include <cstring>
using namespace std;
const int maxn = 1e3 + 5;
const int maxm = 2e5 + 5;
int n, m;
int h[maxn], rh[maxn], e[maxm], w[maxm], ne[maxm], idx;
int dist[maxn], f[maxn], used[maxn];
int S, T, K;
typedef pair<int, int> PII;
typedef pair<int, PII> PIII;

void add(int *h, int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;
}

void dijkstra() {
    priority_queue<PII, vector<PII>, greater<PII>> he;
    memset(dist, 0x3f, sizeof dist);
    dist[T] = 0;
    he.push({0, T});
    while (he.size()) {
        auto t = he.top(); he.pop();
        int ver = t.second;
        if (used[ver])
            continue;
        used[ver] = true;
        for (int i = rh[ver]; ~i; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[ver] + w[i]) {
                dist[j] = dist[ver] + w[i];
                he.push({dist[j], j});
            }
        }
    }
    memcpy(f, dist, sizeof dist);
}

int astar() {
    priority_queue<PIII, vector<PIII>, greater<PIII>> he;
    he.push({f[S], {0, S}});
    memset(used, 0, sizeof used);
    while (he.size()) {
        auto t = he.top(); he.pop();
        int ver = t.second.second, distance = t.second.first;
        if (used[ver] >= K)
            continue;
        used[ver] ++ ;
        if (ver == T && used[ver] == K)
            return distance;
        for (int i = h[ver]; ~i; i = ne[i]) {
            int j = e[i];
            if (used[j] < K)
                he.push({distance + w[i] + f[j], {distance + w[i], j}});
        }
    }
    return -1;
}

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

```
    memset(h, -1, sizeof h);
    memset(rh, -1, sizeof rh);
    for (int i = 0; i < m; ++ i) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        add(h, a, b, c), add(rh, b, a, c);
    }
    scanf("%d %d %d", &S, &T, &K);
    if (S == T)
        K ++ ;
    dijkstra();   // 求启发函数，每个点到终点的最短距离
    printf("%d\n", astar());
    return 0;
}
```

# 4 动态规划

## 4.1 DP 基础

### 4.1.1 硬币问题

#### 4.1.1.1 最少硬币问题.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：最少硬币问题.cpp
 *   创建日期：2021年03月08日 ---- 14时16分
 *   题    目：硬币问题
 *   算    法：动态规划
 *   描    述：有5种硬币，面值分别为1, 5, 10, 25, 50，数量无限，输入非负整数
 *      表示需要付的钱数，要求付钱时选择最少的硬币数
 *
----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 251;              //购物需要的钱数不超过250
const int inf  = 0x3f3f3f3f;
int type[5] = {1, 5, 10, 25, 50}; //5种硬币面值
int minCoins[maxn];                //minCoins[i]代表：付i块钱最少需要多少硬币
#define bug printf("<--->\n");

void solve() {
    for (int i = 0; i < maxn; ++i)  //动态转移方程需要此初始化
        minCoins[i] = inf;
    minCoins[0] = 0;                 //初始条件
    for (int i = 0; i < 5; ++i) {
        for (int j = type[i]; j < maxn; ++j)
            minCoins[j] = min(minCoins[j], minCoins[j-type[i]]+1);
    }
}

int main() {
    solve();
    for (int i = 0; i < maxn; ++i)
        printf("minCoins[%d] = %d\n", i, minCoins[i]);
    return 0;
}
```

#### 4.1.1.2 最少硬币组合.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：最少硬币问题.cpp
 *   创建日期：2021年03月08日 ---- 14时16分
 *   题    目：硬币问题
 *   算    法：动态规划
 *   描    述：有5种硬币，面值分别为1, 5, 10, 25, 50，数量无限，输入非负整数
```

```
 *       表示需要付的钱数，要求付钱时选择最少的硬币数
 *       而且打印出硬币组合
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 251;              //购物需要的钱数不超过250
const int inf  = 0x3f3f3f3f;
int type[5] = {1, 5, 10, 25, 50}; //5种硬币面值
int minCoins[maxn];                //minCoins[i]代表：付i块钱最少需要多少硬币
int coinPath[maxn];                //记录最少硬币路径
#define bug printf("<--->\n");

void solve() {
    for (int i = 0; i < maxn; ++i)  //动态转移方程需要此初始化
        minCoins[i] = inf;
    minCoins[0] = 0;                 //初始条件
    for (int i = 0; i < 5; ++i) {
        for (int j = type[i]; j < maxn; ++j)
            if (minCoins[j] > minCoins[j-type[i]]+1) {
                coinPath[j] = type[i];    //j块钱时选择的最后一块硬币是type[i]
                minCoins[j] = minCoins[j-type[i]] + 1;
            }
    }
}

void print(int money) {
    while (money) {
        printf("%d ", coinPath[money]);
        money -= coinPath[money];
    }
    printf("\n");
}

int main() {
    solve();
    for (int i = 0; i < maxn; ++i) {
        printf("minCoins[%d] = %d\n", i, minCoins[i]);
        printf("==> ");
        print(i);
    }
    return 0;
}
```

#### 4.1.1.3 所有硬币组合-1.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：最少硬币问题.cpp
 *   创建日期：2021年03月08日 ---- 14时16分
 *   题    目：硬币问题
 *   算    法：动态规划
 *   描    述：有5种硬币，面值分别为1, 5, 10, 25, 50，数量无限，输入非负整数
 *       表示需要付的钱数，要求付钱时选择最少的硬币数
 *       而且打印出硬币组合
 *       没有硬币数量限制
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 251;
int type[5] = {1, 5, 10, 25, 50};
int dp[maxn];
void solve() {
    dp[0] = 1;
    for (int i = 0; i < 5; ++i)
        for (int j = type[i]; j < maxn; ++j)
```

```
            dp[j] = dp[j] + dp[j-type[i]];
}

int main() {
    solve();
    for (int i = 0; i < maxn; ++i)
        printf("dp[%d] = %d\n", i, dp[i]);
    return 0;
}
```

#### 4.1.1.4 所有硬币组合-2.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：最少硬币问题.cpp
 *   创建日期：2021年03月08日 ---- 14时16分
 *   题    目：硬币问题
 *   算    法：动态规划
 *   描    述：有5种硬币，面值分别为1, 5, 10, 25, 50，数量无限，输入非负整数
 *       表示需要付的钱数，要求付钱时选择最少的硬币数
 *       而且打印出硬币组合
 *       硬币数量限制为100
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 251;
const int coin = 101;  //限制最多选择100个硬币
int type[5] = {1, 5, 10, 25, 50};
int dp[maxn][coin];
int ans[maxn];
void solve() {
    dp[0][0] = 1;
    for (int i = 0; i < 5; ++i)
        for (int j = 1; j < coin; ++j)
            for (int k = type[i]; k < maxn; ++k)
                dp[k][j] += dp[k-type[i]][j-1];
}

int main() {
    solve();
    for (int i = 0; i < maxn; ++i)
        for (int j = 0; j < coin; ++j)
            ans[i] += dp[i][j];
    for (int i = 0; i < maxn; ++i)
        printf("ans[%d] = %d\n", i, ans[i]);
    return 0;
}
```

### 4.1.2 最长公共子序列

#### 4.1.2.1 最长公共子序列.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-最长公共子序列.cpp
 *   创建日期：2021年03月09日 ---- 15时48分
 *   题    目：hdu1159 Common Subsequence
 *   算    法：动态规划
 *   描    述：求两个序列的最长公共子序列，注意子序列不是子串
 *
 -----------------------------------------------------------------*/

#include <iostream>
#include <string>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1005;
```

```cpp
int dp[maxn][maxn];
string str1, str2;
int LCS() {
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= str1.length(); ++i)
        for (int j = 1; j <= str2.length(); ++j) {
            if (str1[i-1] == str2[j-1])
                str1[i-1] = dp[i-1][j-1] + 1;
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    return dp[str1.length()][str2.length()];
}

int main() {
    while (cin >> str1 >> str2)
        cout << LCS() << endl;
    return 0;
}
```

#### 4.1.3 最长递增子序列

### 4.2 记忆化搜索

#### 4.2.1 The-Triangle.cpp

```cpp
/*------------------------------------------------------------------
 *
 *   文件名称：01-The-Triangle.cpp
 *   创建日期：2021年03月09日 ---- 16时04分
 *   题    目：poj1163 The Triangle
 *   算    法：记忆化搜索
 *   描    述：自下往上的方法更好，但是这里使用递归+记忆化
 *
 ------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 155;
int n;  //三角形的高度
int Tri[maxn][maxn];  //存储三角形塔
int dp[maxn][maxn];
int dfs(int i, int j) {
    if (i == n)
        return Tri[i][j];
    if (dp[i][j] >= 0)
        return dp[i][j];
    return dp[i][j] = max(dfs(i+1, j), dfs(i+1, j+1)) + Tri[i][j];
}

int main() {
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= i; ++j)
            scanf("%d", &Tri[i][j]);
    memset(dp, -1, sizeof(dp));
    printf("%d\n", dfs(0, 0));
    return 0;
}
```

#### 4.2.2 滑雪.cpp

```cpp
/*------------------------------------------------------------------
 *
```

```
 *    文件名称：滑雪.cpp
 *    创建日期：2021年11月01日 星期一 22时21分50秒
 *    题      目：AcWing 0901 滑雪
 *    算      法：记忆化搜索
 *    描      述：给定一个 R 行 C 列的矩阵，表示一个矩形网格滑雪场，
 *    矩阵中第 i 行第 j 列的点表示滑雪场第 i 行第 j 列区域的高度。
 *    一个人从滑雪场中的某个区域出发，每次可以向上下左右任意一个
 *    方向滑动一个单位距离。当然，一个人能够滑动到某相邻区域的前
 *    提是该区域的高度低于自己目前所在区域的高度。
 *
 *     1  2  3  4 5
 *   16 17 18 19 6
 *   15 24 25 20 7
 *   14 23 22 21 8
 *   13 12 11 10 9
 *
 *   在给定矩阵中，一条可行的滑行轨迹为 24 -> 17 -> 2 -> 1，
 *   最长的滑行轨迹为 25 -> 24 -> 23 -> ... -> 3 -> 2 -> 1
 *   沿途共经过 25 个区域。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 300 + 5;
int n, m;
int h[maxn][maxn];
int dp[maxn][maxn];
int dirx[4] = {-1, 0, 1, 0}, diry[4] = {0, 1, 0, -1};

int work(int x, int y) {
    int &v = dp[x][y];
    if (v != -1)
        return v;
    v = 1;
    for (int i = 0; i < 4; ++ i) {
        int newx = x + dirx[i],
            newy = y + diry[i];
        if (newx >= 1 && newx <= n && newy >= 1 && newy <= m && h[newx][newy] < h[x][y])
            v = max(v, work(newx, newy) + 1);
    }
    return v;
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= m; ++ j)
            scanf("%d", &h[i][j]);
    memset(dp, -1, sizeof dp);
    int res = 0;
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= m; ++ j)
            res = max(res, work(i, j));
    printf("%d\n", res);
    return 0;
}
```

## 4.3 背包 DP

### 4.3.1 01 背包

#### 4.3.1.1 01 背包问题-一维.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：0043-01背包问题-一维.cpp
 *    创建日期：2021年10月15日 星期五 23时35分00秒
```

```
 *   题     目: AcWing 0002 01背包问题
 *   算     法: 动态规划
 *   描     述: 有N件物品，背包容积是V，第i件物品的体积是vi，价值wi
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e3 + 5;
int N, V;
int v[maxn], w[maxn];
int dp[maxn];

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++ i)
        scanf("%d %d", &v[i], &w[i]);
    for (int i = 1; i <= N; ++ i)
        for (int j = V; j >= v[i]; -- j)
            dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
    printf("%d\n", dp[V]);
    return 0;
}
```

#### 4.3.1.2 01 背包问题.cpp

```
/*----------------------------------------------------------------
 *
 * ••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
 *
 *   文件名称: 0042-01背包问题.cpp
 *   创建日期: 2021年03月29日 ---- 22时18分
 *   创建日期: 2021年10月15日 星期五 23时16分05秒
 *   题     目: AcWing 0002 01 背包问题
 *   算     法: 动态规划 01背包
 *   描     述: 有N件物品，背包容积是V，第i件物品的体积是vi，价值wi
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e3 + 5;
int N; // N件物品
int V; // 背包容量
int dp[maxn][maxn];
int v[maxn], w[maxn];

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++ i)
        scanf("%d %d", &v[i], &w[i]);
    for (int i = 1; i <= N; ++ i)
        for (int j = 0; j <= V; ++ j) {
            dp[i][j] = dp[i - 1][j];
            if (j >= v[i])
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - v[i]] + w[i]);
        }
    printf("%d\n", dp[N][V]);
    return 0;
}
```

#### 4.3.1.3 Bone-Collector.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称: 01-Bone-Collector.cpp
```

```
 *    创建日期: 2021年03月09日 ---- 15时34分
 *    题      目: hdu2602 Bone Collector
 *    算      法: o1背包
 *    描      述: 骨头收集者带着体积为V的背包去捡骨头，已知每个骨头的
 *       体积和价值，求能装进背包的最大价值
 *       第一行是测试数量，第二行是骨头数量和背包体积，
 *       第三行是每个骨头的价值，第四行是每个骨头的体积
 *
----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1005;
struct Bone {
    int val;
    int vol;
} bone[maxn];
int N;   // 骨头数量
int V;   // 背包体积
// dp[i][j]: 前i件物品放在体积为j的背包中最大价值
int dp[maxn][maxn];

int solve() {
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= N; ++i)
        for (int j = 0; j <= V; ++j) {
            if (bone[i].vol > j)  // 第i个物品太大，装不下
                dp[i][j] = dp[i-1][j];
            else
                dp[i][j] = max(dp[i-1][j], dp[i-1][j-bone[i].vol] + bone[i].val);
        }
    return dp[N][V];
}

int main() {
    int t;
    scanf("%d", &t);
    while (t -- ) {
        scanf("%d %d", &N, &V);
        // dp题还是从1开始吧，因为需要用到dp[i-1][j]
        for (int i = 1; i <= N; ++i)
            scanf("%d", &bone[i].val);
        for (int i = 1; i <= N; ++i)
            scanf("%d", &bone[i].vol);
        printf("%d\n", solve());
    }
    return 0;
}
```

### 4.3.2 完全背包

#### 4.3.2.1 完全背包问题-一维.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: 02-一维.cpp
 *    创建日期: 2021年03月30日 ---- 16时20分
 *    创建日期: 2021年10月18日 星期一 11时53分17秒
 *    题      目: AcWing 0003 完全背包
 *    算      法: 动态规划
 *    描      述: 与01背包问题代码相差无几
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e3 + 5;
```

```cpp
int n, m;
int dp[maxn];
int v[maxn], w[maxn];

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++ i)
        scanf("%d %d", &v[i], &w[i]);
    for (int i = 1; i <= n; ++ i)
        for (int j = v[i]; j <= m; ++ j)
            dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
    printf("%d\n", dp[m]);
    return 0;
}
```

#### 4.3.2.2 完全背包问题.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称: 01.cpp
 *   创建日期: 2021年03月30日 ---- 15时30分
 *   创建日期: 2021年10月18日 星期一 11时47分41秒
 *   题    目: AcWing 0003 完全背包问题
 *   算    法: 动态规划
 *   描    述: 与01背包问题代码相差无几
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e3 + 5;
int n, m;
int dp[maxn][maxn];
int v[maxn], w[maxn];

// dp[i][j] = max(dp[i - 1][j], dp[i][j - v[i]] + w[i]);
int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++ i)
        scanf("%d %d", &v[i], &w[i]);
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= m; ++ j) {
            dp[i][j] = dp[i - 1][j];
            if (j >= v[i])
                dp[i][j] = max(dp[i][j], dp[i][j - v[i]] + w[i]);
        }
    printf("%d\n", dp[n][m]);
    return 0;
}
```

### 4.3.3 多重背包

#### 4.3.3.1 多重背包问题 I-朴素.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称: 多重背包问题I-朴素.cpp
 *   创建日期: 2021年04月10日 ---- 10时39分
 *   创建日期: 2021年10月18日 星期一 18时35分36秒
 *   题    目: AcWing 0004 多重背包
 *   算    法: 多重背包
 *   描    述: 每种物品选ai次转化为有ai个物品选1次，即01背包
 *
 *   0 < n, m < 100
 *   0 < v, w, s < 100
 *
 ----------------------------------------------------------------*/

```

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 105;
int n, m;
int dp[maxn];

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i) {
        int v, w, s;
        scanf("%d %d %d", &v, &w, &s);
        for (int j = m; j >= v; -- j)
            for (int k = 1; k <= s && k * v <= j; ++ k)
                dp[j] = max(dp[j], dp[j - k * v] + k * w);
    }
    printf("%d\n", dp[m]);
    return 0;
}
```

#### 4.3.3.2 多重背包问题 II-二进制分组优化.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：多重背包问题II-二进制分组优化.cpp
 *   创建日期：2021年10月18日 星期一 18时46分46秒
 *   题    目：AcWing 0005 多重背包问题II
 *   算    法：多重背包二进制优化
 *   描    述：有n个物品，一个容量是m的背包
 *   第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。
 *   求解将哪些物品装入背包，可使物品体积总和不超过背包容量
 *   且价值总和最大。
 *
 *   0 < n < 1000
 *   0 < m < 2000
 *   0 < v, w, s < 2000
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
using namespace std;
const int maxn = 1005;
int n, m;
int dp[10 * maxn];

struct Bone {
    int v, w;   // 体积，价值
} bone[10 * maxn];

int main() {
    scanf("%d %d", &n, &m);
    int idx = 1;
    for (int i = 1; i <= n; ++ i) {
        int v, w, s;
        scanf("%d %d %d", &v, &w, &s);
        int b = 1;    // 二进制分组优化，1, 2, 4, 8, 16, ...
        while (s - b > 0) {
            s -= b;
            bone[idx].v = b * v;
            bone[idx].w = b * w;
            idx ++;
            b *= 2;
        }
        if (s) {
            bone[idx].v = s * v;
            bone[idx].w = s * w;
            idx ++;
        }
```

```
    }
    for (int i = 1; i < idx; ++ i)
        for (int j = m; j >= bone[i].v; -- j)
            dp[j] = max(dp[j], dp[j - bone[i].v] + bone[i].w);
    printf("%d\n", dp[m]);
    return 0;
}
```

#### 4.3.3.3 多重背包问题 III-单调队列优化.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：03-单调队列优化.cpp
 *   创建日期：2021年04月07日 ---- 17时52分
 *   创建日期：2021年10月18日 星期一 20时16分52秒
 *   题    目：AcWing 0006 多重背包问题
 *   算    法：多重背包，单调队列优化
 *   描    述：执行单调队列的三个惯例操作
 *       1. 检查队头合法性
 *       2. 取队头为最优策略，更新
 *       3. 把新策略插入队尾，入队前检查队尾单调性，排除无用决策
 *
 *   0 < n < 1000
 *   0 < m < 20000
 *   0 < v, w, s < 20000
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 2e4 + 5;
int n, m;
int dp[maxn], tmp[maxn];
int q[maxn];

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i) {
        int v, w, s;   // 体积，价值，当前物品数量
        scanf("%d %d %d", &v, &w, &s);
        memcpy(tmp, dp, sizeof dp);   // tmp 为 dp[i - 1]，dp 为 dp[i]，有滚动数组的味道了
        /*
         * 枚举余数即等价类
         * 背包体积，j < v 是因为 v 可以由 j = 0 + v 得到
         * 每个等价类有0, 0+1v, 0+2v, 0+3v ...   -> 每个 for(j) 进行单调队列操作，为了找出这个等价类中最大的价值
         *             1, 1+1v, 1+2v, 1+3v ...
         *             2, 2+1v, 2+2v, 2+3v ...
         *             ...
         *             v-1, ...
         * 所有等价类中的数就是0 ~ m，也就是背包的第二维
         */
        for (int j = 0; j < v; ++ j) {
            int hh = 0, tt = -1;
            for (int k = j; k <= m; k += v) {
                // 如果当前位置表示的背包体积 > 队首体积 + 当前物品总体积，很显然队首体积不合法了，出队
                if (hh <= tt && q[hh] < k - s * v)
                    ++ hh;
                /*
                 * 使用tmp，决策上一件物品是队首体积时的最优解+队首体积到当前体积需要的当前物品总价值
                 *
                 *      |------- q[hh]指向的位置，tmp[q[hh]] 存储这里的最优解（背包体积为 q[hh] 时，最大价值）
                 *      v
                 * ---------------------------------------------------------------------------
                 * |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
                 * ---------------------------------------------------------------------------
                 * |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
                 * ---------------------------------------------------------------------------
```

```
                 *      ^                                                    ^
                 *      |---  填充到当前体积需要的物品数(k - q[hh]) / v * w  ---|
                 */
                if (hh <= tt)
                    dp[k] = max(dp[k], tmp[q[hh]] + (k - q[hh]) / v * w);
                /*
                 * tmp[q[tt]]             放入上一件物品体积在队尾体积处时的最大价值
                 * (q[tt] - j) / v * w    等价类的第一个体积到队尾体积全部填充当前物品所需要的总价值
                 * tmp[k]                 放入上一件物品体积在当前体积处时的最大价值
                 * (k - j) / v * w        等价类的第一个体积到当前体积全部填充当前物品所需要的总价值
                 *
                 * 在动态转移方程里是tmp[q[hh]] + (k - q[hh]) / v * w
                 * 也就是最近的k个物品的决策里的最大值
                 * 但为什么在这里就是tmp[q[tt]] - (q[tt] - j) / v * w 呢
                 *
                 * 前 a-1 个数在单调队列中的决策
                 * dp[k] = max(dp[k], dp[k-v]+w, dp[k-2v]+2w, dp[k-3v]+3w, ...)
                 * 在单调队列中无法实现，转化为
                 * dp[k]-(k-j)/v*w, dp[k-v]-(k-v-j)/v*w, dp[k-2v]-(k-2v-j)/v*w, ...
                 * 会发现，上下两式各项都相差(k-j)/v*w，所以可以转化
                 *
                 * 队列中肯定要放最大值，这里的最大值还要根据k的位置确定，因为在求dp[k]时，队首元素到
                 * 当前位置k所需要的(k-q[hh])/v*w是随着k的变化而变化的，所以队列中不能存储确定的值，只能存储位置
                 * 对于当前位置，队列中的元素到达当前位置的总价值是无法确定的，但是到达等价类的第一个位置的总价值是
                 * 确定的，而且等价类的第一个位置到当前位置的总价值也是确定的，所以单调队列中存储的本应是前a个位置到
                 * 达当前位置的总价值由大到小排序(部分不满足由大到小的位置出队)，然而到达当前位置的总价值无法确定，所
    以
                 * 转化为到达等价类第一个位置的总价值
                 */
                while (hh <= tt && tmp[q[tt]] - (q[tt] - j) / v * w <= tmp[k] - (k - j) / v * w)
                    -- tt;
                // 无论如何，将当前位置插入队列
                q[ ++ tt] = k;
            }
        }
    }
    printf("%d\n", dp[m]);
    return 0;
}
```

### 4.3.4 混合背包

#### 4.3.4.1 混合背包问题.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01-混合背包.cpp
 *   创建日期：2021年04月07日 ---- 17时56分
 *   创建日期：2021年10月25日 星期一 08时30分20秒
 *   题    目：AcWing 0007 混合背包问题
 *   算    法：混合背包
 *   描    述：转化为多重背包二进制优化
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1000 + 5;
int dp[10 * maxn];

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i) {
        int v, w, s;   // 体积，价值，数量
        scanf("%d %d %d", &v, &w, &s);
        if (s == 0) {   // 完全背包
            for (int j = v; j <= m; ++ j)
                dp[j] = max(dp[j], dp[j - v] + w);
```

```
        }
        else {   // 多重背包或01背包
            if (s == -1)   // 01背包是特殊的多重背包
                s = 1;
            int b = 1;
            while (s - b > 0) {
                for (int j = m; j >= b * v; -- j)
                    dp[j] = max(dp[j], dp[j - b * v] + b * w);
                s -= b;
                b *= 2;
            }
            if (s) {
                for (int j = m; j >= s * v; -- j)
                    dp[j] = max(dp[j], dp[j - s * v] + s * w);
            }
        }
    }
    printf("%d\n", dp[m]);
    return 0;
}
```

### 4.3.5 二维费用背包

#### 4.3.5.1 二维费用的背包问题.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称: 01-二维费用的背包问题.cpp
 *   创建日期: 2021年04月28日 ---- 10时37分
 *   创建日期: 2021年10月25日 星期一 08时56分17秒
 *   题    目: AcWing 0008 二维费用问题
 *   算    法: 二维费用背包
 *   描    述:
 *   有 N 件物品和一个容量是 V 的背包，背包能承受的最大重量是 M
 *   每件物品只能用一次 体积是 vi，重量是 mi，价值是 wi
 *   求解将哪些物品装入背包，可使物品总体积不超过背包容量
 *   总重量不超过背包可承受的最大重量，且价值总和最大
 *
 *   0 < N <= 1000
 *   0 < V, M <= 100
 *   0 < v, m <= 100
 *   0 < w <= 1000
 *
 *   dp[V][M] 表示在体积不超过 V，重量不超过 M，的最大价值
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e2 + 5;
int dp[maxn][maxn];

int main() {
    int N, V, M;
    scanf("%d %d %d", &N, &V, &M);
    for (int i = 0; i < N; ++ i) {
        int v, m, w;
        scanf("%d %d %d", &v, &m, &w);
        for (int j = V; j >= v; -- j)
            for (int k = M; k >= m; -- k)
                dp[j][k] = max(dp[j][k], dp[j - v][k - m] + w);
    }
    printf("%d\n", dp[V][M]);
    return 0;
}
```

#### 4.3.6 分组背包

##### 4.3.6.1 分组背包问题.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：分组背包问题.cpp
 *   创建日期：2021年10月21日 星期四 19时58分08秒
 *   题    目：AcWing 0009 分组背包问题
 *   算    法：动态规划 分组背包
 *   描    述：有 N 组物品和一个容量是 V 的背包。
 *   每组物品有若干个，同一组内的物品最多只能选一个。
 *   每件物品的体积是 vij，价值是 wij，其中 i 是组号，j 是组内编号。
 *   求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 100 + 5;
int n, m;
int dp[maxn];
int v[maxn], w[maxn];

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i) {   // 物品组数
        // 输入
        int s; scanf("%d", &s);
        for (int j = 0; j < s; ++ j)   // 这组物品信息，体积、价值
            scanf("%d %d", &v[j], &w[j]);

        /* 动态规划
         * 其实和 01背包 差不多不是吗
         * for (int i = 0; i < n; ++ i)
         *     for (int j = m; j >= v[i]; -- j)
         *         dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
         *
         * 分组背包只不过是把一个背包换成一组，多个 for(k) 就行
         **/
        for (int j = m; j >= 0; -- j)   // 体积
            for (int k = 0; k < s; ++ k)   // 遍历每个物品
                if (j >= v[k])
                    dp[j] = max(dp[j], dp[j - v[k]] + w[k]);
    }
    printf("%d\n", dp[m]);
    return 0;
}
```

### 4.4 区间 DP

#### 4.4.1 回文串.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：02-回文串.cpp
 *   创建日期：2021年03月09日 ---- 17时37分
 *   题    目：poj3280 Cheapest Palindrome
 *   算    法：区间DP
 *   描    述：给定字符串s，长度为m，由n个小写字母构成，在s的任意位置
 *      增删字母，把它变为回文串，增删特定字母的花费不同，求最小花费
 *      3 4
 *      abcb
 *      b 350 700
 *      c 200 800
 *      a 1000 1100
 *
 *      900
 *
```

```
 ----------------------------------------------------------------*/

#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
const int maxn = 2005;
int weight[30];
int dp[maxn][maxn];

int main() {
    int n, m;
    while (cin >> n >> m) {
        string str;
        cin >> str;
        for (int i = 0; i < n; ++i) {
            int x, y;
            char ch;
            cin >> ch >> x >> y; //读取每个字符的插入和删除花费
            weight[ch-'a'] = min(x, y); //取其中的最小值
        }
        for (int i = m-1; i >= 0; --i)      //i是子区间的起点
            for (int j = i+1; j < m; ++j) { //j是子区间的终点
                if (str[i] == str[j])
                    dp[i][j] = dp[i+1][j-1];
                else
                    dp[i][j] = min(dp[i+1][j] + weight[str[i]-'a'], dp[i][j-1] + weight[str[j]-'a']);
            }
        cout << dp[0][m-1] << endl;
    }
    return 0;
}
```

### 4.4.2 石子合并.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：石子合并.cpp
 *   创建日期：2021年03月09日 ---- 17时08分
 *   创建日期：2021年10月28日 星期四 16时13分02秒
 *   题    目：AcWing 0282 石子合并
 *   算    法：区间DP
 *   描    述：设有 N 堆石子排成一排，其编号为 1, 2, 3, ..., N
 *   每堆石子有一定的质量，可以用一个整数来描述，现在要将这 N
 *   堆石子合并成为一堆。每次只能合并相邻的两堆，合并的代价为
 *   这两堆石子的质量之和，合并后与这两堆石子相邻的石子将和新堆相邻，
 *   合并时由于选择的顺序不同，合并的总代价也不相同。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 300 + 5;
int preS[maxn], dp[maxn][maxn];
const int INF = 0x3f3f3f3f;

int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; ++ i) {
        scanf("%d", &preS[i]);
        preS[i] += preS[i - 1];
    }
    for (int len = 2; len <= n; ++ len)
        for (int i = 1; i + len - 1 <= n; ++ i) {
            int j = i + len - 1;
            dp[i][j] = INF;
            for (int k = i; k < j; ++ k)
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j] + preS[j] - preS[i - 1]);
```

```
        }
    printf("%d\n", dp[1][n]);
    return 0;
}
```

## 4.5 树形 DP

### 4.5.1 Anniversary-Party.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-Anniversary-Party.cpp
 *   创建日期：2021年03月09日 ---- 18时08分
 *   题    目：hdu1520 Anniversary Party
 *   算    法：树形DP
 *   描    述：一颗有根树上每个结点有一个权值，相邻的父结点和子结点只能选择一个
 *       问如何选择使得总权值之和最大(邀请员工参加宴会，为了避免员工和直属上司发
 *       生尴尬，规定员工和直属上司不能同时出席)
 *   输    入：(规定结点编号从1到N)，输入第一行是一个数字N，后续N行中的每一行都
 *       包含结点的权值，范围是 -128 ~ 127，下面T行分别输入两个结点，后一个结点
 *       是前一个结点的父亲，读到0 0结束
 *       5
 *       1 1 1 1 1
 *       1 3
 *       2 3
 *       4 5
 *       3 5
 *       0 0
 *   输    出：总的最大权值
 *       3
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
const int maxn = 6005;
int n;
int value[maxn];
int dp[maxn][2];
int fa[maxn];
vector<int> tree[maxn];

void dfs(int node) {
    dp[node][0] = 0; //每次都初始化，不参加宴会(不选择当前结点)
    dp[node][1] = value[node]; //参加宴会(选择当前结点)
    for (int i = 0; i < tree[node].size(); ++i) {
        int son = tree[node][i];
        dfs(son);
        dp[node][0] += max(dp[son][1], dp[son][0]); //父结点不选，那么子结点可选可不选
        dp[node][1] += dp[son][0]; //选择父结点，子结点不能选
    }
}

int main() {
    while (~scanf("%d", &n)) {
        for (int i = 1; i <= n; ++i) {
            scanf("%d", &value[i]);
            tree[i].clear();
            fa[i] = -1;
        }
        while (1) {
            int a, b;
            scanf("%d %d", &a, &b);
            if (a == 0 && b == 0)
                break;
            tree[b].push_back(a); //邻接表建树
            fa[a] = b;
```

```
        }
        int t = 1;
        while (fa[t] != -1)
            t = fa[t]; //找到根结点
        dfs(t);
        printf("%d\n", max(dp[t][1], dp[t][0]));
    }
    return 0;
}
```

### 4.5.2 没有上司的舞会.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：没有上司的舞会.cpp
 *   创建日期：2021年11月01日 星期一 21时07分35秒
 *   题    目：AcWing 0285 没有上司的舞会
 *   算    法：树形DP
 *   描    述：某大学有 n 个职员，编号为 [1, n]。他们之间有从属关系
 *   也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司
 *   现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数 ai，
 *   但是呢，如果某个职员的上司来参加舞会了，
 *   那么这个职员就无论如何也不肯来参加舞会了。
 *   所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 6000 + 5;
int n, happy[maxn];
int dp[maxn][2];
int h[maxn], e[maxn], ne[maxn], idx;
bool has_father[maxn];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

void DFS(int u) {
    dp[u][1] = happy[u];
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        DFS(j);
        dp[u][0] += max(dp[j][0], dp[j][1]);
        dp[u][1] += dp[j][0];
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &happy[i]);
    memset(h, -1, sizeof h);
    for (int i = 0; i < n - 1; ++ i) {
        int a, b;   // b 是 a 的上司
        scanf("%d %d", &a, &b);
        has_father[a] = true;
        add(b, a);
    }
    int root = 1;
    while (has_father[root])
        root ++ ;

    DFS(root);
    printf("%d\n", max(dp[root][0], dp[root][1]));
    return 0;
```

```
}
```

## 4.6 状压 DP

### 4.6.1 Corn-Fields.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称: 01-Corn-Fields.cpp
 *   创建日期: 2021年03月10日 ---- 20时08分
 *   题    目: poj3254 Corn Fields
 *   算    法: 状态压缩DP
 *   描    述: 输入一个矩阵，选择不相邻的方格的方案数
 *      | 1 | 2 | 3 |
 *      |   | 4 |   |
 *      可以的方案数有{}、{1}、{2}、{3}、{4}、{1, 3}、{1, 4}、{3, 4}、{1, 3, 4}
 *
 *      单看第一行，使用二进制描述方格，1表示种玉米，0表示不种玉米
 *      | 编号 |  1  |  2  |  3  |  4  |  5  |
 *      | 方案 | 000 | 001 | 010 | 100 | 101 |
 *
 *      单看第二行，使用二进制描述方格，1表示种玉米，0表示不种玉米
 *      | 编号 |  1  |  2  |
 *      | 方案 | 000 | 010 |
 *
 *      如果第二行选编号1，第一行可以选五种不冲突方案
 *      如果第二行选编号2，会与第一行010冲突，其他四种没问题
 *
 *      dp[i][j]表示第i行采用第j种编号的方案时前i行可以得到
 *      的可行方案总数
 *      例如dp[2][2] = 4表示第二行使用第二种方案时的方案总数是4
 *
 *      状态转移方程dp[i][k] = dp[i-1][j] (j From 1 To n)
 *      最后一行的dp[m][k]相加就得到了答案
 *
 -----------------------------------------------------------------*/

//<++>
```

### 4.6.2 最短 Hamiton 路径.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称: 最短Hamiton路径.cpp
 *   创建日期: 2021年11月01日 星期一 20时06分05秒
 *   题    目: AcWing 0091 最短Hamiton路径
 *   算    法: 状压DP
 *   描    述: 给定一张 n 个点的带权无向图，点从 0□n-1 标号，
 *   求起点 0 到终点 n-1 的最短 Hamilton 路径。
 *   Hamilton 路径的定义是从 0 到 n-1 不重不漏地经过每个点恰好一次。
 *
 *   题目输入: 一个数n，然后是 n * n 个整数，第 i 行第 j 列的整数表示
 *   点 i 到 j 的距离
 *
 *   1 <= n <= 20
 *   0 <= dist[i, j] <= 1e7
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 20 + 5, maxm = 1 << 20;
int dist[maxn][maxn];
int n;
int dp[maxm][maxn];
```

```

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < n; ++ j)
            scanf("%d", &dist[i][j]);
    memset(dp, 0x3f, sizeof dp);
    dp[1][0] = 0;
    for (int i = 0; i < 1 << n; ++ i)   // 遍历当前状态
        for (int j = 0; j < n; ++ j)   // 最后一个点是 j
            if (i >> j & 1)   // 显然需要满足当前状态有 j 这个点
                for (int k = 0; k < n; ++ k)   // 当前状态由当前状态去除最后一个点 j 且最后一个点是 k 的路径转移过来
                    if (i - (1 << j) >> k & 1)   // 显然需要满足这个状态有 k 这个点
                        dp[i][j] = min(dp[i][j], dp[i - (1 << j)][k] + dist[k][j]);
    printf("%d\n", dp[(1 << n) - 1][n - 1]);
    return 0;
}
```

### 4.6.3 蒙德里安的梦想.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：蒙德里安的梦想.cpp
 *   创建日期：2021年11月01日 星期一 08时49分37秒
 *   题    目：AcWing 0291 蒙德里安的梦想
 *   算    法：状压DP
 *   描    述：求把 N * M 的棋盘分割成若干个 1 * 2 的长方形，有多少
 *   种方案
 *
 *   例如 N = 2, M = 3 有 3 种方案
 *
 *   |--------|----|     |----|--------|     |----|----|----|
 *   |        |    |     |    |        |     |    |    |    |
 *   |--------|    |     |    |--------|     |    |    |    |
 *   |        |    |     |    |        |     |    |    |    |
 *   |--------|----|     |----|--------|     |----|----|----|
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <cstring>
using namespace std;
const int maxn = 12, maxm = 1 << maxn;
typedef long long ll;
int n, m;
// dp[i][j] 表示已经将前 i - 1 列放好，第 i - 1 列中横放的方块占据第 i 列的状态为 j 的方案数
ll dp[maxn][maxm];
vector<int> state[maxm];
bool used[maxm];


/*
 *  这里是第 i - 1 列 --|   |-- 这里是第 i 列
 *                      V   V
 *    |---|---|---|---|-------|---|---|---|---|---|---|---|---|
 *    |               |       |   |   |   |   |   |   |   |   |
 *    |               |-------|---|---|---|---|---|---|---|---|
 *    |               |       |   |   |   |   |   |   |   |   |
 *    |               |-------|---|---|---|---|---|---|---|---|
 *    |               |   | + |   |   |   |   |   |   |   |   |
 *    |               |   |---|---|---|---|---|---|---|---|---|
 *    |               |   | + |   |   |   |   |   |   |   |   |
 *    |               |-------|---|---|---|---|---|---|---|---|
 *    |               |       |   |   |   |   |   |   |   |   |
 *    |               |-------|---|---|---|---|---|---|---|---|
 *    |               |   | + |   |   |   |   |   |   |   |   |
 *    |               |   |---|---|---|---|---|---|---|---|---|
```

```
 *     |               |   | + |   |   |   |   |   |   |   |   |   |
 *     |               |---|---|---|---|---|---|---|---|---|---|---|
 *     |               |       |   |   |   |   |   |   |   |   |   |
 *     |               |---|---|---|---|---|---|---|---|---|---|---|
 *     |               |   | + |   |   |   |   |   |   |   |   |   |
 *     |               |   |---|---|---|---|---|---|---|---|---|---|
 *     |               |   | + |   |   |   |   |   |   |   |   |   |
 *     |               |---|---|---|---|---|---|---|---|---|---|---|
 *     |               |   | + |   |   |   |   |   |   |   |   |   |
 *     |               |   |---|---|---|---|---|---|---|---|---|---|
 *     |               |   | + |   |   |   |   |   |   |   |   |   |
 *     |               |---|---|---|---|---|---|---|---|---|---|---|
 *     |               |       |   |   |   |   |   |   |   |   |   |
 *     |---|---|---|---|---|---|---|---|---|---|---|---|---|---|
 *
 *    在这个图中，第 i 列可能部分位置被第 i - 1 列横放的方块占据
 *    对于第 i 列来说，被占据的位置用 1 表示，没被占据的位置用 0 表示
 *    则可以用一个二进制数表示这一列当前的状态，在这里是：1100100100001
 *
 *    这个状态成立的条件之一就是没被占据的空位置的长度都必须是偶数
 *
 */

int main() {
    while (scanf("%d %d", &n, &m) && (n || m)) {
        // 预处理当前状态 i 是否存在，也就是没有奇数长度的空位置
        for (int i = 0; i < 1 << n; ++ i) {
            int cnt = 0;   // 这个空位的长度
            bool is_valid = true;   // 判断是否合法，如果有奇数个零，表示不合法
            for (int j = 0; j < n; ++ j) {
                if (i >> j & 1) {
                    if (cnt & 1) {
                        is_valid = false;
                        break;
                    }
                    cnt = 0;
                }
                else {
                    cnt ++ ;
                }
            }
            if (cnt & 1)   // 还要判断最后一个空的长度
                is_valid = false;
            used[i] = is_valid;
        }
        for (int i = 0; i < 1 << n; ++ i) {
            state[i].clear();
            for (int j = 0; j < 1 << n; ++ j)
                if ((i & j) == 0 && used[i | j])
                    // 对于前一列的状态 i 来说，当前列的状态 j 是满足的
                    // 对于当前列的状态 i 来说，前一列的状态 j 是满足的
                    state[i].push_back(j);
        }
        memset(dp, 0, sizeof dp);
        dp[0][0] = 1;
        for (int i = 1; i <= m; ++ i)
            for (int j = 0; j < 1 << n; ++ j)   // i - 1 列横放的方块占据的第 i 列的状态为 j
                for (auto k : state[j])   // 第 i - 1 列是怎么放的
                    dp[i][j] += dp[i - 1][k];
        printf("%lld\n", dp[m][0]);
    }
    return 0;
}
```

## 4.7 数位 DP

### 4.7.1 不要 4-递推.cpp

```
/*-----------------------------------------------------------------
```

```
 *
 *    文件名称：01-不要4-递推.cpp
 *    创建日期：2021年03月09日 ---- 21时57分
 *    题      目：hdu2089 不要62
 *    算      法：数位DP
 *    描      述：一个数字如果包含‘4’或者'62'，它是不吉利的，给定m和n
 *       0 < m < n < 1e6，统计[m, n]范围内的吉利数
 *       只是为了理解数位DP，所以简化题目要求，只排除了'4'
 *       dp[i][j]表示i位数中首位是j，符合要求的数的个数
 *       递推公式dp[i][j] = dp[i-1][k]  (k From 0 To 9) && (j != 4) && (k != 2 && j != 6)
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 12; //可以更大
int dp[maxn+1][10];  //dp[i][j]表示i位数，第1位数是j时符合条件的数字数量
int digit[maxn+1];   //digit[i]存第i位数字
void init() {
    dp[0][0] = 1;
    for (int i = 1; i <= maxn; ++i)
        for (int j = 0; j < 10; ++j)
            for (int k = 0; k < 10; ++k)
                if (j != 4) //排除数字4
                    dp[i][j] += dp[i-1][k];
}

/*计算0 ~ n区间满足条件的数字个数*/
int solve(int len) {
    int res = 0;
    for (int i = len; i >= 1; --i) { //从高位到低位处理
        for (int j = 0; j < digit[i]; ++j)
            if (j != 4)
                res += dp[i][j];
        if (digit[i] == 4) { //第i位是4，以4开头的2数都不行
            --res;
            break;
        }
    }
    return res;
}

int main() {
    int n;
    int len = 0;
    init(); //预计算dp[][]
    scanf("%d", &n);
    while (n) { //len是n的位数，例如n = 324，是3位数，len = 3
        digit[++len] = n % 10; //digit[3] = 3, digit[2] = 2, digit[1] = 4
        n /= 10;
    }
    printf("%d\n", solve(len) + 1); //求0 ~ n不含4的个数
    return 0;
}
```

### 4.7.2 不要 4-记忆化.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：02-不要4-记忆化.cpp
 *    创建日期：2021年03月10日 ---- 08时12分
 *    题      目：hdu2089 不要62
 *    算      法：数位DP
 *    描      述：一个数字如果包含‘4’或者'62'，它是不吉利的，给定m和n
 *       0 < m < n < 1e6，统计[m, n]范围内的吉利数
 *       只是为了理解数位DP，所以简化题目要求，只排除了'4'
 *       记忆化搜索的思路就是在递归dfs()中搜索所有可能的情况，遇到已
 *       经算过的记录在dp[]中的结果就直接使用，不再重复计算
 *       dp[i]表示i位数中符合条件的数字个数
 *
```

```
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 12;
int dp[maxn]; //dp[i]表示i位数符合要求的个数，dp[2]表示00 ~ 99内符合要求的个数
int digit[maxn];
int dfs(int len, int ismax) { //如果ismax == 1，
    int res = 0;
    int maxx;
    if (!len) //已经递归到0位数，返回
        return 1;
    if (!ismax && dp[len] != -1) //记忆化搜索，如果已经计算过，就直接使用
        return dp[len];
    maxx = (ismax ? digit[len] : 9); //用来判断搜索到什么位置，比如324在十位只用搜索到2
    for (int i = 0; i <= maxx; ++i) {
        if (i == 4) //排除4
            continue;
        res += dfs(len-1, ismax && i == maxx);
    }
    if (!ismax)
        dp[len] = res;
    return res;
}

int main() {
    int n;
    int len = 0;
    memset(dp, -1, sizeof(dp));
    scanf("%d", &n);
    while(n) {
        digit[++len] = n % 10;
        n /= 10;
    }
    printf("%d\n", dfs(len, 1));
    return 0;
}
```

### 4.7.3 不要 4-递推.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01-不要4-递推.cpp
 *   创建日期：2021年03月09日 ---- 21时57分
 *   题    目：hdu2089 不要62
 *   算    法：数位DP
 *   描    述：一个数字如果包含‘4’或者'62'，它是不吉利的，给定m和n
 *      0 < m < n < 1e6，统计[m, n]范围内的吉利数
 *      只是为了理解数位DP，所以简化题目要求，只排除了'4'
 *      dp[i][j]表示i位数中首位是j，符合要求的数的个数
 *      递推公式dp[i][j] = dp[i-1][k]  (k From 0 To 9) && (j != 4) && (k != 2 && j != 6)
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 12; //可以更大
int dp[maxn+1][10];  //dp[i][j]表示i位数，第1位数是j时符合条件的数字数量
int digit[maxn+1];   //digit[i]存第i位数字
void init() {
    dp[0][0] = 1;
    for (int i = 1; i <= maxn; ++i)
        for (int j = 0; j < 10; ++j)
            for (int k = 0; k < 10; ++k)
                if (j != 4) //排除数字4
                    dp[i][j] += dp[i-1][k];
}

/*计算0 ~ n区间满足条件的数字个数*/
int solve(int len) {
```

```
    int res = 0;
    for (int i = len; i >= 1; --i) { //从高位到低位处理
        for (int j = 0; j < digit[i]; ++j)
            if (j != 4)
                res += dp[i][j];
        if (digit[i] == 4) { //第i位是4，以4开头的2数都不行
            --res;
            break;
        }
    }
    return res;
}

int main() {
    int n;
    int len = 0;
    init(); //预计算dp[][]
    scanf("%d", &n);
    while (n) { //len是n的位数，例如n = 324，是3位数，len = 3
        digit[++len] = n % 10; //digit[3] = 3, digit[2] = 2, digit[1] = 4
        n /= 10;
    }
    printf("%d\n", solve(len) + 1); //求0 ~ n不含4的个数
    return 0;
}
```

### 4.7.4 计数问题.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：计数问题.cpp
 *   创建日期：2021年10月31日 星期日 22时03分14秒
 *   题    目：AcWing 0338 计数问题
 *   算    法：数位DP
 *   描    述：给定两个整数 a 和 b，求 a 和 b 之间所有数字中 [0, 9]
 *   的出现次数
 *
 *   例如：a = 1024，b = 1032，则 a 和 b 之间的 9 个数：
 *
 *   1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032
 *
 *   其中 0 出现 10 次；1 出现 10 次；2 出现 7 次 ......
 *
 *   0 < a, b < 1e8
 *
 *   多组输入，输入两个 0 结束输入。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
#define NEXTLINE puts("");

// 算出前面的数字 abc
int get(vector<int> num, int l, int r) {
    int res = 0;
    for (int i = l; i >= r; -- i)
        res = res * 10 + num[i];
    return res;
}

// 后面还有 x 个数字，10^x
int power10(int x) {
    int res = 1;
    while (x -- )
        res *= 10;
    return res;
```

```
}


// 从 1 到 n 中 x 出现的次数
int count(int n, int x) {
    if (!n)
        return 0;
    vector<int> num;
    // n 的每一位是什么
    while (n) {
        num.push_back(n % 10);
        n /= 10;
    }
    n = num.size();   // n 现在是 n 的位数
    int res = 0;
    // 从最高位上开始枚举
    for (int i = n - 1 - !x; i >= 0; -- i) {
        // 只有 i < n - 1 才会出现前面有数，abcdefg
        //                                  ^   前面要有数字
        if (i < n - 1) {
            res += get(num, n - 1, i + 1) * power10(i);
            if (!x)
                res -= power10(i);
        }
        if (num[i] == x)
            res += get(num, i - 1, 0) + 1;
        else if (num[i] > x)
            res += power10(i);
    }
    return res;
}

int main() {
    int a, b;
    while (scanf("%d %d", &a, &b) && (a || b)) {
        if (a > b)
            swap(a, b);
        for (int i = 0; i < 10; ++ i)
            printf("%d ", count(b, i) - count(a - 1, i));
        NEXTLINE;
    }
    return 0;
}
```

## 4.8 计数 DP

### 4.8.1 整数划分-完全背包.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：整数划分.cpp
 *   创建日期：2021年10月28日 星期四 16时51分46秒
 *   题    目：AcWing 0900 整数划分
 *   算    法：完全背包
 *   描    述：一个正整数 n 可以表示成若干个正整数之和，
 *   形如：n = n1 + n2 + ... + nk，
 *   其中 n1 >= n2 >=  ...  >= nk,k >= 1
 *   我们将这样的一种表示称为正整数 n 的一种划分。
 *   现在给定一个正整数 n，请你求出 n 共有多少种不同的划分方法。
 *
 *
 *   有一个体积为 n 的背包，和体积分别为 1, 2, 3, ..., n 的物品
 *   问装满这个背包的选法有多少种
 *
 *   dp[i][j]：从 [1, i] 中选总体积恰好为 j 的选法的数量
 *
 *   dp[i][j] = dp[i - 1][j] + dp[i - 1][j - i] + dp[i - 1][j - 2i]
 *              + dp[i - 1][j - 3i] + ... + dp[i - 1][j - si]
 *
```

```
 *    表示从 [1, i - 1] 中选总体积恰好为 j - ki 的选法的数量
 *
 *    dp[i][j - i] =            dp[i - 1][j - i] + dp[i - 1][j - 2i]
 *                + dp[i - 1][j - 3i] + ... + dp[i - 1][j - si]
 *
 *    所以有: dp[i][j] = dp[i - 1][j] + dp[i][j - i];
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e3 + 5, MOD = 1e9 + 7;
int dp[maxn];

int main() {
    int n; scanf("%d", &n);
    dp[0] = 1;   // 总体积恰好为 0 的选法的方案数是一
    for (int i = 1; i <= n; ++ i)
        for (int j = i; j <= n; ++ j)
            dp[j] = (dp[j] + dp[j - i]) % MOD;
    printf("%d\n", dp[n]);
    return 0;
}
```

### 4.8.2 整数划分-计数 DP.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: 整数划分-计数DP.cpp
 *    创建日期: 2021年10月31日 星期日 16时08分19秒
 *    题    目: AcWing 0900 整数划分
 *    算    法: 计数DP
 *    描    述: 一个正整数 n 可以表示成若干个正整数之和,
 *    形如: n = n1 + n2 + ... + nk,
 *    其中 n1 >= n2 >=  ...  >= nk,k >= 1
 *    我们将这样的一种表示称为正整数 n 的一种划分。
 *    现在给定一个正整数 n, 请你求出 n 共有多少种不同的划分方法。
 *
 *    dp[i][j]: 所有总和是 i, 并且恰好表示成 j 个数的和的方案数量
 *
 *    dp[i][j]: dp[i - 1][j - 1] + dp[i - j][j];
 *
 *    分为两种情况:
 *    1. 这 j 个数中最小值是 1, 那么把这个 1 给去掉, 就有 j - 1 个数
 *       的总和是 i - 1
 *    2. 这 j 个数中最小值不是 1, 那么把这 j 个数都减 1, 就还是 j 个
 *       数, 总和变成了 i - j
 *    显然所有的数的组合都被这两种情况所涵盖, 且不重合
 *
 *    就有: dp[i][j]: dp[i - 1][j - 1] + dp[i - j][j];
 *
 *    最后的结果是 dp[n][1] + dp[n][2] + ... + dp[n][n]
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e3 + 5, MOD = 1e9 + 7;
int dp[maxn][maxn];

int main() {
    int n; scanf("%d", &n);
    dp[0][0] = 1;   // 所有总和是 0, 并且恰好表示成 0 个数的和的方案数量
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= i; ++ j)
            dp[i][j] = (dp[i - 1][j - 1] + dp[i - j][j]) % MOD;
    int res = 0;
    for (int j = 1; j <= n; ++ j)
        res = (res + dp[n][j]) % MOD;
    printf("%d\n", res);
    return 0;
```

```
}
```

## 4.9 线性 DP

### 4.9.1 数字三角形.cpp

```
/*--------------------------------------------------------------
 *
 *    文件名称: 数字三角形.cpp
 *    创建日期: 2021年10月25日 星期一 19时36分09秒
 *    题     目: AcWing 0898 数字三角形
 *    算     法: 线性DP
 *    描     述:
 *    给定一个如下图所示的数字三角形，从顶部出发
 *    在每一结点可以选择移动至其左下方的结点或移动至其右下方的结点
 *    一直走到底层，要求找出一条路径，使路径上的数字的和最大。
 *
 *              7
 *            3   8
 *          8   1   0
 *        2   7   4   4
 *      4   5   2   6   5
 *
 --------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 500 + 5;
int a[maxn][maxn], dp[maxn][maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= i; ++ j)
            scanf("%d", &a[i][j]);
    for (int i = 1; i <= n; ++ i)
        dp[n][i] = a[n][i];
    for (int i = n - 1; i; -- i)
        for (int j = 1; j <= i; ++ j)
            dp[i][j] = max(dp[i + 1][j] + a[i][j], dp[i + 1][j + 1] + a[i][j]);
    printf("%d\n", dp[1][1]);
    return 0;
}
```

### 4.9.2 最短编辑距离.cpp

```
/*--------------------------------------------------------------
 *
 *    文件名称: 最短编辑距离.cpp
 *    创建日期: 2021年10月27日 星期三 21时07分54秒
 *    题     目: AcWing 0902 最短编辑距离
 *    算     法: 线性DP
 *    描     述: 给定两个字符串 A 和 B
 *    现在要将 A 经过若干操作变为 B，可进行的操作有:
 *        删除 – 将字符串 A 中的某个字符删除。
 *        插入 – 在字符串 A 的某个位置插入某个字符。
 *        替换 – 将字符串 A 中的某个字符替换为另一个字符。
 *    现在请你求出，将 A 变为 B 至少需要进行多少次操作。
 *
 *    0 <= n <= 1000, 0 <= m <= 1000
 *
 --------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
```

```cpp
const int maxn = 1e3 + 5;
int n, m;
char a[maxn], b[maxn];
int dp[maxn][maxn];

int main() {
    scanf("%d %s", &n, a + 1);
    scanf("%d %s", &m, b + 1);
    // 初始化
    for (int i = 1; i <= n; ++ i)
        dp[i][0] = i;   // 将a的前i个字符与b的前0个字符匹配，删i次
    for (int j = 1; j <= m; ++ j)
        dp[0][j] = j;   // 将a的前0个字符与b的前j个字符匹配，增j次

    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= m; ++ j) {
            dp[i][j] = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
            if (a[i] == b[j])
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1]);
            else
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + 1);
        }
    printf("%d\n", dp[n][m]);
    return 0;
}
```

### 4.9.3 最长上升子序列 I.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：最长上升子序列.cpp
 *   创建日期：2021年10月25日 星期一 19时57分02秒
 *   题    目：AcWing 0895 最长上升子序列
 *   算    法：线性DP
 *   描    述：
 *   给一个长度为 N 的数列，求数值严格单调递增的子序列的长度最长是多少
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1000 + 5;
int sequ[maxn];
int dp[maxn];

int main() {
    int n; scanf("%d", &n);
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &sequ[i]);
    for (int i = 1; i <= n; ++ i) {
        dp[i] = 1;
        for (int j = 1; j < i; ++ j)
            if (sequ[j] < sequ[i])
                dp[i] = max(dp[i], dp[j] + 1);
    }
    int res = 0;
    for (int i = 1; i <= n; ++ i)
        res = max(res, dp[i]);
    printf("%d\n", res);
    return 0;
}
```

### 4.9.4 最长上升子序列 II.cpp

```cpp
/*----------------------------------------------------------------
 *
```

```
 *    文件名称：最长上升子序列II.cpp
 *    创建日期：2021年10月25日 星期一 20时09分03秒
 *    题    目：AcWing 0896 最长上升子序列II
 *    算    法：线性DP
 *    描    述：
 *    给定一个长度为 N 的数列，求数值严格单调递增的子序列的长度最长是多少
 *
 *    还要输出这个最长上升子序列
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int a[maxn],   // 原序列
    mini[maxn];   // 所有长度下结尾的最小值
const int INF = 0x3f3f3f3f;

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d", &a[i]);
    int len = 0;   // 当前的最大长度
    mini[0] = -INF;   // 为了使比当前的数还小的数存在
    for (int i = 0; i < n; ++ i) {
        int l = 0, r = len;
        // 找到在 mini[] 地一个小于 a[i] 的数
        while (l < r) {
            // [l, mid - 1] [mid, r]
            int mid = l + r + 1 >> 1;
            if (mini[mid] < a[i])
                l = mid;
            else
                r = mid - 1;
        }
        len = max(len, r + 1);
        /**
         * 我们二分找到第一个小于 a[i] 的数就是 mini[l] = mini[r]
         * 那么就一定有 a[i] 大于等于 mini[l + 1] = mini[r + 1]
         * 所以更新 mini[r + 1]
         * mini[r + 1] = a[i]
         * 最大长度为 r + 1 的子序列的结尾是 a[i]
         */
        mini[r + 1] = a[i];
    }
    printf("%d\n", len);
    return 0;
}
```

#### 4.9.5 最长公共子序列.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：最长公共子序列.cpp
 *    创建日期：2021年10月26日 星期二 21时28分44秒
 *    题    目：AcWing 0897 最长公共子序列
 *    算    法：线性DP
 *    描    述：给定两个长度分别为 N 和 M 的字符串 A 和 B
 *    求既是 A 的子序列又是 B 的子序列的字符串长度最长是多少。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e3 + 5;
int n, m;
char a[maxn], b[maxn];
```

```
int dp[maxn][maxn];

int main() {
    scanf("%d %d", &n, &m);
    scanf("%s %s", a + 1, b + 1);
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= m; ++ j) {
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            if (a[i] == b[j])
                dp[i][j] = max(dp[i][j], dp[i - 1][j - 1] + 1);
        }
    printf("%d\n", dp[n][m]);
    return 0;
}
```

#### 4.9.6 编辑距离.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：编辑距离.cpp
 *   创建日期：2021年10月27日 星期三 21时54分18秒
 *   题    目：AcWing 0899 编辑距离
 *   算    法：线性DP
 *   描    述：给定 n 个长度不超过 10 的字符串以及 m 次询问
 *   每次询问给出一个字符串和一个操作次数上限。
 *   对于每次询问，请你求出给定的 n 个字符串中有多少个字符串可以
 *   在上限操作次数内经过操作变成询问给出的字符串。
 *   每个对字符串进行的单个字符的插入、删除或替换算作一次操作。
 *
 *   0 <= n <= 1000, 0 <= m <= 1000
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 10 + 5, maxm = 1e3 + 5;;
int n, m;
int dp[maxn][maxn];
char str[maxm][maxn];

int edit_distance(char a[], char b[]) {
    int len_a = strlen(a + 1),
        len_b = strlen(b + 1);
    for (int i = 0; i <= len_a; ++ i)
        dp[i][0] = i;
    for (int j = 0; j <= len_b; ++ j)
        dp[0][j] = j;
    for (int i = 1; i <= len_a; ++ i)
        for (int j = 1; j <= len_b; ++ j) {
            dp[i][j] = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
            dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (a[i] != b[j]));
        }
    return dp[len_a][len_b];
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++ i)
        scanf("%s", str[i] + 1);
    while (m -- ) {
        char s[maxn];
        int limit;
        scanf("%s %d", s + 1, &limit);
        int res = 0;
        for (int i = 0; i < n; ++ i)
            if (edit_distance(str[i], s) <= limit)
                res ++ ;
```

```
        printf("%d\n", res);
    }
    return 0;
}
```

# 5 字符串

## 5.1 字符串哈希

### 5.1.1 字符串哈希.cpp

```
/*------------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年08月11日 星期三 00时17分16秒
 *   题    目：AcWing 0841 字符串哈希
 *   算    法：哈希
 *   描    述：预处理所有前缀的哈希
 *                                hash?
 *   [高位]                      <--------->          [低位]
 *   |-------------------------|-----------|--------------|
 *   1                         L           R
 *   ^
 *   |----  从1开始，
 *
 *
 *   我们现在已知ha[L-1]，ha[R]
 *
 *   ha[L-1]：表示下面这个子串的哈希值
 *   |-----------------------|
 *   1                      L - 1
 *
 *   ha[R]：表示下面这个长的子串的哈希值
 *   |-------------------------|-----------|
 *   1                         L           R
 *
 *   让上面的两个串的高位对齐、相减得到我们想要表示出[L, R]区间的哈希值
 *   $hash = ha[R] - ha[L] * p^{R - L + 1};$
 *
 ------------------------------------------------------------------*/

#include <cstdio>
typedef unsigned long long ull;
const int maxn = 1e5 + 5;
int P = 131;   // P进制数，虽然没写Q = 2^64，但是ull溢出隐藏了这一步
int n, m;
char str[maxn];
ull ha[maxn], p[maxn];   // 发现公式中需要乘一个p的一个指数，所以用一个数组预处理

ull get(int l, int r) {
    return ha[r] - ha[l - 1] * p[r - l + 1];
}

int main() {
    scanf("%d %d", &n, &m);
    scanf("%s", str + 1);
    p[0] = 1;   // p^0 = 1;
    for (int i = 1; i <= n; ++i) {   // 预处理p数组，与前缀子串哈希
        p[i] = p[i-1] * P;
        // P是131，这是一个131进制的数，一个131进制的数的一个位上出现'z' = 122有问题吗?
        // 我告诉你，显然没问题
        ha[i] = ha[i-1] * P + str[i];   // woc，不用str[i] - 'a' + 1，神了
    }
    while (m --) {
        int l1, r1, l2, r2;
        scanf("%d %d %d %d", &l1, &r1, &l2, &r2);
        if (get(l1, r1) == get(l2, r2))
            printf("Yes\n");
        else
```

```
            printf("No\n");
    }
    return 0;
}
```

### 5.1.2 拉链法.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：拉链法.cpp
 *   创建日期：2021年08月10日 星期二 10时57分04秒
 *   题    目：<++>
 *   算    法：<++>
 *   描    述：数学上可以证明，mod一个素数，产生的冲突概率较小
 *       所以我们需要先寻找一个素数，而且这里把最大值设为N，而不是maxn了
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int N = 1e5 + 3;
int ha[N];
int e[N], ne[N], idx;

// 运行之后发现大于1e5的最小的素数是1e5 + 3，所以N = 1e5 + 3;
void get_prime() {
    for (int i = 100000; ; ++i) {
        bool flag = true;
        for (int j = 2; j * j <= i; ++j)
            if (i % j == 0) {
                flag = false;
                break;
            }
        if (flag) {
            printf("%d\n", i);
            return;
        }
    }
}

/**
 * 我们另k = (x % N + N) % N = 2;
 *   0   1   2   3   4   5   6     ha[maxn];
 *  ---------------------------
 * |   |   |   |   |   |   |   |   ha[maxn];   // 里面存的是指针(idx)
 *  ---------------------------
 *           |  <- 往这里插入一个值x(头插法)
 *          ---
 *         |   |
 *          ---
 *                   idx |
 *                       v
 *  ---------------------------
 * | - | - | - | - | - | x |   |   e[maxn];
 *  ---------------------------
 *
 */

void insert(int x) {
    int k = (x % N + N) % N;   // k就是哈希值
    e[idx] = x;       // e数组存储数据
    ne[idx] = ha[k];  // ne数组存储指针，因为ha[k]中就是指针，是上一个数的idx，
    ha[k] = idx++;     // 存储指针，当前的idx
}

bool find(int x) {
    int k = (x % N + N) % N;
    for (int i = ha[k]; i != -1; i = ne[i])
        if (e[i] == x)
```

```
            return true;
    return false;
}

int main() {
    // get_prime();
    int n; scanf("%d", &n);
    memset(ha, -1, sizeof ha);
    while (n--) {
        char op[2];
        int x;
        scanf("%s %d", op, &x);
        if (*op == 'I')
            insert(x);
        else {
            if (find(x))
                puts("Yes");
            else
                puts("No");
        }
    }
    return 0;
}
```

## 5.2 字典树

### 5.2.1 Trie 字符串统计.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：Trie字符串统计.cpp
 *   创建日期：2021年08月07日 星期六 20时16分23秒
 *   题    目：AcWing 0835 Trie字符串统计
 *   算    法：字典树
 *   描    述：
 *   维护一个字符串集合，支持两种操作：
 *   1. I x  向集合中插入一个字符串x
 *   2. Q x  询问一个字符串在集合中出现了多少次
 *
 *   简单，trie就是一个字典树，实际上就是一个多链表
 *   cnt用来标记这个是不是字串的结尾
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;   // 单个字符串的长度
int trie[maxn][26], cnt[maxn], idx;   // 下标是0的点，既是根结点，又是空节点

void insert(char str[]) {
    int p = 0;   // 根结点，变量名随便定义，用root也可以
    for (int i = 0; str[i]; ++ i) {
        int u = str[i] - 'a';
        if (!trie[p][u]) // 创建新结点
            trie[p][u] = ++ idx;
        p = trie[p][u];
    }
    cnt[p] ++ ;   // 画个星星标记
}

// 查询这个字符串出现次数
int query(char str[]) {
    int p = 0;
    for (int i = 0; str[i]; ++i) {
        int u = str[i] - 'a';
        if (!trie[p][u])
            return 0;
        p = trie[p][u];
    }
    return cnt[p];
```

```
}

int main() {
    int n; scanf("%d", &n);
    while (n -- ) {
        char op[2];
        char str[maxn];
        scanf("%s %s", op, str);
        if (op[0] == 'I')
            insert(str);
        else if (op[0] == 'Q')
            printf("%d\n", query(str));
    }
    return 0;
}
```

### 5.2.2 python-trie.py

```
# Python Version
class trie:
    nex = [[0 for i in range(26)] for j in range(100000)]
    cnt = 0
    exist = [False] * 100000 # 该结点结尾的字符串是否存在

    def insert(s, l): # 插入字符串
        p = 0
        for i in range(0, l):
            c = ord(s[i]) - ord('a')
            if nex[p][c] == 0:
                nex[p][c] = cnt # 如果没有，就添加结点
                cnt += 1
            p = nex[p][c]
        exist[p] = True

    def find(s, l): # 查找字符串
        p = 0
        for i in range(0, l):
            c = ord(s[i]) - ord('a')
            if nex[p][c] == 0:
                return False
            p = nex[p][c]
        return exist[p]
```

### 5.2.3 最大异或对.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年08月08日 星期日 17时43分39秒
 *   题    目：AcWing 0143 最大异或对
 *   算    法：Trie
 *   描    述：从n个数中找两个数，它们异或得到的值最大
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
const int maxm = 3e6 + 5;
int son[maxm][2], idx;
int a[maxn];

void insert(int x) {
    int root = 0;
    // 取反i等于i >= 0
    for (int i = 30; ~i; --i) {
```

```
        int &s = son[root][x >> i & 1];
        if (!s)
            s = ++idx;
        root = s;
    }
    // cnt[root]++;   // 不需要这句
}

int query(int x) {
    int res = 0, root = 0;
    for (int i = 30; ~i; --i) {
        int bit = x >> i & 1;
        if (son[root][!bit]) {
            res += 1 << i;
            root = son[root][!bit];
        }
        else
            root = son[root][bit];
    }
    return res;
}

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &a[i]);
        insert(a[i]);
    }
    int res = 0;
    for (int i = 0; i < n; ++i)
        res = max(res, query(a[i]));
    printf("%d\n", res);
    return 0;
}
```

## 5.3 前缀函数与 KMP 算法

### 5.3.1 KMP-OI.cpp

```
#include <cstdio>
#include <cstring>
const int maxn = 1e6 + 5, maxm = 1e6 + 5;
#define bug printf("<-->\n");
#define NEXTLINE puts("");
int n, m;
char text[maxn], pattern[maxm];
int ne[maxm];

void prefix_table(char pattern[]) {
    int n = strlen(pattern);
    for (int i = 1; i < n; ++ i) {
        int j = ne[i - 1];
        while (j > 0 && pattern[i] != pattern[j])
            j = ne[j - 1];
        if (pattern[i] == pattern[j])
            j ++ ;
        ne[i] = j;
    }
}
```

### 5.3.2 KMP 字符串-idx-0.cpp

```
// 字符串下标从0开始
#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5, maxm = 1e6 + 5;
int n, m;
```

```
char text[maxm], pattern[maxn];
int ne[maxn];
#define NEXTLINE puts("");

int main() {
    scanf("%d %s", &n, pattern);
    scanf("%d %s", &m, text);
    ne[0] = -1;
    for (int i = 1, j = -1; i < n; i ++ ) {
        while (j >= 0 && pattern[j + 1] != pattern[i])
            j = ne[j];
        if (pattern[j + 1] == pattern[i])
            j ++ ;
        ne[i] = j;
    }
    for (int i = 0; i <= n; ++ i)
        printf("%d ", ne[i]);
    NEXTLINE;

    for (int i = 0, j = -1; i < m; i ++ ) {
        while (j != -1 && text[i] != pattern[j + 1])
            j = ne[j];
        if (text[i] == pattern[j + 1])
            j ++ ;
        if (j == n - 1) {
            printf("%d ", i - j);
            j = ne[j];
        }
    }
    return 0;
}
```

### 5.3.3 KMP 字符串-idx-1.cpp

```
// 字符串下标从1开始
#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5, maxm = 1e6 + 5;
int ne[maxn];
char pattern[maxn], text[maxm];
#define NEXTLINE puts("");

void get_perfix_table(char pattern[]) {
    int n = strlen(pattern + 1);
    for (int i = 2, j = 0; i <= n; ++ i) {
        while (j && pattern[i] != pattern[j + 1])
            j = ne[j];
        if (pattern[i] == pattern[j + 1])
            j ++ ;
        ne[i] = j;
    }
}

void kmp_search(char text[], char pattern[]) {
    int m = strlen(text + 1),
        n = strlen(pattern + 1);
    for (int i = 1, j = 0; i <= m; ++ i) {
        while (j && text[i] != pattern[j + 1])
            j = ne[j];
        if (text[i] == pattern[j + 1])
            j ++ ;
        if (j == n) {
            printf("%d ", i - n);
            j = ne[j];
        }
    }
}

int main() {
```

```
    int n, m;
    scanf("%d %s", &n, pattern + 1);
    scanf("%d %s", &m, text + 1);
    get_perfix_table(pattern);
    /*
     * printf("%s\n", pattern + 1);
     * for (int i = 1; i <= n; ++ i)
     *     printf("%d", ne[i]);
     * NEXTLINE;
     */
    kmp_search(text, pattern);
    return 0;
}

/*
6
ABAABA
33
ABAABAABAABAABAAABBCCABABCABAABAA
|       |           |
1       9           22

9
ABABCABAA
19
ABABABCABAABABAABAB
*/
```

### 5.3.4 KMP 字符串.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：KMP字符串.cpp
 *   创建日期：2021年08月06日 星期五 21时23分58秒
 *   题    目：AcWing 0831 KMP字符串
 *   算    法：KMP
 *   描    述：ne[0] = 0，没有改变ne的下标
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
// maxn 是文本串的长度，maxm 是模式串的长度
const int maxn = 1e6 + 5, maxm = 1e6 + 5;
#define bug printf("<-->\n");
#define NEXTLINE puts("");
int n, m;
char text[maxn], pattern[maxm];
int ne[maxm];

void prefix_table() {
    ne[0] = 0;
    for (int i = 1, j = 0; i < n;) {
        if (pattern[i] == pattern[j])
            ne[i ++ ] = ++ j;
        else if (j > 0)
            j = ne[j - 1];
        else   // j <= 0，说明前i个字符没有相等的真前后缀
            ne[i ++ ] = 0;
    }
    for (int i = n - 1; i > 0; -- i)
        ne[i] = ne[i - 1];
    ne[0] = -1;
}

void kmp_search() {
    prefix_table();
    for (int i = 0, j = 0; i < m;) {
        if (j == n - 1 && text[i] == pattern[j]) {
```

```
            printf("%d ", i - j);   // Found pattern at (i - j);
            j = ne[j];
        }
        if (text[i] == pattern[j])
            i ++ , j ++ ;
        else {   // 如果不等于的话，使用前缀表得到最小移动位置
            j = ne[j];
            if (j == -1)  // 如果是0表示直接从下一个字符开始匹配
                i ++ , j ++ ;
        }
    }
}

int main() {
    // m 是文本串的长度，n 是模式串的长度
    scanf("%d %s", &n, pattern);
    scanf("%d %s", &m, text);
    /*
     * n = 9, m = 19;
     * strcpy(text, "ABABABCABAABABAABAB");
     * strcpy(pattern, "ABABCABAA");
     */
    kmp_search();
    return 0;
}
```

## 5.4 z 函数 (扩展 KMP)

### 5.4.1 EKMP.cpp

```
//因为个人习惯，在灯笼代码的基础上作了些修改
#include <cstdio>
#include <cstdlib>
#include <cstring>

void prefix_table(char pattern[], int prefix[]) {
    int len   = strlen(pattern);
    prefix[0] = 0;
    int ptr   = 0;

    int i = 1;
    while (i < len) {

    }
}

int main() {
    char pattern[] = "ABABCABAA";
    char text[]    = "ABABABCABAABABAABAB";

    return 0;
}
```

## 5.5 AC 自动机

### 5.5.1 ACautomaton.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-ACautomaton.cpp
 *   创建日期：2021年03月19日 ---- 19时49分
 *   题    目：hdu2222 keywords search
 *   算    法：AC自动机
 *   描    述：第一行测试用例数，每个用例包括一个整数N，表示关键字个数
 *       下面有N个关键词N <= 10000，每个关键词只包括小写字母，长度不超过
 *       50，最后一行是文本，长度不大于1000000
 *       在输出的文本中能找到多少关键词
 *
```

```
 ----------------------------------------------------------------*/

#include <cstdio>
#include <map>
#include <queue>
#include <cstring>
using namespace std;
const int maxn = 1000005;
const int sigma_size = 26; //字符集
const int maxnode = 1000005;
int res;
bool used[maxn];
/*
 * 数组定义的字典树
 * ch[i][j]保存结点i的那个编号为j的子结点
 * ch[i][0] == 0表示结点i的子结点a不存在
 */
int ch[maxnode][sigma_size+5];
int dict[maxnode];
int idx(char c) {return c - 'a';}

/*              _
 *             |_|  root
 *             /|\
 *            / | \
 *           /  |  \
 *      [t] t   a   i [i]
 *         / \       \
 *        /   \       \
 *       /     \       \
 * [to] o       e [te]  n [in]
 *             /|\       \
 *            / | \       \
 *           /  |  \       \
 *          a   d   n       n [inn]
 *      [tea] [ted] [ten]
 */

struct Trie {
    int pos;
    Trie() { pos = 1; memset(ch[0], 0, sizeof(ch[0])); memset(used, 0, sizeof(used)); }
    void insert(char *s) {
        int p = 0; //查字典，前缀的位置pos
        int n = strlen(s);
        for(int i = 0; i < n; i++) {
            int c = idx(s[i]); //c = ch - 'a'
            if(!ch[p][c]) {
                memset(ch[pos], 0, sizeof(ch[pos]));
                dict[pos] = 0;
                ch[p][c] = pos++;
            }
            p = ch[p][c];
        }
        dict[p]++;
    }
};

/*Aho-Corasick automaton*/
int last[maxn], f[maxn];
/*递归打印以结点j结尾的所有字符串*/
void print(int j) {
    if (j && !used[j]) {
        res += dict[j];
        used[j] = 1;
        print(last[j]);
    }
}

/*
 * 计算失配函数
```

```
 * BFS顺序递推
 * 在字典树ch中添加失配边
 */
void getFail() {
    queue<int> quu;
    f[0] = 0;
    for (int c = 0; c < sigma_size; ++c) {
        int p = ch[0][c];
        if (p) {
            f[p] = 0;
            quu.push(p);
            last[p] = 0;
        }
    }
    while (!quu.empty()) {
        int r = quu.front();
        quu.pop();
        for (int c = 0; c < sigma_size; ++c) { //遍历字母
            int p = ch[r][c];
            if (!p) {
                ch[r][c] = ch[f[r]][c];
                continue;
            }
            quu.push(p);
            int v = f[r];
            while(v && !ch[v][c])
                v = f[v];
            f[p] = ch[v][c];
            last[p] = dict[f[p]] ? f[p] : last[f[p]];
        }
    }
}

/*在文本串T中找模板*/
void find_T(char* T) {
    int n = strlen(T);
    int p = 0;
    for (int i = 0; i < n; ++i) {
        int c = idx(T[i]);
        p = ch[p][c];
        if (dict[p])
            print(p);
        else if (last[p])
            print(last[p]);
    }
}

char pattern[55];
char text[1000005];
int main() {
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    int t;
    scanf("%d", &t);
    while(t--) {
        int n;
        scanf("%d", &n);
        Trie trie;
        res = 0;
        for(int i = 0; i < n; i++) {
            scanf("%s", pattern);
            trie.insert(pattern);
        }
        getFail();
        scanf("%s", text);
        find_T(text);
        printf("%d\n", res);
    }
    return 0;
}
```

### 5.5.2 Trie 图.cpp

```cpp
#include <cstdio>
#include <cstring>
const int maxn = 1e4 + 5, maxm = 1e6 + 5, maxs = 55;
int n;
int tr[maxn * maxs][26], cnt[maxn * maxs], idx;
char str[maxm];
int q[maxn * maxs], ne[maxn * maxs];
#define NEXTLINE puts("");
#define bug puts("<-->");
#define p(x) printf("-- %d\n", (x));

void insert() {
    int p = 0;
    for (int i = 0; str[i]; ++ i) {
        int u = str[i] - 'a';
        if (!tr[p][u])
            tr[p][u] = ++ idx;
        p = tr[p][u];
    }
    cnt[p] ++ ;
}

void build() {
    int hh = 0, tt = -1;
    for (int i = 0; i < 26; ++ i)
        if (tr[0][i]) {
            ne[tr[0][i]] = 0;
            q[ ++ tt] = tr[0][i];
        }
    while (hh <= tt) {
        int t = q[hh ++ ];   // 父结点
        for (int i = 0; i < 26; ++ i) {
            int c = tr[t][i];   // 当前结点
            if (!c) {   // 当前结点不存在
                tr[t][i] = tr[ne[t]][i];
            }
            else {
                ne[c] = tr[ne[t]][i];
                q[ ++ tt] = c;
            }
        }
    }
}

int ACautomaton() {
    int res = 0;
    for (int i = 0, j = 0; str[i]; ++ i) {
        int c = str[i] - 'a';
        j = tr[j][c];   // j 是父结点
        int u = j;
        while (u) {
            res += cnt[u];
            cnt[u] = 0;
            u = ne[u];
        }
    }
    return res;
}

int main() {
    int t; scanf("%d", &t);
    while (t -- ) {
        memset(tr, 0, sizeof tr);
        memset(cnt, 0, sizeof cnt);
        memset(ne, 0, sizeof ne);
```

```
        idx = 0;
        scanf("%d", &n);
        for (int i = 0; i < n; ++ i) {
            scanf("%s", str);
            insert();
        }
        build();
        scanf("%s", str);
        int res = ACautomaton();
        printf("%d\n", res);
    }
    return 0;
}
```

### 5.5.3 搜索关键词.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：搜索关键词.cpp
 *   创建日期：2021年11月03日 星期三 22时06分59秒
 *   题    目：hdu 2222 AcWing 1282 搜索关键词
 *   算    法：AC自动机
 *   描    述：给定 n 个长度不超过 50 的由小写英文字母组成的单词，
 *   以及一篇长为 m 的文章，问有多少个单词在文章中出现了。
 *   1 1 she he say shr her text
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e4 + 5, maxm = 1e6 + 5, maxs = 55;
int n;
int tr[maxn * maxs][26], cnt[maxn * maxs], idx;
char str[maxm];
int q[maxn * maxs], ne[maxn * maxs];   // Fail[]
#define NEXTLINE puts("");
#define bug puts("<-->");
#define p(x) printf("-- %d\n", (x));

void insert() {
    int p = 0;
    for (int i = 0; str[i]; ++ i) {
        int u = str[i] - 'a';
        if (!tr[p][u])
            tr[p][u] = ++ idx;
        p = tr[p][u];
    }
    cnt[p] ++ ;
}

/*
 *                    +
 *                s / \ h
 *                 /   \
 *          ne = 0  1     4  ne = 0
 *               h /       \ e
 *                /         \
 *       ne = 4  2           5  ne = 0
 *            e /
 *             /
 *     ne = 5  3
 *
 *  当 t = 2 时，也就是停在sh位置，会遍历它的所有后继结点
 *  这时找到 c = tr[t]['e' - 'a'] = tr[2][4] = 3
 *
 *  j = ne[t] = ne[2] = 4，j 是当前结点 3 的 next，就是因为 'e'
 *  的上面的子串的最长后继已经找到了，没必要从头开始找
 *
 *  接下来就是判断是否有 tr[j][i]，这里的 i 就是 'e' - 'a'，也就代表了 'e'
```

```
 *
 *  发现有 tr[j][i] = tr[4][4] = 5，那么 j = tr[j][i],  ne[c] = ne[3] = j = 5
 *  没发现的话，j 最后会变成 0，ne[c] = 0，指向根结点
 */

void build() {
    int hh = 0, tt = -1;   // 宽度优先搜索
    for (int i = 0; i < 26; ++ i)
        if (tr[0][i]) {
            ne[tr[0][i]] = 0;   // 因为初始ne就是0，这句没必要
            q[ ++ tt] = tr[0][i];
        }
    while (hh <= tt) {
        int t = q[hh ++ ];
        for (int i = 0; i < 26; ++ i) {
            int c = tr[t][i];
            if (!c)
                continue;
            int j = ne[t];   // 当前结点的上一个结点，的next
            // 当前结点代表的字母是i
            while (j && !tr[j][i])   // 也就是next下面是否有同样的i
                j = ne[j];
            if (tr[j][i])
                j = tr[j][i];
            ne[c] = j;
            q[ ++ tt] = c;
        }
    }
}

int ACautomaton() {
    int res = 0;
    for (int i = 0, j = 0; str[i]; ++ i) {
        int c = str[i] - 'a';   // 当前字符
        while (j && !tr[j][c])
            j = ne[j];
        if (tr[j][c])
            j = tr[j][c];
        int u = j;
        while (u) {
            res += cnt[u];
            cnt[u] = 0;
            u = ne[u];
        }
    }
    return res;
}

int main() {
    int t; scanf("%d", &t);
    while (t -- ) {
        memset(tr, 0, sizeof tr);
        memset(cnt, 0, sizeof cnt);
        memset(ne, 0, sizeof ne);
        idx = 0;
        scanf("%d", &n);
        for (int i = 0; i < n; ++ i) {
            scanf("%s", str);
            insert();   // 建立 trie
        }
        build();   // 建立 next
        scanf("%s", str);
        int res = ACautomaton();
        printf("%d\n", res);
    }
    return 0;
}
```

## 5.6 后缀数组 SA

### 5.6.1 后缀数组的使用.cpp

```cpp
#include <cstdio>
#include <string>
#include <iostream>
using namespace std;

/*在text中查找子串pattern，sa是text的后缀数组*/
//原字符串中从第sa[0]位置开始的后缀子串在字典序中排列第0
int find(string text, string pattern, int* sa) {
    int i = 0;
    int j = text.length();
    /*而分发查找*/
    while (j - i >= 1) {
        int k = (i + j) / 2;
        if (text.compare(sa[k], pattern.length(), pattern) < 0)
            i = k;
        else
            j = k;
    }
    /*找到了，返回pattern在text中的位置*/
    if (text.compare(sa[j], pattern.length(), pattern) == 0)
        return sa[j];
    return -1;
}

int main() {
    string text = "vamamadn";
    string pattern = "ad";
    int sa[] = {5, 3, 1, 6, 4, 2, 7, 0}; //sa是text的后缀数组，假设已经得到
    int location = find(text, pattern, sa);
    cout << location << ":" << &text[location] << endl << endl;
    return 0;
}
```

### 5.6.2 sort 函数求 sa 数组.cpp

```cpp
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 200005;
char text[maxn];
int sa[maxn];
int rk[maxn];
int tmp[maxn + 1];
int n, k;

bool comp_sa(int i, int j) {
    if (rk[i] != rk[j])
        return rk[i] < rk[j];
    else {
        int ri = i + k <= n ? rk[i + k] : -1;
        int rj = j + k <= n ? rk[j + k] : -1;
        return ri < rj;
    }
}

void calc_sa() {
    for (int i = 0; i <= n; ++i) {
        rk[i] = text[i];
        sa[i] = i;
    }
    for (k = 1; k <= n; k *= 2) {
        sort(sa, sa+n, comp_sa);
        tmp[sa[0]]= 0;
        for (int i = 0; i < n; ++i)
```

```
            tmp[sa[i+1]] = tmp[sa[i]] + (comp_sa(sa[i], sa[i+1]) ? 1 : 0);
        for (int i = 0; i < n; ++i)
            rk[i] = tmp[i];
    }
}

int main() {
    while (scanf("%s", text) != EOF) {
        n = strlen(text);
        calc_sa();
        for (int i = 0; i < n; ++i)
            printf("%d ", sa[i]);
        printf("\n");
    }
    return 0;
}
```

### 5.6.3 基数排序求 sa 数组.cpp

```
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 200005;
char text[maxn];
int sa[maxn];
int rk[maxn];
int cnt[maxn];
int t1[maxn];
int t2[maxn];
int height[maxn];
int n;

void calc_sa() {
    int m = 127;
    int *x = t1;
    int *y = t2;
    for (int i = 0; i < m; ++i) cnt[i] = 0;
    for (int i = 0; i < n; ++i) cnt[x[i] = text[i]]++;
    for (int i = 1; i < m; ++i) cnt[i] += cnt[i-1];
    for (int i = n-1; i >= 0; --i) sa[--cnt[x[i]]] = i;
    for (int k = 1; k <= n; k *= 2) {
        int p = 0;
        for (int i = n-k; i < n; ++i) y[p++] = i;
        for (int i = 0; i < n; ++i)
            if (sa[i] >= k)
                y[p++] = sa[i] - k;
        for (int i = 0; i < m; ++i) cnt[i] = 0;
        for (int i = 0; i < n; ++i) cnt[x[y[i]]]++;
        for (int i = 1; i < m; ++i) cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; --i) sa[--cnt[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            x[sa[i]] =
                y[sa[i-1]] == y[sa[i]] && y[sa[i-1] + k] == y[sa[i] + k] ? p-1 : p++;
        if (p >= n) break;
        m = p;
    }
}

int main() {
    while (scanf("%s", text) != EOF) {
        n = strlen(text);
        calc_sa();
        for (int i = 0; i < n; ++i)
            printf("%d ", sa[i]);
        printf("\n");
```

```
    }
    return 0;
}
```

### 5.6.4 最长公共子串.cpp

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 200005;
char text[maxn];
int sa[maxn];
int rk[maxn];
int cnt[maxn];
int t1[maxn];
int t2[maxn];
int height[maxn];
int n;

void calc_sa() {
    int m = 127;
    int *x = t1;
    int *y = t2;
    for (int i = 0; i < m; ++i) cnt[i] = 0;
    for (int i = 0; i < n; ++i) cnt[x[i] = text[i]]++;
    for (int i = 1; i < m; ++i) cnt[i] += cnt[i-1];
    for (int i = n-1; i >= 0; --i) sa[--cnt[x[i]]] = i;
    for (int k = 1; k <= n; k *= 2) {
        int p = 0;
        for (int i = n-k; i < n; ++i) y[p++] = i;
        for (int i = 0; i < n; ++i)
            if (sa[i] >= k)
                y[p++] = sa[i] - k;
        for (int i = 0; i < m; ++i) cnt[i] = 0;
        for (int i = 0; i < n; ++i) cnt[x[y[i]]]++;
        for (int i = 1; i < m; ++i) cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; --i) sa[--cnt[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            x[sa[i]] =
                y[sa[i-1]] == y[sa[i]] && y[sa[i-1] + k] == y[sa[i] + k] ? p-1 : p++;
        if (p >= n) break;
        m = p;
    }
}

/*求辅助数组height[]*/
/*定义height[]为sa[i-1]和sa[i]也就是排名相邻的两个后缀的最长公共前缀长度*/
void getheight(int n) {
    int k = 0;
    for (int i = 0; i < n; ++i)
        rk[sa[i]] = i;
    for (int i = 0; i < n; ++i) {
        if (k)
            k--;
        int j = sa[rk[i]-1];
        while (text[i+k] == text[j+k])
            k++;
        height[rk[i]] = k;
    }
}

int main() {
    int len1;
    int res;
    while (scanf("%s", text) != EOF) {
```

```
        n = strlen(text);
        len1 = n;
        text[n] = '$'; //用$分割两个字符串
        scanf("%s", text + n + 1); //将第2个字符串和第1个字符串合并
        n = strlen(text);
        calc_sa();
        getheight(n);
        res = 0;
        for (int i = 0; i < n; ++i)
            //找最大的height[i]，并且它对应的sa[i-1]和sa[i]分别属于前后两个字符串
            if (height[i] > res &&
                    ((sa[i-1] < len1 && sa[i] >= len1) || (sa[i-1] >= len1 && sa[i] < len1)))
                res = height[i];
        printf("%d\n", res);
    }
}
```

## 5.7 Manacher

### 5.7.1 Manacher.cpp

```
#include <cstdio>
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define bug printf("<------>\n");

vector<int> radius, let;  /*回文子串半径*/
string expa_str; /*扩展后的串*/
/*原串、最长回文子串开始位置、最长回文子串长度*/
void Manacher(const string &str, int &pos, int &max_len) {
    int orig_len = str.length(); /*原串长度*/
    int expa_len = (orig_len + 1) << 1; /*扩展串长度*/
    max_len  = 0;
    radius.resize(expa_len + 1);
    expa_str.resize(expa_len + 1);
    //@#0#1#2#3#4#5#6#7#8#9#$
    expa_str[0] = '@';
    expa_str[1] = '#';
    /*expa_len是扩展串的长度减一，不过串从零开始*/
    expa_str[expa_len] = '$';
    for (int i = 1; i <= orig_len; ++i) {
        /*偶位置对应原串字符*/
        expa_str[i << 1] = str[i-1];
        /*不更改奇位置，只更改偶位置*/
        expa_str[i << 1 | 1] = '#';
    }
    radius[1] = 1; /*显然回文子串半径大于等于1*/
    //本应该计算到enpa_len + 1的，但那个位置是'#'，不需要计算了
    for (int max_R = 0, center = 0, i = 2; i < expa_len; ++i) {
        /*根据i探查到的位置是否超过了最右回文子串的右边界，初始化i的回文半径，核心操作*/
        radius[i] = i < max_R ? min(max_R-i, radius[2*center-i]) : 1;
        /*暴力拓展中心在i位置的最长回文子串半径radius[i]*/
        while (expa_str[i-radius[i]] == expa_str[i+radius[i]])
            ++radius[i];
        /*更新最右回文子串的右边界*/
        if (radius[i] + i > max_R) {
            max_R = radius[i] + i;
            center = i;
        }
        /*更新最长回文子串的位置与长度*/
        if (radius[i] - 1 > max_len) {
            /*原串的最长回文子串*/
            max_len = radius[i] - 1;
            /*原串的最长回文子串的起点*/
            pos     = (center - radius[i] + 1) >> 1;
        }
```

```
    }
}

//odd为false，字符串为奇回文串
//以x为中心的最长回文子串的长度
//如果为偶回文串，则以x，x+1中心为中心的最长回文子串的长度
int start_mid(int x, bool odd) {
    return odd ? radius[(x+1) << 1] - 1 : radius[(x+1) << 1 | 1] - 1;
}

//知道回文左边界，且在Manacher函数运行结束后使用
int start_left(int x, string str) {
    //let[i]表示以x为起点，的最长回文子串的中心位置
    int expand_len = (str.length() + 1) << 1;
    //let数组在扩展之后的字符串上，以位置i为起点的最长回文子串的中心在哪里
    let.resize(expand_len + 1);
    //计算维护以每个位置为起点的最长回文串
    for (int i = 0; i <= expand_len; i++)
        let[i] = 0;
    for (int i = 2; i < expand_len; i++)
        if (let[i - radius[i] + 1] < i + 1)
            let[i - radius[i] + 1] = i + 1;
    for (int i = 1; i <= expand_len; i++)
        if (let[i] < let[i - 1])
            let[i] = let[i - 1];
    //返回以x为起点的最长回文子串的长度
    return let[(x + 1) << 1] - ((x + 1) << 1);
}

int main() {
    //string str = "0123456789";
    string str = "DDDDBBDBA|BCBAAABBAAABCB|DB";
    string subStr;
    int pos;
    int max_len;
    Manacher(str, pos, max_len);
    printf("pos     = %d\n", pos);
    printf("max_len = %d\n", max_len);
    cout << "subStr  = " << str.substr(pos, max_len) << endl;
    printf("length  = %d\n", start_left(pos, str));
    printf("length  = %d\n", start_mid(pos+max_len/2-1, false));
    return 0;
}
```

# 6 数学问题

## 6.1 数学公式

### 6.1.1 数学公式.md

```
# 数学公式

## 组合数
- $C_n^m = \frac{n \times (n-1) \times (n-2) \times \cdots \times (n-m+1)}{m!}$

## 约数个数
- $cnt = ({\alpha}_1 + 1) + ({\alpha}_2 + 1) + ({\alpha}_3 + 1) + \dots + ({\alpha}_n + 1)$

## 约数之和

- $(p_1^0 + p_1^1 + p_1^2 + \dots + {p_1}^{{\alpha}_1}) \times (p_2^0 + p_2^1 + p_2^2 + \dots + {p_2}^{{\
    alpha}_2}) \times \dots \times (p_n^0 + p_n^1 + p_n^2 + \dots + {p_n}^{{\alpha}_n})$

## 等比数列之和

- $S_n = \frac{a_1 (1 - q^n)}{1 - q} (q \neq 1)$
```

## 6.2 位运算

### 6.2.1 lowbit.cpp

```cpp
#include <cstdio>
#define lowbit(x) ((x) & -(x))

int main() {
    printf("%d\n", lowbit(12)); // 1100
    return 0;
}
```

### 6.2.2 二进制状态压缩.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：02-二进制状态压缩.cpp
 *   创建日期：2021年05月08日 ---- 22时44分
 *   题    目：CH 0103
 *   算    法：旅行商问题，哈密顿路径
 *   描    述：0~n-1这n个点，从点0开始行走到点n-1结束，每个点都要经过
 *           且每个点到另外一个点有权重，问最小的花费是多少
 *           发现：
 *           假设已经经过了0, 1, 2, 3这四个点，由于要从0开始，那么会有3! = 6种方式
 *           0 -> 1 -> 2 -> 3  18
 *           0 -> 2 -> 1 -> 3  20
 *           ...
 *           在上面的两种方案里，对于后面的点的选择，一定不会选择第二种路径
 *           因为我们只需要在经过相同的点且最后的位置相同的行走方式中找到
 *           权重和最小的那种方式就行了，不管在这些点中是如何行走的，不会影响到
 *           后面点的选择
 *           1. 哪些点被用过
 *           2. 现在停在哪些点上
 *
 *           dp[state][j] = dp[state_k][k] + weight[k][j];
 *           state_k是state去掉j的集合，state_k要包含点k
 *           状态压缩，用一个数二进制中哪些是1来表示这个点被经过
 *           0, 1, 4    ->    10011
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 20;
#define _min(a, b) (a < b ? a : b)
/*
 * dp[i][j]中i的二进制为1的点已经被经过，当前处于点j
 * 如dp[7][1]中7的二进制为0111，则有点0、点1、点2都已经被经过，当前位于点1
 * 状态转移方程：dp[i][j] = min(dp[i][j], dp[i ^ (1<<j)][k] + weight[k][j]);
 * 既然状态为i的点都被经过，而当前位于点j，显然上一个状态是dp[i ^ (1<<j)][k] (k是状态i中非j的点)
 */
int dp[1 << maxn][maxn];
int weight[maxn][maxn];

int hamilton(int n) {
    memset(dp, 0x3f, sizeof(dp));
    dp[1][0] = 0; //起始点为0，所以点0到点0的距离是0
    for (int i = 1; i < (1 << n); ++i)
        for (int j = 0; j < n; ++j) //枚举当前所在的点
            if ((i >> j) & 1) //判断路径i中是否包括当前点j,如果包括当前点，则可以进行状态转移
                for (int k = 0; k < n; ++k) //要完成点k到点j的转移，所以要来枚举k
                    if (i - (1<<j) >> k & 1) //只有去除掉点j后路线中仍然包含点k才能说明路线是在点k的基础上向点j转移的
                        dp[i][j] = _min(dp[i][j], dp[i-(1<<j)][k] + weight[k][j]);
    return dp[(1 << n)-1][n-1]; //由于题目要求计算从点n到点n-1的路径长度，所以(1<<n)-1的二进制形式为111...111[共有(n-1)个1]
}

int main() {
```

```
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", &weight[i][j]);
    int res = hamilton(n);
    printf("%d\n", res);
    return 0;
}
```

#### 6.2.3 整数除以 2.cpp

```
#include <cstdio>
int main() {
    int i = -3;
    printf("%d\n", i / 2);  //得到-1，说明C++编译器实现为除以2向0取整
    return 0;
}
```

### 6.3 快速幂

#### 6.3.1 快速幂.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：快速幂.cpp
 *   创建日期：2021年10月08日 星期五 22时59分30秒
 *   题    目：AcWing 0875 快速幂
 *   算    法：快速幂
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
typedef long long ll;
int MOD;

int binpow(int base, int expo) {
    ll res = 1;
    while (expo) {
        if (expo & 1)
            res = (1LL * res * base) % MOD;
        base = (1LL * base * base) % MOD;
        // 也就是base分别是2, 4, 8, 16, ...
        expo >>= 1;
    }
    return res;
}

int main() {
    int n; scanf("%d", &n);
    while (n -- ) {
        int a, b, p;
        scanf("%d %d %d", &a, &b, &p);
        MOD = p;
        printf("%lld\n", binpow(a, b));
    }
    return 0;
}
```

#### 6.3.2 快速幂测试.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：快速幂.cpp
 *   创建日期：2021年03月10日 ---- 21时43分
```

```
 *   算    法: 快速幂
 *   描    述: 使用了模板template
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int mod = 99991;

// 计算 a^b % m
// 1.17秒
long long binaryPow(long long a, long long b, long long m) {
    if (b == 0) return 1;

    if (b % 2 == 1)
        return a * binaryPow(a, b - 1, m) % m;
    else {
        long long mul = binaryPow(a, b / 2, m);
        return mul * mul % m;
    }
}

//0.45秒
long long binPow(long long base, long long expo) {
    long long res = 1;
    while (expo != 0) {
        if (expo & 1)
            res = (1LL * res * base) % mod;
        base = (1LL * base * base) % mod;
        expo >>= 1;
    }
    return res;
}

/*
 *template<typename T>
 *T binPow(T base, T expo) {
 *    if (!expo) return 1;
 *    if (expo % 2)
 *        return base * binPow(base, expo - 1) % mod;
 *    else
 *        return binPow(base, expo / 2) % mod * binPow(base, expo / 2) % mod;
 *}
 */

template<typename T>
T binPow(T base, T expo) {
    T res = 1;
    while (expo != 0) {
        if (expo & 1)
            res = (1LL * res * base) % mod;
        base = (1LL * base * base) % mod;
        expo >>= 1;
    }
    return res;
}

int main() {
    long long base = 23, expo = 19898283988388888;
    long long res;
    for (int i = 0; i < 1000000; ++i)
        //res = binaryPow(base, expo, mod);
        res = binPow(base, expo);
    printf("%lld\n", res);
    return 0;
}
```

### 6.3.3 矩阵快速幂.cpp

```
#include <cstdio>
```

```cpp
#include <cstring>
const int maxn = 2;  // 定义矩阵的阶
const int MOD  = 99991;

struct Matrix {
    int m[maxn][maxn];
    Matrix() {
        memset(m, 0, sizeof(m));
    }
};

/*矩阵乘法*/
Matrix Multi(Matrix a, Matrix b) {
    Matrix res;
    for (int i = 0; i < maxn; ++i)
        for (int j = 0; j < maxn; ++j)
            for (int k = 0; k < maxn; ++k)
                res.m[i][j] = (res.m[i][j] + a.m[i][k] * b.m[k][j]) % MOD;
    return res;
}

/*矩阵快速幂*/
Matrix fastm(Matrix a, int n) {
    Matrix res;
    for (int i = 0; i < maxn; ++i)
        res.m[i][i] = 1;
    while (n) {
        if (n & 1)
            res = Multi(res, a);
        a = Multi(a, a);
        n >>= 1;
    }
    return res;
}
```

## 6.4 进制转换

### 6.4.1 进制转换.cpp

```cpp
/*除了十进制的数用一个int型变量直接存储之外，其他进制的数都用vector容器存储*/
#include <cstdio>
#include <vector>
#include <cmath>
using namespace std;
// B: bit 二进制
// T: ternary 三进制
// Q: quaternary 四进制
// O: octonary 八进制
// D: decimal 十进制
// H: hexadecimal 十六进制

// 将一个P进制的数转换为D进制的数
int anytoD(vector<int> num_P, int base_P) {
    int num_D  = 0;
    for (int i = 0; i < (int)num_P.size(); ++i)
        num_D += (num_P[i] * pow(base_P, i));
    return num_D;
}

// 将一个D进制的数转换为C进制的数
vector<int> numto;
void Dtoany(int num_D, int base_C) {
    do {
        numto.push_back(num_D % base_C);
        num_D /= base_C;
    }while (num_D != 0);
}

int main() {
```

```
    int num = 101110011;
    int base_P = 2;
    vector<int> num_P;
    while (num) {
        num_P.push_back(num % 10);
        num /= 10;
    }
    int base_C = 8;
    int num_D = anytoD(num_P, base_P);
    Dtoany(num_D, base_C);
    for (auto it = numto.end()-1; it != numto.begin()-1; --it)
        printf("%d", *it);
    return 0;
}
```

### 6.4.2 进制转换.cpp

```
/*--------------------------------------------------------------
 *
 *   文件名称：进制转换.cpp
 *   创建日期：2020年08月10日
 *   描    述：板子
 *
--------------------------------------------------------------*/

//将其他进制的数转换为十进制
#include <cstdio>

/**
 * 一个函数，将一个字符串转化为十进制数
 * 这里不知道数字字符个数,有大小写字母
 * 12345
 * 1 * 10 + 2
 * 12 * 10 + 3
 * 123 * 10 + 4
 * 1234 * 10 + 5
 * .....
 */
int Conversion(char str[], int len) {
    int unit = 0;
    for (int i = 0; i < len; i++) {
        if (str[i] >= 'A' && str[i] <= 'Z')
            unit = 62 * unit + (str[i] - 'A');
        else if (str[i] >= 'a' && str[i] <= 'z')
            unit = 62 * unit + (str[i] - 'a') + 26;
        else
            unit = 62 * unit + (str[i] - '0') + 52;
    }
    return unit;
}

int main() {
    char str[] = "BCD";
    int unit = Conversion(str, 3);
    printf("%d\n", unit);
    return 0;
}
```

## 6.5 高精度计算

### 6.5.1 factN.java

```
/*--------------------------------------------------------------
 *
 *   文件名称：_01fact_N.java
 *   创建日期：2021年03月10日 ---- 20时35分
 *   题    目：hdu1042
```

```
 *    算    法：高精度计算
 *    描    述：输入：N(0 <= N <= 10000)    输出：N!
 *
 ----------------------------------------------------------------*/

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

public class _01fact_N {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        while (input.hasNextInt()) {  //判断是否仍有int型输入
            int n = input.nextInt();  //输入int型变量n
            BigInteger res = BigInteger.ONE;
            for (int i = 1; i <= n; ++i)
                res = res.multiply(BigInteger.valueOf(i)); //BigInteger.valueOf(i)强制类型转换
            System.out.println(res);
        }
    }
}
```

### 6.5.2 二进制方法实现 64 位整数乘法.cpp

```
#include <cstdio>
typedef long long ll;
ll mod;

ll binTimes(ll l1, ll l2) {
    ll res = 0;
    while (l2) {
        if (l2 & 1)
            res = (res + l1) % mod;
        l1 = (l1 * 2) % mod;
        l2 >>= 1;
    }
    return res;
}

int main() {
    ll a, b;
    scanf("%lld %lld %lld", &a, &b, &mod);
    ll res = binTimes(a, b);
    printf("%lld\n", res);
    return 0;
}
```

### 6.5.3 暴躁 py.py

```
n = eval(input())
res = 1

# 以换行结束输入
for i in range(n):
    x = eval(input())
    res = res * x

print(res)
```

### 6.5.4 高精度乘法.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：06-高精度乘法.cpp
 *    创建日期：2021年08月06日 星期五 16时25分58秒
 *    题    目：AcWing 0793 高精度乘法
 *    算    法：高精度乘法
```

```
 *   描    述：给定两个非负整数(不含前导 0) A 和 B，计算 A * B
 *   使用了string输入乘数，其实也可以使用数组存储，代码改一下就可以了
 *
 *   模板化就模板化吧，挺简洁的
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
#define NEXTLINE cout << endl;

// C = A * b, A >= 0, b >= 0
vector<int> mul(vector<int> &A, int b) {
    vector<int> C;
    // 太模板化了，可以拆开写，结束条件是 i < A.size()
    // 这时 t 可能不为0，那就写一个while (t) {c.pb(t % 10); t /= 10}
    for (int i = 0, t = 0; i < A.size() || t; ++i) {
        if (i < A.size())   // 这句要要相应的改
            t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }
    // 去除前导零
    while (C.size() > 1 && C.back() == 0)
        C.pop_back();
    return C;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    string n1;
    int b;
    cin >> n1 >> b;
    vector<int> A;

    // vector 推入的第一个位置是个位
    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');

    auto C = mul(A, b);
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    NEXTLINE;
    return 0;
}
```

### 6.5.5 高精度减法.cpp

```
#include <cstdio>
#include <vector>
#include <string>
#include <iostream>
#include <algorithm>
using namespace std;
#define NEXTLINE cout << endl;

// C = A - B, 满足A >= B, A >= 0, B >= 0
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); ++i) {
        t = A[i] - t;
        if (i < B.size())
            t -= B[i];
        C.push_back((t + 10) % 10);
```

```
        t < 0 ? t = 1 : t = 0;
    }

    while (C.size() > 1 && C.back() == 0)
        C.pop_back();
    return C;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    string n1, n2;
    cin >> n1 >> n2;
    vector<int> A, B;

    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');
    for (int i = n2.length() - 1; i >= 0; --i)
        B.push_back(n2[i] - '0');

    //判断A < B, 则swap
    if (n1.length() < n2.length() || (n1.length() == n2.length() && n1 < n2))
        swap(A, B);

    auto C = sub(A, B);

    if (n1.length() < n2.length() || (n1.length() == n2.length() && n1 < n2))
        cout << "-";
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    NEXTLINE;
    return 0;
}
```

### 6.5.6 高精度加法.cpp

```
#include <cstdio>
#include <vector>
#include <string>
#include <iostream>
#define NEXTLINE cout << endl;
using namespace std;

// C = A + B, A >= 0, B >= 0
vector<int> add(vector<int> &A, vector<int> &B) {
    if (A.size() < B.size())
        return add(B, A);

    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); ++i) {
        t += A[i];
        if (i < B.size())
            t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }

    if (t)
        C.push_back(t);
    return C;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    string n1, n2;
```

```
    vector<int> A, B;
    cin >> n1 >> n2; //n1 = 123456

    // A = [6, 5, 4, 3, 2, 1];
    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');
    for (int i = n2.length() - 1; i >= 0; --i)
        B.push_back(n2[i] - '0');

    auto C = add(A, B);
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    NEXTLINE;
    return 0;
}
```

#### 6.5.7 高精度除法.cpp

```
#include <cstdio>
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define NEXTLINE cout << endl;

// A / b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r) {
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; --i) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }

    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0)
        C.pop_back();
    return C;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    string n1;
    int b;
    cin >> n1 >> b;
    vector<int> A;

    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');

    int r;
    auto C = div(A, b, r);
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    cout << endl << r << endl;
    // NEXTLINE;
    return 0;
}
```

### 6.6 数论

#### 6.6.1 素数

##### 6.6.1.1 埃氏筛法.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：埃氏筛法.cpp
 *   创建日期：2021年08月04日 星期三 00时17分10秒
 *   题    目：AcWing 0868 筛质数
 *   算    法：<++>
 *   描    述：
 *       一个质数定理：[1, n]中有(n/lnn)个质数
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e6;
//存储1 ~ 1e6内的所有素数
int primes[maxn], cnt;
//对1 ~ 1e6内的数标记，未被筛掉的素数标记为false
bool sifter[maxn];

void get_primes(int n) {
    for (int i = 2; i <= n; ++i)
        if (sifter[i] == false) {
            primes[cnt++] = i;
            for (int j = i + i; j <= n; j += i)
                sifter[j] = true;
        }
}

int main() {
    int n; scanf("%d", &n);
    get_primes(n);
    printf("%d\n", cnt);
    return 0;
}
```

#### 6.6.1.2 欧拉筛法.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：欧拉筛法.cpp
 *   创建日期：2021年08月04日 星期三 00时44分47秒
 *   题    目：AcWing 0868 筛质数
 *   算    法：欧拉筛法
 *   描    述：<++>
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e6;
//存储1 ~ n内的所有素数
int primes[maxn], cnt;
//最终为false的数是素数
bool sifter[maxn];

void get_primes(int n) {
    //从2开始，判断i是否为素数
    for (int i = 2; i <= n; ++i) {
        if (sifter[i] == false)
            primes[cnt++] = i;

        for (int j = 0; primes[j] <= n / i; ++ j) {
            // 每一个倍数标记为不是素数
            sifter[primes[j] * i] = true;

            // primes[j]为i的最小素因子，分析下一个i
            if (i % primes[j] == 0)
                break;
        }
    }
}
```

```

int main() {
    int n; scanf("%d", &n);
    get_primes(n);
    printf("%d\n", cnt);
    return 0;
}

/* *
 * n = 4;
 *
 * i = 2, 未筛去, primes[0] = 2
 *      j = 0, primes[j] = 2 <= 4 / 2, 4被筛去, i % primes[j] == 2 % 2 == 0, break
 * i = 3, 未筛去, primes[1] = 3
 *      j = 0, primes[j] = 2 > 4 / 3, break
 * i = 4, 已筛去
 *
 * 这里为什么i % primes[j] == 0, 就要break呢?
 * 如果不break, 4*3 = 12就要被筛去, 实际上12会在i = 6, primes[j] = primes[1] = 2, 6*2 = 12被筛去
 * 2是12的最小素因子, 比如100的最小素因子是2, 所以100会在50 * 2时筛选掉, 避免重复筛选
 * 合数的所有最小因子都是素数, 此算法用到这一点
 */
```

#### 6.6.1.3 试除法判定素数.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称: 试除法判定素数.cpp
 *   创建日期: 2021年08月03日 星期二 13时59分14秒
 *   题    目: AcWing 0866 试除法判定素数
 *   算    法: 试除法判定素数
 *   描    述: 给定 n 个正整数 ai, 判定每个数是否是质数
 *
 ----------------------------------------------------------------*/

#include <cstdio>

bool is_prime(int n) {
    if (n < 2)
        return false;
    for (int i = 2; i <= n / i; ++i)
        if (n % i == 0)
            return false;
    return true;
}

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        int num; scanf("%d", &num);
        if (is_prime(num))
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```

### 6.6.2 最大公约数

#### 6.6.2.1 最大公约数.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称: 01.cpp
 *   创建日期: 2021年09月29日 星期三 16时35分55秒
 *   题    目: AcWing 0872 最大公约数
 *   算    法: 欧几里德算法(辗转相除法)
 *   描    述: 给定 n 对正整数 ai,bi, 请你求出每对数的最大公约数
```

```
 *
 ----------------------------------------------------------------*/

#include <cstdio>

// gcd(a, b) = gcd(b, a mod b)
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int main() {
    int n; scanf("%d", &n);
    while (n --) {
        int a, b;
        scanf("%d %d", &a, &b);
        printf("%d\n", gcd(a, b));
    }
    return 0;
}
```

### 6.6.3 欧拉函数

#### 6.6.3.1 欧拉函数.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：欧拉函数.cpp
 *   创建日期：2021年09月29日 星期三 17时48分21秒
 *   题    目：AcWing 0873 欧拉函数
 *   算    法：欧拉函数
 *   描    述：给定 n 个正整数 ai，请你求出每个数的欧拉函数
 *   若在算数基本定理中，N = pa1 * pa2 * ... * pam，则：
 *   φ(N) = N × (p1-1)/p1 × (p2-1)/p2 × … × (pm-1)/pm
 *
 ----------------------------------------------------------------*/

#include <cstdio>
int main() {
    int n; scanf("%d", &n);
    while (n --) {
        int a; scanf("%d", &a);
        int res = a;
        for (int i = 2; i <= a / i; ++ i) {
            if (a % i == 0) {
                res = res / i * (i - 1);
                while (a % i == 0)
                    a /= i;
            }
        }
        if (a > 1)
            res = res / a * (a - 1);
        printf("%d\n", res);
    }
    return 0;
}
```

#### 6.6.3.2 筛法求欧拉函数.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：筛法求欧拉函数.cpp
 *   创建日期：2021年09月29日 星期三 17时58分25秒
 *   题    目：AcWing 0874 筛法求欧拉函数
 *   算    法：<++>
 *   描    述：给定一个正整数 n，求 1∼n 中每个数的欧拉函数之和
 *
 ----------------------------------------------------------------*/

#include <cstdio>
```

```cpp
const int maxn = 1e6 + 5;
typedef long long ll;
int primes[maxn], cnt;
int phi[maxn];
bool sifter[maxn];

ll get_eulers(int n) {
    phi[1] = 1;
    for (int i = 2; i <= n; ++ i) {
        if (sifter[i] == false) {
            primes[cnt ++ ] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; primes[j] <= n / i; ++ j) {
            sifter[primes[j] * i] = true;
            if (i % primes[j] == 0) {
                /**
                 * i * primes[j] 比 i 多一个质因子 primes[j]
                 * 所以根据欧拉公式，N 变化了，且需要多乘一个 (1 - 1/primes[j])
                 * 但是，又因为 i % primes[j] == 0，所以，第一句是错的，看似多乘了
                 * 一个质因子，实际上，i 本身就有这个 primes[j] 的质因子
                 * 因为 N 由 i 变成 i * primes[j]
                 * 根据欧拉公式，phi[i * primes[j]] = primes[j] * phi[i];
                 */
                phi[i * primes[j]] = primes[j] * phi[i];
                break;
            }
            else {
                // 否则就多乘一个 1 - 1 / primes[j] 嘛
                // phi[i * primes[j]] = primes[j] * phi[i] / primes[j] * (primes[j] - 1);
                phi[i * primes[j]] = phi[i] * (primes[j] - 1);
            }
        }
    }

    ll res = 0;
    for (int i = 1; i <= n; ++ i)
        res += phi[i];
    return res;
}

int main() {
    int n; scanf("%d", &n);
    printf("%lld\n", get_eulers(n));
    return 0;
}
```

### 6.6.4 类欧几里德算法

#### 6.6.4.1 扩展欧几里德算法.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：扩展欧几里德算法.cpp
 *   创建日期：2021年10月09日 星期六 19时18分35秒
 *   题    目：AcWing 0877 扩展欧几里德算法
 *   算    法：扩展欧几里德算法
 *   描    述：求 ax + by = gcd(a, b) 任意一个解
 *
 ----------------------------------------------------------------*/

#include <cstdio>

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int exgcd(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
```

```
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

int main() {
    int n; scanf("%d", &n);
    while (n -- ) {
        int a, b, x, y;
        scanf("%d %d", &a, &b);
        exgcd(a, b, x, y);
        printf("%d %d\n", x, y);
    }
    return 0;
}
```

### 6.6.5 乘法逆元

#### 6.6.5.1 递推.cpp

```
#include<cstdio>
int inv[100001];
int mod = 99991;

//在(mod 99991)的意义下，2 - 10000分别对应的逆元
void Inv() {
    inv[1] = 1;
    for (int i = 2; i < 10000; ++i) {
        inv[i] = - mod/i * inv[mod%i] % mod;
        inv[i] = (mod + inv[i]) % mod;
        /*inv[i] = (mod - mod/i) * inv[mod%i] % mod;*/ /*会溢出*/
    }
}

int main() {
    Inv();
    for (int i = 0; i < 20; ++i)
        printf("%d ", inv[i]);
    return 0;
}
```

#### 6.6.5.2 终极递推.cpp

```
#include <cstdio>
typedef long long ll;
const int maxn = 1e7;
ll inv[maxn];
ll mod = 999983;

//生成一个表
void Inv() {
    inv[1] = 1;
    for (int i = 2; i < maxn; ++i)
        inv[i] = (mod - mod/i) % mod * inv[mod%i] % mod;
}

//求base的逆元
ll InvKB(ll base) {
    if (base == 1)
        return 1;
    return InvKB(mod%base) * (mod-mod/base) % mod;
}

int main() {
    Inv();
    for (int i = 0; i < 20; ++i)
        printf("%lld ", inv[i]);
```

```
25
26      printf("%lld\n", InvKB(9));
27      return 0;
28  }
```

#### 6.6.5.3 快速幂求逆元-费马小定理.cpp

```
1   /*-----------------------------------------------------------------
2    *
3    *   文件名称：快速幂求逆元.cpp
4    *   创建日期：2021年10月09日 星期六 17时54分32秒
5    *   题      目：AcWing 0876 快速幂求逆元
6    *   算      法：费马小定理，快速幂
7    *   描      述：不与 MOD 互质的数没有在 (mod MOD) 下的逆元
8    *   所以快速幂要求 MOD 一定是一个素数，而扩展欧几里德算法只要求
9    *   gcd(a, MOD) = 1，两数互质，MOD 不一定是素数
10   *
11   -----------------------------------------------------------------*/
12
13  #include <cstdio>
14  typedef long long ll;
15  int MOD;
16
17  int binpow(int base, int expo) {
18      ll res = 1;
19      while (expo) {
20          if (expo & 1)
21              res = (1LL * res * base) % MOD;
22          base = (1LL * base * base) % MOD;
23          // 也就是base分别是2, 4, 8, 16, ...
24          expo >>= 1;
25      }
26      return res;
27  }
28
29  // 返回0表示无解
30  int inv(int x) {
31      return binpow(x, MOD - 2);
32  }
33
34  int main() {
35      int n; scanf("%d", &n);
36      while (n -- ) {
37          int a, p;   // 求a在(mod p)下的逆元
38          scanf("%d %d", &a, &p);
39          MOD = p;
40          int inverse = inv(a);
41          // p = 2 情况特殊
42          if (a % p) printf("%d\n", inverse);
43          else puts("impossible");
44      }
45      return 0;
46  }
```

#### 6.6.5.4 扩展欧几里德算法求逆元.cpp

```
1   /*-----------------------------------------------------------------
2    *
3    *   文件名称：扩展欧几里德算法求逆元.cpp
4    *   创建日期：2021年10月11日 星期一 17时43分47秒
5    *   题      目：<++>
6    *   算      法：扩展欧几里德算法
7    *   描      述：前提条件：a 与 MOD 互质，如果 MOD 本身就是就是素数
8    *   就不用考虑这个问题
9    *
10   *   ax \equiv 1 (mod MOD)   x 就是 a 的逆元，逆元就是这样定义的
11   *
12   -----------------------------------------------------------------*/
```

```

#include <cstdio>
int MOD;

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

int inv(int a) {
    int x, y;
    int gcd = exgcd(a, MOD, x, y);    // 此时得到的x是方程的一个解，但不一定是方程的最小正整数解，x可能为负
    gcd += 1;
    // exgcd(a, mod, x, y);
    return (x % MOD + MOD) % MOD;    // (x % m + m) % m 是方程最小正整数解，也就是a模m的逆元
}

int main() {
    MOD = 99991;
    for (int i = 2; i < 20; ++ i)
        printf("%d ", inv(i));   // 求解 i 在 (mod MOD) 下的逆元
    return 0;
}
```

### 6.6.6 线性同余方程

#### 6.6.6.1 线性同余方程.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：线性同余方程.cpp
 *   创建日期：2021年10月11日 星期一 00时29分07秒
 *   题    目：AcWing 0878 线性同余方程
 *   算    法：线性同余方程 扩展欧几里德算法
 *   描    述：求满足 a * x \equiv b (mod m) 的一个解
 *   转化为 ax + my = b，求x，这个式子有解的充要条件是 gcd(a, m) | b
 *
 *   扩展欧几里德算法求的是 ax + my = d (d = gcd(a, b)
 *   这里要求 ax + my = b 所以等式两边同时扩大 d/b 倍
 *   -> (d/b)(ax + my) = d
 *   ax' + my' = d
 *   最后求出的 x = x' * b/d
 *
 ----------------------------------------------------------------*/

#include <cstdio>
typedef long long ll;

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

int main() {
    int n; scanf("%d", &n);
    while (n -- ) {
        int a, b, m;
        scanf("%d %d %d", &a, &b, &m);
        int x, y;
```

```
        int d = exgcd(a, m, x, y);
        if (b % d)
            puts("impossible");
        else
            printf("%lld\n", (ll)x * (b / d) % m);
    }
    return 0;
}
```

### 6.6.7 中国剩余定理

#### 6.6.7.1 表达整数的奇怪方式-模数不一定互质.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年10月11日 星期一 21时08分01秒
 *   题    目：AcWing 0204 表达整数的奇怪方式
 *   算    法：中国剩余定理
 *   描    述：下面是推导过程
 *   看推导过程
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long ll;
const int maxn = 25 + 5;

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

// Chinese Remainder Theorem
// k: 方程个数，n: 除数数组，a: 余数数组
ll CRT(int k, ll n[], ll a[]) {
    bool has_result = true;
    ll n1 = n[0], a1 = a[0];
    for (int i = 1; i < k; ++ i) {
        int n2 = n[i], a2 = a[i];
        ll k1, k2;   // k1', k2'
        ll gcd = exgcd(n1, n2, k1, k2);
        if ((a2 - a1) % gcd)
            return -1;
        k1 = k1 * (a2 - a1) / gcd;
        // 此句不需要，因为只要知道 k1 就知道 x 了
        // k2 = k2 * (a2 - a1) / gcd;
        ll t = n2 / gcd;
        k1 = (k1 % t + t) % t;
        a1 = n1 * k1 + a1;
        n1 = abs(n1 / gcd * n2);   // lcm
    }
    return (a1 % n1 + n1) % n1;
}

int main() {
    int n; scanf("%d", &n);
    ll a[maxn], m[maxn];   // 除数数组a[]，余数数组m[]
    for (int i = 0; i < n; ++ i)
        scanf("%lld %lld", &a[i], &m[i]);
    ll res = CRT(n, a, m);
    printf("%lld\n", res);
    return 0;
```

```
}
```

#### 6.6.7.2 表达整数的奇怪方式-模数互质.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：表达整数的奇怪方式.cpp
 *   创建日期：2021年10月11日 星期一 17时08分43秒
 *   题    目：AcWing 0204 表达整数的奇怪方式
 *   算    法：中国剩余定理
 *   算法描述：
 *             x \equiv a1 (mod n1)
 *             x \equiv a2 (mod n2)
 *             x \equiv a3 (mod n3)
 *             ...
 *             x \equiv ak (mod nk)
 *
 *             中国剩余定理：给出上面的 a1~ak，n1~nk，求出 x
 *
 *             由于 ni 不一定是素数，所以需要使用扩展欧几里德算法求逆元
 *
 *   算法流程：1. 计算所有模数的积 n = n1 * n2 * n3 * ... * nk
 *             2. 对于第 i 个方程：
 *                  1. 计算 mi = n / ni
 *                  2. 计算 mi 在模 ni 意义下的逆元 mi^-1
 *                  3. 计算 ci = mi * mi^-1   (不要对 ni 取模)
 *             3. 方程组的唯一解为 x = a1*c1%n + a2*c2%n + ... + ak*ck%n;
 *
 *   题目描述：
 *             给定 2n 个整数，a1, a2, ... an 和 m1, m2, ... mn，求最小的
 *             非负整数 x 满足
 *             x \equiv mi (mod ai)
 *
 *   输入样例：
 *             2
 *             8 7
 *             11 9
 *
 *             x mod 8 = 7
 *             x mod 11 = 9
 *             -> x = 31
 *
 -----------------------------------------------------------------*/

#include <cstdio>
typedef long long ll;
const int maxn = 25 + 5;

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

// Chinese Remainder Theorem
// k: k 方程个数
// n[]: n 数组(除数)
// a[]: a 数组(余数)
ll CRT(int k, ll n[], ll a[]) {
    ll prod_n = 1, res = 0;
    for (int i = 0; i < k; ++ i)
        prod_n *= n[i];
    for (int i = 0; i < k; ++ i) {
        ll mi = prod_n / n[i];
        // 计算 mi 在模 ni 意义下的逆元 inv_mi
```

```
        ll inv_mi, y;
        exgcd(mi, n[i], inv_mi, y);
        ll ci = mi * inv_mi;
        res = (res + a[i] * ci % prod_n) % prod_n;
    }
    return (res % prod_n + prod_n) % prod_n;
}

int main() {
    int n; scanf("%d", &n);
    ll a[maxn], m[maxn];
    for (int i = 0; i < n; ++ i)
        scanf("%lld %lld", &a[i], &m[i]);
    ll res = CRT(n, a, m);
    printf("%lld\n", res);
    return 0;
}
```

### 6.6.8 分解质因数

#### 6.6.8.1 分解质因数.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：分解质因数.cpp
 *   创建日期：2021年08月04日 星期三 00时03分26秒
 *   题    目：AcWing 0867 分解质因数
 *   算    法：试除法分解质因数
 *   描    述：使用试除法分解质因数
 *       n中至多只包含一个大于sqrt(n)的质因子
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#define NEXTLINE puts("");

void divide(int n) {
    // 从小到大枚举，i是底数，expo是指数
    for (int i = 2; i <= n / i; ++i)
        if (n % i == 0) {
            int expo = 0;
            while (n % i == 0) {
                n /= i;
                expo ++ ;
            }
            printf("%d %d\n", i, expo);
        }
    if (n > 1)
        printf("%d %d\n", n, 1);
}

int main() {
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        int num; scanf("%d", &num);
        divide(num);
        NEXTLINE;
    }
    return 0;
}
```

### 6.6.9 约数

#### 6.6.9.1 约数个数.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年09月28日 星期二 21时24分53秒
```

```
 *    结束日期: 2021年09月28日 星期二 22时37分59秒
 *    题    目: AcWing 0970 约数个数
 *    算    法: <++>
 *    描    述: 给定 n 个正整数 ai，请你输出这些数的乘积的约数个数
 *    答案对 1e9 + 7 取模
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <unordered_map>
using namespace std;
typedef long long ll;
const int MOD = 1e9 + 7;

int main() {
    int n; scanf("%d", &n);
    unordered_map<int, int> primes;
    while (n --) {
        int a; scanf("%d", &a);
        for (int i = 2; i <= a / i; ++ i) {
            while (a % i == 0) {
                a /= i;
                primes[i] ++ ;
            }
        }
        if (a > 1)
            primes[a] ++ ;
    }
    ll res = 1;
    for (auto prime : primes)
        res = res * (prime.second + 1) % MOD;
    printf("%lld\n", res);
    return 0;
}
```

#### 6.6.9.2 约数之和.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: 约数之和.cpp
 *    创建日期: 2021年09月29日 星期三 16时24分18秒
 *    题    目: AcWing 0871 约数之和
 *    算    法: <++>
 *    描    述: 给定 n 个正整数 ai，请你输出这些数的乘积的约数之和
 *    答案对 109+7 取模。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <unordered_map>
using namespace std;
typedef long long ll;
const int MOD = 1e9 + 7;

int main() {
    int n; scanf("%d", &n);
    unordered_map<int, int> primes;
    while (n --) {
        int a; scanf("%d", &a);
        for (int i = 2; i <= a / i; ++ i) {
            while (a % i == 0) {
                a /= i;
                primes[i] ++ ;
            }
        }
        if (a > 1)
            primes[a] ++ ;
    }
    ll res = 1;
```

```
    for (auto prime : primes) {
        int p = prime.first, a = prime.second;
        ll t = 1;
        while (a --)
            t = (t * p + 1) % MOD;
        res = res * t % MOD;
    }
    printf("%lld\n", res);
    return 0;
}
```

#### 6.6.9.3 试除法求约数.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年09月28日 星期二 18时28分47秒
 *   结束日期：2021年09月28日 星期二 18时37分09秒
 *   题    目：AcWing 0869 试除法求约数
 *   算    法：<++>
 *   描    述：给定 n 个正整数 ai，对于每个整数 ai，
 *   请你按照从小到大的顺序输出它的所有约数。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
#define NEXTLINE puts("");

vector<int> get_divisors(int n) {
    vector<int> res;
    for (int i = 1; i <= n / i; ++ i)
        if (n % i == 0) {
            res.push_back(i);
            if (n / i != i)
                res.push_back(n / i);
        }
    sort(res.begin(), res.end());
    return res;
}

int main() {
    int n; scanf("%d", &n);
    while (n --) {
        int a; scanf("%d", &a);
        vector<int> res = get_divisors(a);
        for (auto it = res.begin(); it != res.end(); ++ it)
            printf("%d ", *it);
        NEXTLINE;
    }
    return 0;
}
```

## 6.7 多项式

### 6.7.1 快速傅里叶变换

#### 6.7.1.1 Cooley-Tukey.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：Cooley-Tukey.cpp
 *   创建日期：2021年07月23日 星期五 12时07分52秒
 *   题    目：<++>
 *   算    法：快速傅里叶变换
 *   描    述：nlOgn的时间下得到两个序列的和的序列
 *       [2 3 4], [2 3 4]  -->  [4 5 5 6 6 6 7 7 8]
```

```
 *       如果是差的序列，那么先加上一个偏移量
 *       [2 3 4], [-4 -3 -2]
 *          -- +5 -->  [2 3 4], [1, 2, 3]
 *          -------->  [3 4 4 5 5 5 6 6 7]
 *          -- -5 -->  [-2 -1 -1 0 0 0 1 1 2]
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cmath>
using namespace std;
const int maxn = 1 << 10;   // 定义maxn时一定要让maxn是2的指数
const double PI = acos(-1.0);
int a[maxn], b[maxn];
#define bug printf("<-->");
#define NEXTLINE puts("");

struct Complex {
    double r, i;   // 一定注意，使用%f输出
    Complex() {}
    Complex(double _r, double _i) : r (_r), i (_i) {}
    inline void real(const double& x) {r = x;}
    inline double real() {return r;}
    inline Complex operator + (const Complex& rhs) const {
        return Complex (r + rhs.r, i + rhs.i) ;
    }
    inline Complex operator - (const Complex& rhs) const {
        return Complex (r - rhs.r, i - rhs.i);
    }
    inline Complex operator * (const Complex& rhs) const {
        return Complex (r*rhs.r - i*rhs.i, r*rhs.i + i*rhs.r);
    }
    inline void operator /= (const double& x) {
        r /= x, i /= x ;
    }
    inline void operator *= (const Complex& rhs) {
        *this = Complex (r*rhs.r - i*rhs.i, r*rhs.i + i*rhs.r);
    }
    inline void operator += (const Complex& rhs) {
        r += rhs.r, i += rhs.i;
    }
    inline Complex conj() {    // 共轭复数
        return Complex (r, -i) ;
    }
};

struct FastFourierTransform {
    // 自己封装的复数类
    Complex omega[maxn], omegaInverse[maxn];

    void init(const int& n) {
        for (int i = 0; i < n; ++i) {
            omega[i] = Complex(cos(2*PI / n*i), sin(2*PI / n*i));
            omegaInverse[i] = omega[i].conj();
        }
    }

    void transform(Complex *a, const int& n, const Complex* omega) {
        for (int i = 0, j = 0; i < n; ++i) {
            if (i > j)
                swap(a[i], a[j]);
            for (int l = n >> 1; (j ^= l) < l; l >>= 1);
        }

        for (int l = 2; l <= n; l <<= 1) {
            int m = l / 2;
            for (Complex *p = a; p != a + n; p += l)
                for (int i = 0; i < m; ++i) {
                    Complex t = omega[n / l * i] * p [m + i];
```

```
                    p[m + i] = p[i] - t;
                    p[i] += t;
                }
        }
    }

    // 由系数表达式离散为点值表达式
    void dft(Complex *a, const int& n) {
        transform(a, n, omega);
    }

    // 由点值表达式转化为系数表达式
    void idft(Complex *a, const int& n) {
        transform(a, n, omegaInverse);
        for (int i = 0; i < n; ++i)
            a[i] /= n;
    }
} fft;

int main() {
    /*
     * 这是两个多项式的系数:
     * A(x) = 5 + 2x + 3x^2
     * B(x) = 2 + 6x + x^2
     * 下面的len1，len2分别是两个多项式最大项的最高次幂+1
     * FFT中len一定要是2^k这种形式，否则在进行分治时会左右不均失败
     */
    a[0] = 5; a[1] = 2; a[2] = 3;
    b[0] = 2; b[1] = 6; b[2] = 1;
    int len1 = 3, len2 = 3;
    int len = 1;
    while (len < len1 * 2 || len < len2 * 2)
        len <<= 1;
    fft.init(len);   // 初始化\omega

    Complex x1[maxn], x2[maxn];   // 存储两个多项式的系数，幂由低到高
    // 将原本的系数用复数表达，实部是0.0
    for (int i = 0; i < len; ++i) {
        x1[i].r = a[i]; x1[i].i = 0.0;
        x2[i].r = b[i]; x2[i].i = 0.0;
    }
    // 由系数表达式离散为点值表达式
    fft.dft(x1, len); fft.dft(x2, len);
    // O(n)时间算出结果的点值表达式
    for (int i = 0; i < len; ++i)
        x1[i] = x1[i] * x2[i];
    // 根据结果的点值表达式得到系数表达式
    fft.idft(x1, len);

    int res[maxn];   // 用不用这个存无所谓，存一下吧，和开头的系数也是数组统一
    for (int i = 0; i < len; ++i)
        res[i] = (int)(x1[i].r + 0.5);
    // 可能高位上系数为0，我们给它预留的位置还是比较多的，它没用完
    while (len && res[len-1] == 0)
        --len;
    // 结果就是C(x) = 10 + 34x + 23x^2 + 20x^3 + 3x^4
    for (int i = 0; i < len; ++i)
        printf("%d ", res[i]);
    NEXTLINE
    return 0;
}
```

## 6.8 生成函数

### 6.8.1 普通生成函数

#### 6.8.1.1 递归求整数划分.cpp

```
/*-----------------------------------------------------------------
 *
```

```
 *    文件名称：01-hdu1082-递归求整数划分.cpp
 *    创建日期：2021年03月12日 ---- 11时05分
 *    题    目：hdu1028
 *    算    法：递归
 *    描    述：题目是1 <= n <= 120所以会TLE
 *
 ----------------------------------------------------------------*/

#include <cstdio>
/**
 * 将n划分成最大数不超过m的划分数
 * 比如当n == 4时
 * 划分有5种{1, 1, 1, 1}, {1, 1, 2}, {2, 2}, {1, 3}, {4}
 * 如果有条件限制，比如将4划分成不超过2的划分数
 * 则只有3种{1, 1, 1, 1}, {1, 1, 2}, {2, 2}
 *
 * 不过还是难以置信，怎么想到会有两个参数，难想到
 * 而且想不到通过比较n与m的大小分支
 */
int part(int n, int m) {
    if (n == 1 || m == 1) //结束条件，如果无论n == 1还是m == 1，都只有一种情况
        return 1;
    else if (n < m) //如果n < m也做不到划分比n大的数
        return part(n, n);
    /*
     * 如果n == m，显然有part(n, n-1) + 1
     * +1是因为要不就是划分不超过n - 1的数
     * 要不就是划分等于n的数，而划分等于n的划分{n}只有一种划分
     */
    else if (n == m)
        return part(n, n-1) + 1;
    /*
     * 特殊情况讨论完就是一般情况
     * 划分数中要不就是至少有一个m
     * 要不就是没有m，那么就是m-1
     */
    else
        return part(n-m, m) + part(n, m-1);
}

int main() {
    int n;
    while (scanf("%d", &n))
        printf("%d\n", part(n, n));
    return 0;
}
```

#### 6.8.1.2 DP 求整数划分.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：02-DP求整数划分.cpp
 *    创建日期：2021年03月12日 ---- 11时16分
 *    题    目：hud1028
 *    算    法：动态规划
 *    描    述：与递归的代码类似
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 200;
int dp[maxn][maxn];
void part() {
    for (int n = 1; n <= maxn; ++n)
        for (int m = 1; m <= maxn; ++m) {
            if (n == 1 || m == 1)
                dp[n][m] = 1;
            else if (n < m)
                dp[n][m] = dp[n][n];
```

```
            else if (n == m)
                dp[n][m] = dp[n][m-1] + 1;
            else
                dp[n][m] = dp[n][m-1] + dp[n-m][m];
        }
}

int main() {
    int n;
    part();
    while (scanf("%d", &n))
        printf("%d\n", dp[n][n]);
    return 0;
}
```

#### 6.8.1.3 生成函数求整数划分.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：03-生成函数求整数划分.cpp
 *   创建日期：2021年03月13日 ---- 03时33分
 *   题    目：hdu1028
 *   算    法：生成函数
 *   描    述：如果说一定要至少用一张2分的邮票，那就在j循环那里加上
 *       if (k == 3 && j == 0) continue;
 *       以此类推
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 200;
int c1[maxn+1]; //记录展开后第X^n项的系数，即划分n的种类数有c1[n]种
int c2[maxn+1]; //记录临时计算结果
void part() {
    for (int i = 0; i <= maxn; ++i) //初始化，即第一部分(1 + x + x^2 + ...)的系数都是1
        c1[i] = 1, c2[i] = 0;
    /*
     * 从第2部分(1 + x^2 + x^4 + ...)开始展开
     * 也就是说我有面值1分的邮票，有面值2分的邮票，... 有面值200分的邮票，但是面值200分的邮票只有一张
     * 不过也不会题目也不会出到200，因为到123就溢出了
     * 如果题目面值小一点，即可认为邮票是可以重复贴的
     */
    for (int k = 2; k <= maxn; ++k) {
        for (int i = 0; i <= maxn; ++i)
            //k = 2时，i循环第1部分(1 + x + x^2 + ...)，j循环第2部分(1 + x^2 + x^4 + ...)
            for (int j = 0; j + i <= maxn; j += k)
                c2[i+j] += c1[i];
        for (int i = 0; i <= maxn; ++i) {
            c1[i] = c2[i];
            c2[i] = 0;
        }
    }
}

int main() {
    part();
    for (int i = 0; i < 122; ++i)
        printf("%d\n", c1[i]);
    return 0;
}
```

### 6.8.2 指数生成函数

#### 6.8.2.1 排列组合.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01-排列组合.cpp
 *   创建日期：2021年03月12日 ---- 22时16分
```

```
 *    题    目：hdu1521 排列组合
 *    算    法：指数型母函数
 *    描    述：有n种物品，并且知道每种物品的数量，求从中选出m件物品
 *        的排列数，例如有两种物品A, B，并且数量都是1,从中选两件物品
 *        则排列有"AB"和"BA"两种
 *
 *        输入：每组输入数据有两行，第一行是两个数n和m(1 <= m, n <= 10)
 *        表示物品数；第二行有n个数，分别表示这n件物品的数量
 *
 *        输出：对应每组数据输出排列数(任何运算不会超过2^31的范围)
 *
 ----------------------------------------------------------------*/
```

## 6.9 线性代数

### 6.9.1 高斯消元

#### 6.9.1.1 高斯消元解异或线性方程组.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称：高斯消元解异或线性方程组.cpp
 *    创建日期：2021年10月13日 星期三 00时06分27秒
 *    题    目：AcWing 0884 高斯消元解异或线性方程组
 *    算    法：高斯消元
 *    描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 105;
int n;
int a[maxn][maxn];

int guass() {
    int r, c;
    for (r = c = 0; c < n; ++ c) {
        int temp = r;
        for (int i = r; i < n; ++ i)
            if (a[i][c]) {
                temp = i;
                break;
            }
        if (!a[temp][c])
            continue;
        for (int i = c; i < n + 1; ++ i)
            swap(a[r][i], a[temp][i]);
        for (int i = r + 1; i < n; ++ i)
            if (a[i][c])
                for (int j = c; j < n + 1; ++ j)
                    a[i][j] ^= a[r][j];
        r ++ ;
    }
    if (r < n) {
        for (int i = r; i < n; ++ i)
            if (a[i][n])
                return 2;
        return 1;
    }
    for (int i = n - 1; i >= 0; -- i)
        for (int j = i + 1; j < n; ++ j)
            a[i][n] ^= a[i][j] & a[j][n];
    return 0;
}

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

```
        for (int j = 0; j < n + 1; ++ j)
            scanf("%d", &a[i][j]);
    int t = guass();
    if (t == 0)
        for (int i = 0; i < n; ++ i)
            printf("%d\n", a[i][n]);
    else if (t == 1)
        puts("Multiple sets of solutions");
    else
        puts("No solution");
    return 0;
}
```

#### 6.9.1.2 高斯消元解线性方程组.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：高斯消元解线性方程组.cpp
 *   创建日期：2021年10月12日 星期二 23时16分11秒
 *   题    目：AcWing 0883 高斯消元解线性方程组
 *   算    法：高斯消元
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const int maxn = 105;
int n;
double a[maxn][maxn];   // Augmented matrix 增广矩阵
const double eps = 1e-6;

inline int sgn(double x) {   // 判断x是否等于0
    if (fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}

/**
 * 枚举每一列
 * 1. 找到绝对值最大的一行
 * 2. 将该行换到最上面
 * 3. 将该行的第一个数变成1
 * 4. 将下面所有行的第 c 列清成0
 */

// 0: 唯一解，1: 无穷组解，2: 无解
int guess() {
    int r, c;   // 枚举第 r(ow) 行，第 c(olumn) 列
    for (c = 0, r = 0; c < n; ++ c) {
        int temp = r;
        // 找到当前列绝对值最大的一行
        for (int i = r; i < n; ++ i)
            if (fabs(a[i][c]) > fabs(a[temp][c]))
                temp = i;
        // 如果找到的当前列最大的一行是0，表示这一列全是0，就不用将该行换到上面了
        if (sgn(fabs(a[temp][c])) == 0)
            continue;

        for (int i = c; i < n + 1; ++ i)
            swap(a[r][i], a[temp][i]);
        for (int i = n; i >= c; -- i)
            a[r][i] /= a[r][c];
        for (int i = r + 1; i < n; ++ i)
            if (sgn(abs(a[i][c])) != 0)
                for (int j = n; j >= c; -- j)
                    a[i][j] -= a[r][j] * a[i][c];
        r ++ ;
```

```
    }
    if (r < n) {
        for (int i = r; i < n; ++ i)
            if (sgn(fabs(a[i][n])) != 0)
                return 2;
        return 1;
    }
    for (int i = n - 1; i >= 0; -- i)
        for (int j = i + 1; j < n; ++ j)
            a[i][n] -= a[i][j] * a[j][n];
    return 0;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < n + 1; ++ j)   // n + 1
            scanf("%lf", &a[i][j]);
    int t = guess();
    if (t == 0)
        for (int i = 0; i < n; ++ i)
            printf("%.2lf\n", a[i][n]);
    else if (t == 1)
        puts("Infinite group solutions");
    else
        puts("No solution");
    return 0;
}
```

## 6.10 组合数学

### 6.10.1 排列组合

#### 6.10.1.1 求组合数 I.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：求组合数I.cpp
 *   创建日期：2021年10月13日 星期三 14时25分48秒
 *   题    目：AcWing 0885 求组合数I
 *   算    法：组合数
 *   描    述：给定 n 组询问，每组询问给定两个整数 a，b，
 *   请你输出 C_a^b mod (1e9 +7) 的值
 *   0 < n < 100000
 *   0 < a, b < 2000
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 2000 + 5;
const int MOD = 1e9 + 7;
int c[maxn][maxn];

void init() {
    // 使用递推公式预处理组合数
    for (int a = 0; a < maxn; ++ a)
        for (int b = 0; b <= a; ++ b) {
            if (!b)
                c[a][b] = 1;
            else
                c[a][b] = (c[a - 1][b] + c[a - 1][b - 1]) % MOD;
        }
}

int main() {
    init();
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i) {
        int a, b;
        scanf("%d %d", &a, &b);
```

```
        printf("%d\n", c[a][b]);
    }
    return 0;
}
```

#### 6.10.1.2 求组合数 II.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：求组合数II.cpp
 *   创建日期：2021年10月13日 星期三 16时35分18秒
 *   题    目：AcWing 0886 求组合数II
 *   算    法：组合数
 *   描    述：给定 n 组询问，每组询问给定两个整数 a，b，
 *   请你输出 C_a^b mod (1e9 +7) 的值
 *   0 < n < 10000
 *   0 < a, b < 1e5
 *
 -----------------------------------------------------------------*/

// 同样是预处理，这次处理的是阶乘
// C_a^b = a! / ((a - b)! * b!)
#include <cstdio>
typedef long long ll;
typedef long long LL;
const int MOD = 1e9 + 7;
const int maxn = 1e5 + 5;
int fact[maxn], inv_fact[maxn];

int binpow(int base, int expo) {
    int res = 1;
    while (expo) {
        if (expo & 1)
            res = (1LL * res * base) % MOD;
        base = (1LL * base * base) % MOD;
        expo >>= 1;
    }
    return res;
}

int inv(int x) {
    return binpow(x, MOD - 2);
}

void init() {
    fact[0] = 1, inv_fact[0] = 1;
    for (int i = 1; i < maxn; ++ i) {
        fact[i] = (LL)fact[i - 1] * i % MOD;
        inv_fact[i] = (LL)inv_fact[i - 1] * inv(i) % MOD;
    }
}

int main() {
    init();
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i) {
        int a, b;
        scanf("%d %d", &a, &b);
        printf("%d\n", (LL)fact[a] * inv_fact[a - b] % MOD * inv_fact[b] % MOD);
    }
    return 0;
}
```

#### 6.10.1.3 求组合数 III.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：求组合数III.cpp
```

```
 *   创建日期: 2021年10月13日 星期三 16时52分04秒
 *   题     目: AcWing 0887 求组合数III
 *   算     法: 组合数
 *   描     述: 给定 n 组询问，每组询问给定三个整数 a, b, p
 *   其中 p 是质数，请你输出 C_a^b % p 的值。
 *   0 <= n <= 21
 *   1 <= b < a <= 1e18
 *   1 <= p <= 1e5
 *
 ----------------------------------------------------------------*/

/**
 * C_a^b % p = C_{a/p}^{b/p} * C_{a%p}^{b%p} % p;
 */

#include <cstdio>
typedef long long ll;
int p;
int MOD;

int binpow(int base, int expo) {
    int res = 1;
    while (expo) {
        if (expo & 1)
            res = (1LL * res * base) % MOD;
        base = (1LL * base * base) % MOD;
        expo >>= 1;
    }
    return res;
}

int inv(int x) {
    return binpow(x, MOD - 2);
}

int C(int a, int b) {
    int res = 1;
    for (int i = 1, j = a; i <= b; ++ i, -- j) {
        res = (ll)res * j % p;
        res = (ll)res * inv(i) % p;
    }
    return res;
}

int lucas(ll a, ll b) {
    if (a < p && b < p)
        return C(a, b);
    else
        return (ll)C(a % p, b % p) * lucas(a / p, b / p) % p;
}

int main() {
    int n; scanf("%d", &n);
    while (n -- ) {
        ll a, b;
        scanf("%lld %lld %d", &a, &b, &p);
        MOD = p;
        printf("%d\n", lucas(a, b));
    }
    return 0;
}
```

#### 6.10.1.4 求组合数 IV.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称: 求组合数IV.cpp
 *   创建日期: 2021年10月13日 星期三 17时44分06秒
 *   题     目: AcWing 0888 求组合数IV
```

```
 *   算     法：组合数
 *   描     述：输入 a, b，求 C_a^b 的值
 *   注意结果可能很大，需要使用高精度计算。
 *
 *   1. 得到质数表
 *   2. 将结果分解质因子
 *
 *   a! 中素因子 p 的个数是 cnt = a / p + a / p^2 + a / p^3 + ...
 *
 *   3. 高精度乘法
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
using namespace std;
const int maxn = 5000 + 5;
int primes[maxn], cnt;
int sum[maxn];
bool sifter[maxn];
#define NEXTLINE puts("");

void get_primes(int n) {
    for (int i = 2; i <= n; ++ i) {
        if (sifter[i] == false)
            primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; ++ j) {
            sifter[primes[j] * i] = true;
            if (i % primes[j] == 0)
                break;
        }
    }
}

// n! 中包含的素因子 p 的个数
int get(int n, int p) {
    int res = 0;
    while (n) {
        res += n / p;
        n /= p;
    }
    return res;
}

vector<int> mul(vector<int> a, int b) {
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); ++ i) {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while (t) {
        c.push_back(t % 10);
        t /= 10;
    }
    return c;
}

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    get_primes(a);
    for (int i = 0; i < cnt; ++ i) {
        int p = primes[i];
        sum[i] = get(a, p) - get(b, p) - get(a - b, p);
    }
    vector<int> res;
    res.push_back(1);

```

```
    for (int i = 0; i < cnt; ++ i)
        for (int j = 0; j < sum[i]; ++ j)
            res = mul(res, primes[i]);

    for (int i = res.size() - 1; i >= 0; -- i)
        printf("%d", res[i]);
    NEXTLINE;
    return 0;
}
```

### 6.10.2 卡特兰数

#### 6.10.2.1 满足条件的 01 序列.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：满足条件的01序列.cpp
 *   创建日期：2021年10月13日 星期三 22时19分09秒
 *   题    目：AcWing 0889 满足条件的01序列
 *   算    法：卡特兰数
 *   描    述：给定 n 个 0 和 n 个 1，它们将按照某种顺序排成长度为
 *   2n 的序列，求它们能排列成的所有序列中，
 *   能够满足任意前缀序列中 0 的个数都不少于 1 的个数的序列有多少个。
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int MOD = 1e9 + 7;

int binpow(int base, int expo) {
    int res = 1;
    while (expo) {
        if (expo & 1)
            res = (1LL * res * base) % MOD;
        base = (1LL * base * base) % MOD;
        expo >>= 1;
    }
    return res;
}

int inv(int x) {
    return binpow(x, MOD - 2);
}

int main() {
    int n; scanf("%d", &n);
    int a = 2 * n, b = n;
    int res = 1;
    for (int i = a; i > a - b; -- i)
        res = 1LL * res * i % MOD;
    for (int i = 1; i <= b; ++ i)
        res = 1LL * res * inv(i) % MOD;
    res = 1LL * res * inv(n + 1) % MOD;
    printf("%d\n", res);
    return 0;
}
```

### 6.10.3 斯特林数

#### 6.10.3.1 Stirling.cpp

```
#ifndef _FEISTDLIB_POLY_
#define _FEISTDLIB_POLY_

/*
 * This file is part of the fstdlib project.
 * Version: Build v0.0.2
 * You can check for details at https://github.com/FNatsuka/fstdlib
 */
```

```

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

namespace fstdlib{

    typedef long long ll;
    int mod = 998244353, grt = 3;

    class poly{
        private:
            std::vector<int> data;
            void out(void){
                for(int i = 0; i < (int)data.size(); ++i) printf("%d ", data[i]);
                puts("");
            }
        public:
            poly(std::size_t len = std::size_t(0)){data = std::vector<int>(len); }
            poly(const std::vector<int> &b){data = b; }
            poly(const poly &b){data = b.data; }
            void resize(std::size_t len, int val = 0){data.resize(len, val); }
            std::size_t size(void)const{return data.size(); }
            void clear(void){data.clear(); }
#if __cplusplus >= 201103L
            void shrink_to_fit(void){data.shrink_to_fit(); }
#endif
            int &operator[](std::size_t b){return data[b]; }
            const int &operator[](std::size_t b)const{return data[b]; }
            poly operator*(const poly& h)const;
            poly operator*=(const poly &h);
            poly operator*(const int &h)const;
            poly operator*=(const int &h);
            poly operator+(const poly &h)const;
            poly operator+=(const poly &h);
            poly operator-(const poly &h)const;
            poly operator-=(const poly &h);
            poly operator<<(const std::size_t &b)const;
            poly operator<<=(const std::size_t &b);
            poly operator>>(const std::size_t &b)const;
            poly operator>>=(const std::size_t &b);
            poly operator/(const int &h)const;
            poly operator/=(const int &h);
            poly operator==(const poly &h)const;
            poly operator!=(const poly &h)const;
            poly operator+(const int &h)const;
            poly operator+=(const int &h);
            poly inv(void)const;
            poly inv(const int &h)const;
            friend poly sqrt(const poly &h);
            friend poly log(const poly &h);
            friend poly exp(const poly &h);
    };

    int qpow(int a, int b, int p = mod){
        int res = 1;
        while(b){if(b & 1) res = (ll)res * a % p; a = (ll)a * a % p, b >>= 1; }
        return res;
    }

    std::vector<int> rev;
    void dft_for_module(std::vector<int> &f, int n, int b){
        static std::vector<int> w;
        w.resize(n);
        for(int i = 0; i < n; ++i) if(i < rev[i]) std::swap(f[i], f[rev[i]]);
        for(int i = 2; i <= n; i <<= 1){
            w[0] = 1, w[1] = qpow(grt, (mod - 1) / i); if(b == -1) w[1] = qpow(w[1], mod - 2);
            for(int j = 2; j < i / 2; ++j) w[j] = (ll)w[j - 1] * w[1] % mod;
            for(int j = 0; j < n; j += i)
```

```
                for(int k = 0; k < i / 2; ++k){
                    int p = f[j + k], q = (ll)f[j + k + i / 2] * w[k] % mod;
                    f[j + k] = (p + q) % mod, f[j + k + i / 2] = (p - q + mod) % mod;
                }
            }
        }

        poly poly::operator*(const poly &h)const{
            int N = 1; while(N < (int)(size() + h.size() - 1)) N <<= 1;
            std::vector<int> f(this->data), g(h.data); f.resize(N), g.resize(N);
            rev.resize(N);
            for(int i = 0; i < N; ++i) rev[i] = (rev[i >> 1] >> 1) | (i & 1 ? N >> 1 : 0);
            dft_for_module(f, N, 1), dft_for_module(g, N, 1);
            for(int i = 0; i < N; ++i) f[i] = (ll)f[i] * g[i] % mod;
            dft_for_module(f, N, -1), f.resize(size() + h.size() - 1);
            for(int i = 0, inv = qpow(N, mod - 2); i < (int)f.size(); ++i) f[i] = (ll)f[i] * inv % mod;
            return f;
        }

        poly poly::operator*=(const poly &h){
            return *this = *this * h;
        }

        poly poly::operator*(const int &h)const{
            std::vector<int> f(this->data);
            for(int i = 0; i < (int)f.size(); ++i) f[i] = (ll)f[i] * h % mod;
            return f;
        }

        poly poly::operator*=(const int &h){
            for(int i = 0; i < (int)size(); ++i) data[i] = (ll)data[i] * h % mod;
            return *this;
        }

        poly poly::operator+(const poly &h)const{
            std::vector<int> f(this->data);
            if(f.size() < h.size()) f.resize(h.size());
            for(int i = 0; i < (int)h.size(); ++i) f[i] = (f[i] + h[i]) % mod;
            return f;
        }

        poly poly::operator+=(const poly &h){
            std::vector<int> &f = this->data;
            if(f.size() < h.size()) f.resize(h.size());
            for(int i = 0; i < (int)h.size(); ++i) f[i] = (f[i] + h[i]) % mod;
            return f;
        }

        poly poly::operator-(const poly &h)const{
            std::vector<int> f(this->data);
            if(f.size() < h.size()) f.resize(h.size());
            for(int i = 0; i < (int)h.size(); ++i) f[i] = (f[i] - h[i] + mod) % mod;
            return f;
        }

        poly poly::operator-=(const poly &h){
            std::vector<int> &f = this->data;
            if(f.size() < h.size()) f.resize(h.size());
            for(int i = 0; i < (int)h.size(); ++i) f[i] = (f[i] - h[i] + mod) % mod;
            return f;
        }

        poly poly::operator<<(const std::size_t &b)const{
            std::vector<int> f(size() + b);
            for(int i = 0; i < (int)size(); ++i) f[i + b] = data[i];
            return f;
        }

        poly poly::operator<<=(const std::size_t &b){
            return *this = (*this) << b;
```

```
149     }
150
151     poly poly::operator>>(const std::size_t &b)const{
152         std::vector<int> f(size() - b);
153         for(int i = 0; i < (int)f.size(); ++i) f[i] = data[i + b];
154         return f;
155     }
156
157     poly poly::operator>>=(const std::size_t &b){
158         return *this = (*this) >> b;
159     }
160
161     poly poly::operator/(const int &h)const{
162         std::vector<int> f(this->data); int inv = qpow(h, mod - 2);
163         for(int i = 0; i < (int)f.size(); ++i) f[i] = (ll)f[i] * inv % mod;
164         return f;
165     }
166
167     poly poly::operator/=(const int &h){
168         int inv = qpow(h, mod - 2);
169         for(int i = 0; i < (int)data.size(); ++i) data[i] = (ll)data[i] * inv % mod;
170         return *this;
171     }
172
173     poly poly::inv(void)const{
174         int N = 1; while(N < (int)(size() + size() - 1)) N <<= 1;
175         std::vector<int> f(N), g(N), d(this->data);
176         d.resize(N), f[0] = qpow(d[0], mod - 2);
177         for(int w = 2; w < N; w <<= 1){
178             for(int i = 0; i < w; ++i) g[i] = d[i];
179             rev.resize(w << 1);
180             for(int i = 0; i < w * 2; ++i) rev[i] = (rev[i >> 1] >> 1) | (i & 1 ? w : 0);
181             dft_for_module(f, w << 1, 1), dft_for_module(g, w << 1, 1);
182             for(int i = 0; i < w * 2; ++i) f[i] = (ll)f[i] * (2 + mod - (ll)f[i] * g[i] % mod) % mod;
183             dft_for_module(f, w << 1, -1);
184             for(int i = 0, inv = qpow(w << 1, mod - 2); i < w; ++i) f[i] = (ll)f[i] * inv % mod;
185             for(int i = w; i < w * 2; ++i) f[i] = 0;
186         }
187         f.resize(size());
188         return f;
189     }
190
191     poly poly::operator==(const poly &h)const{
192         if(size() != h.size()) return 0;
193         for(int i = 0; i < (int)size(); ++i) if(data[i] != h[i]) return 0;
194         return 1;
195     }
196
197     poly poly::operator!=(const poly &h)const{
198         if(size() != h.size()) return 1;
199         for(int i = 0; i < (int)size(); ++i) if(data[i] != h[i]) return 1;
200         return 0;
201     }
202
203     poly poly::operator+(const int &h)const{
204         poly f(this->data);
205         f[0] = (f[0] + h) % mod;
206         return f;
207     }
208
209     poly poly::operator+=(const int &h){
210         return *this = (*this) + h;
211     }
212
213     poly poly::inv(const int &h)const{
214         poly f(*this);
215         f.resize(h);
216         return f.inv();
217     }
218
```

```
219     int modsqrt(int h, int p = mod){
220         return 1;
221     }
222
223     poly sqrt(const poly &h){
224         int N = 1; while(N < (int)(h.size() + h.size() - 1)) N <<= 1;
225         poly f(N), g(N), d(h); d.resize(N), f[0] = modsqrt(d[0]);
226         for(int w = 2; w < N; w <<= 1){
227             g.resize(w);
228             for(int i = 0; i < w; ++i) g[i] = d[i];
229             f = (f + f.inv(w) * g) / 2;
230             f.resize(w);
231         }
232         f.resize(h.size());
233         return f;
234     }
235
236     poly log(const poly &h){
237         poly f(h);
238         for(int i = 1; i < (int)f.size(); ++i) f[i - 1] = (ll)f[i] * i % mod;
239         f[f.size() - 1] = 0, f = f * h.inv(), f.resize(h.size());
240         for(int i = (int)f.size() - 1; i > 0; --i) f[i] = (ll)f[i - 1] * qpow(i, mod - 2) % mod;
241         f[0] = 0;
242         return f;
243     }
244
245     poly exp(const poly &h){
246         int N = 1; while(N < (int)(h.size() + h.size() - 1)) N <<= 1;
247         poly f(N), g(N), d(h);
248         f[0] = 1, d.resize(N);
249         for(int w = 2; w < N; w <<= 1){
250             f.resize(w), g.resize(w);
251             for(int i = 0; i < w; ++i) g[i] = d[i];
252             f = f * (g + 1 - log(f));
253             f.resize(w);
254         }
255         f.resize(h.size());
256         return f;
257     }
258
259     struct comp{
260         long double x, y;
261         comp(long double _x = 0, long double _y = 0) : x(_x), y(_y) {}
262         comp operator*(const comp &b)const{return comp(x * b.x - y * b.y, x * b.y + y * b.x); }
263         comp operator+(const comp &b)const{return comp(x + b.x, y + b.y); }
264         comp operator-(const comp &b)const{return comp(x - b.x, y - b.y); }
265         comp conj(void){return comp(x, -y); }
266     };
267
268     const int EPS = 1e-9;
269
270     template <typename FLOAT_T>
271         FLOAT_T fabs(const FLOAT_T &x){
272             return x > 0 ? x : -x;
273         }
274
275     template <typename FLOAT_T>
276         FLOAT_T sin(const FLOAT_T &x, const long double &EPS = fstdlib::EPS){
277             FLOAT_T res = 0, delt = x;
278             int d = 0;
279             while(fabs(delt) > EPS){
280                 res += delt, ++d;
281                 delt *= - x * x / ((2 * d) * (2 * d + 1));
282             }
283             return res;
284         }
285
286     template <typename FLOAT_T>
287         FLOAT_T cos(const FLOAT_T &x, const long double &EPS = fstdlib::EPS){
288             FLOAT_T res = 0, delt = 1;
```

```
            int d = 0;
            while(fabs(delt) > EPS){
                res += delt, ++d;
                delt *= - x * x / ((2 * d) * (2 * d - 1));
            }
            return res;
        }

    const long double PI = std::acos((long double)(-1));

    void dft_for_complex(std::vector<comp> &f, int n, int b){
        static std::vector<comp> w;
        w.resize(n);
        for(int i = 0; i < n; ++i) if(i < rev[i]) std::swap(f[i], f[rev[i]]);
        for(int i = 2; i <= n; i <<= 1){
            w[0] = comp(1, 0), w[1] = comp(cos(2 * PI / i), b * sin(2 * PI / i));
            for(int j = 2; j < i / 2; ++j) w[j] = w[j - 1] * w[1];
            for(int j = 0; j < n; j += i)
                for(int k = 0; k < i / 2; ++k){
                    comp p = f[j + k], q = f[j + k + i / 2] * w[k];
                    f[j + k] = p + q, f[j + k + i / 2] = p - q;
                }
        }
    }

    class arbitrary_module_poly{
        private:
            std::vector<int> data;
            int construct_element(int D, ll x, ll y, ll z)const{
                x %= mod, y %= mod, z %= mod;
                return ((ll)D * D * x % mod + (ll)D * y % mod + z) % mod;
            }
        public:
            int mod;
            arbitrary_module_poly(std::size_t len = std::size_t(0), int module_value = 1e9 + 7){mod =
    module_value; data = std::vector<int>(len); }
            arbitrary_module_poly(const std::vector<int> &b, int module_value = 1e9 + 7){mod = module_value;
    data = b; }
            arbitrary_module_poly(const arbitrary_module_poly &b){mod = b.mod; data = b.data; }
            void resize(std::size_t len, const int &val = 0){data.resize(len, val); }
            std::size_t size(void)const{return data.size(); }
            void clear(void){data.clear(); }
#if __cplusplus >= 201103L
            void shrink_to_fit(void){data.shrink_to_fit(); }
#endif
            int &operator[](std::size_t b){return data[b]; }
            const int &operator[](std::size_t b)const{return data[b]; }
            arbitrary_module_poly operator*(const arbitrary_module_poly& h)const;
            arbitrary_module_poly operator*=(const arbitrary_module_poly &h);
            arbitrary_module_poly operator*(const int &h)const;
            arbitrary_module_poly operator*=(const int &h);
            arbitrary_module_poly operator+(const arbitrary_module_poly &h)const;
            arbitrary_module_poly operator+=(const arbitrary_module_poly &h);
            arbitrary_module_poly operator-(const arbitrary_module_poly &h)const;
            arbitrary_module_poly operator-=(const arbitrary_module_poly &h);
            arbitrary_module_poly operator<<(const std::size_t &b)const;
            arbitrary_module_poly operator<<=(const std::size_t &b);
            arbitrary_module_poly operator>>(const std::size_t &b)const;
            arbitrary_module_poly operator>>=(const std::size_t &b);
            arbitrary_module_poly operator/(const int &h)const;
            arbitrary_module_poly operator/=(const int &h);
            arbitrary_module_poly operator==(const arbitrary_module_poly &h)const;
            arbitrary_module_poly operator!=(const arbitrary_module_poly &h)const;
            arbitrary_module_poly inv(void)const;
            arbitrary_module_poly inv(const int &h)const;
            friend arbitrary_module_poly sqrt(const arbitrary_module_poly &h);
            friend arbitrary_module_poly log(const arbitrary_module_poly &h);
    };

    arbitrary_module_poly arbitrary_module_poly::operator*(const arbitrary_module_poly& h)const{
```

```
        int N = 1; while(N < (int)(size() + h.size() - 1)) N <<= 1;
        std::vector<comp> f(N), g(N), p(N), q(N);
        const int D = std::sqrt(mod);
        for(int i = 0; i < (int)size(); ++i) f[i].x = data[i] / D, f[i].y = data[i] % D;
        for(int i = 0; i < (int)h.size(); ++i) g[i].x = h[i] / D, g[i].y = h[i] % D;
        rev.resize(N);
        for(int i = 0; i < N; ++i) rev[i] = (rev[i >> 1] >> 1) | (i & 1 ? N >> 1 : 0);
        dft_for_complex(f, N, 1), dft_for_complex(g, N, 1);
        for(int i = 0; i < N; ++i){
            p[i] = (f[i] + f[(N - i) % N].conj()) * comp(0.50, 0) * g[i];
            q[i] = (f[i] - f[(N - i) % N].conj()) * comp(0, -0.5) * g[i];
        }
        dft_for_complex(p, N, -1), dft_for_complex(q, N, -1);
        std::vector<int> r(size() + h.size() - 1);
        for(int i = 0; i < (int)r.size(); ++i)
            r[i] = construct_element(D, p[i].x / N + 0.5, (p[i].y + q[i].x) / N + 0.5, q[i].y / N + 0.5);
        return arbitrary_module_poly(r, mod);
    }

    arbitrary_module_poly arbitrary_module_poly::operator*=(const arbitrary_module_poly &h){
        return *this = *this * h;
    }

    arbitrary_module_poly arbitrary_module_poly::operator*(const int &h)const{
        std::vector<int> f(this->data);
        for(int i = 0; i < (int)f.size(); ++i) f[i] = (ll)f[i] * h % mod;
        return arbitrary_module_poly(f, mod);
    }

    arbitrary_module_poly arbitrary_module_poly::operator*=(const int &h){
        for(int i = 0; i < (int)size(); ++i) data[i] = (ll)data[i] * h % mod;
        return *this;
    }

    arbitrary_module_poly arbitrary_module_poly::operator+(const arbitrary_module_poly &h)const{
        std::vector<int> f(this->data);
        if(f.size() < h.size()) f.resize(h.size());
        for(int i = 0; i < (int)h.size(); ++i) f[i] = (f[i] + h[i]) % mod;
        return arbitrary_module_poly(f, mod);
    }

    arbitrary_module_poly arbitrary_module_poly::operator+=(const arbitrary_module_poly &h){
        if(size() < h.size()) resize(h.size());
        for(int i = 0; i < (int)h.size(); ++i) data[i] = (data[i] + h[i]) % mod;
        return *this;
    }

    arbitrary_module_poly arbitrary_module_poly::operator-(const arbitrary_module_poly &h)const{
        std::vector<int> f(this->data);
        if(f.size() < h.size()) f.resize(h.size());
        for(int i = 0; i < (int)h.size(); ++i) f[i] = (f[i] + mod - h[i]) % mod;
        return arbitrary_module_poly(f, mod);
    }

    arbitrary_module_poly arbitrary_module_poly::operator-=(const arbitrary_module_poly &h){
        if(size() < h.size()) resize(h.size());
        for(int i = 0; i < (int)h.size(); ++i) data[i] = (data[i] + mod - h[i]) % mod;
        return *this;
    }

    arbitrary_module_poly arbitrary_module_poly::operator<<(const std::size_t &b)const{
        std::vector<int> f(size() + b);
        for(int i = 0; i < (int)size(); ++i) f[i + b] = data[i];
        return arbitrary_module_poly(f, mod);
    }

    arbitrary_module_poly arbitrary_module_poly::operator<<=(const std::size_t &b){
        return *this = (*this) << b;
    }
```

```
427     arbitrary_module_poly arbitrary_module_poly::operator>>(const std::size_t &b)const{
428         std::vector<int> f(size() - b);
429         for(int i = 0; i < (int)f.size(); ++i) f[i] = data[i + b];
430         return arbitrary_module_poly(f, mod);
431     }
432
433     arbitrary_module_poly arbitrary_module_poly::operator>>=(const std::size_t &b){
434         return *this = (*this) >> b;
435     }
436
437     arbitrary_module_poly arbitrary_module_poly::inv(void)const{
438         int N = 1; while(N < (int)(size() + size() - 1)) N <<= 1;
439         arbitrary_module_poly f(1, mod), g(N, mod), h(*this), f2(1, mod);
440         f[0] = qpow(data[0], mod - 2, mod), h.resize(N), f2[0] = 2;
441         for(int w = 2; w < N; w <<= 1){
442             g.resize(w);
443             for(int i = 0; i < w; ++i) g[i] = h[i];
444             f = f * (f * g - f2) * (mod - 1);
445             f.resize(w);
446         }
447         f.resize(size());
448         return f;
449     }
450
451     arbitrary_module_poly arbitrary_module_poly::inv(const int &h)const{
452         arbitrary_module_poly f(*this);
453         f.resize(h);
454         return f.inv();
455     }
456
457     arbitrary_module_poly arbitrary_module_poly::operator/(const int &h)const{
458         int inv = qpow(h, mod - 2, mod);
459         std::vector<int> f(this->data);
460         for(int i = 0; i < (int)f.size(); ++i) f[i] = (ll)f[i] * inv % mod;
461         return arbitrary_module_poly(f, mod);
462     }
463
464     arbitrary_module_poly arbitrary_module_poly::operator/=(const int &h){
465         int inv = qpow(h, mod - 2, mod);
466         for(int i = 0; i < (int)size(); ++i) data[i] = (ll)data[i] * inv % mod;
467         return *this;
468     }
469
470     arbitrary_module_poly arbitrary_module_poly::operator==(const arbitrary_module_poly &h)const{
471         if(size() != h.size() || mod != h.mod) return 0;
472         for(int i = 0; i < (int)size(); ++i) if(data[i] != h[i]) return 0;
473         return 1;
474     }
475
476     arbitrary_module_poly arbitrary_module_poly::operator!=(const arbitrary_module_poly &h)const{
477         if(size() != h.size() || mod != h.mod) return 1;
478         for(int i = 0; i < (int)size(); ++i) if(data[i] != h[i]) return 1;
479         return 0;
480     }
481
482     arbitrary_module_poly sqrt(const arbitrary_module_poly &h){
483         int N = 1; while(N < (int)(h.size() + h.size() - 1)) N <<= 1;
484         arbitrary_module_poly f(1, mod), g(N, mod), d(h);
485         f[0] = modsqrt(h[0], mod), d.resize(N);
486         for(int w = 2; w < N; w <<= 1){
487             g.resize(w);
488             for(int i = 0; i < w; ++i) g[i] = d[i];
489             f = (f + f.inv(w) * g) / 2;
490             f.resize(w);
491         }
492         f.resize(h.size());
493         return f;
494     }
495
496     arbitrary_module_poly log(const arbitrary_module_poly &h){
```

```
        arbitrary_module_poly f(h);
        for(int i = 1; i < (int)f.size(); ++i) f[i - 1] = (ll)f[i] * i % f.mod;
        f[f.size() - 1] = 0, f = f * h.inv(), f.resize(h.size());
        for(int i = (int)f.size() - 1; i > 0; --i) f[i] = (ll)f[i - 1] * qpow(i, f.mod - 2, f.mod) % f.mod;
        f[0] = 0;
        return f;
    }
    typedef arbitrary_module_poly m_poly;
}

#endif
using namespace fstdlib;
int n;
const int maxn = 1e6;
int fact[maxn];
int ifact[maxn];
typedef long long ll;
int main() {
    scanf("%d", &n);
    fact[0] = 1;
    for (int i = 1; i <= n; ++i) fact[i] = (ll)fact[i - 1] * i % mod;
    exgcd(fact[n], mod, ifact[n], ifact[0]),
        ifact[n] = (ifact[n] % mod + mod) % mod;
    for (int i = n - 1; i >= 0; --i) ifact[i] = (ll)ifact[i + 1] * (i + 1) % mod;
    poly f(n + 1), g(n + 1);
    for (int i = 0; i <= n; ++i)
        g[i] = (i & 1 ? mod - 1ll : 1ll) * ifact[i] % mod,
            f[i] = (ll)qpow(i, n) * ifact[i] % mod;
    f *= g, f.resize(n + 1);
    for (int i = 0; i <= n; ++i) printf("%d ", f[i]);
    return 0;
}
```

### 6.10.4 康托展开

#### 6.10.4.1 八数码.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：八数码.cpp
 *   创建日期：2021年11月25日 星期四 11时27分52秒
 *   题    目：AcWing 0179 八数码
 *   算    法：BFS + cantor 也可以使用astar，代码在 搜索 -> A*
 *   描    述：将输入的一个八数码通过移动 x 来转换为初始八数码
 *
 *   1 2 3            1 2 3
 *   x 4 6     ->     4 5 6
 *   7 5 8            7 8 x
 *
 *   u : 向上
 *   d : 向下
 *   l : 向左
 *   r : 向右
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <unordered_map>
#include <queue>
#include <cstring>
#include <algorithm>
using namespace std;
int grid[10], over[10], n = 9;
int fact[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};   // 阶乘
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};
// const char conv[] = "lurd";
const char conv[] = "rdlu";
unordered_map<int, int> used;
// 从 pre.second.first 到 pre.first 的变换是 pre.second.second
```

```
unordered_map<int, pair<int, char>> pre;
#define NEXTLINE puts("");
#define bug puts("<-->");

/*
 * n 个数，这个 str 不是字符串类型，而是一个数组
 * 里面是 0 - 8 这 9 个数
 * 康托展开公式：res = a[n] * (n - 1)! + a[n - 1] * (n - 2)! + ... + a[1] * 0!;
 * 这里的 a[i] 表示的是第 i 个位置上的数在 i 后面的数中排第几
 */
int Contor(int* sequ, int n) {
    int res = 0;
    for (int i = 0; i < n; ++ i) {
        int cnt = 0;
        for (int j = i + 1; j < n; ++ j)
            if (sequ[i] > sequ[j])
                cnt ++ ;
        res += cnt * fact[n - i - 1];
    }
    return res + 1;
}

// 只能返回 [1, n] 的排列，也容易转换为指定的集合
int* revContor(int ranking, int n) {
    ranking = ranking - 1;   // 有 ranking - 1 个排列比目标序列要小
    bool used[10];
    static int permutation[10];
    memset(used, false, sizeof used);
    memset(permutation, -1, sizeof permutation);

    for (int i = 0; i < n; ++ i) {
        int r = ranking / fact[n - i - 1];
        ranking = ranking % fact[n - i - 1];

        for (int j = 1; j <= n; ++ j)
            if (!used[j] && !(r -- )) {
                used[j] = true;
                permutation[i] = j;
                break;
            }
    }
    // sequ 排个序
    // 如果 permutaion[i] = 1，需要指定集合的话，就输出 sequ[permutation[i] - 1];
    return permutation;
}

bool BFS(int rankbegin, int rankover) {
    queue<int> q;
    q.push(rankbegin);   // 初始的 Contor 值存入队列中
    used[rankbegin] ++ ;
    while (q.size()) {
        int t = q.front(); q.pop();   // 队顶的 Contor 值
        if (t == rankover)
            return true;
        int *g = revContor(t, n);
        int x, y, idx;
        for (int i = 0; i < 9; ++ i)
            if (g[i] == 9) {
                idx = i;
                // 从 0 开始
                x = i / 3;
                y = i % 3;
                // printf("-- %d %d %d\n", i, x, y);
            }
        /*
         * for (int i = 0; i < 9; ++ i) {
         *     printf("%d ", g[i]);
         *     if (i == 2 || i == 5 || i == 8)
         *         NEXTLINE;
         * }
```

```
         */
        for (int i = 0; i < 4; ++ i) {
            int nx = x + dx[i],
                ny = y + dy[i];
            if (nx < 0 || nx >= 3 || ny < 0 || ny >= 3)
                continue;
            int newidx = nx * 3 + ny;
            swap(g[idx], g[newidx]);
            int ranking = Contor(g, n);
            swap(g[idx], g[newidx]);
            if (used.count(ranking))
                continue;
            // printf("-- %d %d %d\n", newidx, nx, ny);
            q.push(ranking);
            pre[ranking] = {t, char(conv[i])};
            used[ranking] ++ ;
        }
        // NEXTLINE;
    }
    return false;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif
    for (int i = 0; i < 9; ++ i) {
        char ch[2]; scanf("%s", ch);
        grid[i] = *ch == 'x' ? 9 : (int)(*ch - '0');
    }
    for (int i = 0; i < 9; ++ i)
        over[i] = i + 1;
    int rankover = Contor(over, n);   // 结束时的 Contor
    int rankbegin = Contor(grid, n);   // 初始的 Contor
    if (BFS(rankover, rankbegin)) {
        while (rankbegin != rankover) {
            char ch = pre[rankbegin].second;
            printf("%c", ch);
            rankbegin = pre[rankbegin].first;
        }
        NEXTLINE;
    }
    else
        puts("unsolvable");
    return 0;
}
```

#### 6.10.4.2 康托展开-vector.cpp

```
#include <cstdio>
#include <vector>
using namespace std;
#define NEXTLINE puts("");
const int fact[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};

int Contor(const vector<int>& permutation) {
    int num = 0;
    int len = permutation.size();
    for (int i = 0; i < len; ++ i) {
        int cnt = 0;
        for (int j = i + 1; j < len; ++ j)
            if (permutation[i] > permutation[j])
                cnt ++ ;
        num += cnt * fact[len - i - 1];
    }
    return num + 1;
}

```

```
vector<int> revContor(int ranking, int n) {
    ranking = ranking - 1;   // 有 ranking - 1 个排列比目标序列要小
    vector<bool> used(n + 1, false);
    vector<int> permutation(n, -1);

    for (int i = 0; i < n; ++ i) {
        int r = ranking / (fact[n - i - 1]);
        ranking = ranking % (fact[n - i - 1]);

        for (int j = 1; j <= n; ++ j)
            if (!used[j] && !(r -- )) {
                used[j] = true;
                permutation[i] = j;
                break;
            }
    }
    // sequ 排个序
    // 如果 permutaion[i] = 1，需要指定集合的话，就输出 sequ[permutation[i] - 1];
    return permutation;
}

int main() {
    vector<int> vec;
    int n; scanf("%d", &n);
    for (int i = 0; i < n; ++ i) {
        int _; scanf("%d", &_);
        vec.push_back(_);
    }
    int ranking = Contor(vec);
    printf("%d\n", ranking);

    vector<int> res = revContor(ranking, n);
    for (auto i : res)
        printf("%d ", i);
    NEXTLINE;
    return 0;
}
```

#### 6.10.4.3 康托展开数组.cpp

```
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
const int maxn = 10;
#define NEXTLINE puts("");
int fact[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};   // 阶乘

/*
 * 返回长度为 n 的这个序列在这 n 个数的全排列中是第几位
 * 1, 2, 3, 4, 5, ..., n 是第一位
 * n, n - 1, ..., 3, 2, 1 是最后一位，也是第 A_n^n 位
 * 康托展开公式：res = a[n] * (n - 1)! + a[n - 1] * (n - 2)! + ... + a[1] * 0!;
 * 这里的 a[i] 表示的是第 i 个位置上的数在 i 后面的数中排第几
 */
int Contor(int* sequ, int n) {
    int res = 0;
    for (int i = 0; i < n; ++ i) {
        int cnt = 0;
        for (int j = i + 1; j < n; ++ j)
            if (sequ[i] > sequ[j])
                cnt ++ ;
        res += cnt * fact[n - i - 1];
    }
    return res + 1;
}

// 只能返回 [1, n] 的排列，也容易转换为指定的集合
int* revContor(int ranking, int n) {
```

```
    ranking = ranking - 1;   // 有 ranking - 1 个排列比目标序列要小
    bool used[maxn];
    static int permutation[maxn];
    memset(used, false, sizeof used);
    memset(permutation, -1, sizeof permutation);

    for (int i = 0; i < n; ++ i) {
        int r = ranking / fact[n - i - 1];
        ranking = ranking % fact[n - i - 1];

        for (int j = 1; j <= n; ++ j)
            if (!used[j] && !(r -- )) {
                used[j] = true;
                permutation[i] = j;
                break;
            }
    }
    // sequ 排个序
    // 如果 permutaion[i] = 1，需要指定集合的话，就输出 sequ[permutation[i] - 1];
    return permutation;
}

int main() {
    int a[maxn], n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++ i)
        scanf("%d", &a[i]);
    int ranking = Contor(a, n);
    printf("%d\n", ranking);
    int *rank_sequ;
    rank_sequ = revContor(ranking, n);
    for (int i = 0; i < n; ++ i)
        printf("%d ", rank_sequ[i]);
    NEXTLINE;
    return 0;
}
```

### 6.10.5 容斥原理

#### 6.10.5.1 能被整除的数.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：能被整除的数.cpp
 *   创建日期：2021年10月14日 星期四 17时55分02秒
 *   题    目：AcWing 0890 能被整除的数
 *   算    法：容斥原理
 *   描    述：给定一个整数 n 和 m 个不同的质数 p1, p2, ..., pm
 *   求[1, n]中能被 p1, p2, ..., pm 中至少一个数整除的整数有多少个
 *   1 <= m <= 16
 *   1 <= n, pi <= 1e9
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 20;
int p[maxn];
typedef long long ll;

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++ i)
        scanf("%d", &p[i]);
    int res = 0;
    // inclusion-exclusion principle
    // 使用二进制的方式枚举每种选法
    for (int i = 1; i < 1 << m; ++ i) {
        // t: 所有质数的乘积
        // cnt: 这个二进制数中位为1的个数
```

```
        int t = 1, cnt = 0;
        for (int j = 0; j < m; ++ j)
            if (i >> j & 1) {
                cnt ++ ;
                if ((ll)t * p[j] > n) {
                    t = -1;
                    break;
                }
                t *= p[j];
            }
        if (t != -1) {
            if (cnt % 2)
                res += n / t;
            else
                res -= n / t;
        }
    }
    printf("%d\n", res);
    return 0;
}
```

## 6.11 斐波那契数列

### 6.11.1 Fibonacci.c

```c
#include <stdio.h>

int Fibonacci(int n) {
    if (n == 1 || n == 0)
        return 1;
    else
        return Fibonacci(n - 1) + Fibonacci(n - 2);
}

int main() {
    //n的值再高可能递归太深，出不来
    int n = 41;
    printf("%d\n", Fibonacci(n));
    return 0;
}
```

### 6.11.2 Fibonacci.cpp

```cpp
#include <cstdio>
#include <cmath>
#include <vector>
using namespace std;

template<typename T>
T Fibo0(T n) {
    if (n <= 1) return 1;
    else return Fibo(n-1) + Fibo(n-2);
}

template<typename T>
T Fibo1(T n) {
    if (n <= 1) return 1;

    std::vector<int> table(n + 1);
    table[0] = table[1] = 1;
    for (int i = 2; i <= n; ++i)
        table[i] = table[i-1] + table[i-2];
    return table.back();
}

template<typename T>
T Fibo2(T n) {
```

```
    const double sqrt5 = std::sqrt(5);
    const double phi = (1 + sqrt5) / 2;
    return (T)(std::pow(phi, n+1) / sqrt5 + 0.5);
}

template<typename T>
T Fibo3(T n) {
    static std::vector<T> arr;
    if (n <= 1) return 1;
    else if (n >= (T)arr.size())
        arr.resize(n+1);

    if (!arr[n])
        arr[n] = Fibo3(n-1) + Fibo3(n-2);
    return arr[n];
}

int main() {
    for (int i = 0; i < 10; ++i)
        printf("%d ", Fibo1(i));

    printf("\n");
    for (int i = 0; i < 10; ++i)
        printf("%d ", Fibo2(i));

    printf("\n");
    for (int i = 0; i < 10; ++i)
        printf("%d ", Fibo3(i));

    printf("\n");
    return 0;
}
```

## 6.12 博弈论

### 6.12.1 Bash-Game-sg.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01-Bash-Game-sg.cpp
 *   创建日期：2021年03月19日 ---- 10时17分
 *   题    目：hdu1846
 *   算    法：sg函数
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1005;
int n; //石子数量
int m; //一次最多可拿多少石子
int sg[maxn];
int st[maxn]; //后继结点

void SG() {
    memset(sg, 0, sizeof(sg));
    for (int i = 1; i <= n; ++i) {
        memset(st, 0, sizeof(st));
        for (int j = 1; j <= m && i-j >= 0; ++j)
            st[sg[i-j]] = 1;  //把i的后继结点(i-1, i-2, i-3, ... , i-j)放到集合st中
        for (int j = 0; j <= n; ++j)  //计算sg[i]
            if (!st[j]) {
                sg[i] = j;
                break;
            }
    }
}

```

```
int main() {
    int c;
    scanf("%d", &c);
    while (c--) {
        scanf("%d %d", &n, &m);
        SG();
        /*sg != 0 先手胜; sg == 0 后手胜*/
        /*if sg != 0 胜*/
        sg[n] ? printf("first\n") : printf("second\n");
    }
    return 0;
}
```

### 6.12.2 Nim-Game-sg.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称: 02-Nim-Game-sg.cpp
 *   创建日期: 2021年03月19日 ---- 14时46分
 *   题    目: hdu1848
 *   算    法: sg函数
 *   描    述: <++>
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1005;
int sg[maxn];
int st[maxn];
int fibo[15] = {1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987};

/*预计算每堆有0~1005石子时的sg函数*/
void SG() {
    for (int i = 0; i <= maxn; ++i) {
        sg[i] = i;
        memset(st, 0, sizeof(st));
        for (int j = 0; j < 15 && fibo[j] <= i; ++j) {
            st[sg[i-fibo[j]]] = 1;  //把i的后继结点(i-fibo[1], i-fibo[2], ... , i-fibo[j])放到集合st中
            for (int j = 0; j <= i; ++j)
                if (!st[j]) {
                    sg[i] = j;
                    break;
                }
        }
    }
}

int main() {
    SG();
    int n, m, p;
    while (scanf("%d %d %d", &n, &m, &p) && n + m + p) {
        if (sg[n] ^ sg[m] ^ sg[p])
            printf("Fibo\n");
        else
            printf("Nacci\n");
    }
    return 0;
}
```

### 6.12.3 wythoff-Game.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称: 03-wythoff-Game.cpp
 *   创建日期: 2021年03月19日 ---- 14时59分
 *   题    目: hdu1527
```

```
 *    算    法: 博弈论
 *    描    述: <++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
double gold = (1 + sqrt(5)) / 2; //黄金分割 = 1.61803398...

int main() {
    int n, m;
    while (scanf("%d %d", &n, &m)) {
        int mini = min(n, m);
        int maxi = max(n, m);
        double k = double(maxi - mini);
        int test = (int)(k * gold);  //乘以黄金分割数，然后取整
        /*test == mini 先手败*/
        test == mini ? printf("0\n") : printf("1\n");
    }
    return 0;
}
```

### 6.12.4 Nim 游戏.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: Nim游戏.cpp
 *    创建日期: 2021年10月15日 星期五 00时10分07秒
 *    题    目: AcWing 0891 Nim游戏
 *    算    法: 博弈论
 *    描    述:  给定 n 堆石子，两位玩家轮流操作，每次操作可以从
 *    任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），
 *    最后无法进行操作的人视为失败。
 *    问如果两人都采用最优策略，先手是否必胜。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
int main() {
    int n; scanf("%d", &n);
    int res = 0;
    while (n -- ) {
        int _;
        scanf("%d", &_);
        res ^= _;
    }
    if (res)
        puts("Yes");
    else
        puts("No");
    return 0;
}
```

### 6.12.5 Nim 游戏.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: 台阶-Nim游戏.cpp
 *    创建日期: 2021年10月15日 星期五 00时24分32秒
 *    题    目: AcWing 0892 台阶-Nim游戏
 *    算    法: 博弈论
 *    描    述: 只看奇数台阶石子，就是经典的Nim游戏
 *    对手拿偶数台阶的石子，那就把刚才拿下的石子再往下拿，
 *    保证奇数台阶的石子数不变
 *    对手拿奇数台阶石子，那就也拿奇数台阶石子，使之异或为零
```

```
 *
 ----------------------------------------------------------------*/

#include <cstdio>
int main() {
    int n; scanf("%d", &n);
    int idx = 1;
    int res = 0;
    while (n -- ) {
        int _; scanf("%d", &_);
        if (idx & 1)
            res ^= _;
        idx ++ ;
    }
    if (res)
        puts("Yes");
    else
        puts("No");
    return 0;
}
```

### 6.12.6 Nim 游戏.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：拆分-Nim游戏.cpp
 *   创建日期：2021年10月15日 星期五 21时42分41秒
 *   题    目：AcWing 0894 拆分-Nim游戏
 *   算    法：博弈论 sg函数
 *   描    述：
 *   给定 n 堆石子，两位玩家轮流操作，每次操作可以取走其中的一堆石子
 *   然后放入两堆规模更小的石子（新堆规模可以为 0
 *   且两个新堆的石子总数可以大于取走的那堆石子数）
 *   最后无法进行操作的人视为失败。
 *   问如果两人都采用最优策略，先手是否必胜。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <unordered_set>
using namespace std;
const int maxn = 100 + 5;
int f[maxn];

int sg(int x) {
    if (f[x] != -1)
        return f[x];
    unordered_set<int> S;
    // 分成两堆
    for (int i = 0; i < x; ++ i)
        for (int j = 0; j <= i; ++ j)
            S.insert(sg(i) ^ sg(j));
    // mex
    for (int i = 0; ; ++ i)
        if (S.count(i) == 0)
            return f[x] = i;
}

int main() {
    memset(f, -1, sizeof f);
    int n; scanf("%d", &n);
    int res = 0;
    for (int i = 0; i < n; ++ i) {
        int _; scanf("%d", &_);
        res ^= sg(_);
    }
    if (res == 0)
        puts("No");
```

```
    else
        puts("Yes");
    return 0;
}
```

### 6.12.7 Nim 游戏-sg 函数.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：集合-Nim游戏-sg函数.cpp
 *   创建日期：2021年10月15日 星期五 12时47分50秒
 *   题    目：AcWing 0893 集合-Nim游戏
 *   算    法：博弈论 sg函数
 *   描    述：
 *   给定 n 堆石子以及一个由 k 个不同正整数构成的数字集合 S。
 *   现在有两位玩家轮流操作，每次操作可以从任意一堆石子中拿取石子，
 *   每次拿取的石子数量必须包含于集合 S，最后无法进行操作的人视为失败。
 *   问如果两人都采用最优策略，先手是否必胜。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <unordered_set>
using namespace std;
const int maxn = 105, maxm = 1e4 + 5;
int n, m;
int s[maxn];
int used[maxm];

int sg(int x) {
    if (used[x] != -1)
        return used[x];
    // 哈希表存储所有能到的局面
    unordered_set<int> S;
    for (int i = 0; i < m; ++ i) {
        int sum = s[i];
        if (x >= sum)
            S.insert(sg(x - sum));
    }
    for (int i = 0; ; ++ i)
        if (!S.count(i))
            return used[x] = i;
}

int main() {
    // 集合 S 中的数的个数
    scanf("%d", &m);
    for (int i = 0; i < m; ++ i)
        scanf("%d", &s[i]);
    memset(used, -1, sizeof used);
    scanf("%d", &n);
    int res = 0;
    for (int i = 0; i < n; ++ i) {
        int x; scanf("%d", &x);
        res ^= sg(x);
    }
    if (res == 0)
        puts("No");
    else
        puts("Yes");
    return 0;
}
```

# 7 数据结构

## 7.1 栈

### 7.1.1 模拟栈.cpp

```cpp
/*--------------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年07月28日 星期三 21时51分40秒
 *   题    目：AcWing 0828 模拟栈
 *   算    法：栈
 *   描    述：<++>
 *       实现一个栈，栈初始为空，支持四种操作：
 *       - push x – 向栈顶插入一个数 x；
 *       - pop – 从栈顶弹出一个数；
 *       - empty – 判断栈是否为空；
 *       - query – 查询栈顶元素。
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5;
int stk[maxn], tt = 0;

// 插入：stk[++tt] = x;
// 弹出：stk[tt--] = x;
// 为什么stk[0]这个位置不用呢？把它留着作为边界，或者在stk[0]这个位置放一个特殊的值

int main() {
    int t; scanf("%d", &t);
    char op[10];
    while (t--) {
        scanf("%s", op);
        if (!strcmp(op, "push")) {
            int x; scanf("%d", &x);
            stk[++tt] = x;
        }
        else if (!strcmp(op, "pop")) {
            tt--;
        }
        else if (!strcmp(op, "empty")) {
            printf(tt ? "NO\n" : "YES\n");
        }
        else if (!strcmp(op, "query")) {
            printf("%d\n", stk[tt]);
        }
    }
    return 0;
}
```

### 7.1.2 表达式求值.cpp

```cpp
#include <cstdio>
#include <stack>
#include <string>
#include <algorithm>
#include <cctype>
#include <iostream>
#include <unordered_map>
using namespace std;

stack<int> num;
stack<char> op;

void eval() {
    auto b = num.top(); num.pop();
    auto a = num.top(); num.pop();
```

```
    auto c = op.top(); op.pop();
    int x;
    if (c == '+') x = a + b;
    else if (c == '-') x = a - b;
    else if (c == '*') x = a * b;
    else x = a / b;
    num.push(x);
}

int main() {
    unordered_map<char, int> pr{{'+', 1}, {'-', 1}, {'*', 2}, {'/', 2}};
    string str;
    cin >> str;
    for (int i = 0; i < (int)str.length(); ++i) {
        auto c = str[i];
        if (isdigit(c)) {
            int x = 0, j = i;
            while (j < str.size() && isdigit(str[j]))
                x = x * 10 + str[j++] - '0';
            i = j - 1;
            num.push(x);
        }
        else if (c == '(')
            op.push(c);
        else if (c == ')') {
            while (op.top() != '(')
                eval();
            op.pop();
        }
        else {
            while (op.size() && pr[op.top()] >= pr[c])
                eval();
            op.push(c);
        }
    }
    while (op.size())
        eval();
    cout << num.top();
    return 0;
}
```

## 7.2 队列

### 7.2.1 模拟队列.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年07月29日 星期四 00时24分27秒
 *   题    目：AcWing 0829 模拟队列
 *   算    法：队列
 *   描    述：
 *       栈可能会出现边界问题，所以可能在栈底放一个特殊元素
 *       而队列不会出现这样的边界
 *       为了使这两个数据结构插入元素时都使用++tt，所以队列的tt初始为-1
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5;
int q[maxn], hh = 0, tt = -1;

// 插入：q[++tt] = x;
// 弹出：hh++;

int main() {
    int t; scanf("%d", &t);
    char op[10];
```

```
    while (t--) {
        scanf("%s", &op);
        if (!strcmp(op, "push")) {
            int x; scanf("%d", &x);
            q[ ++ tt] = x;
        }
        else if (!strcmp(op, "pop")) {
            hh ++ ;
        }
        else if (!strcmp(op, "empty")) {
            // if (hh <= tt)  不为空
            printf(hh > tt ? "YES\n" : "NO\n");
        }
        else if (!strcmp(op, "query")) {
            printf("%d\n", q[hh]);
        }
    }
    return 0;
}
```

## 7.3 链表 + 邻接表

### 7.3.1 双链表.cpp

### 7.3.2 邻接表.cpp

### 7.3.3 链式前向星.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：04-邻接表.cpp
 *   创建日期：2021年04月14日 ---- 16时30分
 *   题    目：算法竞赛
 *   算    法：邻接表
 *   描    述：这种的邻接表和一般的邻接表不一样，这个是按输入的倒序
 *      存储的
 *      如果从h[x]开始搜索，得到的是最后一次加入到链表x的边tot
 *      ve[tot]，ed[tot]存储的是与x相连的节点和权值，然后进行操作
 *      i = ne[i]，此时i是x结点上一次添加的边，所以是倒序打印
 *
 *      !一定要有memset(h, -1, sizeof h)，因为你的tot是从0开始的
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5;
int ve[maxn], ed[maxn], ne[maxn], h[maxn];
int tot;

//加入有向边(x, y)，权值为z
void add(int x, int y, int z) {
    ve[tot] = y;
    ed[tot] = z;
    ne[tot] = h[x];
    h[x] = tot++;
}

int main() {
    // freopen("in.txt", "r", stdin);
    int n, m;
    memset(h, -1, sizeof h);   // 这条链表的最后一个是-1
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
```

```
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
    }
    printf("<-->\n");
    for (int x = 1; x <= n; ++x)
        //访问从x出发的所有边
        for (int i = h[x]; i != -1; i = ne[i]) {
            int y = ve[i];
            int z = ed[i];
            printf("%d %d %d\n", x, y, z);
        }
    return 0;
}
/*
6 7
4 3 1
4 6 1
3 6 1
3 2 1
1 2 1
6 1 1
6 5 1
*/
```

### 7.3.4 单链表.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年07月26日 星期一 22时02分51秒
 *   题    目：Acwing 0826 单链表
 *   算    法：单链表
 *   描    述：因为数组下标是从0开始的，所以下标是k-1的数是第k个插入的
 *   实现一个单链表，链表初始为空，支持三种操作：
 *       - 向链表头插入一个数；
 *       - 删除第 k 个插入的数后面的数；
 *       - 在第 k 个插入的数后插入一个数。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;

// head是头指针的下标
int head, e[maxn], ne[maxn];
// 也是相当于一个指针，一开始这个链表是空的，如果要往里面新建结点，就需要知道哪个结点是空的，idx指向的位置就是空的位置，将idx位置的结点填入数据之后，将idx向后移动一位，继续指向一个空的结点
int idx;

void init() {
    head = -1; idx  = 0;
}

/**
 * head -> node1 -> node2 -> ... -> nodek
 *      ^
 *      |  要在这里插入一个结点(头结点处，头插法)
 */
void add_to_head(int x) {
    e[idx] = x;
    ne[idx] = head;
    head = idx;
    idx++;
}

/**
 * head -> node1 -> node2 -> ... -> nodek -> ...
 *                                   ^
```

```
 *      在下标是k的结点后面插入一个结点      |
 *
 *  错了，不是上面这个操作，图画错了，不是第k个结点后面，是数组下标为k的后面
 *  没错，仔细体会，因为idx只有递增且每次加一，数组下标为k的结点就是第k个插入的
 */
void add(int k, int x) {
    e[idx] = x;
    ne[idx] = ne[k];
    ne[k] = idx;
    idx++;
}

// 将下标为k的点后面的点删除
// 这里没有特判头节点删除的情况，题目中要是出现了remove(head)，需要在代码中判断
void remove(int k) {
    ne[k] = ne[ne[k]];
}

int main() {
    init();
    int m; scanf("%d", &m);
    while (m--) {
        getchar();
        char ch; scanf("%c", &ch);
        int x, k;
        if (ch == 'H') {
            scanf("%d", &x);
            add_to_head(x);
        }
        else if (ch == 'D') {
            scanf("%d", &k);
            if (k == 0) {
                head = ne[head];
                continue;
            }
            remove(k - 1);
        }
        else if (ch == 'I') {
            scanf("%d %d", &k, &x);
            add(k - 1, x);
        }
    }
    int i = head;
    while (i != -1) {
        printf("%d ", e[i]);
        i = ne[i];
    }
    return 0;
}
```

## 7.4 堆

### 7.4.1 二叉堆

#### 7.4.1.1 对顶堆.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：对顶堆.cpp
 *   创建日期：2021年06月02日 星期三 22时21分50秒
 *   题    目：AcWing 0106 动态中位数
 *   算    法：对顶堆
 *   描    述：维护一个动态中位数
 *          一般堆用he(heap)表示，大根堆用hE表示
 *          哈希用ha(hash)表示
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <queue>
```

```cpp
#include <vector>
using namespace std;
#define NEXTLINE puts("");

int main() {
    // t组数，m是各组编号，n是各组数据个数
    int t, m, n;
    scanf("%d", &t);
    while (t--) {
        priority_queue<int> hE; // hE为大根堆
        priority_queue<int, vector<int>, greater<int>> he; // he为小根堆
        scanf("%d %d", &m, &n);
        printf("%d %d\n", m, (n + 1) / 2);
        int cnt = 0;
        for (int i = 1; i <= n; ++i) {
            int _;
            scanf("%d", &_);
            he.push(_);
            if (hE.size() && hE.top() > he.top()) {
                int a = he.top(),
                    b = hE.top();
                he.pop(), hE.pop();
                he.push(b), hE.push(a);
            }
            while (he.size() > hE.size()) {
                hE.push(he.top());
                he.pop();
            }
            if (i & 1)//奇数
                printf("%d%c", hE.top(), ++cnt % 10 == 0 ? '\n' : ' ');//每10个数换一行
        }
        if (cnt % 10)
            NEXTLINE
    }
    return 0;
}
```

#### 7.4.1.2 模拟堆.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：模拟堆.cpp
 *   创建日期：2021年08月09日 星期一 10时41分38秒
 *   题    目：<++>
 *   算    法：<++>
 *   描    述：从1开始，树的基本知识
 *
 ----------------------------------------------------------------*/

// h[N]存储堆中的值，h[1]是堆顶，x的左儿子是2x，右儿子是2x + 1
// ph[k]存储第k个插入的点在堆中的位置
// hp[k]存储堆中下标是k的点是第几个插入的
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int he[maxn], tot, idx;
int ph[maxn], hp[maxn];

void heap_swap(int a, int b) {
    swap(ph[hp[a]], ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(he[a], he[b]);
}

void down(int u) {
    int minidx = u;
    if (u * 2 <= tot && he[u * 2] < he[minidx])
```

```
        minidx = u * 2;
    if (u * 2 + 1 <= tot && he[u * 2 + 1] < he[minidx])
        minidx = u * 2 + 1;
    if (u != minidx) {
        heap_swap(u, minidx);
        down(minidx);
    }
}

void up(int u) {
    while (u / 2 && he[u / 2] > he[u]) {
        heap_swap(u / 2, u);
        u /= 2;
    }
}

int main() {
    int n; scanf("%d", &n);
    while (n--) {
        char op[10];
        int k, x;
        scanf("%s", op);
        if (!strcmp(op, "I")) {          // 插入一个数
            scanf("%d", &x);
            tot ++ ;
            idx ++ ;
            ph[idx] = tot, hp[tot] = idx;
            he[tot] = x;
            up(tot);
        }
        else if (!strcmp(op, "PM"))      // 输出当前集合最小值
            printf("%d\n", he[1]);
        else if (!strcmp(op, "DM")) {    // 删除当前集合最小值
            heap_swap(1, tot);
            tot--;
            down(1);
        }
        else if (!strcmp(op, "D")) {     // 删除第k个插入的数
            scanf("%d", &k);
            k = ph[k];
            heap_swap(k, tot);
            tot--;
            down(k), up(k);
        }
        else {
            scanf("%d %d", &k, &x);      // 修改第k个插入的数，将其变为x
            k = ph[k];
            he[k] = x;
            down(k), up(k);
        }
    }
    return 0;
}
```

## 7.5 并查集

### 7.5.1 合并集合.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年08月08日 星期日 17时48分14秒
 *   题    目：AcWing 0836 合并集合
 *   算    法：并茶集
 *   描    述：路径压缩，按秩合并代码比这个长，而且效率低
 *   M a b，将编号为 a 和 b 的两个数所在的集合合并
 *   Q a b，询问编号为 a 和 b 的两个数是否在同一个集合中
 *
 *   0 <= n, m <= 1e5
```

```
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int n, m;
int fa[maxn];

// 返回x的祖宗结点 + 路径压缩
int find(int x) {
    if (fa[x] == x)
        return x;
    return fa[x] = find(fa[x]);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++i)  // 灯笼的init()函数，根据题目要求选择从下标1开始还是0开始
        fa[i] = i;
    while (m--) {
        char op[2];
        int a, b;
        scanf("%s %d %d", op, &a, &b);
        if (op[0] == 'M')
            fa[find(a)] = find(b);  // 灯笼的union_vert()函数，给a的祖宗认个爹
        else if (op[0] == 'Q') {
            if (find(a) == find(b))
                puts("Yes");
            else
                puts("No");
        }
    }
    return 0;
}
```

### 7.5.2 食物链.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：02.cpp
 *   创建日期：2021年08月08日 星期日 21时21分15秒
 *   题    目：AcWing 0240 食物链
 *   算    法：并查集
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 5e4 + 5;
int fa[maxn];  // 父节点
int d[maxn];   // 距离根节点的距离

// 并查集查询操作，维护一个d数组
int find(int x) {
    if (fa[x] != x) {
        int u = find(fa[x]);
        d[x] += d[fa[x]];
        fa[x] = u;
    }
    return fa[x];
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++i)
        fa[i] = i;
    int res = 0;
    while (m--) {
```

```
        int s, x, y;
        scanf("%d %d %d", &s, &x, &y);
        if (x > n || y > n)
            res++;
        else {
            int rootx = find(x), rooty = find(y);
            if (s == 1) {
                if (rootx == rooty && (d[x] - d[y]) % 3)  // 说明都已经在集合中
                        res++;
                else if (rootx != rooty) {
                    fa[rootx] = rooty;
                    d[rootx] = d[y] - d[x];
                }
            }
            else {
                if (rootx == rooty && (d[x] - d[y] - 1) % 3)
                    res++;
                else if (rootx != rooty) {
                    fa[rootx] = rooty;
                    d[rootx] = d[y] + 1 - d[x];
                }
            }
        }
    }
    printf("%d\n", res);
    return 0;
}
```

## 7.6 数状数组

### 7.6.1 一个简单的整数问题.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：一个简单的整数问题.cpp
 *   创建日期：2021年11月16日 星期二 16时34分00秒
 *   题    目：AcWing 0242 一个简单的整数问题
 *   算    法：树状数组
 *   描    述：长度为 n 的数组，m 个指令
 *   - C l r d   [l, r] 中的数都加上 d
 *   - Q x       查询第 x 个数的值
 *
 *   区间修改，单点查询
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int n, m;
int a[maxn];
typedef long long ll;
ll tr[maxn];   // 实际上是差分数组
#define lowbit(x) ((x) & -(x))

void add(int x, int c) {
    for (int i = x; i <= n; i += lowbit(i))
        tr[i] += c;
}

ll sum(int x) {
    ll res = 0;
    for (int i = x; i; i -= lowbit(i))
        res += tr[i];
    return res;
}

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

```
        scanf("%d", &a[i]);
    // 就是直接的初始化，在第i个位置放上(a[i] - a[i - 1]);
    // 是建树的过程
    for (int i = 1; i <= n; ++ i)
        add(i, a[i] - a[i - 1]);
    while (m -- ) {
        char op[2];
        int l;
        scanf("%s %d", op, &l);
        if (*op == 'C') {
            int r, d;
            scanf("%d %d", &r, &d);
            add(l, d);
            if (r + 1 <= n)
                add(r + 1, -d);
        }
        else if (*op == 'Q') {
            int x = l;
            printf("%lld\n", sum(x));
        }
    }
    return 0;
}
```

### 7.6.2 一个简单的整数问题 2.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年10月05日 星期二 21时58分37秒
 *   题    目：AcWing 0243 一个简单的整数问题
 *   算    法：线段树
 *   描    述：
 *     给定一个长度为 N 的数列 A，以及 M 条指令，每条指令可能是以下两种之一：
 *       - C l r d，表示把 A[l],A[l+1],…,A[r] 都加上 d。
 *       - Q l r，表示询问数列中第 l∼r 个数的和。
 *     对于每个询问，输出一个整数表示答案。
 *
 *   - 区间最大连续子段和   区间和
 *   - 最大数   区间最大数
 *   - 区间最大公约数   区间加
 *   - 本题   区间加 + 区间和
 *   - 区间加 ---> 差分
 *
 *   差分 diff[i]
 *
 *   第i个数 arr[i] = diff[1] + diff[2] + ... + diff[i]
 *
 *   前x项的和 sum = arr[1] + arr[2] + ... + arr[i]
 *
 *   arr1 = diff1
 *   arr2 = diff1 diff2
 *   arr3 = diff1 diff2 diff3
 *   ...   = ...
 *   arrx = diff1 diff2 diff3 ... diffx
 *
 *   |
 *   V
 *
 *           |-------------------------------------|
 *           | diff1   diff2   diff3   ...   diffx |  ->  这个矩形的左下三角形区域为所求
 *           |-------|                             |
 *   arr1 -> | diff1 | diff2   diff3   ...   diffx |
 *           |       |-------|                     |
 *   arr2 -> | diff1   diff2 | diff3   ...   diffx |
 *           |               |-------|             |
 *   arr3 -> | diff1   diff2   diff3 | ...   diffx |
 *           | ...                                 |
 *           | ...                                 |
```

```
/*
 *           | ...                                  |
 *           | diff1   diff2   diff3   ... | diffx |
 *           |                             |-------|
 *   arrx -> | diff1   diff2   diff3   ...   diffx |
 *           |-------------------------------------|
 *
 *   也就是说，整个矩形区域减去右上三角形区域为所求
 *
 *   (diff1 + diff2 + ... diffx) * (x + 1) - (1*diff1 + 2*diff2 + ... x*diffx)
 *
 *   所以使用线段树维护一个差分 diffi 的前缀和
 *                      和一个 i * diffi 的前缀和
 *
 ----------------------------------------------------------------*/

#include <cstdio>
int n, m;
const int maxn = 1e5 + 5;
int arr[maxn];
typedef long long ll;
ll tr1[maxn];   // 维护 diff[i] 的前缀和
ll tr2[maxn];   // 维护 diff[i] * i 的前缀和

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

void add(ll tr[], int x, ll c) {
    for (int i = x; i <= n; i += lowbit(i))
        tr[i] += c;
}

ll sum(ll tr[], int x) {
    ll res = 0;
    for (int i = x; i; i -= lowbit(i))
        res += tr[i];
    return res;
}

ll get_preS(int x) {
    return sum(tr1, x) * (x + 1) - sum(tr2, x);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &arr[i]);
    for (int i = 1; i <= n; ++ i) {
        int diff = arr[i] - arr[i - 1];
        add(tr1, i, diff);
        add(tr2, i, (ll)diff * i);
    }
    while (m -- ) {
        char op[2];
        int l, r, d;
        scanf("%s", op);
        scanf("%d %d", &l, &r);
        if (*op == 'C') {
            scanf("%d", &d);
            // arr[l] += d;
            add(tr1, l, d), add(tr2, l, l * d);
            // arr[r + 1] -= d;
            add(tr1, r + 1, -d), add(tr2, r + 1, (r + 1) * -d);
        }
        else if (*op == 'Q') {
            printf("%lld\n", get_preS(r) - get_preS(l - 1));
        }
    }
    return 0;
}
```

### 7.6.3 楼兰图腾.cpp

```cpp
/*--------------------------------------------------------------------
 *
 *   文件名称：楼兰图腾.cpp
 *   创建日期：2021年11月16日 星期二 13时46分47秒
 *   题    目：AcWing 0241 楼兰图腾
 *   算    法：树状数组
 *   描    述：坐标轴上有 n 个点，
 *   v : (i, yi), (j, yj), (k, yk) 满足 1 <= i < j < k <= n 且 yi > yj, yj < yk
 *   ^ : (i, yi), (j, yj), (k, yk) 满足 1 <= i < j < k <= n 且 yi < yj, yj > yk
 *
 *   求出有多少个 v 和 ^
 *
 *   [y1, yn] 是 [1, n] 的一个排列
 *
 *   实际上就是单点修改，区间查询
 *
 *   a[x] += c
 *   query(l, r);
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 2e5 + 5;
int n, y[maxn], tr[maxn];
#define lowbit(x) ((x) & -(x))
typedef long long ll;

// 在位置 x 上加上 c
void add(int x, int c) {
    for (int i = x; i <= n; i += lowbit(i))
        tr[i] += c;
}

// 返回 x 的前缀和
int sum(int x) {
    int res = 0;
    for (int i = x; i; i -= lowbit(i))
        res += tr[i];
    return res;
}

// 在 [i] 的位置左边，有多少个数大于 y[i]
int greater[maxn], lower[maxn];

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &y[i]);
    for (int i = 1; i <= n; ++ i) {
        greater[i] = sum(n) - sum(y[i]);
        lower[i] = sum(y[i] - 1);
        add(y[i], 1);
    }
    memset(tr, 0, sizeof tr);
    ll res1 = 0, res2 = 0;
    for (int i = n; i; -- i) {
        res1 += greater[i] * (ll)(sum(n) - sum(y[i]));
        res2 += lower[i] * (ll)(sum(y[i] - 1));
        add(y[i], 1);
    }
    printf("%lld %lld\n", res1, res2);
    return 0;
}
```

### 7.6.4 迷一样的牛.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：迷一样的牛.cpp
 *   创建日期：2021年11月16日 星期二 17时09分21秒
 *   题    目：AcWing 0244 迷一样的牛
 *   算    法：树状数组
 *   描    述：n 头牛站成一列，输入的第一个数是牛的个数，下面 n - 1
 *   行数代表的是第 i + 1 头牛前面有多少个比它矮的牛的个数，输出身高
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int n, h[maxn], res[maxn], tr[maxn];
#define lowbit(x) ((x) & -(x))

void add(int x, int c) {
    for (int i = x; i <= n; i += lowbit(i))
        tr[i] += c;
}

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

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++ i) {
        tr[i] = lowbit(i);   // add(i, 1);
        if (i != 1)
            scanf("%d", &h[i]);
    }
    for (int i = n; i; -- i) {
        int k = h[i] + 1;
        int l = 1, r = n;
        while (l < r) {
            int mid = l + r >> 1;
            if (sum(mid) < k)
                l = mid + 1;
            else
                r = mid;
        }
        res[i] = l;
        add(l, -1);
    }
    for (int i = 1; i <= n; ++ i)
        printf("%d\n", res[i]);
    return 0;
}
```

## 7.7 单调栈

### 7.7.1 直方图中最大矩形.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年07月31日 星期六 13时54分56秒
 *   题    目：luogu SP1805 HISTOGRA - LARGEST Rectangle in a Histogram
 *   算    法：单调栈
 *   描    述：Luogu题解
 *
 -----------------------------------------------------------------*/

```

```cpp
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
typedef long long ll;
using namespace std;
const int maxn = 1e5 + 5;
int n, tt;
ll res;
// 直方体中矩形的高度，直方体中矩形的宽度
int a[maxn], width[maxn];
int stk[maxn];

int main() {
    while(scanf("%d", &n) && n) {
        res = 0; tt = 0;

        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        a[n+1] = 0;

        for (int i = 1; i <= n+1; i++) {
            if (a[i] > stk[tt]) {
                stk[++tt] = a[i];
                width[tt] = 1;
            }
            else {
                int cnt = 0;
                while (stk[tt] > a[i]) {
                    cnt += width[tt];
                    res = max(res, (ll)cnt * stk[tt]);
                    tt--;
                }
                stk[++tt] = a[i];
                width[tt] = cnt + 1;
            }

        }
        printf("%lld\n", res);
    }
    return 0;
}
```

## 7.8 单调队列

### 7.8.1 最大子序列和.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：02-最大子序列和.cpp
 *   创建日期：2021年04月08日 ---- 21时02分
 *   题    目：ch1201
 *   算    法：单调队列
 *   描    述：给定一个长度为N的整数序列(可能有负数)，从中找出一段
 *       长度不超过M的连续子序列，使得所有子序列中所有数的和最大
 *
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 3e5 + 5;
int sum[maxn];
int quu[maxn];
int head = 0;
int tail = 0;
int res;

```

```
int main() {
    // freopen("in.txt", "r", stdin);
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &sum[i]);
        if (i)
            sum[i] += sum[i-1];
    }
    for (int i = 0; i < n; ++i) {
        while (head <= tail && i-quu[i] > m) //超出M的范围
            ++head;
        res = max(res, sum[i] - sum[quu[head]]);
        //为了保证队列单调，要使quu[tail]的位置的前缀和小于sum[i]才行
        while (head <= tail && sum[i] <= sum[quu[tail]])
            --tail;
        quu[++tail] = i;
    }
    printf("%d\n", res);
    return 0;
}
```

### 7.8.2 滑动窗口.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年08月06日 星期五 20时19分44秒
 *   题    目：AcWing 0154 滑动窗口
 *   算    法：单调队列
 *   描    述：给定一个1e6大小的数组，有一个大小为k的滑动窗口，它从
 *   数组的最左边移动到最右边，输出窗口移动过程中的最大值与最小值
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e6 + 5;
#define NEXTLINE puts("");
int quu[maxn], a[maxn];
int n, k;

void get_min() {
    int hh = 0, tt = -1;
    for (int i = 0; i < n; ++i) {
        if (hh <= tt && quu[hh] < i - k + 1)   // 超出窗口范围
            hh++;
        // 队列中一定是单调递增的，前面的比后面大，那就会删去前面的
        while (hh <= tt && a[quu[tt]] >= a[i])
            tt--;
        quu[++tt] = i;
        if (i >= k - 1)
            printf("%d ", a[quu[hh]]);
    }
}

void get_max() {
    int hh = 0, tt = -1;
    for (int i = 0; i < n; ++i) {
        if (hh <= tt && quu[hh] < i - k + 1)
            hh++;
        while (hh <= tt && a[quu[tt]] <= a[i])
            tt--;
        quu[++tt] = i;
        if (i >= k - 1)
            printf("%d ", a[quu[hh]]);
    }
}

int main() {
```

```
    scanf("%d %d", &n, &k);   // n个数，窗口大小为k
    for (int i = 0; i < n; ++i)
        scanf("%d", &a[i]);

    get_min();
    NEXTLINE
    get_max();
    return 0;
}
```

## 7.9 线段树

### 7.9.1 模板

#### 7.9.1.1 Segment-Tree(灯笼).cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：Segment_Tree.cpp
 *   创建日期：2021年04月23日 ---- 21时06分
 *   题    目：<++>
 *   算    法：线段树
 *   描    述：线段树结点中没有存储区间，但由于build, update, query
 *       操作的参数都包含了start与end，所以相当于存储了区间
 *
 -----------------------------------------------------------------*/

#include <cstdio>
using namespace std;
const int maxn = 1000;
int arr[] = {1, 3, 5, 7, 9, 11};
int tree[maxn];
int n = 6; //数组中元素的个数

// build(0, 0, n-1)，从根节点开始建树，从arr数组的start到end，node是树的结点
void build(int node, int start, int end) {
    if (start == end)
        tree[node] = arr[start];
    else {
        int mid = (start + end) / 2;
        int left_node  = 2 * node + 1;
        int right_node = 2 * node + 2;

        build(left_node,  start, mid);
        build(right_node, mid+1, end);

        tree[node] = tree[left_node] + tree[right_node];
    }
}

// update(0, 0, n-1, idx, val)，arr[idx] = val
void update(int node, int start, int end, int idx, int val) {
    if (start == end) {
        arr[idx] = val;
        tree[node] = val;
    }
    else {
        int mid = (start + end) / 2;
        int left_node  = 2 * node + 1;
        int right_node = 2 * node + 2;

        if (idx <= mid)
            update(left_node, start, mid, idx, val);
        else
            update(right_node, mid+1, end, idx, val);

        tree[node] = tree[left_node] + tree[right_node];
    }
}

```

```
// query(0, 0, n-1, L, R)，求arr[L], arr[L+1], ... arr[R]之间的和
int query(int node, int start, int end, int L, int R) {
    printf("start = %d\n", start);
    printf("end   = %d\n", end);

    if (R < start || L > end)
        return 0;
    else if (L <= start && R >= end)
        return tree[node];
    else if (start == end)
        return tree[node];
    else {
        int mid = (start + end) / 2;
        int left_node  = 2 * node + 1;
        int right_node = 2 * node + 2;
        int sum_left  = query(left_node,  start, mid, L, R);
        int sum_right = query(right_node, mid+1, end, L, R);
        return sum_left + sum_right;
    }
}

int main() {
    build(0, 0, n-1);
    for (int i = 0; i <= 2*n+2; ++i)
        printf("tree[%d] = %d\n", i, tree[i]);
    printf("\n");

    update(0, 0, n-1, 4, 6);
    for (int i = 0; i <= 2*n+2; ++i)
        printf("tree[%d] = %d\n", i, tree[i]);
    printf("\n");

    int s = query(0, 0, n-1, 2, 5);
    printf("%d\n", s);
    return 0;
}
```

#### 7.9.1.2 区间修改.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：0058-一个简单的整数问题.cpp
 *   创建日期：2021年11月11日 星期四 13时28分11秒
 *   题    目：AcWing 0243 一个简单的整数问题2
 *   算    法：线段树
 *   描    述：
 *     给定一个长度为 N 的数列 A，以及 M 条指令，每条指令可能是以下两种之一：
 *       - C l r d，表示把 A[l], A[l+1], ..., A[r] 都加上 d。
 *       - Q l r，表示询问数列中第 [l, r] 个数的和。
 *     对于每个询问，输出一个整数表示答案。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
typedef long long ll;
int arr[maxn];

struct Node {
    int l, r;
    ll sum, add;
} tr[maxn * 4];

void pushup(int u) {
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

void build(int u, int l, int r) {
    if (l == r) {
```

```
        tr[u] = {l, r, arr[l], 0};
    }
    else {
        tr[u].l = l, tr[u].r = r;
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void pushdown(int u) {
    auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
    if (root.add) {
        left.add += root.add, left.sum += (ll)(left.r - left.l + 1) * root.add;
        right.add += root.add, right.sum += (ll)(right.r - right.l + 1) * root.add;
        root.add = 0;
    }
}

void modify(int u, int l, int r, int d) {
    if (l <= tr[u].l && tr[u].r <= r) {
        tr[u].sum += (ll)(tr[u].r - tr[u].l + 1) * d;
        tr[u].add += d;
    }
    else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid)
            modify(u << 1, l, r, d);
        if (r > mid)
            modify(u << 1 | 1, l, r, d);
        pushup(u);
    }
}

ll query(int u, int l, int r) {
    if (l <= tr[u].l && tr[u].r <= r) {
        return tr[u].sum;
    }
    else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        ll sum = 0;
        if (l <= mid)
            sum += query(u << 1, l, r);
        if (r > mid)
            sum += query(u << 1 | 1, l, r);
        return sum;
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++ i)
        scanf("%d",  &arr[i]);
    build(1, 1, n);
    while (m -- ) {
        char op[2];
        int l, r;
        scanf("%s %d %d", op, &l, &r);
        if (*op == 'C') {
            int d; scanf("%d", &d);
            modify(1, l, r, d);
        }
        else if (*op == 'Q') {
            printf("%lld\n", query(1, l, r));
        }
    }
    return 0;
```

```
}
```

#### 7.9.1.3 单点修改.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：最大数.cpp
 *   创建日期：2021年09月24日 星期五 14时00分56秒
 *   结束日期：2021年09月24日 星期五 23时00分19秒
 *   题    目：AcWing 1257 最大数
 *   算    法：线段树
 *   描    述：给定一个正整数数列a1, a2, ..., an，每一个数都在
 *   0 ~ p - 1 之间，两种操作：
 *   1. 添加操作：序列后添加一个数，序列长度变成n + 1
 *   2. 询问操作：询问这个序列最后L个数中最大的数是多少
 *   线段树，动态的修改一个数，动态的查询一个区间内的最大数
 *
 *   一共有 m 次操作
 *
 *   ! 线段树的区间都是左闭右闭
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 2e5 + 5;

struct Node {
    int l, r;
    int maxi;       // 本题存储的是区间[l, r]最大值
} tr[maxn * 4];

// 由子结点的信息计算父结点的信息
void pushup(int u) {
    tr[u].maxi = max(tr[u << 1].maxi, tr[u << 1 | 1].maxi);
}

// 这里的u是l, r的一个表示，也就是每一个u一一映射一个区间
// 这棵树的所有叶子是原数组
// 左闭右闭
void build(int u, int l, int r) {
    tr[u] = {l, r};
    if (l == r)   // 叶结点
        return;
    int mid = l + r >> 1;
    build(u << 1, l, mid);
    build(u << 1 | 1, mid + 1, r);
}

// 从u结点开始查询，一般从根结点开始查询，查询[l, r]之间的最大值
// 这个函数的作用是找到既在结点u管理的区域又在在[l, r]区域中的最大值
// 左闭右闭
int query(int u, int l, int r) {
    // 要查询的区间把这个结点所管理的区间包含了，那就返回这个结点的最大值
    if (l <= tr[u].l && tr[u].r <= r)
        return tr[u].maxi;
    // 否则这个结点管理的一部分区域不在[l, r]中
    int mid = tr[u].l + tr[u].r >> 1;
    int maxi = 0;
    /*
     *          l (先判断左边，右边r无所谓)
     *  |---------------------------------------|
     * tr[u].l             mid              tr[u].r
     *
     */
    if (l <= mid)
        maxi = query(u << 1, l, r);
    /*
```

```
     *                              r (这是判断右边，左边l无所谓)
     *  |-------------------------------------|
     * tr[u].l             mid              tr[u].r
     *
     */
    if (r > mid)
        maxi = max(maxi, query(u << 1 | 1, l, r));
    return maxi;
}

// update，左闭右闭
void modify(int u, int idx, int val) {
    // 这是叶子，所以是数组中的数
    // 是数组还不行，需要tr[u].l == idx才表明找到了这个下标的数
    if (tr[u].l == tr[u].r && tr[u].l == idx)
        tr[u].maxi = val;
    else {   // 所以当前结点不是叶子结点，就需要判断往左还是往右递归
        int mid = tr[u].l + tr[u].r >> 1;
        if (idx <= mid)
            modify(u << 1, idx, val);
        else
            modify(u << 1 | 1, idx, val);
        pushup(u);
    }
}

int main() {
    int m, MOD;
    scanf("%d %d", &m, &MOD);
    build(1, 1, m);
    // idx是数的个数，last是上一个区间的答案
    int idx = 0, last = 0;
    while (m -- ) {
        char op[2]; int x;
        scanf("%s %d", op, &x);
        if (*op == 'Q') {   // Query
            last = query(1, idx - x + 1, idx);
            printf("%d\n", last);
        }
        else if (*op == 'A') {   // Append
            modify(1, ++ idx, ((long long)x + last) % MOD);
        }
    }
    return 0;
}
```

### 7.9.2 进阶

#### 7.9.2.1 区间乘-区间加.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：区间乘.cpp
 *   创建日期：2021年11月10日 星期三 19时17分03秒
 *   题    目：AcWing 1277 维护序列
 *   算    法：线段树
 *   描    述：有长为 n 的数列 a1, a2, a3, ... an
 *   一共有 m 个操作
 *   操作1：1 l r c，把 [l, r] 中的数 ai * c
 *   操作2：2 l r c，把 [l, r] 中的数 ai + c
 *   操作3：3 l r，询问所有 [l, r] 中的数的和模 MOD 的值
 *
 -----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int n, m, MOD;
typedef long long ll;
int arr[maxn];

```

```
struct Node {
    int l, r;
    int sum , add, mul;
} tr[maxn * 4];

void pushup(int u) {
    tr[u].sum = (tr[u << 1].sum + tr[u << 1 | 1].sum) % MOD;
}

void eval(Node &u, int mul, int add) {
    u.sum = ((ll)u.sum * mul + (ll)(u.r - u.l + 1) * add) % MOD;
    u.mul = (ll)u.mul * mul % MOD;
    u.add = ((ll)u.add * mul + add) % MOD;
}

void pushdown(int u) {
    eval(tr[u << 1], tr[u].mul, tr[u].add);
    eval(tr[u << 1 | 1], tr[u].mul, tr[u].add);
    tr[u].add = 0, tr[u].mul = 1;
}

void build(int u, int l, int r) {
    if (l == r)
        tr[u] = {l, r, arr[r], 0, 1};
    else {
        tr[u] = {l, r, 0, 0, 1};
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, int add, int mul) {
    if (l <= tr[u].l && tr[u].r <= r)
        eval(tr[u], mul, add);
    else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid)
            modify(u << 1, l, r, add, mul);
        if (r > mid)
            modify(u << 1 | 1, l, r, add, mul);
        pushup(u);
    }
}

int query(int u, int l, int r) {
    if (l <= tr[u].l && tr[u].r <= r)
        return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    int sum = 0;
    if (l <= mid)
        sum = query(u << 1, l, r);
    if (r > mid)
        sum = (sum + query(u << 1 | 1, l, r)) %  MOD;
    return sum;
}

int main() {
    scanf("%d %d", &n, &MOD);
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &arr[i]);
    build(1, 1, n);
    scanf("%d", &m);
    while (m -- ) {
        int op, l, r;
        scanf("%d %d %d", &op, &l, &r);
        if (op == 1) {
            int x; scanf("%d", &x);
```

```
            modify(1, l, r, 0, x);
        }
        else if (op == 2) {
            int x; scanf("%d", &x);
            modify(1, l, r, x, 1);
        }
        else
            printf("%d\n", query(1, l, r));
    }
    return 0;
}
```

#### 7.9.2.2 区间最大公约数.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：区间最大公约数.cpp
 *   创建日期：2021年10月02日 星期六 03时15分36秒
 *   题    目：AcWing 0246 区间最大公约数
 *   算    法：线段树
 *   描    述：
 *   C l r d，表示把[l, r]都加上 d。
 *   Q l r，表示询问[l, r]的最大公约数(GCD)。
 *   gcd(a1, a2, a3, ..., an) = gcd(a1, a2-a1, a3-a2, ..., an-a(n-1))
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long ll;
const int maxn = 5e5 + 5;
int n, m;   // 初始数组个数，操作数
ll arr[maxn];

struct Node {
    int l, r;
    ll diff, gcd;
} tr[maxn * 4];

// gcd(a, b) = gcd(b, a mod b)
ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}

void pushup(Node &u, Node &l, Node &r) {
    // 如果现在 u.l == 1，那么这个 u.diff 就是 arr[r];
    // 为了让我们后面求 a[l] 方便
    u.diff = l.diff + r.diff;
    // 这里的 gcd 是 gcd(a2 - a1, a3 - a2, a4 - a3, ... an - a(n-1))
    u.gcd = gcd(l.gcd, r.gcd);
}

void pushup(int u) {
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r) {
    if (l == r) {
        ll dif = arr[r] - arr[r - 1];
        tr[u] = {l, r, dif, dif};
    }
    else {
        tr[u].l = l, tr[u].r = r;
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
```

```

void modify(int u, int idx, ll val) {
    if (tr[u].l == idx && tr[u].r == idx) {
        ll b = tr[u].diff + val;
        tr[u] = {idx, idx, b, b};
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (idx <= mid)
            modify(u << 1, idx, val);
        else
            modify(u << 1 | 1, idx, val);
        pushup(u);
    }
}

Node query(int u, int l, int r) {
    if (l <= tr[u].l && tr[u].r <= r) {
        return tr[u];
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (mid < l) {
            return query(u << 1 | 1, l, r);
        }
        else if (r <= mid) {
            return query(u << 1, l, r);
        }
        else {
            auto left = query(u << 1, l, r),
                 right = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++ i)
        scanf("%lld", &arr[i]);
    build(1, 1, n);
    while (m -- ) {
        char op[2];
        int l, r;
        scanf("%s %d %d", op, &l, &r);
        if (*op == 'C') {
            ll x; scanf("%lld", &x);
            modify(1, l, x);
            if (r + 1 <= n)
                modify(1, r + 1, -x);
        }
        else if (*op == 'Q') {
            auto left = query(1, 1, l),
                 right = query(1, l + 1, r);
            printf("%lld\n", abs(gcd(left.diff, right.gcd)));
        }
    }
    return 0;
}
```

#### 7.9.2.3 区间最大连续子段和.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：区间最大连续子段和.cpp
 *   创建日期：2021年10月02日 星期六 02时10分51秒
 *   题    目：AcWing 0245 你能回答这些问题吗
```

```
 *   算    法: 线段树
 *   描    述:
 *   开始给出数组 arr
 *   1 x y 查询区间[x, y]的最大连续子段和
 *   2 x y 把A[x] 修改为 y
 *
 *
 *   这里的pushup操作的是三个结点
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 5e5 + 5;
int n, m;
int arr[maxn];
struct Node {
    int l, r;
    int sum;    // 区间和
    int lmax;   // 区间的最大前缀和
    int rmax;   // 区间的最大后缀和
    int maxi;   // 区间的最大连续子序列和
} tr[maxn * 4];

void pushup(Node &u, Node &l, Node &r) {
    u.sum = l.sum + r.sum;
    u.lmax = max(l.lmax, l.sum + r.lmax);
    u.rmax = max(r.rmax, r.sum + l.rmax);
    // |---------|   |---------|
    // 新序列最大连续子序列可能跨区间
    u.maxi = max(max(l.maxi, r.maxi), l.rmax + r.lmax);
}

void pushup(int u) {
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r) {
    if (l == r)
        tr[u] = {l, r, arr[r], arr[r], arr[r], arr[r]};
    else {
        tr[u].l = l, tr[u].r = r;
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int x, int v) {
    if (tr[u].l == x && tr[u].r == x)
        tr[u] = {x, x, v, v, v, v};
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid)
            modify(u << 1, x, v);
        else
            modify(u << 1 | 1, x, v);
        pushup(u);
    }
}

Node query(int u, int l, int r) {
    //  tr[u] |------------------------|
    //  ^ l                             r ^
    if (l <= tr[u].l && tr[u].r <= r)
        return tr[u];
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        // |------------|------------|
```

```
        //        <- ^ r
        if (r <= mid)
            return query(u << 1, l, r);
        // |------------|------------|
        //                    l ^ ->
        else if (l > mid)
            return query(u << 1 | 1, l, r);
        // |------------|------------|
        //    ^ l           r ^
        else {
            auto left  = query(u << 1, l, r);
            auto right = query(u << 1 | 1, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    // 线段树从1开始
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &arr[i]);
    build(1, 1, n);
    while (m --) {
        int k, x, y;
        scanf("%d %d %d", &k, &x, &y);
        if (k == 1) {
            if (x > y) swap(x, y);
            printf("%d\n", query(1, x, y).maxi);
        }
        else
            modify(1, x, y);
    }
    return 0;
}
```

### 7.9.3 扫描线

#### 7.9.3.1 扫描线.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：扫描线.cpp
 *   创建日期：2021年11月10日 星期三 17时41分18秒
 *   题    目：AcWing 0247 亚特兰蒂斯
 *   算    法：扫描线
 *   描    述：n个矩形，每个输入为 x1, y1, x2, y2，矩形左上，右下
 *   坐标，输出总面积
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;

struct Segment {
    double x, y1, y2;
    int k;   // 权值 +1, -1
    bool operator< (const Segment &a) const {
        return x < a.x;
    }
} seg[maxn * 2];

struct Node {
    int l, r;
    int cnt;
```

```
    double len;
} tr[maxn * 2 * 4];

vector<double> ys;   // 对纵坐标离散化

// 离散化后通过这个函数找到位置
int find(double y) {
    return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
}

void pushup(int u) {
    if (tr[u].cnt)
        tr[u].len = ys[tr[u].r + 1] - ys[tr[u].l];
    else if (tr[u].l != tr[u].r) {
        tr[u].len = tr[u << 1].len + tr[u << 1 | 1].len;
    }
    else
        tr[u].len = 0;
}

void build(int u, int l, int r) {
    tr[u] = {l, r, 0, 0};
    if (l != r) {
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    }
}

void modify(int u, int l, int r, int k) {
    if (l <= tr[u].l && tr[u].r <= r) {
        tr[u].cnt += k;
        pushup(u);
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid)
            modify(u << 1, l, r, k);
        if (r > mid)
            modify(u << 1 | 1, l, r, k);
        pushup(u);
    }
}

int main() {
    int n, t = 1;
    while (scanf("%d", &n), n) {
        ys.clear();
        for (int i = 0, idx = 0; i < n; ++ i) {
            double x1, y1, x2, y2;
            scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
            seg[idx ++ ] = {x1, y1, y2, 1};
            seg[idx ++ ] = {x2, y1, y2, -1};
            ys.push_back(y1), ys.push_back(y2);
        }
        // 去重模板
        sort(ys.begin(), ys.end());
        ys.erase(unique(ys.begin(), ys.end()), ys.end());

        build(1, 0, ys.size() - 2);

        sort(seg, seg + n * 2);
        double res = 0;
        for (int i = 0; i < n * 2; ++ i) {
            if (i > 0)   // 从第二条线开始加
                res += tr[1].len * (seg[i].x - seg[i - 1].x);
            modify(1, find(seg[i].y1), find(seg[i].y2) - 1, seg[i].k);
        }
        printf("Test case #%d\n", t ++ );
        printf("Total explored area: %.2lf\n\n", res);
    }
```

```
    return 0;
}
```

## 7.10 二叉搜索树-平衡树

### 7.10.1 二叉搜索树简介

### 7.10.2 Treap

#### 7.10.2.1 普通平衡树.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：普通平衡树.cpp
 *   创建日期：2021年11月18日 星期四 11时16分53秒
 *   题    目：AcWing 0253 普通平衡树
 *   算    法：Treap
 *   描    述：
 *   维护一些数：
 *   1. 插入一个数x
 *   2. 删除数值x（若有多个相同的x，只删除一个）
 *   3. 查询数值x的排名（若有多个相同的数，只输出最小的排名）
 *   4. 查询排名为x的数值
 *   5. 求数值x的前驱（前驱定义为小于x的最大的数）
 *   6. 求数值x的后继（后继定义为大于x的最小的数）
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct Node {
    int l, r;
    int key;
    int val;   // priority
    int cnt;   // 结点重复，使用一个结点表示cnt个权值相同的结点
    int size;   // 下面子树中有多少个结点，用于计算排名
} tr[maxn];

int root, idx;   // 根结点和链表

void pushup(int p) {
    tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
}

/** 右旋
 *        x              y
 *       / \            / \
 *      y   +   ->     -   x
 *     / \               / \
 *    -   z             z   +
 */
void zig(int &p) {
    int q = tr[p].l;   // 也就是y
    tr[p].l = tr[q].r, tr[q].r = p, p = q;
    pushup(tr[p].r), pushup(p);
}

// 左旋
void zag(int &p) {
    int q = tr[p].r;
    tr[p].r = tr[q].l, tr[q].l = p, p = q;
    pushup(tr[p].l), pushup(p);
}

int get_node(int key) {
    tr[ ++ idx].key = key;
```

```
    tr[idx].val = rand();
    tr[idx].cnt = tr[idx].size = 1;
    return idx;
}

void build() {
    // 初始化时需要两个哨兵
    get_node(-INF), get_node(INF);
    // 手动创建
    root = 1, tr[1].r = 2;
    pushup(root);
    if (tr[1].val < tr[2].val)
        zag(root);
}

void insert(int &p, int key) {
    if (!p)
        p = get_node(key);
    else if (key == tr[p].key)
        tr[p].cnt ++ ;
    else if (key < tr[p].key) {
        insert(tr[p].l, key);
        if (tr[tr[p].l].val > tr[p].val)
            zig(p);
    }
    else {
        insert(tr[p].r, key);
        if (tr[tr[p].r].val > tr[p].val)
            zag(p);
    }
    pushup(p);
}

void remove(int &p, int key) {
    if (!p)
        return;
    if (key == tr[p].key) {
        if (tr[p].cnt > 1)
            tr[p].cnt -- ;
        else if (tr[p].l || tr[p].r) {
            if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
                zig(p);
                remove(tr[p].r, key);
            }
            else {
                zag(p);
                remove(tr[p].l, key);
            }
        }
        else
            p = 0;
    }
    else if (key < tr[p].key)
        remove(tr[p].l, key);
    else
        remove(tr[p].r, key);
    pushup(p);
}

// 通过数值找排名
int get_rank_by_key(int p, int key) {
    if (!p)
        return 0;
    if (tr[p].key == key)
        return tr[tr[p].l].size + 1;
    else if (key < tr[p].key) {
        return get_rank_by_key(tr[p].l, key);
    }
    else {
        return tr[tr[p].l].size + tr[p].cnt + get_rank_by_key(tr[p].r, key);
```

```
    }
}

// 通过排名找数值
int get_key_by_rank(int p, int rank) {
    if (!p)
        return INF;
    if (tr[tr[p].l].size >= rank)
        return get_key_by_rank(tr[p].l, rank);
    else if (tr[tr[p].l].size + tr[p].cnt >= rank)
        return tr[p].key;
    else
        return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].size - tr[p].cnt);
}

// 前驱
int get_prev(int p, int key) {
    if (!p)
        return -INF;
    if (key <= tr[p].key)
        return get_prev(tr[p].l, key);
    else
        return max(tr[p].key, get_prev(tr[p].r, key));
}

// 后继
int get_next(int &p, int key) {
    if (!p)
        return INF;
    if (key >= tr[p].key)
        return get_next(tr[p].r, key);
    else
        return min(tr[p].key, get_next(tr[p].l, key));
}

int main() {
    build();
    int n; scanf("%d", &n);
    while (n -- ) {
        int opt, x;
        scanf("%d %d", &opt, &x);
        if (opt == 1) insert(root, x);
        else if (opt == 2) remove(root, x);
        else if (opt == 3) printf("%d\n", get_rank_by_key(root, x) - 1);
        else if (opt == 4) printf("%d\n", get_key_by_rank(root, x + 1));
        else if (opt == 5) printf("%d\n", get_prev(root, x));
        else if (opt == 6) printf("%d\n", get_next(root, x));
    }
    return 0;
}
```

## 7.11 可持久化数据结构

## 7.12 K-D-Tree

### 7.12.1 K-D-Tree.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-K-D-Tree.cpp
 *   创建日期：2021年04月23日 ---- 23时03分
 *   题    目：luogu p1429 平面最近点对(加强版)
 *   算    法：K-D Tree
 *   描    述：最近邻问题
 *       给定平面上n个点，找出其中的一对点的距离，使得在这n个点的所
 *       有点对中，该距离为所有点对中最小的
 *
 -----------------------------------------------------------------*/

```

```cpp
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <queue>
using namespace std;
typedef long long ll;
const int maxn = 200010;
// const int inf  = 0x3f3f3f3f;
#define bug printf("<-->\n");
#define _max(a, b) (a > b ? a : b)
#define _min(a, b) (a < b ? a : b)
/*
 * n个点；d数组表示将当前超长方体按什么维度分割的
 * lc[i]表示这一小段以i为中位数分隔之后左侧的中位数(是坐标的中位数)
 * rc[i]表示这一小段以i为中位数分隔之后右侧的中位数(是坐标的中位数)
 * lc[i]表示结点i的左孩子
 * rc[i]表示结点i的右孩子
 */
int n, lc[maxn], rc[maxn];
bool opt;
double res = 2e18;
struct Point {double x, y;} p[maxn];
double L[maxn], R[maxn], D[maxn], U[maxn];

// double dist(int a, int b) {return hypot((p[a].x - p[b].x), (p[a].y - p[b].y));} //wrong
inline double dist(int a, int b) {return (p[a].x - p[b].x)*(p[a].x - p[b].x) + (p[a].y - p[b].y)*(p[a].y - p[
    b].y);}
inline bool cmp(Point a, Point b) {return opt ? a.x < b.x : a.y < b.y;}

void maintain(int idx) { //维持，重构
    L[idx] = R[idx] = p[idx].x;
    D[idx] = U[idx] = p[idx].y;
    if (lc[idx])
        //L[idx]：当前结点与左孩子较小的横坐标，R[idx]：当前结点与左孩子较大的横坐标
        L[idx] = min(L[idx], L[lc[idx]]), R[idx] = max(R[idx], R[lc[idx]]),
            //D[idx]：当前结点与左孩子较小的纵坐标，U[idx]：当前结点与左孩子较大的纵坐标
            D[idx] = min(D[idx], D[lc[idx]]), U[idx] = max(U[idx], U[lc[idx]]);
    if (rc[idx])
        //L[idx]：当前结点与右孩子较小的横坐标，R[idx]：当前结点与右孩子较大的横坐标
        L[idx] = min(L[idx], L[rc[idx]]), R[idx] = max(R[idx], R[rc[idx]]),
            //D[idx]：当前结点与右孩子较小的纵坐标，U[idx]：当前结点与右孩子较大的纵坐标
            D[idx] = min(D[idx], D[rc[idx]]), U[idx] = max(U[idx], U[rc[idx]]);
}


void note() {
    /*
     *          ^
     *       10 |             |
     *        9 |             |
     *        8 |       B     |
     *        7 |       *     |   E
     *        6 |             |---*-----
     *        5 |         C   |
     *        4 |---------*---|
     *        3 |   *         |
     *        2 |   A       D *
     *        1 |             | * F
     *          |------------------------>
     *        0   1 2 3 4 5 6 7 8 9 10
     *
     *
     *                     (7, 2) x
     *                     /    \
     *                    /      \
     *             y (5, 4)      (9, 6) y
     *               /    \      /
     *              /      \    /
     *         (2, 3)  (4, 7)  (8, 1)
     */
```

```
}

/*如果在l到r这个区间里x维度的方差大，就返回true*/
bool va(int l, int r) {
    double avx = 0, avy = 0, vax = 0, vay = 0;  // average variance
    for (int i = l; i < r; ++i)
        avx += p[i].x, avy += p[i].y;
    avx /= (double)(r - l);  //横坐标平均值
    avy /= (double)(r - l);  //纵坐标平均值
    for (int i = l; i < r; ++i)
        vax += (p[i].x - avx) * (p[i].x - avx),  // 方差(x) = vax / (r - l + 1);
            vay += (p[i].y - avy) * (p[i].y - avy);  // 方差(y) = vay / (r - l + 1);
    return vax >= vay;
}

/*选择p[l]到p[r]的点建树，左闭右开*/
int build(int l, int r) {
    if (l >= r) return 0;
    int mid = (l + r) >> 1;
    opt = va(l, r);
    nth_element(p+l, p+mid, p+r, cmp); //d数组表示当前维度选择的分割的方向
    lc[mid] = build(l, mid), rc[mid] = build(mid + 1, r); //左孩子，右孩子
    maintain(mid);
    return mid;
}

double f(int a, int b) {
    double ret = 0;
    //结点b与其左孩子中较小的横坐标大于结点a的横坐标，则加上横坐标二次方
    if (L[b] > p[a].x) ret += (L[b] - p[a].x) * (L[b] - p[a].x);
    //结点b与其左孩子中较大的横坐标小于结点a的横坐标，则加上横坐标二次方
    if (R[b] < p[a].x) ret += (p[a].x - R[b]) * (p[a].x - R[b]);
    //结点b与其右孩子中较小的纵坐标大于结点a的横坐标，则加上纵坐标二次方
    if (D[b] > p[a].y) ret += (D[b] - p[a].y) * (D[b] - p[a].y);
    //结点b与其右孩子中较大的纵坐标小于结点a的横坐标，则加上纵坐标二次方
    if (U[b] < p[a].y) ret += (p[a].y - U[b]) * (p[a].y - U[b]);
    return ret;
}

/*
 * 函数名是query但不是查询，没有return，左闭右开
 * 可以求离p[idx]最近的点离p[idx]的距离，是在函数中更新res的值
 */
void query(int l, int r, int idx) {
    if (l > r) return;
    int mid = (l + r) >> 1;
    if (mid != idx)
        res = min(res, dist(idx, mid));
    if (l == r) return;
    //虽然函数f的理论依据不知，但猜测知是一种高维下的相对距离的表示方法
    //到达点mid的region的距离
    double distl = f(idx, lc[mid]), distr = f(idx, rc[mid]);
    //当然是离点p[idx]越近的区域存在离点p[idx]最近的点的概率大
    if (distl < res && distr < res) {
        if (distl < distr) { query(l, mid, idx);
            if (distr < res) query(mid+1, r, idx);
        }
        else { query(mid + 1, r, idx);
            if (distl < res) query(l, mid, idx);
        }
    }
    else {
        if (distl < res) query(l, mid, idx);
        if (distr < res) query(mid+1, r, idx);
    }
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
```

```
    // freopen("out.txt", "w", stdout);
#endif
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%lf %lf", &p[i].x, &p[i].y);
    build(0, n);
    for (int i = 0; i < n; ++i)
        query(0, n, i);
    printf("%.4lf\n", sqrt(res));
    return 0;
}
```

### 7.12.2 K-D-Tree.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-K-D-Tree.cpp
 *   创建日期：2021年04月24日 ---- 12时05分
 *   题    目：luogu p4357 k远点对
 *   算    法：K-D tree  对顶堆
 *   描    述：k远点对问题
 *      堆中有k个元素，最小的就是第k远的
 *      缺点，起始下标是1
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <queue>
#include <algorithm>
#include <ctime>
#include <vector>
using namespace std;
typedef long long ll;
const int maxn = 1e6 + 5;
const int inf  = 0x3f3f3f3f;
#define _max(a, b) (a > b ? a : b)
#define _min(a, b) (a < b ? a : b)
bool opt;
int n, k;
int lc[maxn<<1], rc[maxn<<1], cnt;
int L[maxn], R[maxn], D[maxn], U[maxn];
priority_queue<ll> pqu; //默认数字大的优先级高

inline ll squ(int x) {return 1ll * x * x;}
struct Point{int x, y;} p[maxn], d[maxn << 1];
inline bool cmp(Point a, Point b) {return opt ? a.x < b.x : a.y < b.y;}
inline ll dist(Point a, Point b) {return squ(a.x-b.x) + squ(a.y-b.y);}
inline ll f(Point a, int b) {
    return _max(squ(a.x-L[b]), squ(a.x-R[b])) + _max(squ(a.y-D[b]), squ(a.y-U[b]));
}

void maintain(int idx) {
    L[idx] = R[idx] = d[idx].x;
    D[idx] = U[idx] = d[idx].y;
    if (lc[idx])
        L[idx] = min(L[idx], L[lc[idx]]), R[idx] = max(R[idx], R[lc[idx]]),
            D[idx] = min(D[idx], D[lc[idx]]), U[idx] = max(U[idx], U[lc[idx]]);
    if (rc[idx])
        L[idx] = min(L[idx], L[rc[idx]]), R[idx] = max(R[idx], R[rc[idx]]),
            D[idx] = min(D[idx], D[rc[idx]]), U[idx] = max(U[idx], U[rc[idx]]);
}

/*方差*/
bool va(int l, int r) {
    double avx = 0, avy = 0, vax = 0, vay = 0;  // average variance
    for (int i = l; i < r; ++i)
        avx += p[i].x, avy += p[i].y;
    avx /= (double)(r - l);  //横坐标平均值
    avy /= (double)(r - l);  //纵坐标平均值
```

```
    for (int i = l; i < r; ++i)
        vax += (p[i].x - avx) * (p[i].x - avx),  // 方差(x) = vax / (r - l + 1);
            vay += (p[i].y - avy) * (p[i].y - avy);  // 方差(y) = vay / (r - l + 1);
    return vax >= vay;
}

/*d数组是输入的点的dfs序*/
void build(int L, int R, int &x) {
    if (L > R) return;
    x = ++cnt;
    int mid = (L+R) >> 1;
    opt = va(L, R);//或者opt=rand()%2,或者轮换维度切割也好
    nth_element(p+L, p+mid, p+R+1, cmp);
    d[x] = p[mid];
    build(L, mid-1, lc[x]);
    build(mid+1, R, rc[x]);
    maintain(x);
}

/*查找离下标idx最近的点*/
void query(int i, int idx) {
    ll distl = -inf, distr = -inf;
    if (lc[i]) distl = f(p[idx], lc[i]);
    if (rc[i]) distr = f(p[idx], rc[i]);
    ll di = dist(p[idx], d[i]);
    if (-pqu.top() < di) {
        pqu.pop();
        pqu.push(-di);
    }
    if (distl > distr) {
        if (-pqu.top() < distl) query(lc[i], idx);
        if (-pqu.top() < distr) query(rc[i], idx);
    }
    else {
        if (-pqu.top() < distr) query(rc[i], idx);
        if (-pqu.top() < distl) query(lc[i], idx);
    }
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    // freopen("out.txt", "w", stdout);
#endif
    srand(time(NULL));
    scanf("%d %d", &n, &k);
    for (int i = 1; i <= n; ++i)
        scanf("%d %d", &p[i].x, &p[i].y);
    //距离是相对的，对于两个点会有两个同样的距离
    //所有距离都比较一遍，则会出现第k大的距离
    for (int i = 0; i < 2*k; ++i)
        pqu.push(0);
    build(1, n, lc[0]);
    for (int i = 1; i <= n; ++i)
        query(1, i);
    printf("%lld\n", -pqu.top());
    return 0;
}
```

# 8 图论

## 8.1 图的存储

### 8.1.1 图论技巧.md

```
+ 稠密图用邻接矩阵来存
+ 稀疏图用邻接表存
+ 最小生成树稠密图用prim
+ 最小生成树稀疏图用Kruskal
```

## 8.2 DFS(图论)

### 8.2.1 树的重心.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *    文件名称：树的重心.cpp
 *    创建日期：2021年10月07日 星期四 15时58分34秒
 *    题      目：AcWing 0846 树的重心
 *    算      法：图论 DFS
 *    描      述：
 *    重心定义：重心是指树中的一个结点，
 *              重心是指树中的一个结点如果将这个点删除后，
 *              剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 2e5 + 5;   // 无向图，所以乘2
int h[maxn], e[maxn], ne[maxn], idx;
int n;
bool used[maxn];

int res = maxn;

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

// 以u为根的子树中点的数量
int DFS(int u) {
    used[u] = true;
    int self = 1, max_part = 0;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!used[j]) {
            int s = DFS(j);
            max_part = max(max_part, s);
            self += s;
        }
    }
    max_part = max(max_part, n - self);
    res = min(res, max_part);
    return self;
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d", &n);
    for (int i = 0; i < n - 1; ++ i) {
        int a, b;
        scanf("%d %d", &a, &b);
        add(a, b), add(b, a);
    }
    DFS(1);
    printf("%d\n", res);
    return 0;
}
```

### 8.2.2 邻接矩阵.cpp

```cpp
#include <cstdio>
#include <cstring>
```

```cpp
const int maxn = 1000; /*顶点数再多就不再建议使用邻接矩阵了*/
int n, m; /*分别是顶点个数，边的个数*/
int g[maxn][maxn];
bool used[maxn]; /*DFS搜索需要标记已访问过的顶点*/

void DFS(int vert, int depth) {
    printf("%d ", vert);
    used[vert] = true; /*将已访问过的顶点标记为已被访问*/
    for (int i = 0; i < n; ++i)
        /*邻接矩阵初始化时就需要初始化为无穷大*/
        /*所以也就是在这里判断是否是未输入的边*/
        if (used[i] == false && g[vert][i])
            DFS(i, depth + 1);
}

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    memset(g, 0, sizeof(g));
    for (int i = 0; i < m; ++i) {
        int ver1;
        int ver2;
        scanf("%d %d", &ver1, &ver2);
        g[ver1][ver2] = 1;
        g[ver2][ver1] = 1;
    }
    /*遍历图需要搜索每个顶点，首先需要判断是否已经访问*/
    for (int i = 0; i < n; ++i)
        if (used[i] == false)
            DFS(i, 1); /*1表示第一层*/
    return 0;
}
```

### 8.2.3 邻接表.cpp

```cpp
#include <cstdio>
#include <vector>
using namespace std;
const int maxn = 1e4; //顶点数比1e3少的话，就可以使用邻接矩阵
int n, m; //顶点的个数，边的个数
vector<int> g[maxn];  //邻接矩阵与邻接表都使用一个变量名，好记
bool used[maxn];      //DFS搜索需要标记已访问过的顶点

void DFS(int vertex) {
    used[vertex] = true;
    for (int i = 0; i < (int)g[vertex].size(); ++i) {
        int vert = g[vertex][i];
        if (used[vert] == false)
            DFS(vert);
    }
}

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int ver1, ver2;
        scanf("%d %d", &ver1, &ver2);
        g[ver1].push_back(ver2);
    }
    for (int i = 0; i < n; ++i)
        if (used[i] == false)
            DFS(i);
    return 0;
}
```

### 8.2.4 链式前向星.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：02-图的深度优先搜索.cpp
 *   创建日期：2021年04月14日 ---- 19时30分
 *   题    目：算法竞赛
 *   算    法：图论，链式前向星
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int n, m;
int used[maxn];
int head[maxn], edge[maxn], nxet[maxn], vert[maxn];
int cnt, tot;

//加入有向边(x, y)，权值为z
void add(int x, int y, int z) {
    vert[++tot] = y;
    edge[tot]   = z;
    nxet[tot]   = head[x];
    head[x] = tot;
}

void DFS(int x) {
    used[x] = true;
    for (int i = head[x]; i; i = nxet[i]) {
        int y = vert[i];
        if (used[y]) continue;
        printf("%d\n", y);
        DFS(y);
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    for (int i = 0; i < n; ++i)
        if (!used[i]) {
            ++cnt;
            DFS(i);
        }
    return 0;
}
```

## 8.3 BFS(图论)

### 8.3.1 BFS-邻接矩阵.cpp

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int maxn = 1000; /*如果顶点数大于1000，就不再建议使用邻接矩阵*/
int n, m; /*分别是顶点个数，边的个数*/
bool g[maxn][maxn];
bool inq[maxn]; /*当前在队列中的顶点*/

void BFS(int vert) {
    queue<int> quu;
    quu.push(vert);
    inq[vert] = true;
```

```
    while (!quu.empty()) {
        int vert = quu.front();
        printf("%d ", vert);
        quu.pop();
        for (int i = 0; i < n; ++i)
            if (inq[i] == false && g[vert][i]) {
                quu.push(i);
                inq[i] = true;
            }
    }
}

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    memset(g, 0, sizeof(g));
    for (int i = 0; i < m; ++i) {
        int ver1;
        int ver2;
        scanf("%d %d", &ver1, &ver2);
        g[ver1][ver2] = 1;
        g[ver2][ver1] = 1;
    }
    /*一般图为连通图，那么只需要一次广搜遍历就行了*/
    /*如果为非连通图，需要检查每一个顶点*/
    for (int i = 0; i < n; ++i)
        if (inq[i] == false)
            BFS(i);
    return 0;
}
```

### 8.3.2 BFS-邻接表.cpp

```
#include <cstdio>
#include <queue>
#include <vector>
using namespace std;
const int maxn = 1e4; //顶点比1e3少的话，就可以使用邻接矩阵
int n; //顶点的个数
//没有边权
vector<int> g[maxn];
bool inq[maxn]; //当前在队列中的顶点

void BFS(int vertex) {
    queue<int> quu;
    quu.push(vertex);
    inq[vertex] = true;
    while (!quu.empty()) {
        int vert = quu.front();
        quu.pop();
        for (int i = 0; i < (int)g[vert].size(); ++i) {
            int v = g[vert][i];
            if (inq[v] == false) {
                quu.push(v);
                inq[v] = true;
            }
        }
    }
}

int main() {
    //一般图为连通图，那么只需要一次广搜遍历就行了
    //如果为非连通图，需要检查每一个顶点
    for (int i = 0; i < n; ++i)
        if (inq[i] == false)
            BFS(i);
    return 0;
}
```

### 8.3.3 BFS-链式前向星.cpp

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>
using namespace std;
const int maxn = 1e4; //顶点比1e3少的话，就可以使用邻接矩阵
int n, m; //顶点的个数
int vert[maxn], edge[maxn], nxet[maxn], head[maxn], tot;
bool used[maxn];

//加入有向边(x, y)，权值为z
void add(int x, int y, int z) {
    vert[++tot] = y;
    edge[tot]   = z;
    nxet[tot]   = head[x];
    head[x] = tot;
}

void BFS() {
    queue<int> quu;
    quu.push(0);
    used[0] = true;
    while (!quu.empty()) {
        int x = quu.front();
        printf("%d\n", x);
        quu.pop();
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            if (used[y]) continue;
            quu.push(y);
            used[y] = true;
        }
    }
}

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    BFS();
    return 0;
}
```

## 8.4 拓扑排序

### 8.4.1 拓扑排序.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：01.cpp
 *   创建日期：2021年07月29日 星期四 00时43分56秒
 *   题    目：AcWing 0848 有向图的拓扑序列
 *   算    法：拓扑
 *   描    述：拓扑序在队列中，
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5;

int n, m;
```

```cpp
int h[maxn], ve[maxn], ne[maxn], tot;
int quu[maxn], degree[maxn];   // 一个点的入度数

// ve[]存储结点，ne[]存储下一个结点，是一个指针，h[]是头结点
void add(int a, int b) {
    ve[tot] = b, ne[tot] = h[a], h[a] = tot++;
}
/*
 * 把当前入度为0的点加入到队列，并把这个点指向的点入度减一
 * 因为我们把这个点放入队列了，自然要把这个点的所有连线删除，故指向的点入度减一
 * 这时又会生成入度为0的点，继续上述操作
 */
bool topsort() {
    int hh = 0, tt = 0;
    // 遍历每个点的入度，因为题目规定了点是[1, n]，所以for循环遍历[1, n]
    // 把入度为0的点添加到队列中
    for (int i = 1; i <= n; ++i)
        if (!degree[i])
            quu[tt++] = i;

    while (hh < tt) {
        int x = quu[hh++];  // 将已找到的入度为0的点指向的点入度减一

        for (int i = h[x]; i != -1; i = ne[i]) {
            int j = ve[i];
            degree[j]--;
            if (degree[j] == 0)
                quu[tt++] = j;
        }
    }
    return tt == n;   // 如果队列中添加过n个点，说明是拓扑图
}

int main() {
    scanf("%d %d", &n, &m);
    memset(h, -1, sizeof h);   // 这条链表的最后一个是-1
    for (int i = 0; i < m; ++i) {
        int ve1, ve2;
        scanf("%d %d", &ve1, &ve2);
        add(ve1, ve2);
        degree[ve2]++;
    }
    if (topsort()) {  // 队列中的点还在数组中
        for (int i = 0; i < n; ++i)
            printf("%d ", quu[i]);
        puts("");
    }
    else
        puts("-1\n");
    return 0;
}
```

## 8.5 最小生成树

### 8.5.1 Prim

#### 8.5.1.1 Prim 算法求最小生成树.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：Prim算法求最小生成树.cpp
 *   创建日期：2021年08月12日 星期四 19时52分32秒
 *   题    目：AcWing 0858 Prim算法求最小生成树
 *   算    法：prim(稠密图，边多)
 *   描    述：注意：本题下标从1开始，本题可以解决自环、重环、边权为负数问题
 *
 ----------------------------------------------------------------*/

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

```cpp
#include <cstring>
using namespace std;
const int maxn = 500 + 5;
const int INF = 0x3f3f3f3f;
int n, m;
int g[maxn][maxn];
int dist[maxn];
bool used[maxn];

/**
 * 这里res存储的是最小生成树上所有边权之和
 * 如果确定有最小生成树，可以返回空void
 * 然后把所有的res删除，dist[]中是这棵树的所有边权
 */
int prim() {
    memset(dist, 0x3f, sizeof dist);
    memset(used, 0, sizeof used);
    int source = 1; dist[source] = 0;   // 设置源点，不然下面的判断语句要额外判断是否是源点
    int res = 0;
    for (int i = 0; i < n; ++i) {   // 这个循环只是为了将n个点都加入集合中，下标无所谓
        int vert = -1;  // 用vert找到集合外最近的点
        for (int j = 1; j <= n; ++j)   // 根据题目要求设置下标开始的位置
            if (!used[j] && (vert == -1 || dist[j] < dist[vert]))
                vert = j;
        used[vert] = true;   // 找到后标记true
        if (dist[vert] == INF)   // 如果不能形成最小生成树，一般不会，除非这个点不连通
            return INF;
        // 用新加入集合中的点vert更新未进入集合的点到集合的距离
        for (int j = 1; j <= n; ++j)
            if (!used[j])
                dist[j] = min(dist[j], g[vert][j]);
        res += dist[vert];
    }
    return res;
}

int main() {
    scanf("%d %d", &n, &m);
    memset(g, 0x3f, sizeof g);
    while (m--) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    int res = prim();
    if (res >= INF)
        printf("impossible\n");
    else
        printf("%d\n", res);
    return 0;
}
```

### 8.5.2 Kruskal

#### 8.5.2.1 Kruskal 算法求最小生成树.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：Kruskal算法求最小生成树.cpp
 *   创建日期：2021年08月12日 星期四 23时59分12秒
 *   题    目：AcWing 0859 Kruskal算法求最小生成树
 *   算    法：Kruskal(稀疏图，点多边少)
 *   描    述：由于我们只需要知道边就可以进行Kruskal，所以用一个
 *       结构体存储就可以
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using std::sort;
```

```cpp
const int maxn = 2e5 + 5;
int n, m;
int fa[maxn];

struct Edge {
    int a, b, c;
    bool operator < (const Edge &E) const {
        return c < E.c;
    }
} edge[maxn];

int find(int x) {
    if (fa[x] == x)
        return x;
    return fa[x] = find(fa[x]);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        edge[i] = {a, b, c};
    }
    sort(edge, edge + m);
    for (int i = 1; i <= n; ++i)
        fa[i] = i;
    int res = 0, cnt = 0;  // 所有边权之和，加入到集合中边的个数
    for (int i = 0; i < m; ++i) {
        int a = edge[i].a,
            b = edge[i].b,
            c = edge[i].c;
        int root_a = find(a),
            root_b = find(b);
        if (root_a != root_b) {
            fa[root_a] = root_b;
            res += c;
            cnt ++;
        }
    }
    if (cnt < n - 1)
        printf("impossible\n");
    else
        printf("%d\n", res);
    return 0;
}
```

## 8.6 最短路

### 8.6.1 Dijstra

#### 8.6.1.1 Dijkstra 求最短路 I.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：Dijkstra求最短路I.cpp
 *   创建日期：2021年08月13日 星期五 11时07分24秒
 *   题    目：AcWing 0849 Dijkstra求最短路I
 *   算    法：Dijkstra
 *   描    述：朴素版Dijkstra，可以解决重边，自环问题，但是不能解决负环问题
 *       因为没有负数，所以重边也就没有意义了，因为重边是正的，就不会更新这
 *       条边。
 *       本题是有向图，转化为无向图很容易，时间复杂度O(n^2)
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
```

```cpp
const int maxn = 500 + 5;
const int INF = 0x3f3f3f3f;
int n, m;
int g[maxn][maxn], dist[maxn];
bool used[maxn];

// dist[n]表示从源点到第n个点的最短路
void dijkstra() {
    memset(dist, 0x3f, sizeof dist);
    memset(used, 0, sizeof used);
    int source = 1; dist[source] = 0;  // 设置源点
    for (int i = 0; i < n; ++i) {
        int vert = -1;
        for (int j = 1; j <= n; ++j)
            if (!used[j] && (vert == -1 || dist[j] < dist[vert]))
                vert = j;
        used[vert] = true;
        for (int j = 1; j <= n; ++j)
            if (!used[j])
                dist[j] = min(dist[j], dist[vert] + g[vert][j]);
    }
}

int main() {
    scanf("%d %d", &n, &m);
    memset(g, 0x3f, sizeof g);
    while (m--) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        g[a][b] = min(g[a][b], c);
    }
    dijkstra();
    if (dist[n] == INF)
        printf("-1\n");
    else
        printf("%d\n", dist[n]);
    return 0;
}
```

#### 8.6.1.2 Dijkstra 求最短路 II.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：Dijkstra求最短路II.cpp
 *   创建日期：2021年08月13日 星期五 11时41分15秒
 *   题    目：AcWing 0850 Dijkstra求最短路II
 *   算    法：Dijkstra
 *   描    述：Dijkstra不可以有负边，堆优化的时间复杂度为O(mlogm)
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
typedef pair<int, int> PII;
const int maxn = 2e5 + 5;
const int INF = 0x3f3f3f3f;
int n, m;
int h[maxn], e[maxn], ne[maxn], w[maxn], idx;  // 邻接表，还要存权重
int dist[maxn];
bool used[maxn];

void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

void dijkstra() {
```

```
    memset(dist, 0x3f, sizeof dist);
    memset(used, 0, sizeof used);
    int source = 1; dist[source] = 0;
    // 小根堆，优先队列会按结构体的第一位排序，所以将距离放在第一位，它就会把距离最小的放在最前面，得到最近的点
    priority_queue<PII, vector<PII>, greater<PII>> he;
    he.push({0, source});  // 距离：0，源点：source
    while (he.size()) {
        auto t = he.top();  // t = top，小根堆的堆顶，距离集合最近的点
        he.pop();
        int d = t.first, vert = t.second;   // d = dist，堆顶元素vert距离源点的距离
        if (used[vert])   // 冗余
            continue;
        used[vert] = true;
        // 用vert更新所有它邻接的点
        // 因为一个点可能同时被上一个vert和当前vert更新，就会压入堆中两次，所以就有了上面的消除冗余
        for (int i = h[vert]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > d + w[i]) {
                dist[j] = d + w[i];
                he.push({dist[j], j});
            }
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    memset(h, -1, sizeof h);  // 邻接表
    while (m --) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        add(a, b, c);
    }
    dijkstra();
    if (dist[n] == INF)
        printf("-1\n");
    else
        printf("%d\n", dist[n]);
    return 0;
}
```

### 8.6.2 Bellman-Ford

#### 8.6.2.1 有边数限制的最短路.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：有边数限制的最短路.cpp
 *   创建日期：2021年08月13日 星期五 13时46分17秒
 *   题    目：<++>
 *   算    法：<++>
 *   描    述：Bellman-Ford算法只要能够遍历到所有边就可以，所以
 *      存储时使用结构体就可以，Bellman-Ford可以存在负权边，也可以存在
 *      负权回路
 *            2           4            2
 *      1 ---------- 2 ---------- 3 ---------- 5
 *                    \         /
 *                     \       /
 *                  -2  \     /  -3
 *                       \   /
 *                        4
 *
 *   - 如果出现负权回路，就不存在从1到5的最短路，负权回路不是只有负边，
 *     还要这个回路是的权是负的
 *   - 但是如果存在负环，但是负环又不在我要求的最短路上，那就不影响最后结果
 *   本题只能使用Bellman-Ford算法来做，因为我们限制了这条最短路需要经过k条
 *   边
 *   时间复杂度O(nm)
 *
 ----------------------------------------------------------------*/
```

```

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int N = 500 + 5, M = 1e4 + 5;
const int INF = 0x3f3f3f3f;
int n, m, k;
int dist[N], backup[N];

struct Edge {
    int a, b, c;
} edge[M];

void bellman_ford() {
    memset(dist, 0x3f, sizeof dist);
    int source = 1; dist[source] = 0;
    for (int i = 0; i < k; ++i) {
        /**
         * 迭代k次，为什么要备份dist数组呢?
         * 假如输入:
         * 1 2 1
         * 2 3 1
         * 1 3 3
         * 又k = 1，那么就有1到3的距离是3，虽然可以先从1到2距离为1，然后从2到3距离为1，得出更短的距离 1 -> 3 = 2
         * 但是有限制k = 1，即只能经过一条边从1到3，如果我们不备份的话，更新了1 -> 2 = 1，然后循环继续更新2的出边
         * 就会发生串联，我们只希望第k次迭代dist[x]时使用的是第k-1次的dist
         */
        memcpy(backup, dist, sizeof dist);
        for (int j = 0; j < m; ++ j) {
            int a = edge[j].a, b = edge[j].b, c = edge[j].c;
            dist[b] = min(dist[b], backup[a] + c);
        }
    }
}

int main() {
    scanf("%d %d %d", &n, &m, &k);
    for (int i = 0; i < m; ++i) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        edge[i] = {a, b, c};
    }
    bellman_ford();
    // 为什么除以2，是因为可能有 t -> n 这条边是负权的，虽然源点到不了n，但是n会被别的点更新小一点
    if (dist[n] > INF / 2)
        printf("impossible\n");
    else
        printf("%d\n", dist[n]);
    return 0;
}
```

### 8.6.3 SPFA

#### 8.6.3.1 spfa 判断负环.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：spfa判断负环.cpp
 *   创建日期：2021年08月13日 星期五 16时09分25秒
 *   题    目：AcWing 0852 spfa判断负环
 *   算    法：spfa
 *   描    述：给定一个n个点m条边的有向图，图中可能存在重边和自环，
 *        边权可能为负数，判断图中是否存在负权回路
 *        当有一个cnt[x] >= n时，就说明出现了负环
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
```

```cpp
#include <algorithm>
#include <queue>
using namespace std;
const int maxn = 2e5 + 5;
const int INF = 0x3f3f3f3f;
int n, m;
int h[maxn], e[maxn], ne[maxn], w[maxn], idx;  // 邻接表，还要存权重
int dist[maxn], cnt[maxn];   // 这个cnt数组是多加的，cnt[x]记录从源点到节点x的最短路经过了多少条边
bool used[maxn];

void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

bool spfa() {
    // 有没有发现这里没有初始化了，因为我们不是求距离了
    // 也没有源点source了，这是因为我们需要找到负环，而不只是从源点到达的负环
    queue<int> quu;
    for (int i = 1; i <= n; ++i) {
        used[i] = true;
        quu.push(i);
    }
    while (quu.size()) {
        int vert = quu.front();
        quu.pop();
        used[vert] = false;
        for (int i = h[vert]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[vert] + w[i]) {
                dist[j] = dist[vert] + w[i];
                cnt[j] = cnt[vert] + 1;
                if (cnt[j] >= n)
                    return true;
                if (!used[j]) {
                    quu.push(j);
                    used[j] = true;
                }
            }
        }
    }
    return false;
}

int main() {
    scanf("%d %d", &n, &m);
    memset(h, -1, sizeof h);  // 邻接表
    while (m --) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        add(a, b, c);
    }
    spfa() ? printf("Yes\n") : printf("No\n");
    return 0;
}
```

#### 8.6.3.2 spfa 求最短路.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：spfa求最短路.cpp
 *   创建日期：2021年08月13日 星期五 15时13分46秒
 *   题    目：AcWing 0851 spfa求最短路
 *   算    法：spfa
 *   描    述：Bellman-Ford算法是对每个点每条边都更新，而spfa只更新
 *       那些在第一次更新时变小的点，不可以操作有负权回路的图
 *
 ----------------------------------------------------------------*/

#include <cstdio>
```

```cpp
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
const int maxn = 2e5 + 5;
const int INF = 0x3f3f3f3f;
int n, m;
int h[maxn], e[maxn], ne[maxn], w[maxn], idx;  // 邻接表，还要存权重
int dist[maxn];
bool used[maxn];

void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

void spfa() {
    memset(dist, 0x3f, sizeof dist);
    int source = 1; dist[source] = 0;
    queue<int> quu;
    quu.push(source); used[source] = true;  // used表示当前元素是否在队列中，防止出现重复的点
    while (quu.size()) {
        int vert = quu.front();
        quu.pop();
        used[vert] = false;
        for (int i = h[vert]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[vert] + w[i]) {
                dist[j] = dist[vert] + w[i];
                if (!used[j]) {
                    quu.push(j);
                    used[j] = true;
                }
            }
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    memset(h, -1, sizeof h);  // 邻接表
    while (m --) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        add(a, b, c);
    }
    spfa();
    if (dist[n] == INF)
        printf("impossible\n");
    else
        printf("%d\n", dist[n]);
    return 0;
}
```

### 8.6.4 Floyd

#### 8.6.4.1 Floyd 求最短路.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：Floyd求最短路.cpp
 *   创建日期：2021年08月13日 星期五 16时23分01秒
 *   题    目：AcWing 0854 Floyd求最短路
 *   算    法：Floyd
 *   描    述：给定一个 n 个点 m 条边的有向图，图中可能存在重边和自环
 *   边权可能为负数。再给定 k 个询问，每个询问包含两个整数 x 和 y，
 *   表示查询从点 x 到点 y 的最短距离，如果路径不存在，则输出 impossible
 *
 *   可以有负权边，不可有负权回路
 *
 *   算法过程：检查是否存在一个点k使得i与j之间的最短路能够更新
```

```
 *    基于动态规划理解: d[k, i, j]表示从i点只经过[i, k]这些点到达j的最短路径
 *                      d[k-1, i, j] = d[k-1, i, k] + d[k-1, k, j];
 *                      发现第一维可以去掉 d[i, j] = d[i, k] + d[k, j];
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 200 + 5;
const int INF = 0x3f3f3f3f;
int n, m, Q;
int d[maxn][maxn];   // 使用d数组而不是g数组，因为后面会更改里面的值，最后存的是距离

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

int main() {
    scanf("%d %d %d", &n, &m, &Q);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            if (i == j)
                d[i][j] = 0;  // 防止出现自环
            else
                d[i][j] = INF;
    while (m --) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        d[a][b] = min(d[a][b], c);  // 取消重边
    }
    floyd();
    while (Q --) {
        int a, b;
        scanf("%d %d", &a, &b);
        if (d[a][b] > INF / 2)
            printf("impossible\n");
        else
            printf("%d\n", d[a][b]);
    }
    return 0;
}
```

## 8.7 连通性相关

### 8.7.1 强连通分量

#### 8.7.1.1 Kosaraju.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: 01-Kosaraju.cpp
 *    创建日期: 2021年03月26日 ---- 19时39分
 *    题    目: hdu1269 迷宫城堡
 *    算    法: Kosaraju
 *    描    述: 一个有向图，有n个点(n <= 10000)和m条边(m <= 100000)
 *       判断整个图是否强连通，如果是，输出Yes，否则输出No
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <cstring>
using namespace std;
const int maxn = 10005;
```

```
vector<int> g[maxn], rg[maxn];
vector<int> S; //存第一次dfs1的结果：标记点的先后顺序
int vis[maxn]; //在dfs1()中，用vis[i]记录点i是否被访问
int sccno[maxn]; //sccno[i]是第i个点所属的SCC，在dfs2()中，sccno[i]也被用于记录点i是否被访问
int cnt; // cnt：强连通分量的个数

void dfs1(int u) {
    if (vis[u])
        return;
    vis[u] = 1;
    for (int i = 0; i < g[u].size(); ++i)
        dfs1(g[u][i]);
    S.push_back(u);  //记录点的先后顺序，标记大的放在S的后面
}

void dfs2(int u) {
    if (sccno[u])
        return;
    sccno[u] = cnt;
    for (int i=0; i < rg[u].size(); ++i)
        dfs2(rg[u][i]);
}

void Kosaraju(int n) {
    cnt = 0;
    S.clear();
    memset(sccno, 0, sizeof(sccno));
    memset(vis, 0, sizeof(vis));
    for (int i = 1; i <= n; ++i)
        dfs1(i);  //点的编号：1~n。递归所有点
    for (int i = n-1; i >= 0; --i)
        if (!sccno[S[i]]) {
            cnt++;
            dfs2(S[i]);
        }
}

int main() {
    int n, m, u, v;
    while (scanf("%d %d", &n, &m), n != 0 || m != 0) {
        for(int i = 0; i < n; ++i) {
            g[i].clear();
            rg[i].clear();
        }
        for(int i = 0; i < m; ++i){
            scanf("%d %d", &u, &v);
            g[u].push_back(v);    //原图
            rg[v].push_back(u);   //反图
        }
        Kosaraju(n);
        printf("%s\n", cnt == 1 ? "Yes" : "No");
    }
    return 0;
}
```

#### 8.7.1.2 Tarjan.cpp

```
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;
const int maxn = 10005;
int cnt; // 强连通分量的个数
int low[maxn];
int num[maxn];
int dfn;
int sccno[maxn];
int stack[maxn];
int top; // 用stack[]处理栈，top是栈顶
```

```
vector<int> g[maxn];

void dfs(int u) {
    stack[top++] = u; //u进栈
    low[u] = num[u] = ++dfn;
    for (int i = 0; i < g[u].size(); ++i) {
        int v = g[u][i];
        if (!num[v]) {            //未访问过的点，继续dfs
            dfs(v);               //dfs的最底层，是最后一个SCC
            low[u] = min(low[v], low[u]);
        }
        else if (!sccno[v])       //处理回退边
            low[u]= min(low[u], num[v]);
    }
    if (low[u] == num[u]) {       //栈底的点是SCC的祖先，它的low = num
        cnt++;
        while (1) {
            int v = stack[--top]; //v弹出栈
            sccno[v] = cnt;
            if (u == v)           //栈底的点是SCC的祖先
                break;
        }
    }
}

void Tarjan(int n) {
        cnt = top = dfn = 0;
        memset(sccno, 0, sizeof(sccno));
        memset(num, 0, sizeof(num));
        memset(low, 0, sizeof(low));
        for (int i = 1; i <= n; ++i)
            if (!num[i])
                dfs(i);
}

int main() {
    int n, m, u, v;
    while(scanf("%d %d", &n, &m), n != 0 || m != 0) {
        for (int i = 1; i <= n; ++i)
            g[i].clear();
        for (int i = 0; i < m; ++i) {
            scanf("%d %d", &u, &v);
            g[u].push_back(v);
        }
        Tarjan(n);
        printf("%s\n", cnt == 1 ? "Yes" : "No" );
    }
    return 0;
}
```

### 8.7.2 双连通分量

#### 8.7.2.1 边双连通分量.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-边双连通分量.cpp
 *   创建日期：2021年03月26日 ---- 15时13分
 *   题    目：poj3352 Road Construction
 *   算    法：边双连通分量
 *   描    述：给定一个无向图，图中没有重边，问添加几条边才能使无
 *       向图变成边双连通分量
 *
 -----------------------------------------------------------------*/

#include<cstring>
#include<vector>
#include<stdio.h>
using namespace std;
const int maxn = 1005;
```

```cpp
int n, m, low[maxn], dfn;
vector<int> g[maxn];

void dfs(int u, int fa) {  //计算每个点的low值
    low[u] = ++dfn;
    for (int i = 0; i < g[u].size(); ++i) {
        int v = g[u][i];
        if (v == fa)
            continue;
        if (!low[v])
            dfs(v, u);
        low[u] = min(low[u], low[v]);
    }
}

int tarjan() {
    int degree[maxn];  //计算每个缩点的度数
    memset(degree, 0, sizeof(degree));
    for (int i = 1; i <= n; ++i)  //把有相同low值的点看成一个缩点
        for (int j = 0; j < g[i].size(); ++j)
            if (low[i] != low[g[i][j]])
                degree[low[i]]++;
    int res = 0;
    for(int i = 1; i <= n; ++i)  //统计度数为1的缩点个数
        if (degree[i] == 1)
            ++res;
    return res;
}

int main() {
    while(~scanf("%d%d", &n, &m)) {
        memset(low, 0, sizeof(low));
        for (int i = 0; i <= n; ++i)
            g[i].clear();
        for (int i = 1; i <= m; ++i) {
            int ver1, ver2;
            scanf("%d%d", &ver1, &ver2);
            g[ver1].push_back(ver2);
            g[ver2].push_back(ver1);
        }
        dfn = 0;
        dfs(1, -1);
        int ans = tarjan();
        printf("%d\n", (ans+1)/2);
    }
    return 0;
}
```

### 8.7.3 割点和桥

#### 8.7.3.1 判断是否是割点.cpp

```cpp
/*------------------------------------------------------------------
 *
 *   文件名称：01-判断是否是割点.cpp
 *   创建日期：2021年03月24日 ---- 20时51分
 *   题    目：poj1144 network
 *   算    法：割点
 *   描    述：输入一个无向图，求割点的数量
 *       使用深搜优先生成树求割点
 *       定义num[u]记录DFS对每个点的访问顺序，num值随着递推深度增加
 *       而变大
 *       定义low[v]记录v和u的后代能连回到的祖先的num
 *       只要low[v] >= num[u]，就说明在v这个支路上没有回退边连回u的
 *       祖先，最多退到u本身
 *       把程序中的if (low[v] >= num[u] && u != 1) 改为 if (low[v] > num[u] && u != 1)
 *       其他程序不变，就是求割边的数量
 *
 ------------------------------------------------------------------*/

```

```cpp
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
using namespace std;
const int maxn = 109;
int low[maxn];
int num[maxn];
int dfn; //时间戳，记录进入递归的顺序
bool iscut[maxn]; //标记是否为割点
vector <int> g[maxn];

void dfs(int u, int fa) {     //u的父结点是fa
    low[u] = num[u] = ++dfn;  //初始值
    int child = 0;            //孩子数目
    for (int i = 0; i < g[u].size(); ++i) { //处理u的所有子结点
        int v = g[u][i];
        if (!num[v]) {        //v没访问过
            child++;
            dfs(v, u);
            low[u] = min(low[v], low[u]);  //用后代的返回值更新low值
            if (low[v] >= num[u] && u !=1)
                iscut[u] = true;           //标记割点
        }
        else if (num[v] < num[u] && v != fa)
            //处理回退边，注意这里v != fa，fa是u的父结点，
            //fa也是u的邻居，但是前面已经访问过，不需要处理它
            low[u] = min(low[u], num[v]);
    }
    if (u == 1 && child >= 2) //根结点，有两个以上不相连的子树
        iscut[1] = true;
}

/*根结点从1开始*/
int main() {
    int res;
    int n;
    while (scanf("%d", &n) != -1) {
        if (n == 0)
            break;
        memset(low, 0, sizeof(low));
        memset(num, 0, sizeof(num));
        dfn = 0;
        for (int i = 0; i <= n; ++i)
            g[i].clear();
        int ver1, ver2;
        while (scanf("%d", &ver1) && ver1)
            while (getchar() != '\n') {
                scanf("%d", &ver2);
                g[ver1].push_back(ver2);
                g[ver2].push_back(ver1);
            }
        memset(iscut, false, sizeof(iscut));
        res = 0;
        dfs(1, 1);
        for (int i = 1; i <= n; ++i)
            res += iscut[i];
        printf("%d\n", res);
    }
    return 0;
}
```

## 8.8 二分图

### 8.8.1 二分图的最大匹配-匈牙利算法.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：二分图的最大匹配(匈牙利算法).cpp
```

```
 *    创建日期: 2021年10月08日 星期五 21时35分08秒
 *    题     目: AcWing 0861 二分图的最大匹配
 *    算     法: 匈牙利算法
 *    描     述: 左半部包含n1个点，右半部包含n1个点，二分图共有m个边
 *    输出最大匹配
 *    虽然是无向边，但是因为只会为男生找女生，所以只需要存一条有向边就可以
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5;
int n1, n2, m;
int h[maxn], e[maxn], ne[maxn], idx;
int match[maxn];   // 每个女孩找到的哪个男孩
bool used[maxn];   // 不重复搜一个点，就是有可能两个点之间有多条边

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

bool find(int u) {
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!used[j]) {   // 如果这个点未搜过
            used[j] = true;
            // 如果这个女孩还没找到男朋友，或者她的男朋友可以换人
            if (match[j] == 0 || find(match[j])) {
                match[j] = u;
                return true;
            }
        }
    }
    return false;
}

int main() {
    scanf("%d %d %d", &n1, &n2, &m);
    memset(h, -1, sizeof h);
    while (m -- ) {
        int a, b;
        scanf("%d %d", &a, &b);
        add(a, b);
    }

    int res = 0;   // 匹配数量
    // 给男生找女朋友
    for (int i = 1; i <= n1; ++ i) {
        memset(used, false, sizeof used);
        if (find(i))
            res ++ ;
    }
    printf("%d\n", res);
    return 0;
}
```

### 8.8.2 染色法判定二分图.cpp

```
/*----------------------------------------------------------------
 *
 *    文件名称: 染色法判定二分图.cpp
 *    创建日期: 2021年10月08日 星期五 16时32分49秒
 *    题     目: AcWing 0860 染色法判定二分图
 *    算     法: 染色法
 *    描     述:
 *    给定一个 n 个点 m 条边的无向图，图中可能存在重边和自环。
 *    请你判断这个图是否是二分图。
 *
 ----------------------------------------------------------------*/
```

```

#include <cstdio>
#include <cstring>
const int maxn = 1e5 + 5, maxe = 2e5 + 5;
int h[maxn], e[maxe], ne[maxe], idx;
int n, m;
int color[maxn];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

// 返回false表示有矛盾
bool DFS(int u, int c) {
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!color[j]) {
            if (!DFS(j, 3 - c))
                return false;
        }
        else if (color[j] == c) {
            return false;
        }
    }
    return true;
}

int main() {
    scanf("%d %d", &n, &m);
    memset(h, -1, sizeof h);
    while (m -- ) {
        int a, b;
        scanf("%d %d", &a, &b);
        add(a, b), add(b, a);
    }
    bool flag = true;
    for (int i = 1; i <= n; ++ i)
        if (!color[i])
            // 有两种颜色，1, 2
            if (!DFS(i, 1)) {
                flag = false;
                break;
            }

    if (flag) puts("Yes");
    else puts("No");
    return 0;
}
```

## 8.9 网络流

### 8.9.1 最大流

#### 8.9.1.1 Edmonds-Karp.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-Edmonds-Karp.cpp
 *   创建日期：2021年03月26日 ---- 21时44分
 *   题    目：hdu1532 Drainage Ditches
 *   算    法：最大流Edmonds-Karp
 *   描    述：源点到终点的最大流速
 *
 -----------------------------------------------------------------*/

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

```cpp
using namespace std;
const int inf = 1e9;
const int maxn = 300;
int n, m;
int g[maxn][maxn];
int pre[maxn];
// g[][]不仅记录图，还是残留网络

int bfs(int s, int t) {
    int flow[maxn];
    memset(pre, -1, sizeof pre);
    flow[s] = inf;
    pre[s] = 0;          //初始化起点
    queue<int> Q;
    Q.push(s);           //起点入栈，开始bfs
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        if (u == t)
            break;                  //搜到一个路径，这次bfs结束
        for (int i = 1; i <= m; ++i) {        //bfs所有的点
            if (i != s && g[u][i] > 0 && pre[i] == -1) {
                pre[i] = u;         //记录路径
                Q.push(i);
                flow[i] = min(flow[u], g[u][i]); //更新结点流量
            }
        }
    }
    if (pre[t] == -1)               //没有找到新的增广路
        return -1;
    return flow[t];                 //返回这个增广路的流量
}

int maxflow(int s, int t) {
    int Maxflow = 0;
    while (1) {
        int flow = bfs(s, t);
        //执行一次bfs，找到一条路径，返回路径的流量
        if (flow == -1)             //没有找到新的增广路，结束
            break;
        int cur = t;                //更新路径上的残留网络
        while (cur!=s) {            //一直沿路径回溯到起点
            int father = pre[cur];  //pre[]记录路径上的前一个点
            g[father][cur] -= flow; //更新残留网络：正向减
            g[cur][father] += flow; //更新残留网络：反向加
            cur = father;
        }
        Maxflow += flow;
    }
    return Maxflow;
}

int main() {
    while(~scanf("%d %d", &n, &m)) {
        memset(g, 0, sizeof g);
        for (int i = 0; i < n; ++i) {
            int u, v, w;
            scanf("%d %d %d", &u, &v, &w);
            g[u][v] += w;            //可能有重边
        }
        printf("%d\n", maxflow(1, m));
    }
    return 0;
}
```

### 8.9.2 费用流

#### 8.9.2.1 最小费用最大流.cpp

```cpp
/*-----------------------------------------------------------------
```

```
 *
 *    文件名称：01-最小费用最大流.cpp
 *    创建日期：2021年03月29日 ---- 11时56分
 *    题    目：poj 2135 Farm Tour
 *    算    法：Ford-Fulkerson SPFA
 *    描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <stdio.h>
#include <algorithm>
#include <cstring>
#include <queue>
using namespace std;
const int inf = 0x3f3f3f3f;
const int maxn = 1010;
int n, m;
int dis[maxn];
int pre[maxn];
int preve[maxn];
//dis[i]记录起点到i的最短距离。pre和 preve见下面注释

struct edge {
    int to, cost, capacity, rev;        //rev用于记录前驱点
    edge (int to_,int cost_,int c,int rev_) {
        to = to_;
        cost = cost_;
        capacity = c;
        rev = rev_;}
};

vector<edge> e[maxn];           //e[i]：存第i个结点连接的所有的边
void addedge(int from,int to,int cost,int capacity){//把1个有向边再分为2个
    e[from].push_back(edge(to, cost, capacity, e[to].size()));
    e[to].push_back(edge(from, -cost, 0, e[from].size()-1));
}

bool spfa(int s, int t, int cnt) {          //套SPFA模板
    bool inq[maxn];
    memset(pre, -1, sizeof(pre));
    for (int i = 1; i <= cnt; ++i) {
        dis[i]=inf;
        inq[i]=false;
    }
    dis[s] = 0;
    queue <int> Q;
    Q.push(s);
    inq[s] = true;
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        inq[u] = false;
        for (int i=0; i < e[u].size(); ++i)
            if (e[u][i].capacity > 0){
                int v = e[u][i].to, cost = e[u][i].cost;
                if (dis[u]+cost < dis[v]) {
                    dis[v] = dis[u]+cost;
                    pre[v] = u;         //v的前驱点是u
                    preve[v] = i;       // u的第i个边连接v点
                    if (!inq[v]) {
                        inq[v] = true;
                        Q.push(v);
                    }
                }
            }
    }
    return dis[t] != inf;   //s到t的最短距离（或者最小费用）是dis[t]
}

int mincost(int s, int t, int cnt) {      //基本上是套最大流模板
```

```
    int cost = 0;
    while (spfa(s, t, cnt)) {
        int v = t, flow = inf;        //每次增加的流量
        while(pre[v] != -1) {          //回溯整个路径，计算路径的流
            int u = pre[v], i = preve[v];
            //u是v的前驱点，u的第i个边连接v
            flow = min(flow, e[u][i].capacity);
            //所有边的最小容量就是这条路的流
            v = u;                    //回溯，直到源点
        }
        v = t;
        while (pre[v] != -1) {                      //更新残留网络
            int u = pre[v], i = preve[v];
            e[u][i].capacity -= flow;            //正向减
            e[v][e[u][i].rev].capacity += flow;//反向加，注意rev的作用
            v = u;                                   //回退，直到源点
        }
        cost += dis[t]*flow;
        //费用累加。如果程序需要输出最大流，在这里累加flow
    }
    return cost;                //返回总费用
}
int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 0; i < maxn; ++i)
            e[i].clear();   //清空待用
        for (int i = 1; i <= m; ++i) {
            int u, v, w;
            scanf("%d%d%d", &u, &v, &w);
            addedge(u, v, w, 1);        //把1个无向边分为2个有向边
            addedge(v, u, w, 1);
        }
        int s = n+1, t = n+2;
        addedge(s, 1, 0, 2);           //添加源点
        addedge(n, t, 0, 2);           //添加汇点
        printf("%d\n", mincost(s, t, n+2));
    }
    return 0;
}
```

# 9 杂项

## 9.1 离散化

### 9.1.1 离散化-区间和.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：02.cpp
 *   创建日期：2021年06月01日 星期二 13时25分21秒
 *   题    目：AcWing 0802 区间和
 *   算    法：离散化
 *   描    述：这个代码质量更高一点
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <algorithm>
#include <utility>
using namespace std;
// typedef pair<int, int> PII;
using PII = pair<int, int>;
#define pb push_back
#define fi first
#define se second
#define bug printf("<-->\n");
#define NEXTLINE puts("");
const int maxn = 3e5 + 5;
```

```
int n, m;
vector<PII> op, query;
vector<int> alls;  //存储所有待离散化的值
int I[maxn], preS[maxn];

int find(int x) {  //二分求出x对应的离散化的值
    int l = 0, r = alls.size() - 1;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (alls[mid] >= x)
            r = mid;
        else
            l = mid + 1;
    }
    return r + 1;  //映射到1, 2, 3, ..., n
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++i) {
        int x, c;
        scanf("%d %d", &x, &c);
        op.pb({x, c});
        alls.pb(x);
    }
    for (int i = 0; i < m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        query.pb({l, r});
        alls.pb(l);
        alls.pb(r);
    }

    sort(alls.begin(), alls.end());
    alls.erase(unique(alls.begin(), alls.end()), alls.end());

    for (auto item : op) {  //离散化成功
        int idx = find(item.fi);
        I[idx] += item.se;
    }

    for (int i = 1; i <= (int)alls.size(); ++i)  //前缀和
        preS[i] = preS[i-1] + I[i];

    /*
     * for (int i = 0; i <= alls.size(); ++i)
     *     printf("%d ", preS[i]);
     * NEXTLINE;
     */

    for (auto item : query) {
        int l = find(item.fi),
            r = find(item.se);
        int res = preS[r] - preS[l-1];
        printf("%d\n", res);
    }
    return 0;
}
```

## 9.2 数字和为 sum

### 9.2.1 01.cpp

```
#include <cstdio>
#include <cstring>
int n, sum;
int arr[n];
long long dp[sum+1]; //dp[i]表示和为i时的方案数

```

```cpp
int main() {
    scanf("%d %d", &n, &sum);
    for (int i = 0; i < n; ++i)
        scanf("%d", &arr[i]);

    memset(dp, 0, sizeof dp);

    for (int i = 0; i < n; ++i) {
        for (int j = sum; j >= 0; --j)
            if (arr[i] + j <= sum)
                dp[arr[i]+j] += dp[j];

        if (arr[i] >= 0 && arr[i] <= sum)
            dp[arr[i]]++;
    }
    printf("%lld\n", dp[sum]);
    return 0;
}
```

## 9.3 随机化

### 9.3.1 模拟退火

#### 9.3.1.1 模拟退火求函数值.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：模拟退火求函数值.cpp
 *   创建日期：2021年03月07日 ---- 16时06分
 *   题    目：hdu1899
 *   算    法：模拟退火
 *   描    述：函数F(x) = 6x^7 + 8x^6 + 7x^3 + 5x^2 - yx;
 *   x 的范围是0 <= x <= 100
 *   输入y值，输出F(x)最小值
 *
----------------------------------------------------------------*/

#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
const double eps = 1e-8;    //终止温度
double y;
double func(double x) {
    return 6*pow(x, 7.0) + 8*pow(x, 3.0) + 5*pow(x, 2.0) - y*x;
}

double solve() {
    double T = 100;         //初始温度
    double delta = 0.98;    //降温系数
    double x = 10.0;        //x初始值
    double now = func(x);   //计算初始函数值
    double ans = now;       //返回值
    while (T > eps) {       //eps是终止温度
        int f[2] = {1, -1};
        double newx = x + f[rand()%2] * T;  //按概率改变x，随T的降温而减少
        if (newx >= 0 && newx <= 100) {
            double nxt = func(newx);
            ans = min(ans, nxt);
            if (now - nxt > eps)   //更新x
                x = newx, now = nxt;
        }
        T *= delta;
    }
    return ans;
}

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

```
    while (cas--) {
        scanf("%lf", &y);
        printf("%.4f\n", solve());
    }
    return 0;
}
```

### 9.3.2 mt19937.cpp

```
#include <cstdio>
#include <ctime>
#include <random>
using namespace std;
int main() {
    //std::mt19937 myrand(seed) ，seed可不填，不填seed则会使用默认随机种子
    mt19937 myrand(time(0));
    printf("%ld\n", myrand());
    return 0;
}
```

### 9.3.3 shuffle.cpp

```
#include <cstdio>
#include <random>
#include <algorithm>
using namespace std;
int arr[100];

int main() {
    mt19937 myrand(time(0));        //默认随机种子是rand()，这里使用了time(0)
    int n = myrand() % 10 + 10;     //n是(10-19)之间的随机数

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

    shuffle(arr+3, arr+9, myrand);  //将数组(arr[3] - arr[9])之间的数随机打乱
    for (int i = 0; i < n; ++i) {
        if (i == 3 || i == 9)
            printf("| ");
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

### 9.3.4 随机字符串.cpp

```
#include <cstdio>
#include <time.h>
#include <stdlib.h>
using namespace std;
int main() {
    srand(time(0));                         //产生随机化种子
    int n=rand()%15+1;                      //在1-15的范围内随机产生字符串个数
    printf("%d",n);
    while(n--) {                            //依次产生n个字符串
        printf("\n");
        int k=rand()%50+1;                  //随机生成一个字符串的长度
        for(int i=1;i<=k;i++) {
            int x,s;                        //x表示这个字符的ascii码 ，s表示这个字符的大小写
            s=rand()%2;                     //随机使s为1或0，为1就是大写，为0就是小写
            if(s==1)                        //如果s=1
                x=rand()%('Z'-'A'+1)+'A';   //将x赋为大写字母的ascii码 //注意加一
            else
                x=rand()%('z'-'a'+1)+'a';   //如果s=0，x赋为小写字母的ascii码
            printf("%c",x);                 //将x转换为字符输出
```

```
        }
    }
    return 0;
}
```

### 9.3.5 随机数.cpp

```
#include <cstdio>
#include <cstdlib> //srand() rand()
#include <ctime>   //time() time是C语言获取当前系统时间的函数，以秒作单位，代表当前时间自Unix标准时间戳(1970年1月1日0点0分0秒，GMT)经过了多少秒。
#define MAXNUM 20000   //MAXNUM/MINNUM是随机数范围
#define MINNUM 10000.0 //MINNUM是随机数保留到小数点后多少位
using namespace std;
int main() {
    srand(time(0)) ; //可以srand(10000*time(0)) 否则直接输出的随机数接近，
    for(int i=0;i<10;i++)
        printf("%f\n", (rand()%MAXNUM)/MINNUM);
    return 0;
}

/*
 * 编译运行后产生十个随机数，然后再次编辑运行产生的数还是这十个，如要改变，需使用srand()函数
 * 产生的随机数范围是0~RAND_MAX，即0~2147483647 ，RAND_MAX定义在<cstdlib>中
 * # define    NUMMOD  20000
 * # define    NUMDEV  10000.0
 * cout<<(rand()%NUMMOD)/NUMDEV<<endl;
 * NUMMOD 决定生成随机数的值的范围，NUMDEV 决定取余后生成的小数的范围
 */
```

### 9.3.6 随机选择算法.cpp

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <cmath>
using namespace std;
const int N = 100010;
int num[N];

//随机核心
int randPartition(int num[], int left, int right) {
    int random = (int)(round(1.0 * rand() / RAND_MAX * (right - left) + left));
    swap(num[random], num[left]);
    int temp = num[left];

    while (left < right) {
        while (left < right && num[right] > temp)
            right--;

        num[left] = num[right];

        while (left < right && num[left] <= temp)
            left++;

        num[right] = num[left];
    }
    num[left] = temp;
    return left;
}

//找寻第k大的数
int randSelect(int num[], int left, int right, int k) {
    if (left == right)
        return num[left];

```

```
    int p = randPartition(num, left, right);
    int m = p - left + 1;

    if (k == m)
        return num[p];
    else
        return randSelect(num, p + 1, right, k - m);
}

int main() {
    int num[] = {5, 8, 2, 7, 3, 6, 0, 1, 9, 4};
    printf("%d\n", randSelect(num, 0, 9, 6));
    return 0;
}
```

## 9.4 悬线法

### 9.4.1 直方图中最大矩形.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：02-悬线法.cpp
 *   创建日期：2021年08月02日 星期一 15时29分53秒
 *   题    目：AcWing 0131 直方图中最大矩形
 *   算    法：悬线法
 *   描    述：使用单调栈也可以做，但是悬线法更简单
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using std::max;
const int maxn = 1e5 + 5;
int h[maxn];
// l[i]：表示i位置的悬线能达到的最左边的位置
// r[i]：表示i位置的悬线能达到的最右边的位置
int l[maxn], r[maxn];
long long res;

int main() {
    int n;
    while (scanf("%d", &n) && n) {
        res = 0;
        for (int i = 0; i < n; i++) {
            scanf("%d", &h[i]);
            l[i] = r[i] = i;
        }

        // 得到这根线向左能达到的位置
        for (int i = 0; i < n; i++)
            // 首先不能越界，而且要比我高这根线才能继续向左前进
            while (l[i] > 0 && h[i] <= h[l[i] - 1])
                // 比我高的线都能通过，我也能
                l[i] = l[l[i] - 1];

        // 得到这根线向右能达到的位置
        for (int i = n-1; i >= 0; i--)
            while (r[i] < n-1 && h[i] <= h[r[i] + 1])
                r[i] = r[r[i] + 1];

        // 计算所有矩形面积
        for (int i = 0; i < n; i++)
            res = max(res, (long long)(r[i] - l[i] + 1) * h[i]);
        printf("%lld\n", res);
    }
    return 0;
}
```

## 9.5 约瑟夫环

### 9.5.1 约瑟夫环.cpp

```cpp
#include <cstdio>

int Joseph(int n,int m) {
    int J = 0;
    for(int i = 2; i <= n; i++)
        J = (J + m) % i;
    /*树组中的下标*/
    return J;
}

int main() {
    char car[12] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'};
    printf("%c\n", car[Joseph(11, 3)]);
    return 0;
}
```

# 10 计算几何

## 10.1 模板

### 10.1.1 template.cpp

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <iostream>
using namespace std;
const double pi = acos(-1.0); //高精度圆周率
const double eps = 1e-8;      //偏差值
const int maxp = 1010;        //点的数量
int sgn(double x) {           //判断x是否等于0
    if (fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}

//比较两个浮点数: 0 相等; -1 小于; 1 大于
int dcmp(double x, double y) {
    if (fabs(x - y) < eps) return 0;
    else return x < y ? -1 : 1;
}

//---------------平面几何: 点和线--------
/*定义点和基本运算*/
struct Point {
    double x, y;
    Point() {}
    Point(double x, double y) : x(x), y(y) {}
    Point operator + (Point B)  {return Point(x+B.x, y+B.y);}
    Point operator - (Point B)  {return Point(x-B.x, y-B.y);}
    Point operator * (double k) {return Point(x*k, y*k);}    //长度增大k倍
    Point operator / (double k) {return Point(x/k, y/k);}    //长度缩小k倍
    bool  operator == (Point B) {return sgn(x - B.x) == 0 && sgn(y - B.y) == 0;}
    bool  operator < (Point B)  {return sgn(x - B.x) < 0 || (sgn(x - B.x) == 0 && sgn(y - B.y) < 0);} //用于凸包
};
typedef Point Vector;   //定义向量
double Dot(Vector A, Vector B) {return A.x*B.x + A.y*B.y;} //点积
double Len(Vector A) {return sqrt(Dot(A, A));}             //向量的长度
double Len2(Vector A) {return Dot(A, A);}                  //向量长度的平方
double Angle(Vector A, Vector B) {return acos(Dot(A,B)/Len(A)/Len(B));}  //A与B的夹角
double Cross(Vector A, Vector B) {return A.x*B.y - A.y*B.x;}             //叉积
double Area2(Point A, Point B, Point C) {return Cross(B-A, C-A);}        //三角形ABC面积的2倍
double Distance(Point A, Point B) {return hypot(A.x-B.x, A.y-B.y);}      //两点的距离
double Dist(Point A, Point B) {return sqrt((A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y));}
Vector Normal(Vector A) {return Vector(-A.y/Len(A), A.x/Len(A));}        //向量A的单位法向量
```

```

/*向量平行或重合*/
bool Parallel(Vector A, Vector B) {return sgn(Cross(A, B)) == 0;}

/*向量A逆时针旋转rad度*/
Vector Rotate(Vector A, double rad) {
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}

struct Line {
    Point p1, p2;//线上的两个点
    Line() {}
    Line(Point p1, Point p2) : p1(p1), p2(p2) {}
    // Line(Point x,Point y) {p1 = x;p2 = y;}
    // Point(double x,double y):x(x),y(y) {}
    // 根据一个点和倾斜角 angle 确定直线,0<=angle<pi
    Line(Point p, double angle) {
        p1 = p;
        if (sgn(angle - pi/2) == 0) {p2 = (p1 + Point(0,1));}
        else {p2 = (p1 + Point(1, tan(angle)));}
    }
    // ax+by+c=0
    Line(double a, double b, double c) {
        if (sgn(a) == 0) {
            p1 = Point(0, -c/b);
            p2 = Point(1, -c/b);
        }
        else if (sgn(b) == 0) {
            p1 = Point(-c/a, 0);
            p2 = Point(-c/a, 1);
        }
        else{
            p1 = Point(0, -c/b);
            p2 = Point(1, (-c-a)/b);
        }
    }
};

typedef Line Segment;   //定义线段，两端点是Point p1,p2

/*返回直线倾斜角 0 <= angle < pi*/
double Line_angle(Line v) {
    double k = atan2(v.p2.y-v.p1.y, v.p2.x-v.p1.x);
    if (sgn(k) < 0) k += pi;
    if (sgn(k-pi) == 0) k -= pi;
    return k;
}

/*点和直线关系:1 在左侧;2 在右侧;0 在直线上*/
int Point_line_relation(Point p, Line v) {
    int c = sgn(Cross(p-v.p1, v.p2-v.p1));
    if (c < 0) return 1;     //1：p在v的左边
    if (c > 0) return 2;     //2：p在v的右边
    return 0;                //0：p在v上
}
/*点和线段的关系：0 点p不在线段v上；1 点p在线段v上*/
bool Point_on_seg(Point p, Line v) {
    return sgn(Cross(p-v.p1, v.p2-v.p1)) == 0 && sgn(Dot(p - v.p1, p- v.p2)) <= 0;
}

/*两直线关系:0 平行,1 重合,2 相交*/
int Line_relation(Line v1, Line v2) {
    if (sgn(Cross(v1.p2-v1.p1, v2.p2-v2.p1)) == 0) {
        if (Point_line_relation(v1.p1, v2) == 0)  return 1; //1 重合
        else return 0;  //0 平行
    }
    return 2;           //2 相交
}

/*点到直线的距离*/
```

```
double Dis_point_line(Point p, Line v) {
    return fabs(Cross(p-v.p1, v.p2-v.p1))/Distance(v.p1, v.p2);
}

/*点在直线上的投影*/
Point Point_line_proj(Point p, Line v) {
    double k = Dot(v.p2-v.p1, p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
}

/*点p对直线v的对称点*/
Point Point_line_symmetry(Point p, Line v) {
    Point q = Point_line_proj(p, v);
    return Point(2*q.x-p.x, 2*q.y-p.y);
}

/*点到线段的距离*/
double Dis_point_seg(Point p, Segment v) {
    if (sgn(Dot(p- v.p1, v.p2-v.p1)) < 0 || sgn(Dot(p- v.p2,v.p1-v.p2)) < 0) //点的投影不在线段上
        return min(Distance(p, v.p1), Distance(p, v.p2));
    return Dis_point_line(p, v); //点的投影在线段上
}

/*求两直线ab和cd的交点*/
/*调用前要保证两直线不平行或重合*/
Point Cross_point(Point a, Point b, Point c, Point d) { //Line1:ab,  Line2:cd
    double s1 = Cross(b-a, c-a);
    double s2 = Cross(b-a, d-a);  //叉积有正负
    return Point(c.x*s2-d.x*s1, c.y*s2-d.y*s1)/(s2-s1);
}
/*两线段是否相交：1 相交，0不相交*/
bool Cross_segment(Point a, Point b, Point c, Point d) { //Line1:ab,  Line2:cd
    double c1 = Cross(b-a, c-a), c2 = Cross(b-a, d-a);
    double d1 = Cross(d-c, a-c), d2 = Cross(d-c, b-c);
    return sgn(c1)*sgn(c2) < 0 && sgn(d1)*sgn(d2) < 0; //注意交点是端点的情况不算在内
}

//---------------平面几何：多边形----------------
struct Polygon {
    int n;   //多边形的顶点数
    Point p[maxp];  //多边形的点
    Line v[maxp];   //多边形的边
};

/*判断点和任意多边形的关系：3 点上；2 边上；1 内部；0 外部*/
int Point_in_polygon(Point pt, Point *p, int n) { //点pt，多边形Point *p
    for (int i = 0; i < n; ++i)  //点在多边形的顶点上
        if (p[i] == pt)
            return 3;

    for (int i = 0; i < n; ++i) {//点在多边形的边上
        Line v = Line(p[i], p[(i+1)%n]);
        if (Point_on_seg(pt, v))
            return 2;
    }
    int num = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i+1) % n;
        int c = sgn(Cross(pt-p[j], p[i]-p[j]));
        int u = sgn(p[i].y - pt.y);
        int v = sgn(p[j].y - pt.y);
        if (c > 0 && u < 0 && v >= 0) ++num;
        if (c < 0 && u >= 0 && v < 0) --num;
    }
    return num != 0; //1 内部；0 外部
}

/*Point *p表示多边形。从原点开始划分三角形*/
double Polygon_area(Point *p, int n) {
    double area = 0;
```

```
    for (int i = 0; i < n; ++i)
        area += Cross(p[i], p[(i+1)%n]);
    return area/2;    //面积有正负，不能简单地取绝对值
}

/*求多边形重心。Point *p表示多边形*/
Point Polygon_center(Point *p, int n) {
    Point ans(0, 0);
    if (Polygon_area(p, n) == 0)
        return ans;
    for (int i = 0; i < n; ++i)
        ans = ans + (p[i]+p[(i+1)%n]) * Cross(p[i], p[(i+1)%n]); //面积有正负
    return ans/Polygon_area(p, n)/6.;
}

/*Convex_hull()求凸包。凸包顶点放在ch中，返回值是凸包的顶点数*/
int Convex_hull(Point *p, int n, Point *ch) {
    sort(p, p+n);         //对点排序：按x从小到大排序，如果x相同，按y排序
    n = unique(p, p+n)-p; //去除重复点
    int v = 0;
    //求下凸包。如果p[i]是右拐弯的，这个点不在凸包上，往回退
    for (int i = 0; i < n; ++i) {
        while (v > 1 && sgn(Cross(ch[v-1]-ch[v-2], p[i]-ch[v-2])) <= 0)
            --v;
        ch[v++] = p[i];
    }
    int j = v;
    //求上凸包
    for (int i = n-2;i >= 0;i--) {
        while (v > j && sgn(Cross(ch[v-1]-ch[v-2], p[i]-ch[v-2])) <= 0)
            --v;
        ch[v++] = p[i];
    }
    if (n > 1) --v;
    return v;   //返回值v是凸包的顶点数
}

//---------------平面几何：圆----------------
struct Circle {
    Point c;//圆心
    double r;//半径
    Circle() {}
    Circle(Point c, double r) : c(c), r(r) {}
    Circle(double x, double y, double _r) {c = Point(x, y) ; r = _r;}
};

/*点和圆的关系：0 点在圆内，1 圆上，2 圆外*/
int Point_circle_relation(Point p, Circle C) {
    double dst = Distance(p, C.c);
    if (sgn(dst - C.r) < 0) return 0;  //点在圆内
    if (sgn(dst - C.r) ==0) return 1;   //圆上
    return 2;   //圆外
}

/*直线和圆的关系：0 直线在圆内，1 直线和圆相切，2 直线在圆外*/
int Line_circle_relation(Line v, Circle C) {
    double dst = Dis_point_line(C.c, v);
    if (sgn(dst-C.r) < 0) return 0; //直线在圆内
    if (sgn(dst-C.r) ==0) return 1;  //直线和圆相切
    return 2; //直线在圆外
}

/*线段和圆的关系：0 线段在圆内，1 线段和圆相切，2 线段在圆外*/
int Seg_circle_relation(Segment v, Circle C) {
    double dst = Dis_point_seg(C.c, v);
    if (sgn(dst-C.r) < 0) return 0; //线段在圆内
    if (sgn(dst-C.r) ==0) return 1;  //线段和圆相切
    return 2;    //线段在圆外
}
```

```
/*直线和圆的交点   hdu 5572*/
int Line_cross_circle(Line v, Circle C, Point &pa, Point &pb) {//pa, pb是交点。返回值是交点个数
    if (Line_circle_relation(v, C) == 2) return 0;//无交点
    Point q = Point_line_proj(C.c, v);  //圆心在直线上的投影点
    double d = Dis_point_line(C.c, v);   //圆心到直线的距离
    double k = sqrt(C.r*C.r-d*d);   //
    if (sgn(k) == 0) {   //1个交点，直线和圆相切
        pa = q;
        pb = q;
        return 1;
    }
    Point n=(v.p2-v.p1)/ Len(v.p2-v.p1);  //单位向量
    pa = q + n*k;
    pb = q - n*k;
    return 2;//2个交点
}

//-------------------三维几何----------------
/*三维：点*/
struct Point3 {
    double x, y, z;
    Point3() {}
    Point3(double x, double y, double z) : x(x), y(y), z(z) {}
    Point3 operator + (Point3 B) {return Point3(x+B.x, y+B.y, z+B.z);}
    Point3 operator - (Point3 B) {return Point3(x-B.x, y-B.y, z-B.z);}
    Point3 operator * (double k) {return Point3(x*k, y*k, z*k);}
    Point3 operator / (double k) {return Point3(x/k, y/k, z/k);}
    bool operator == (Point3 B) {return sgn(x-B.x)==0 && sgn(y-B.y)==0 && sgn(z-B.z)==0;}
};
typedef Point3 Vector3;
/*点积。和二维点积函数同名。C++允许函数同名*/
double Dot(Vector3 A, Vector3 B) {return A.x*B.x+A.y*B.y+A.z*B.z;}
/*叉积*/
Vector3 Cross(Vector3 A, Vector3 B) {return Point3(A.y*B.z-A.z*B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x);}
double Len(Vector3 A) {return sqrt(Dot(A, A));}   //向量的长度
double Len2(Vector3 A) {return Dot(A, A);}        //向量长度的平方
double Distance(Point3 A, Point3 B) {
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y)+(A.z-B.z)*(A.z-B.z));
}
double Angle(Vector3 A, Vector3 B) {return acos(Dot(A, B)/Len(A)/Len(B));}   //A与B的夹角
/*三维：线*/
struct Line3 {
    Point3 p1, p2;
    Line3() {}
    Line3(Point3 p1, Point3 p2) : p1(p1), p2(p2) {}
};
typedef Line3 Segment3;   //定义线段，两端点是Point p1,p2

/*三角形面积的2倍*/
double Area2(Point3 A, Point3 B, Point3 C) {return Len(Cross(B-A, C-A));}

/*三维：点到直线距离*/
double Dis_point_line(Point3 p, Line3 v) {
    return Len(Cross(v.p2-v.p1, p-v.p1))/Distance(v.p1, v.p2);
}

/*三维：点在直线上*/
bool Point_line_relation(Point3 p, Line3 v) {
    return sgn( Len(Cross(v.p1-p, v.p2-p))) == 0 && sgn(Dot(v.p1-p, v.p2-p)) == 0;
}
/*三维：点到线段距离*/
double Dis_point_seg(Point3 p, Segment3 v) {
    if (sgn(Dot(p- v.p1, v.p2-v.p1)) < 0 || sgn(Dot(p- v.p2, v.p1-v.p2)) < 0)
        return min(Distance(p, v.p1), Distance(p, v.p2));
    return Dis_point_line(p, v);
}
/*三维：点 p 在直线上的投影*/
Point3 Point_line_proj(Point3 p, Line3 v) {
    double k = Dot(v.p2-v.p1, p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
```

```
}
/*三维：平面*/
struct Plane {
    Point3 p1, p2, p3;//平面上的三个点
    Plane() {}
    Plane(Point3 p1, Point3 p2, Point3 p3) : p1(p1), p2(p2), p3(p3) {}
};
/*平面法向量*/
Point3 Pvec(Point3 A, Point3 B, Point3 C) {return Cross(B-A,C-A);}
Point3 Pvec(Plane f) {return Cross(f.p2-f.p1, f.p3-f.p1);}
/*四点共平面*/
bool Point_on_plane(Point3 A, Point3 B, Point3 C, Point3 D) {
    return sgn(Dot(Pvec(A, B, C), D-A)) == 0;
}
/*两平面平行*/
int Parallel(Plane f1, Plane f2) {
    return Len(Cross(Pvec(f1), Pvec(f2))) < eps;
}
/*两平面垂直*/
int Vertical (Plane f1, Plane f2) {
    return sgn(Dot(Pvec(f1), Pvec(f2)))==0;
}
/*直线与平面的交点p，返回值是交点个数*/
int Line_cross_plane(Line3 u, Plane f, Point3 &p) {
    Point3 v = Pvec(f);
    double x = Dot(v, u.p2-f.p1);
    double y = Dot(v, u.p1-f.p1);
    double d = x-y;
    if (sgn(x) == 0 && sgn(y) == 0) return -1;//-1：v在f上
    if (sgn(d) == 0) return 0;                  //0：v与f平行
    p = ((u.p1*x) - (u.p2*y))/d;              //v与f相交
    return 1;
}

/*四面体有向体积*6*/
double volume4(Point3 A, Point3 B, Point3 C, Point3 D) {return Dot(Cross(B-A, C-A), D-A);}


int main() {
    Point a(0, 1), b(0, 0), c(1, 1), d(1, 2), p(1.5, 1);
    Line k(a, b), k2(c, d);
    Point pr(1, 1), cr(1, 1); double r = 1;
    Circle C(cr, r);

    cout << endl << "Line_circle_relation=" << Line_circle_relation(k, C);
    cout << endl << "Seg_circle_relation=" << Seg_circle_relation(k, C);
    cout << endl << "Point_circle_relation=" << Point_circle_relation(pr, C);
    cout << endl << "parallel=" << Parallel(a, b) << endl;
    cout << "dot=" << Dot(a, b) << endl << " angle=" << Angle(a, b) << endl;
    cout << "90:" << sgn(Rotate(a, -pi/2).x) << endl << Rotate(a, -pi/2).y;
    cout << endl << "line angle=" << Line_angle(k)*4;
    cout << endl << "line place=" << Point_line_relation(p, k);
    cout << endl << "point_on_seg=" << Point_on_seg(p, k);
    cout << endl << "dis_point_line=" << Dis_point_line(p, k);
    cout << endl << "dis_point_line=" << Dis_point_seg(p, k);
    Point pp = Cross_point(a, b, c, d);
    cout << endl << "crosspoint=" << pp.x << "  " << pp.y;
    cout << endl << "cross seg=" << Cross_segment(a, b, c, d);
    cout << endl << "distance=" << Distance(a, b);
    cout << endl << "line_relation=" << Line_relation(k, k2);
    Point g[4];
    g[0] = a; g[1] = b; g[2] = c; g[3] = d;
    cout << endl << "Point_in_polygon=" << Point_in_polygon(p, g, 4);
    cout << endl << "Polygon_area=" << Polygon_area(g, 4);
    cout << endl << endl;
    return 0;
}
```