南昌大学 **ACM** 校队模板

2021 年 9 月 25 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;
#define bug printf("<-->\n");
#define lowbit(x) ((x) & -(x)) //lowbit(ob0010) = 2
#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 int long long
// const int maxp = 1010;    //点的数量
// const int maxn = <++>;
const int INF    = 0x3f3f3f3f;
const ll  INF_LL = 0x3f3f3f3f3f3f3f3fLL;
// const int dir[][2]={{0, 1}, {1, 0}, {0, -1}, {-1, 0}, {1, 1}, {1, -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...
priority_queue<int, vector<int>, less<int>> pqu_int; /*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...));
}
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

```
#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 重载运算符

1.2.1 重载运算符.cpp

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

```cpp
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;
}
```

1.3 快读

1.3.1 普通读取.cpp

```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;
}
```

1.3.2 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;
}
```

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

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

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

//对于第x个数做出选择
void calc(int x) {
    if (x == n + 1) {
        for (int i = 0; i < chosen.size(); ++i)
            printf("%d ", chosen[i]);
        puts("");
        return ;
    }
    calc(x + 1); //没有选择x的分支
    chosen.push_back(x); //选择x
    calc(x + 1); //选择x的分支
    chosen.pop_back(); //回溯
}

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

2.1.1.2 递归-哈希.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>
#define NEXTLINE puts("");
const int maxn = 20;
int n;
bool st[maxn];

void dfs(int u) {
    if (u > n) {
        for (int i = 1; i <= n; ++i)
            if (st[i])
                printf("%d ", i);
        NEXTLINE;
        return ;
    }
    st[u] = true;
    dfs(u + 1);

    st[u] = false;
    dfs(u + 1);
}

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

2.1.1.3 递归-状压.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
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#define NEXTLINE puts("");
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);
        NEXTLINE;
        return ;
    }
    dfs(u + 1, state);
    dfs(u + 1, state | 1 << u);
}

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-for 循环.cpp

```cpp
#include <cstdio>
const int maxn = 30;
#define NEXTLINE puts("");
int n, m;
```

```
int path[maxn];

//u表示层数，start表示从第几个数开始搜
void dfs(int u, int start) {
    if (u > m) {
        for (int i = 1; i <= m; ++i)
            printf("%d ", path[i]);
        NEXTLINE;
    }
    else {
        for (int i = start; i <= n; ++i) {
            path[u] = i;
            dfs(u + 1, i + 1);
            // path[u] = 0;
        }
    }
}

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

2.1.2.2 递归 + 状压.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：使用递归
 *
 ----------------------------------------------------------------*/

#include <cstdio>
using namespace std;
#define bug printf("<++>\n");
#define NEXTLINE puts("");
int n, m;

void dfs(int u, int cnt, int state) {
    // if (cnt + (n - u) < m || cnt > m)  //之所以不需要cnt > m这句剪枝，是因为当cnt == m时会return，所以不可能到达cnt == 4
    if (cnt + n - u < m)
        return ;
    if (cnt == m) {
        for (int i = 0; i < n; ++i)
            if (state >> i & 1)
                printf("%d ", i + 1);
        NEXTLINE;
        return ;
    }

    dfs(u + 1, cnt + 1, state | 1 << u);
    dfs(u + 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;
#define bug printf("<++>\n");
#define NEXTLINE puts("");
int n, m;
struct State {
    int pos; //记录递归运行的位置，递归的关键
    int u, cnt, state;
};

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

    DFS(u + 1, cnt + 1, state | 1 << u);
    // 1:
    DFS(u + 1, cnt, state);
    // 2:
}

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

```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：<++>
 *   算    法：递归
 *   描    述：可以使用状态压缩
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 11;
int n, site[maxn], haxh[maxn];


void DFS(int idx) {
    if (idx > n) {
        for (int i = 1; i <= n; ++i)
            printf("%d", site[i]);
        NEXTLINE;
        return ;
    }
    // 这里for中的i是具体的数，数1，数2，数3，是这三个数全排列
    for (int i = 1; i <= n; ++i) {
        if (!haxh[i]) {
            site[idx] = i;    //存储全排列
            haxh[i] = true;   //深度优先搜索的两个分支
            //每进一层，idx++
            //递归的深度，每一层深度里填入对应的site[idx]
            DFS(idx + 1);     //深度优先搜索
            haxh[i] = false;  //深度优先搜索的另一个分支
        }
        //下面这个右括号，会阻隔部分返回上一层的idx
    }

}

int main() {
    n = 3;
    DFS(1);
    return 0;
}
```

2.1.3.2 swap.cpp

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
int cnt;
void Perm(int* arr, int size, int n) {
    if (n == size) {
        for(int i = 0; i < size; ++i)
            printf("%d", arr[i]);
        printf("\n");
        ++cnt;
    }
    else {
        for(int i = n; i < size; ++i) {
            swap(arr[i], arr[n]);
            Perm(arr, size, n+1);
            swap(arr[i], arr[n]);
        }
    }
}

```

```
int main() {
    int arr[5] = {1,2,3,4,5};
    Perm(arr, 5, 0);
    printf("cnt = %d\n", cnt);
    return 0;
}
```

2.1.3.3 递归全排列 + 状压.cpp

```
#include <cstdio>
#include <vector>
using namespace std;
#define NEXTLINE puts("");
vector<int> path;
int n;

void DFS(int u, int state) {
    if (u == n) {
        for (auto _ : path)
            printf("%d ", _);
        NEXTLINE;
        return ;
    }
    for (int i = 0; i < n; ++i)
        if (!(state >> i & 1)) {
            path.push_back(i + 1);
            DFS(u + 1, state | 1 << i);
            path.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 yxc.cpp

```
// 舍不得丢，用下面lhr的代码
#include <cstdio>
#include <vector>
#include <algorithm>
#include <utility>
using namespace std;
typedef pair<int, int> PII;
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) {
    sort(segs.begin(), segs.end());
    vector<PII> res;
    int st = -2e9, ed = -2e9;
    for (auto seg : segs) {
        if (ed < seg.first) {
            if (st != -2e9)
                res.push_back({st, ed});
            st = seg.first;
            ed = seg.second;
        }
        else
            ed = max(ed, seg.second);
    }
```

```

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

int main() {
    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());
    return 0;
}
```

2.2.1.2 lhr.cpp

```
#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 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>
```

```cpp
using namespace std;
const int maxn = 1e6 + 5;
typedef long long ll;
int n;
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;
}
```

2.2.3.2 七夕祭.cpp

```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;
}
```

2.3 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: <++>
 *
 ----------------------------------------------------------------*/

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

void insertSort() {
    int insertNum;

    for (int i = 0; i < n; i++) {
        insertNum = num[i];
        int j;
        for (j = i; insertNum < num[j - 1]; j--) {
            num[j] = num[j - 1];
        }
        num[j] = insertNum;
    }
}

int main() {
    insertSort();
    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

```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

```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

```
#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

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

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

```
#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>
```

```cpp
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;
}
```

2.5 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：01-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;
}
```

3.2 OLD

3.2.1 八数码问题与状态图搜索

3.2.1.1 BFS+Cantor.cpp

```
#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>
using namespace std;
const int maxn = 362880;  //状态共9! = 362880种
struct node {
    int state[9];
    int dis;
};
int dir[4][2] = {{-1, 0}, {0, -1}, {1, 0}, {0, 1}};
int used[maxn];
int start[9];
int goal[9];
int fact[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};

bool cantor(int str[], int n) {
    int res = 0;
    for (int i = 0; i < n; ++i) {
        int counted = 0;
        for (int j = i+1; j < n; ++j)
            if (str[i] > str[j])
                ++counted;
        res += counted * fact[n-1-1];
    }

    if (!used[res]) {
        used[res] = 1;
        return 1;
    }
    return 0;
}
```

```

int BFS() {
    node head;
    memcpy(head.state, start, sizeof(head.state));
    head.dis = 0;
    queue<node> quu;
    cantor(head.state, 9);
    quu.push(head);
    while (!quu.empty()) {
        head = quu.front();
        if (memcmp(head.state, goal, sizeof(goal)) == 0)
            return head.dis;
        quu.pop();
        int coord;
        for (coord = 0; coord < 9; ++coord)
            if (head.state[coord] == 0)
                break;
        int x = coord % 3;
        int y = coord / 3;
        for (int i = 0; i < 4; ++i) {
            int newx = x + dir[i][0];
            int newy = y + dir[i][1];
            int ncoord = newx + 3*newy;
            if (newx >= 0 && newx < 3 && newy >= 0 && newy < 3) {
                node newnode;
                memcpy(&newnode, & head, sizeof(node));
                swap(newnode.state[coord], newnode.state[ncoord]);
                ++newnode.dis;
                if (cantor(newnode.state, 9))
                    quu.push(newnode);
            }
        }
    }
    return -1;
}

int main() {
    for (int i = 0; i < 9; ++i)
        scanf("%d", &start[i]);
    for (int i = 0; i < 9; ++i)
        scanf("%d", &goal[i]);
    int num = BFS();
    num != -1 ? printf("%d\n", num) : printf("Impossible");
    return 0;
}
```

3.2.2 子集生成与组合问题

3.2.2.1 子集与二进制关系.cpp

```
#include <cstdio>
void print_subset(int n) {
    // 0~2^n, 每个i的二进制数代表一个子集，比如3的二进制为101，子集是{0, 2}
    for (int i = 0; i < (1 << n); ++i) {
        for (int j = 0; j < n; ++j)
            if (i & (1 << j))
                printf("%d ", j);
        printf("\n");
    }
}

int main() {
    int n = 3;
    // scanf("%d", &n);
    print_subset(n);
    return 0;
}
```

3.2.2.2 组合与二进制关系.cpp

```cpp
#include <cstdio>
void print_set(int n, int m) {
    for (int i = 0; i < (1 << n); ++i) {
        int num = 0;
        int kk = i;
        while (kk) {
            kk = kk & (kk-1);
            ++num;
        }
        if (num == m) {
            for (int j = 0; j < n; ++j)
                if (i & (1 << j))
                    printf("%d ", j);
            printf("\n");
        }
    }
}

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

3.2.3 树与图的遍历

3.2.3.1 树的深度优先搜索.cpp

```cpp
/*-----------------------------------------------------------------
 *
 *   文件名称：01-树的深度优先搜索.cpp
 *   创建日期：2021年04月14日 ---- 19时53分
 *   题    目：算法竞赛
 *   算    法：树的深度优先搜索，链式前向星
 *   描    述：树的深度优先搜索，树的dfs序，树的深度，树的重心
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
const int inf  = 0x3f3f3f3f;
int n, m;
int vert[maxn], edge[maxn], nxet[maxn], head[maxn], tot;
int cnt, dfs[maxn]; //记录DFS序
bool used[maxn];
int d[maxn]; //记录结点深度
int size[maxn];
int res = inf, pos = -1; //res记录重心对应的max_part值，pos记录了重心

//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;
}

/*
 * 树的深度优先搜索
 *
 * 求树的dfs序
 * 但由于邻接表的倒序搜索
 * 所以dfs序也是倒序的
 *
```

```
 * 求树的深度
 */
void DFS(int x) {
    dfs[cnt++] = x;
    used[x] = true;
    for (int i = head[x]; i; i = nxet[i]) {
        int y = vert[i]; //因为链式前向星是倒序的，所以此处y = 是x的父结点
        if (used[y]) continue;
        d[y] = d[x] + 1;
        DFS(y);
    }
    dfs[cnt++] = x;
}

/*
 *        0
 *       /|\
 *      / | \
 *     1  6 [3]
 *    /|    / \
 *   / |   /   \
 *  7  4  2     5
 *        |
 *        |
 *        8
 */


//树的重心 --- 树的重心与边的权值无关
void gravity(int x) {
    used[x] = true;
    size[x] = 1;
    int max_part = 0;
    for (int i = head[x]; i; i = nxet[i]) { //遍历x的子结点
        int y = vert[i];
        if (used[y]) continue;
        gravity(y);  //这句很巧妙，没有返回值，而是递归结束条件是子结点的size为1
        size[x] += size[y];
        max_part = max(max_part, size[y]);  //比较所有子树的size
    }
    max_part = max(max_part, n-size[x]); //删去结点x后父结点处还有一棵树，size为n-size[x]
    if (max_part < res) {
        res = max_part;
        pos = x;
    }
}

int main() {
    freopen("in/in-01.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);
    }
    DFS(0);
    //DFS深度优先搜索
    for (int x = 0; x < n; ++x)
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            // int z = edge[i];
            printf("%d %d\n", x, y);
        }

    //dfs序
    printf("dfs序: ");
    for (int i = 0; i < 2*n; ++i)
        printf("%d ", dfs[i]);
    printf("\n");

```

```
    //结点深度
    for (int i = 0; i < n; ++i)
        printf("depth[%d] = %d\n", i, d[i]);

    memset(used, 0, sizeof(used));
    gravity(0);
    printf("重心以及重心对应的max_part值: %d %d\n", pos, res);
    return 0;
}
```

3.2.3.2 图的深度优先搜索.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: 图的连通块划分
 *
 -----------------------------------------------------------------*/

#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() {
    freopen("in/in-02.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);
    }
    for (int i = 0; i < n; ++i)
        if (!used[i]) {
            ++cnt;
            DFS(i);
        }
    printf("%d\n", cnt);
    return 0;
}
```

3.2.3.3 树的广度优先搜索.cpp

```
#include <cstdio>
```

```cpp
#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;
int d[maxn]; //对于一张图来讲，d[x]被称为点x的层次(从起点 1 走到点x需要经过的最少点数)

//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() {
    memset(d, -1, sizeof(d));
    queue<int> quu;
    quu.push(0);
    d[0] = 0;
    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 (d[y] != -1) continue;
            d[y] = d[x] + 1;
            quu.push(y);
        }
    }
}

int main() {
    freopen("in-03.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;
}
```

3.2.3.4 图的广度优先搜索.cpp

3.2.4 深度优先搜索

3.2.4.1 算法笔记.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：First.cpp
 *   创建日期：2020年09月09日 ---- 16时35分
 *   题    目：算法笔记
 *             给定N(N < 1e6)个整数(可能有负数)，从中选择K个数，使得这K个数
 *             之和恰好等于一个给定的整数X，如果有多种方案，选择它
 *             们中元素平方和最大的一个，题目保证这样的方案唯一
 *   输    入：N K X
 *             1 2 3 ... N
 *   输    出：i i+1 i+2 i+3 ... i+k
 *   例      ：
 *             输入：4 2 6
 *                   2 3 3 4
```

```
 *             输出: 2 4
 *   算    法: 深度优先搜索
 *   描    述: void DFS(int index, int count, int sum, int sumSquare)
 *
----------------------------------------------------------------*/

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

const int maxn = 1e6;
int N;              // 输入的数个数
int K;              // 选择的数个数
int X;              // 选择的数总和
int count = 0;      // 目前已选择的数个数
int idx = 0;        // 存储下标
int sum = 0;        // 目前已选择的数总和
int sumSquare = 0;  // 目前已选择的数平方和
int maxSumSquare;   // 已知最大平方和
int num[maxn];      // 存储输入的数
vector<int> tem;    // 存储当前K个数
vector<int> ans;    // 存储答案

void DFS(int index, int count, int sum, int sumSquare) {
    if (count == K && sum == X) {
        if (sumSquare > maxSumSquare) {
            maxSumSquare = sumSquare;
            ans = tem;
        }
        return ;
    }
    //if (index == N || count == K || sum == X)
    if (index == N || count > K || sum > X)
        return ;
    tem.push_back(num[index]);
    DFS(index + 1, count + 1, sum + num[index], sumSquare + num[index] * num[index]);
    tem.pop_back();
    DFS(index + 1, count, sum, sumSquare);
}

int main() {
    scanf("%d", &N);
    scanf("%d", &K);
    scanf("%d", &X);
    for (int i = 0; i < N; i++)
        scanf("%d", &num[i]);

    DFS(idx, count, sum, sumSquare);
    for (auto it = ans.begin(); it != ans.end(); it++)
        printf("%d ", *it);

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

3.2.4.2 木棍 dfs 剪枝.cpp

```
// POJ1011Sticks
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 105;
int sticks[maxn];
int vis[maxn];    /*记录每根木棍的访问*/
int n;            /*多组输入*/
int max_len;      /*假设最长木棍长度*/
int max_cnt;      /*最大木棍数量*/

```

```
/*
 * 正在拼接第stick根原始木棍(已经拼好了stick-1根)
 * 第stick根木棍的当前长度为now_len
 * 拼接到第stick根木棍中的上一根小木棍为last
 */
bool DFS(int stick, int now_len, int last) {
    /*所有原始木棍都已拼好，搜索成功*/
    if (stick > max_cnt)
        return true;
    /*第stick根木棍已经拼好，去拼下一根*/
    if (now_len == max_len)
        return DFS(stick+1, 0, 1);

    /*记录尝试向当前原始木棍拼接的最近的失败的木棍长度*/
    int record = 0;
    for (int i = last; i <= n; ++i)
        if (!vis[i] && now_len + sticks[i] <= max_len && record != sticks[i]) {
            vis[i] = 1;
            if (DFS(stick, now_len + sticks[i], i + 1))
                return true;
            record = sticks[i];
            vis[i] = 0; /*还原现场*/
            /*贪心，再用1根木棍恰好拼完当前原始木棍必然比再用若干根木棍拼完更好*/
            if (now_len == 0 || now_len + sticks[i] == max_len)
                return false;
        }
    /*所有分支均尝试过，搜索失败*/
    return false;
}

int main() {
    while (scanf("%d", &n) && n) {
        int sum = 0;
        int val = 0;
        for (int i = 1; i <= n; ++i) {
            scanf("%d", &sticks[i]);
            sum += sticks[i];
            val = max(sticks[i], val);
        }
        sort(sticks+1, sticks+1+n);
        reverse(sticks+1, sticks+1+n);
        for (int max_len = val; max_len <= sum; ++max_len) {
            if (sum % max_len)
                continue;
            max_cnt = sum / max_len;
            memset(vis, 0, sizeof(vis));
            if (DFS(1, 0, 1))
                break;
        }
        printf("%d\n", max_len);
    }
    return 0;
}
```

3.2.4.3 Lake.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：P1596[USACO10OCT]Lake Counting.cpp
 *   创建日期：2020年11月18日 ---- 21时03分
 *   题    目：luogu
 *   算    法：深度优先搜索
 *   描    述：算法课上的代码
 *
 ----------------------------------------------------------------*/

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

```cpp
using namespace std;

int n;
int m;
const int maxn = 101;
char water[maxn][maxn];
int puddle = 0;
vector<int> vec;
int dirx[8] = {-1, 0, 1, -1, 1, -1, 0, 1};
int diry[8] = {-1, -1, -1, 0, 0, 1, 1, 1};
bool vis[maxn][maxn];

bool judge(int x, int y) {
    if (x < 1 || x > n || y < 1 || y > m)
        return false;

    return true;
}

void DFS(int nowX, int nowY, int &size) {
    vis[nowX][nowY] = true;
    for (int i = 0; i < 8; ++i)
        if (water[nowX+dirx[i]][nowY+diry[i]] == 'W' && judge(nowX+dirx[i], nowY+diry[i]) && !vis[nowX+dirx[i]][nowY+diry[i]])
            DFS(nowX+dirx[i], nowY+diry[i], ++size);

    return ;
}

int main() {
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    scanf("%d", &n);
    scanf("%d", &m);
    getchar();
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j)
            scanf("%c", &water[i][j]);
        getchar();
    }
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            if (water[i][j] == 'W' && !vis[i][j]) {
                int size = 0;
                DFS(i, j, ++size);
                ++puddle;
                vec.push_back(size);
            }
    printf("%d ", puddle);
    sort(vec.begin(), vec.end());
    for (auto it = vec.begin(); it != vec.end(); ++it)
        it == vec.end() - 1 ? printf("%d", *it) : printf("%d ", *it);

    /*
     *printf("\n");
     *for (int i = 1; i <= n; ++i) {
     *    for (int j = 1; j <= m; ++j) {
     *        printf("%d", vis[i][j]);
     *    }
     *    printf("\n");
     *}
     */
    return 0;
}
```

3.2.5 广度优先搜索

3.2.5.1 算法笔记.cpp

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

```
 *
 *    文件名称：First.cpp
 *    创建日期：2020年09月10日 ---- 16时25分
 *    题      目：算法笔记
 *                给出一个m*n (m, n < 100)的矩阵，矩阵中的元素为0或1
 *                称位置(x, y)与其上下左右四个位置
 *                (x, y+1) (x, y-1) (x+1, y) (x-1, y)是相邻的
 *                如果矩阵中有若干个1是相邻的(不必两两相邻)，那么称
 *                这些1构成了一个"块"，求给定的矩阵中块的个数
 *    输      入：m  n
 *                1 ... 1
 *                ...
 *                ...
 *                1 ... 1
 *    例        ：
 *                输入：
 *                      6 7
 *                      0 1 1 1 0 0 1
 *                      0 0 1 0 0 0 0
 *                      0 0 0 0 1 0 0
 *                      0 0 0 1 1 1 0
 *                      1 1 1 0 1 0 0
 *                      1 1 1 1 0 0 0
 *                输出：4
 *    算      法：广度优先搜索
 *    描      述：<++>
 *
-------------------------------------------------------------------*/

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

const int maxn = 100;
struct position {
    int x;
    int y;
}pos;
int m;
int n;
int matrix[maxn][maxn];
bool inq[maxn][maxn];
int nextX[] = {0, 0, 1, -1};
int nextY[] = {1, -1, 0, 0};

bool judge(int x, int y) {
    //越界返回false
    if (x >= n || x < 0 || y >= m || y < 0)
        return false;

    //当前位置是0，或(x, y)已入过队，返回false
    if (matrix[x][y] == 0 || inq[x][y] == true)
        return false;

    //以上都不满足
    return true;
}

//BFS访问位置(x, y)所在的块，将该块内的所有1的inq都设置为true
void BFS(int x, int y) {
    queue<position> quu;
    pos.x = x;
    pos.y = y;
    quu.push(pos);
    inq[x][y] = true;
    //不为空
    while (quu.empty() == false) {
        position top = quu.front(); //取出队首元素
        quu.pop(); //队首元素出队
        for (int i = 0; i < 4; i++) {
```

```
            int newX = top.x + nextX[i];
            int newY = top.y + nextY[i];
            if (judge(newX, newY) == true) {
                pos.x = newX;
                pos.y = newY;
                quu.push(pos);
                inq[newX][newY] = true;
            }
        }
    }
}

int main() {
    scanf("%d", &m);
    scanf("%d", &n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &matrix[i][j]);

    int ans = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (matrix[i][j] == 1 && inq[i][j] == false)
                (ans++, BFS(i, j));

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

3.2.5.2 算法笔记.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称: Second.cpp
 *   创建日期: 2020年09月10日 ---- 17时37分
 *   题    目: 算法笔记
 *             给定一个m*n (m, n < 100)大小的迷宫，其中
 *             * 代表不可通过的墙壁
 *             . 代表平地
 *             S 表示起点
 *             T 代表终点
 *             移动过程中，如果当前位置是(x, y)(下标从0开始)，且
 *             每次只能上下左右移动一个位置
 *             (x, y+1) (x, y-1) (x+1, y) (x-1, y)
 *             求从起点S到达终点T的最小步数
 *             如果无法到达，返回-1
 *   输    入: m  n
 *             * * * * * *
 *             S * * * * *
 *             . . . . . .
 *             * * * * * T
 *             * * * * * *
 *   输    出: 7
 *   例      :
 *             输入:
 *                    . . . . .
 *                    . * . * .
 *                    . * S * .
 *                    . * * * .
 *                    . . . T *
 *             输出: 11
 *   算    法: 广度优先搜索
 *   描    述: <++>
 *   错    误: - std::begin
 *             - std::stop
 *
-----------------------------------------------------------------*/

#include <cstdio>
```

```cpp
#include <queue>
using namespace std;
const int maxn = 100;

struct Node {
    int x;
    int y;
    int step;
}start, stop, node;
int m;
int n;
char maze[maxn][maxn];
bool inq[maxn][maxn];
int nextX[4] = {0, 0, 1, -1};
int nextY[4] = {1, -1, 0, 0};

//判断(x, y)位置是否有效
bool judge(int x, int y) {
    if (x >= n || x < 0 || y >= m || y < 0)
        return false;
    if (maze[x][y] == '*')
        return false;
    if (inq[x][y] == true)
        return false;
    return true;
}

int BFS() {
    queue<Node> quu;
    quu.push(start);
    while (quu.empty() == false) {
        Node top = quu.front();
        quu.pop();
        if (top.x == stop.x && top.y == stop.y)
            return top.step;

        for (int i = 0; i < 4; i++) {
            int newX = top.x + nextX[i];
            int newY = top.y + nextY[i];
            if (judge(newX, newY) == true) {
                node.x = newX;
                node.y = newY;
                node.step = top.step + 1;
                quu.push(node);
                inq[newX][newY] = true;
            }
        }
    }
    return -1;
}

int main() {
    scanf("%d", &m);
    scanf("%d", &n);
    for (int i = 0; i < m; i++) {
        getchar();
        for (int j = 0; j < n; j++)
            maze[i][j] = getchar();

        maze[i][m+1] = '\0';
    }
    scanf("%d", &start.x);
    scanf("%d", &start.y);
    scanf("%d", &stop.x);
    scanf("%d", &stop.y);
    start.step = 0;
    printf("%d\n", BFS());
    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

```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

```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;
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

```
/*-----------------------------------------------------------------
 *
 *   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;
}
```

4.3 背包 DP

4.3.1 01 背包

4.3.1.1 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：01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>
```

```cpp
#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;
}
```

4.3.1.2 一维.cpp

```cpp
/*----------------------------------------------------------------
 *
 *   文件名称：02-一维.cpp
 *   创建日期：2021年03月30日 ---- 16时14分
 *   题    目：AcWing 0003 01背包
 *   算    法：动态规划
 *   描    述：使用一维数组
 *
 ----------------------------------------------------------------*/

#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
int dp[maxn];

int solve() {
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= N; ++i)
        for (int j = V; j >= bone[i].vol; --j)
            /*
```

```
             * dp[j-bone[i].vol]此处体积一定是上个物品的最优解
             * 因为还没有dp[j-bone[i].vol] = max(dp[j-bone[i].vol], dp[j-2*bone[i].vol]+2*bone[i].val)
             * 如果j从小到大循环的话，dp[j-bone[i].vol]中是从dp[j-2*bone[i].vol]转移来的
             * 表示在背包体积为j时，可以放入的当前物品数量不定，可以无穷件，只要不超过背包容积
             */
            dp[j] = max(dp[j], dp[j-bone[i].vol] + bone[i].val);
    return dp[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;
}
```

4.3.1.3 AcWing.cpp

```
/*----------------------------------------------------------------
 *
 * ••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
 *
 *    文件名称：01.cpp
 *    创建日期：2021年03月29日 ---- 22时18分
 *    题      目：AcWing 0002 01 背包问题
 *    算      法：动态规划 01背包
 *    描      述：看过了骨头收集者的代码后，联系01背包
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1005;
int N; //N件物品
int V; //背包容量
struct Bone {
    int val;
    int vol;
} bone[maxn];
int dp[maxn][maxn];

int solve() {
    for (int i = 1; i <= N; ++i)
        for (int j = 0; j <= V; ++j) {
            if (bone[i].vol > j)
                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() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++i)
        scanf("%d %d", &bone[i].vol, &bone[i].val);
    printf("%d\n", solve());
    return 0;
}
```

4.3.2 完全背包

4.3.2.1 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
 *   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 = 1005;
struct Bone {
    int val;
    int vol;
} bone[maxn];
int N, V;
int dp[maxn][maxn];

int solve() {
    //这里必须是从1开始，因为从0开始会dp[i-1][j]出界
    for (int i = 1; i <= N; ++i)
        for (int j = 0; j <= V; ++j) {
            if (bone[i].vol > j)
                dp[i][j] =  dp[i-1][j];
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-bone[i].vol] + bone[i].val);
        }
    return dp[N][V];
}

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++i)
        scanf("%d %d", &bone[i].vol, &bone[i].val);
    printf("%d\n", solve());
    return 0;
}
```

4.3.2.2 一维.cpp

```cpp
/*--------------------------------------------------------------------
 *
 *   文件名称：02-一维.cpp
 *   创建日期：2021年03月30日 ---- 16时20分
 *   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 = 1005;
struct Bone {
    int val;
    int vol;
} bone[maxn];
int N, V;
int dp[maxn];

int solve() {
    for (int i = 1; i <= N; ++i)
```

```
        for (int j = 0; j <= V-bone[i].vol; ++j)
            /*
             * dp[j] = max(dp[j], dp[j-bone[i].vol] + bone[i].val);
             * 为什么不使用上面一句呢?
             * 因为不是从0开始的，dp[j-bone[i].vol]的开始位置不能确定
             */
            dp[j+bone[i].vol] = max(dp[j+bone[i].vol], dp[j]+bone[i].val);
    return dp[V];
}

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++i)
        scanf("%d %d", &bone[i].vol, &bone[i].val);
    printf("%d\n", solve());
    return 0;
}
```

4.3.3 多重背包

4.3.3.1 多重背包.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-多重背包.cpp
 *   创建日期：2021年04月10日 ---- 10时39分
 *   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
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 105;
int N, V; //骨头数量，背包体积
int dp[maxn];

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

4.3.3.2 二进制分组优化.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-二进制分组优化.cpp
 *   创建日期：2021年03月30日 ---- 20时05分
 *   题    目：hdu2191 悼念512汶川大地震
 *   算    法：多重背包
 *   描    述：输入数据首先包含一个正整数t，表示有t组测试用例
 *       每组测试用例第一行是两个整数V和N(1 <= V <= 100, 1 <= N <= 100)
 *       分别表示背包的容量和大米的种类，然后是m行数据
 *       每行包含3个数vo，va和a(1 <= vo <= 20, 1 <= va <= 200, 1 <= a <= 20)
 *       分别表示大米的体积、每袋的价值以及对应种类大米的袋数
 *       输出能够购买的大米的最大价值(题意有更改)
 *
 -----------------------------------------------------------------*/
```

```

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
//虽然题目的N <= 100，但二进制分组后会变多，所以需要更多存储空间
const int maxn = 1005;
struct Bone {
    int vol;
    int val;
} bone[maxn];
int N, V;
int dp[maxn];

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        scanf("%d %d", &V, &N);
        int idx = 1; //拆分后物品总数
        for (int i = 1; i <= N; ++i) {
            int vo, va, a; //体积，价值，大米袋数
            scanf("%d %d %d", &vo, &va, &a);
            int b = 1;     //二进制分组优化，1, 2, 4, 8, 16, ...
            while (a - b > 0) {
                a -= b;
                bone[idx].vol = b * vo;
                bone[idx].val = b * va;
                ++idx;
                b *= 2;
            }
            bone[idx].vol = a * vo; //补充不足指数的差值
            bone[idx].val = a * va;
            ++idx;
        }
        memset(dp, 0, sizeof(dp));
        for (int i = 1; i < idx; ++i) //对拆分后的物品进行0-1背包
            for (int j = V; j >= bone[i].vol; --j)
                dp[j] = max(dp[j], dp[j-bone[i].vol] + bone[i].val);
        printf("%d\n", dp[V]);
    }
    return 0;
}
```

4.3.3.3 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
 *   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
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 20005;
int N, V;
int dp[maxn], tmp[maxn];
int quu[maxn]; //队首元素是dp[j], dp[j-v], dp[j-2v], ... dp[j-(k-v)v)里最大的数

int main() {
```

```
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    scanf("%d %d", &N, &V);
    for (int i = 0; i < N; ++i) {
        int vo, va, a; //体积，价值，当前物品数量
        scanf("%d %d %d", &vo, &va, &a);
        memcpy(tmp, dp, sizeof(dp)); //tmp为dp[i-1], dp为dp[i]
        /*
         * 枚举余数即等价类
         * 背包体积，j < vo是因为vo可以由j = 0 + vo得到
         * 再下面一个for循环就是将每个等价类进行单调队列操作
         * 每个等价类有0, 0+1vo, 0+2vo, 0+3vo ...
         *             1, 1+1vo, 1+2vo, 1+3vo ...
         *             2, 2+1vo, 2+2vo, 2+3vo ...
         *             ...
         * 所有等价类中的数就是0 ~ V，也就是背包的第二维
         */
        for (int j = 0; j < vo; ++j) {
            int hh = 0, tt = -1;                //hh为队首 tt为队尾
            for (int k = j; k <= V; k += vo) {  //枚举同一等价类的背包体积
                dp[k] = tmp[k];                 //此句意义不大，确实意义不大
                //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 && quu[hh] < k-a*vo)
                    ++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
                 *
                 *      |------- quu[hh]指向的位置，tmp[quu[hh]存储这里的最优解
                 *      v
                 * ------------------------------------------------------------------------
                 * |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
                 * ------------------------------------------------------------------------
                 * |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
                 * ------------------------------------------------------------------------
                 *      ^                                                       ^
                 *      |---  填充到当前体积需要的物品数(k-quu[hh])/vo*va  ---|
                 */
                if (hh <= tt)
                    dp[k] = max(dp[k], tmp[quu[hh]]+(k-quu[hh])/vo*va);
                /*
                 * tmp[quu[tt]]          放入上一件物品体积在队尾体积处时的最优解
                 * (quu[tt]-j)/vo*va     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优解
                 * (k-j)/vo*va           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[quu[hh]] + (k-quu[h])/vo*va
                 * 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[quu[tt]] - (quu[tt]-j)/vo*va呢
                 *
                 * 前a-1个数在单调队列中的决策
                 * dp[k] = max(dp[k], dp[k-vo]+va, dp[k-2vo]+2va, dp[k-3vo]+3va, ...
                 * 在单调队列中无法实现，转化为
                 * dp[k]-(k-j)/vo*va, dp[k-vo]-(k-vo-j)/vo*va, dp[k-2vo]-(k-2vo-j)/vo*va, ...
                 * 会发现，上下两式各项都相差(k-j)/vo*va，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-quu[hh])/vo*va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[quu[tt]] - (quu[tt]-j)/vo*va <= tmp[k] - (k-j)/vo*va)
                    --tt;
                //无论如何，将当前位置插入队列
                quu[++tt] = k;
            }
        }
    }
```

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

4.3.4 混合背包

4.3.4.1 混合背包.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
 *   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 <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int N, V;//物品数量，背包体积
int v[maxn];//体积
int w[maxn];//价值
int dp[maxn];

int main() {
    scanf("%d %d", &N, &V);
    int idx = 1; //拆分后物品总数
    for (int i = 1; i <= N; ++i) {
        int vo, va, a;
        scanf("%d %d %d", &vo, &va, &a);
        int b = 1; //二进制分组优化，1, 2, 4, 8, 16, ...
        //将混合背包转化为多重背包
        if (a < 0)
            a = 1;
        else if (a == 0) //如果是完全背包，则在最优情况下，只能取总体积/该物品体积向下取整
            a = V / vo;
        //如果是a - b > 0，那么下面就不需要if语句判断，直接执行if语句里面的语句
        while (a - b >= 0) {
            a -= b;
            v[idx] = vo * b;
            w[idx] = va * b;
            ++idx;
            b *= 2;
        }
        if (a > 0) {
            v[idx] = a * vo;
            w[idx] = a * va;
            ++idx;
        }
    }
    for (int i = 1; i <= idx; ++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;
}
```

4.3.5 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
```

```
 *   题    目：AcWing 0008 二维费用问题
 *   算    法：二维费用背包
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#define _max(a, b) (a > b ? a : b)
#define _min(a, b) (a < b ? a : b)
const int maxn = 1e3 + 5;
int N, V, M;//物品数量，背包体积，背包最大承重
int vo[maxn], we[maxn], va[maxn]; //第i件物品体积、重量和价值
int dp[maxn][maxn];

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
    // freopen("out.txt", "w", stdout);
#endif
    scanf("%d %d %d", &N, &V, &M);
    for (int i = 1; i <= N; ++i)
        scanf("%d %d %d", &vo[i], &we[i], &va[i]);
    for (int i = 1; i <= N; ++i)
        for (int j = V; j >= vo[i]; --j)
            for (int k = M; k >= we[i]; --k)
                dp[j][k] = _max(dp[j][k], dp[j-vo[i]][k-we[i]] + va[i]);//动态转移方程，01 背包的思路
    printf("%d\n", dp[V][M]);
    return 0;
}
```

4.3.6 分组背包

4.3.6.1 AcWing.cpp

```
#include <cstdio>
const int maxn=110;
#define _max(a, b) (a > b ? a : b)
int N, V;
int vo[maxn][maxn], va[maxn][maxn], s[maxn];    //v为体积，w为价值，s代表第i组物品的个数
int dp[maxn][maxn];  //只从前i组物品中选，当前体积小于等于j的最大值

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++i) {
        scanf("%d", &s[i]);
        for(int j = 0; j < s[i]; ++j)
            scanf("%d %d", &vo[i][j], &va[i][j]);
    }
    for (int i = 1; i <= N; ++i) {
        for (int j = 0; j <= V; ++j) {
            dp[i][j] = dp[i-1][j]; //不选
            for (int k = 0; k < s[i]; ++k)
                if (j >= vo[i][k])
                    dp[i][j] = _max(dp[i][j], dp[i-1][j-vo[i][k]] + va[i][k]);
        }
    }
    printf("%d\n", dp[N][V]);
}
```

4.3.6.2 一维.cpp

```
#include <cstdio>
const int maxn = 110;
#define _max(a, b) (a > b ? a : b)
int N, V;
int vo[maxn][maxn], va[maxn][maxn], s[maxn];
int dp[maxn];

int main() {
```

```
    scanf("%d %d", &N, &V);
    for (int i = 0; i < N; i++) {
        scanf("%d", &s[i]);
        for(int j = 0; j < s[i]; j++)
            scanf("%d %d", &vo[i][j], &va[i][j]);
    }
    for (int i = 0; i < N; ++i)
        for (int j = V; j >= 0; --j)
            for (int k = 0; k < s[i]; k++)     //for(int k=s[i];k>=1;k--)也可以
                if (j >= vo[i][k])
                    dp[j] = _max(dp[j], dp[j-vo[i][k]] + va[i][k]);
    printf("%d\n", dp[V]);
}
```

4.4 区间 DP

4.4.1 石子合并.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：01-石子合并.cpp
 *   创建日期：2021年03月09日 ---- 17时08分
 *   题    目：石子合并
 *   算    法：区间DP
 *   描    述：每次只能合并相邻的两堆石子，合并的花费为这两堆石子的总数
 *       每组数据第一行是n个整数，接下来有n堆石子
 *       状态转移方程：dp[i][j] = min(dp[i][k] + dp[k+1][j]) + sum[i][j-i+1];
 *
 -----------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 255;
const int inf  = 0x3f3f3f3f;
int n;
int sum[maxn]; //前i堆的和

int minval() {
    int dp[maxn][maxn]; //从第i堆石子到第j堆石子的最小花费
    for (int i = 1; i <= n; ++i)
        dp[i][i] = 0;
    for (int len = 1; len <= n; ++len) //len是i与j之间的距离，也就是说区间DP是从小区间到大区间DP
        for (int i = 1; i <= n-len; ++i) { //从第i堆开始
            int j = i + len;               //从第j堆结束
            dp[i][j] = inf;
            for (int k = i; k < j; ++k)    //i与j之间用k进行分割
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + (sum[j]-sum[i-1]));
        }
    return dp[1][n];
}

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

4.4.2 回文串.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;
}
```

4.5 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;
}
```

4.6 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
 *
 ----------------------------------------------------------------*/

//<++>
```

4.7 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;
}

```

```cpp
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

```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;
}
```

5 字符串

5.1 字符串哈希

5.1.1 字符串哈希.cpp

```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

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

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

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 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 字符串.cpp

```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：最后还是使用了灯笼的模板
 *
 --------------------------------------------------------------------*/

#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 preT[maxm];

/**
 * KMP灵魂：生成前缀表
 *
 * a b a b c
 * 假如是这样一个模式串，我们来生成它的前缀表。
 * a
 * a b
 * a b a
 * a b a b
 * a b a b c
 * 对于模式串的所有前缀，得到他们的最长公共前后缀长度。
 *
 * 对于[a b a b]这个原串的前缀(前后缀长度不能等于原串的长度)：
 * 他的最长前缀是：a b a；它的最长后缀是：b a b. 最长前缀不等于最长后缀；
 * 它的第二长前缀：a b  ；它的第二长后缀：a b. 第二长前缀等于第二长后缀；
 * 所以这个原串的前缀的最长公共前后缀长度为2。
 *
 * 同理就可以得到下面这个前缀表：
 * -1
 * 0  a
 * 0  a b
 * 1  a b a
 * 2  a b a b
 * 0  a b a b c
 *
 * 为了处理边界，我们在最上面添加一个-1，并删除最后一个0，得到最终前缀表：-1, 0, 0, 1, 2
 *
 *  a b a b c
 * -1 0 0 1 2
 *  !!!(注意上面这个前缀表，灯笼与雪菜此处下标不同)
 *
 */
void prefix_table() {
    preT[0] = 0;
    int len = 0;
    int i = 1;
    while (i < n) {
        if (pattern[i] == pattern[len]) {
            len++;
            preT[i] = len;
            i++;
        }
        else if (len > 0)
            len = preT[len-1];
        else {
            preT[i] = 0;
            i++;
```

```
        }
    }
    // 向后移动一位
    for (int i = m-1; i > 0; --i)
        preT[i] = preT[i-1];
    preT[0] = -1;
}

void kmp_search() {
    prefix_table();
    // text[i], pattern[j];
    int i = 0, j = 0;
    while (i < n) {
        if (j == m - 1 && text[i] == pattern[j]) {
            printf("%d ", i - j);
            // printf("Found pattern at %d\n", i - j);
            j = preT[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 = preT[j];
            if (j == -1)  // 如果是-1表示直接匹配从下一个字符开始匹配
                i++, j++;
        }
    }
}

int main() {
    scanf("%d %s", &m, pattern);
    scanf("%d %s", &n, text);
    // n = 19, m = 9;
    // 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

```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;
}
```

5.6 后缀数组 SA

5.6.1 后缀数组的使用.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

```
#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

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

6.2 位运算

6.2.1 整数除以 2.cpp

```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;
}
```

6.2.2 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 lowbit.cpp

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

int main() {
    printf("%d\n", lowbit(6));
    return 0;
}
```

6.3 快速幂

6.3.1 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;
}

```

```cpp
/*
 *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;
}
```

6.3.2 矩阵快速幂.cpp

```cpp
#include <cstdio>
#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;
}
```

6.3.3 快速幂模板.cpp

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

/* 看题目会不会超过int，选择合适的参数类型
 * 这里还是尽量不要用T，显得很蠢，自己判断会不会超过int，决定使用ll还是int
 */
template<typename> T
T binPow(T base, T expo) {
    T 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 % mod;
}

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

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 02.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)
```

6.5.2 03. 二进制方法实现 64 位整数乘法.cpp

```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 高精度加法.cpp

```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;
}
```

6.5.4 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;
}
```

6.5.5 高精度乘法.cpp

```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：<++>
 *   算    法：高精度乘法
 *   描    述：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
 *
 -----------------------------------------------------------------*/

#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;
    for (int i = 0, t = 0; i < A.size() || t; ++i) {
        if (i < A.size())
            t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }

    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');

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

6.5.6 高精度除法.cpp

```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;
}
```

6.5.7 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);
        }
    }
}
```

6.6 数论

6.6.1 素数

6.6.1.1 n! 中质因子 p 的个数.cpp

```
int factor(int n, int p) {
    if (n < p)
        return 0;
```

```

    return n / p + factor(n / p , p);
}
```

6.6.1.2 素因子.cpp

```cpp
#include <cstdio>
#include <cmath>
#include <vector>
using namespace std;
const int maxn = 1e2;
/*int primeFactor[maxn];*/

vector<int> primeFactor;
template<typename T>
void PrimeFactor(T num) {
    for (T i = 2; pow(i, 2) <= num; ++i)
        if (!(num % i)) {
            primeFactor.push_back(i);
            /*primeFactor[cnt++] = i;*/
            while (!(num % i)) num /= i;
        }
    if (num > 1) primeFactor.push_back(num);
    /*primeFactor.clear();*/
}

int main() {
    int num;
    scanf("%d", &num);
    PrimeFactor(num);
    printf("cnt = %d\n", (int)primeFactor.size());
    for (int i = 0; i < (int)primeFactor.size(); ++i)
        printf("%d ", primeFactor[i]);
    return 0;
}
```

6.6.1.3 质因子分解.cpp

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

struct Factor {
    int x;
    int cnt;
} factor[10];

const int maxn    = 1e4;
bool sifter[maxn] = {0};
int pnum = 0;
int j    = 0;
int prime[maxn];


void findPrime() {
    for (int i = 2; i <= maxn; i++)
        //发现素数
        if (sifter[i] == false) {
            prime[pnum++] = i;
            for (int j = i + i; j < maxn; j += i)
                sifter[j] = true;
        }
}

void findFactor(int num) {
    int sqrnum = sqrt(1.0 * num);

    for (int i = 0; i < sqrnum; i++)
        if (num % prime[i] == 0) {
```

```
            factor[j].x   = prime[i];
            factor[j].cnt = 0;
            while (num % prime[i] == 0) {
                factor[j].cnt++;
                num /= prime[i];
            }
            j++;
        }
    if (num != 1) {
        factor[j].x   = num;
        factor[j].cnt = 1;
    }
}

int main() {
    int num = 12345;
    findPrime();
    findFactor(num);

    for (int i = 0; i <= j; i++)
        printf("%d^%d + ", factor[i].x, factor[i].cnt);

    printf("\b= ");
    printf("%d\n", num);
    return 0;
}
```

6.6.1.4 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;
}
```

6.6.1.5 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
 */
```

6.6.1.6 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 欧几里得算法

6.6.2.1 gcd-lcm.cpp

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

int Gcd(int num1, int num2) {
    return !num2 ? num1 : Gcd(num2, num1 % num2);
}

int LCM(int num1, int num2) {
    return num1 / Gcd(num1, num2) * num2;
}

int main() {
    printf("%d\n", Gcd(12, 32));
    int gcd = __gcd(12, 32);
    printf("%d\n", gcd);
    return 0;
}
```

6.6.2.2 扩展欧几里得算法.cpp

```
//可以使用扩展欧几里得算法对方程 ax + by = c 求解
#include<iostream>
using namespace std;

int exgcd(int a, int b, int &x, int &y /*使用引用*/ ) {
    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
//欧几里得算法中直接return，而这里由于需要递归得到 x 与 y ，所以不能直接return
    int gcd = exgcd(b, a % b, x, y);
    int temp = x;   //为了得到 y_(n) 需要保存 x_(n+1)
    x = y;
    y = temp - a / b * y;
    return gcd;
}

int main() {
    int a, b, x, y;
    cin >> a >> b;
    int gcd = exgcd(a, b, x, y);

    cout << "a   = " << a << endl << "b   = " << b << endl;
    cout << "x   = " << x << endl << "y   = " << y << endl;
```

```
    cout << "gcd = " << gcd << endl;
    cout << "a * x + b * y = " << gcd << endl;
    cout << a << " * " << x << " + " << b << " * " << y << " = " << gcd << endl;

    return 0;
}
```

6.6.2.3 扩展欧几里德.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：02-扩展欧几里德.cpp
 *   创建日期：2021年03月10日 ---- 22时25分
 *   算    法：扩展欧几里德
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

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

/*ax + by = gcd(a, b)*/
/*有解的充分必要条件是gcd(a, b)可以整除n*/
void exgcd(int a, int b, int &x, int &y) {
    if (b ==0) {
        x = 1;
        y = 0;
        return;
    }
    exgcd(b, a%b, x, y);
    int tmp = x;
    x = y;
    y = tmp - (a/b) * y;
}

/**
 * cx + dy = 1
 * 其中c = a / gcd(a, b), d = b / gcd(a, b)
 * 通解：x = x0 + dt
 *       y = y0 - ct;
 * x0, y0是上面的x, y, t是任意整数
 **/
void simplyExgcd(int c, int d, int &x, int &y) {
    int tmpc = c;
    int tmpd = d;
    exgcd(c, d, x, y);
    int t = 1; //t可以是任何数
    x = x + tmpd * t;
    y = y - tmpc * t;
}

/**
 * ax + by = n
 * 有整数解的条件是gcd(a, b)可以整除n
 * 一个解：x0' = x0 * n / gcd(a, b)
 *         y0' = y0 * n / gcd(a, b)
 **/
void anyExgcd(int a, int b, int n, int &x, int &y) {
    exgcd(a, b, x, y);
    x = x * n / __gcd(a, b);
    y = y * n / __gcd(a, b);
}
```

6.6.3 乘法逆元

6.6.3.1 费马小定理.cpp

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

```
 *
 *    文件名称: 费马小定理.cpp
 *    创建日期: 2021年03月11日 ---- 20时07分
 *    算    法: 费马小定理求逆元
 *    描    述: x^(mod-2)就是逆元
 *
 ----------------------------------------------------------------*/

#include<cstdio>
typedef long long  ll;
const int mod = 99991;

/*
 *ll binPow(ll base, ll expo, ll mod) {
 *    if (expo == 0) return 1;
 *
 *    if (expo % 2 == 1)
 *        return base * binPow(base, expo-1, mod) % mod;
 *    else {
 *        ll mul = binPow(base, expo/2, mod) % mod;
 *        return mul % mod * mul % mod;
 *    }
 *}
 */

ll binPow(ll base, ll expo, ll mod) {
    ll res = 1;
    while (expo != 0) {
        if (expo & 1)
            res = (1ll * res * base) % mod;

        base = (1ll * base * base) % mod;
        expo >>= 1;
    }
    return res;
}

ll inv(ll x) {
    return binPow(x, mod-2, mod);
}

int main() {
    for (int i = 0; i < 20; ++i)
        printf("%lld ", inv((ll)i));
    return 0;
}
```

6.6.3.2 扩展欧几里得.cpp

```
#include <cstdio>
const int mod = 99991;

int exGcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }

    int gcd  = exGcd(b, a%b, x, y);
    int temp = x;
    x = y;
    y = temp - a / b * y;
    return gcd;
}

/*返回值是a模m的逆元*/
int inv(int a) {
    int x;
```

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

6.6.3.3 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;
}
```

6.6.3.4 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]);

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

6.6.4 分解质因数

6.6.4.1 分解质因数.cpp

```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>

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);
        puts("");
    }
    return 0;
}
```

6.7 多项式

6.7.1 快速傅里叶变换

6.7.1.1 Cooley-Tukey.cpp

```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);
```

```cpp
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

```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

```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 组合数学

6.9.1 排列组合

6.9.1.1 组合数.md

```
计算组合数
1. 通过定义直接计算
2. 通过递推公式计算
3. 通过定义式的变形计算

计算 $ C_n^m % p $
1. 通过递推公式计算
2. 根据定义式计算
3. 通过定义式的变形计算
4. Lucas定理
```

6.9.1.2 杨辉三角.cpp

```cpp
#include <cstdio>
#include <vector>
using namespace std;
const int maxn = 20;
/*杨辉三角：第n行有n个数，如果要得到20行杨辉三角，需要数组宽度为21*/
int triangle[maxn+1][maxn+1];
/*如果要得到5行杨辉三角，需要数组宽度为6*/
/**
 * 0 1
 * 0 1 1
 * 0 1 2 1
 * 0 1 3 3 1
 * 0 1 4 6 4 1
 * 第一列都是0，第五行有6列
 */
void YHtriangle(int border) {
    triangle[0][1] = 1;
    for (int i = 1; i < border; ++i)
        for (int j = 1; j <= i; ++j)
            triangle[i][j] = triangle[i-1][j-1] + triangle[i-1][j];
}

/*仅生成第n行的杨辉三角*/
vector<int> angle(20, 0);
void lineAngle(int n) {
    angle[0] = angle[n-1] = 1;
    for (int i = 1; i < n; ++i)
        angle[i] = angle[n-i-1] = (n-i) * angle[i-1] / i;
}

int main() {
    YHtriangle(20);
    for (int i = 0; i < maxn; ++i) {
        for (int j = 0; j <= maxn; ++j)
            if (triangle[i][j] != 0)
                printf("%6d ", triangle[i][j]);
        printf("\n");
    }

    lineAngle(10);
    for (int i = 0; i < (int)angle.size(); ++i)
        if (angle[i])
            printf("%d ", angle[i]);
    printf("\n");
    return 0;
}
```

6.9.2 卡特兰数

6.9.2.1 Catalan.cpp

```cpp
#include <cstdio>
#include <vector>
using namespace std;
typedef long long LL;
const LL maxn = 100;
const LL MOD = 1e9 + 9;
vector <long long> fact(maxn + 1, 1LL);
vector <long long> inv(maxn + 1, 1LL);
LL Catalan[maxn];

template<typename T>
/*快速幂*/
T binPow(T a, T n) {
    T res = 1;
    while (n) {
        if (n & 1) res = (1LL * res * a) % MOD;
        a = (1LL * a * a) % MOD;
        n >>= 1;
    }
    return res;
}

/*阶乘*/
void Factorial() {
    for (int i = 1; i <= maxn; ++i) {
        fact[i] = (fact[i - 1] * i) % MOD;
        inv[i] = binPow(fact[i], MOD - 2);
    }
}

/*组合数*/
LL C(int k, int n) {
    if (k > n) return 0;
    int multiply = (1LL * fact[n] * inv[k]) % MOD;
    multiply = (1LL * multiply * inv[n - k]) % MOD;
    return multiply;
}

/*用来求n较小的卡特兰数，相比较不容易溢出*/
void Catalan_Num() {
    Catalan[0] = Catalan[1] = 1;
    for (int i = 2; i <= 35; ++i) {
        for (int j = 0; j < i; ++j)
            Catalan[i] += Catalan[j] * Catalan[i-j-1];
        printf("%d -> %lld\n", i, Catalan[i]);
    }
}

/*用来求n非常大的卡特兰数，需要求模*/
LL Catalan_Num(int n) {
    return C(n, 2*n) / (n+1);
}

int main() {
    Catalan_Num();
    Factorial();
    printf("num = %lld\n", Catalan[10]);
    printf("num = %lld\n", Catalan_Num(10));
    return 0;
}
```

6.9.3 斯特林数

6.9.3.1 Stirling.cpp

```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;
}

poly poly::operator>>(const std::size_t &b)const{
    std::vector<int> f(size() - b);
    for(int i = 0; i < (int)f.size(); ++i) f[i] = data[i + b];
    return f;
}

poly poly::operator>>=(const std::size_t &b){
    return *this = (*this) >> b;
}

poly poly::operator/(const int &h)const{
    std::vector<int> f(this->data); int inv = qpow(h, mod - 2);
    for(int i = 0; i < (int)f.size(); ++i) f[i] = (ll)f[i] * inv % mod;
    return f;
}

poly poly::operator/=(const int &h){
    int inv = qpow(h, mod - 2);
    for(int i = 0; i < (int)data.size(); ++i) data[i] = (ll)data[i] * inv % mod;
    return *this;
}

poly poly::inv(void)const{
    int N = 1; while(N < (int)(size() + size() - 1)) N <<= 1;
    std::vector<int> f(N), g(N), d(this->data);
    d.resize(N), f[0] = qpow(d[0], mod - 2);
    for(int w = 2; w < N; w <<= 1){
        for(int i = 0; i < w; ++i) g[i] = d[i];
        rev.resize(w << 1);
        for(int i = 0; i < w * 2; ++i) rev[i] = (rev[i >> 1] >> 1) | (i & 1 ? w : 0);
        dft_for_module(f, w << 1, 1), dft_for_module(g, w << 1, 1);
        for(int i = 0; i < w * 2; ++i) f[i] = (ll)f[i] * (2 + mod - (ll)f[i] * g[i] % mod) % mod;
        dft_for_module(f, w << 1, -1);
        for(int i = 0, inv = qpow(w << 1, mod - 2); i < w; ++i) f[i] = (ll)f[i] * inv % mod;
        for(int i = w; i < w * 2; ++i) f[i] = 0;
    }
    f.resize(size());
    return f;
}

poly poly::operator==(const poly &h)const{
    if(size() != h.size()) return 0;
    for(int i = 0; i < (int)size(); ++i) if(data[i] != h[i]) return 0;
    return 1;
}

poly poly::operator!=(const poly &h)const{
    if(size() != h.size()) return 1;
    for(int i = 0; i < (int)size(); ++i) if(data[i] != h[i]) return 1;
    return 0;
}

poly poly::operator+(const int &h)const{
    poly f(this->data);
    f[0] = (f[0] + h) % mod;
    return f;
}

poly poly::operator+=(const int &h){
    return *this = (*this) + h;
}

poly poly::inv(const int &h)const{
    poly f(*this);
```

```
        f.resize(h);
        return f.inv();
    }

    int modsqrt(int h, int p = mod){
        return 1;
    }

    poly sqrt(const poly &h){
        int N = 1; while(N < (int)(h.size() + h.size() - 1)) N <<= 1;
        poly f(N), g(N), d(h); d.resize(N), f[0] = modsqrt(d[0]);
        for(int w = 2; w < N; w <<= 1){
            g.resize(w);
            for(int i = 0; i < w; ++i) g[i] = d[i];
            f = (f + f.inv(w) * g) / 2;
            f.resize(w);
        }
        f.resize(h.size());
        return f;
    }

    poly log(const poly &h){
        poly f(h);
        for(int i = 1; i < (int)f.size(); ++i) f[i - 1] = (ll)f[i] * i % 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, mod - 2) % mod;
        f[0] = 0;
        return f;
    }

    poly exp(const poly &h){
        int N = 1; while(N < (int)(h.size() + h.size() - 1)) N <<= 1;
        poly f(N), g(N), d(h);
        f[0] = 1, d.resize(N);
        for(int w = 2; w < N; w <<= 1){
            f.resize(w), g.resize(w);
            for(int i = 0; i < w; ++i) g[i] = d[i];
            f = f * (g + 1 - log(f));
            f.resize(w);
        }
        f.resize(h.size());
        return f;
    }

    struct comp{
        long double x, y;
        comp(long double _x = 0, long double _y = 0) : x(_x), y(_y) {}
        comp operator*(const comp &b)const{return comp(x * b.x - y * b.y, x * b.y + y * b.x); }
        comp operator+(const comp &b)const{return comp(x + b.x, y + b.y); }
        comp operator-(const comp &b)const{return comp(x - b.x, y - b.y); }
        comp conj(void){return comp(x, -y); }
    };

    const int EPS = 1e-9;

    template <typename FLOAT_T>
        FLOAT_T fabs(const FLOAT_T &x){
            return x > 0 ? x : -x;
        }

    template <typename FLOAT_T>
        FLOAT_T sin(const FLOAT_T &x, const long double &EPS = fstdlib::EPS){
            FLOAT_T res = 0, delt = x;
            int d = 0;
            while(fabs(delt) > EPS){
                res += delt, ++d;
                delt *= - x * x / ((2 * d) * (2 * d + 1));
            }
            return res;
        }
```

```
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;
289             int d = 0;
290             while(fabs(delt) > EPS){
291                 res += delt, ++d;
292                 delt *= - x * x / ((2 * d) * (2 * d - 1));
293             }
294             return res;
295         }
296
297     const long double PI = std::acos((long double)(-1));
298
299     void dft_for_complex(std::vector<comp> &f, int n, int b){
300         static std::vector<comp> w;
301         w.resize(n);
302         for(int i = 0; i < n; ++i) if(i < rev[i]) std::swap(f[i], f[rev[i]]);
303         for(int i = 2; i <= n; i <<= 1){
304             w[0] = comp(1, 0), w[1] = comp(cos(2 * PI / i), b * sin(2 * PI / i));
305             for(int j = 2; j < i / 2; ++j) w[j] = w[j - 1] * w[1];
306             for(int j = 0; j < n; j += i)
307                 for(int k = 0; k < i / 2; ++k){
308                     comp p = f[j + k], q = f[j + k + i / 2] * w[k];
309                     f[j + k] = p + q, f[j + k + i / 2] = p - q;
310                 }
311         }
312     }
313
314     class arbitrary_module_poly{
315         private:
316             std::vector<int> data;
317             int construct_element(int D, ll x, ll y, ll z)const{
318                 x %= mod, y %= mod, z %= mod;
319                 return ((ll)D * D * x % mod + (ll)D * y % mod + z) % mod;
320             }
321         public:
322             int mod;
323             arbitrary_module_poly(std::size_t len = std::size_t(0), int module_value = 1e9 + 7){mod =
    module_value; data = std::vector<int>(len); }
324             arbitrary_module_poly(const std::vector<int> &b, int module_value = 1e9 + 7){mod = module_value;
    data = b; }
325             arbitrary_module_poly(const arbitrary_module_poly &b){mod = b.mod; data = b.data; }
326             void resize(std::size_t len, const int &val = 0){data.resize(len, val); }
327             std::size_t size(void)const{return data.size(); }
328             void clear(void){data.clear(); }
329 #if __cplusplus >= 201103L
330             void shrink_to_fit(void){data.shrink_to_fit(); }
331 #endif
332             int &operator[](std::size_t b){return data[b]; }
333             const int &operator[](std::size_t b)const{return data[b]; }
334             arbitrary_module_poly operator*(const arbitrary_module_poly& h)const;
335             arbitrary_module_poly operator*=(const arbitrary_module_poly &h);
336             arbitrary_module_poly operator*(const int &h)const;
337             arbitrary_module_poly operator*=(const int &h);
338             arbitrary_module_poly operator+(const arbitrary_module_poly &h)const;
339             arbitrary_module_poly operator+=(const arbitrary_module_poly &h);
340             arbitrary_module_poly operator-(const arbitrary_module_poly &h)const;
341             arbitrary_module_poly operator-=(const arbitrary_module_poly &h);
342             arbitrary_module_poly operator<<(const std::size_t &b)const;
343             arbitrary_module_poly operator<<=(const std::size_t &b);
344             arbitrary_module_poly operator>>(const std::size_t &b)const;
345             arbitrary_module_poly operator>>=(const std::size_t &b);
346             arbitrary_module_poly operator/(const int &h)const;
347             arbitrary_module_poly operator/=(const int &h);
348             arbitrary_module_poly operator==(const arbitrary_module_poly &h)const;
349             arbitrary_module_poly operator!=(const arbitrary_module_poly &h)const;
350             arbitrary_module_poly inv(void)const;
351             arbitrary_module_poly inv(const int &h)const;
352             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);
}
```

```
423     arbitrary_module_poly arbitrary_module_poly::operator<<=(const std::size_t &b){
424         return *this = (*this) << b;
425     }
426
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());
```

```
        return f;
    }

    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 斐波那契数列

6.10.1 Fibonacci.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;
}
```

6.10.2 Fibonacci.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;
}
```

6.11 博弈论

6.11.1 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;
}
```

6.11.2 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;
}
```

6.11.3 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;
}
```

7 数据结构

7.1 栈

7.1.1 模拟栈.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

```
#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")) {
            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

```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

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

#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 并查集

7.4.1 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
 *
 --------------------------------------------------------------*/

#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;
}
```

7.4.2 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：AcWing视频
 *
 ----------------------------------------------------------------*/

#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;
}
```

7.5 堆

7.5.1 二叉堆

7.5.1.1 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>
#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;
}
```

7.5.1.2 模拟堆.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：<++>
 *
 ----------------------------------------------------------------*/

// 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;
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);
    int m = 0;
    while (n--) {
        char op[10];
        int k, x;
        scanf("%s", op);
        if (!strcmp(op, "I")) {
            scanf("%d", &x);
            tot++;
            m++;
            ph[m] = tot, hp[tot] = m;
            he[tot] = x;
            up(tot);
        }
        else if (!strcmp(op, "PM"))
            printf("%d\n", he[1]);
        else if (!strcmp(op, "DM")) {
            heap_swap(1, tot);
            tot--;
            down(1);
        }
        else if (!strcmp(op, "D")) {
            scanf("%d", &k);
            k = ph[k];
            heap_swap(k, tot);
            tot--;
            down(k), up(k);
        }
        else {
            scanf("%d %d", &k, &x);
            k = ph[k];
            he[k] = x;
            down(k), up(k);
        }
    }
    return 0;
}
```

7.6 数状数组

7.6.1 数状数组.cpp

```
/*------------------------------------------------------------------
 *
 *   文件名称：数状数组.cpp
 *   创建日期：2021年03月07日 ---- 11时36分
 *   题    目：poj2182
 *   算    法：数状数组
```

```
 *   描    述: <++>
 *
--------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e6;
int tree[maxn];
int n;
#define lowbit(x) ((x) & -(x))

/*tree[x] += d*/
void add(int x, int d) {
    while (x <= n) {
        tree[x] += d;
        x += lowbit(x);
    }
}

/*sum(tree[1], tree[x])*/
int sum(int x) {
    int sum = 0;
    while (x > 0) {
        sum += tree[x];
        x -= lowbit(x);
    }
    return sum;
}

int findpos(int x) {
    int L = 1;
    int R = n;
    while (L < R) {
        int mid = (L + R) >> 1;
        sum(mid) < x ? L = mid+1 : R = mid;
    }
    return L;
}

int main() {
    int pre[maxn];
    int ans[maxn];
    scanf("%d", &n);
    pre[1] = 0;
    for (int i = 2; i <= n; ++i)
        scanf("%d", &pre[i]);
    for (int i = 1; i <= n; ++i)
        tree[i] = lowbit(i);
    for (int i = n; i > 0; --i) {
        int x = findpos(pre[i] + 1);
        add(x, -1);
        ans[i] = x;
    }
    for (int i = 1; i <= n; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

7.7 单调栈

7.7.1 直方图中最大矩形.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
 *
```

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

#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

```
/*----------------------------------------------------------------
 *
 *   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：<++>
 *
 ----------------------------------------------------------------*/

#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)
            hh++;
        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);
    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 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;
}
```

7.9.2 SegmentTree.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：SegmentTree.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
 *
 *   ! 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 v;       // 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].v = max(tr[u << 1].v, tr[u << 1 | 1].v);
}
```

```

// 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 (tr[u].l >= l && tr[u].r <= r)
        return tr[u].v;
    // 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].v = 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() {
    // idx是数的个数，last是上一个区间的答案
    int m, p;
    scanf("%d %d", &m, &p);
    int idx = 0, last = 0;
    build(1, 1, m);
    int x;
    char op[5];
    while (m --) {
        scanf("%s %d", op, &x);
        if (*op == 'Q') {
            last = query(1, idx - x + 1, idx);
```

```
            printf("%d\n", last);
        }
        else if (*op == 'A') {
            modify(1, idx + 1, ((long long)last + (long long)x) % p);
            idx ++;
        }
    }
    return 0;
}
```

7.10 二叉搜索树-平衡树

7.10.1 一般意义的树

7.10.1.1 1-树的静态写法.cpp

```
struct Node {
    int data;
    vector<int> child;
}node[maxn];


int index = 0;
int newNode(int value) {
    node[index].data = value;
    node[index].child.clear();
    return index++;
}
```

7.10.1.2 2-树的先根遍历.cpp

```
void preorder(int root) {
    printf("%d ", node[root].data);
    for (int i = 0; i < node[root].child.size(); i++)
        preorder(node[root].child[i]);
}
```

7.10.1.3 3-树的层序遍历.cpp

```
void layerorder(int root) {
    queue<int> quu;
    quu.push(root);
    while (quu.empty() == false) {
        int front = quu.front();
        printf("%d ", node[front].data);
        quu.pop();
        for (int i = 0; i < n; i++)
            quu.push(node[front].child[i]);
    }
}
```

7.10.1.4 复杂建树.cpp

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

typedef struct node {
    int data;
    struct node* left;
    struct node* right;
}Node;

void preorder(Node* node) {
    if (node != NULL) {
        printf("%d\n", /* node.data */ node -> data);
        preorder(node -> left );
        preorder(node -> right);
```

```
    }
}

void inorder(Node* node) {
    if (node != NULL) {
        inorder(node -> left );
        printf("%d\n", node -> data);
        inorder(node -> right);
    }
}

void postorder(Node* node) {
    if (node != NULL) {
        postorder(node -> left );
        postorder(node -> right);
        printf("%d\n", node -> data);
    }
}

int main() {
    struct node n1;
    struct node n2;
    Node n3;
    Node n4;

    n1.data = 5;
    n2.data = 6;
    n3.data = 7;
    n4.data = 8;

    n1.left  = &n2;
    n1.right = &n3;
    n2.left  = &n4;
    n2.right = NULL;
    n3.left  = NULL;
    n3.right = NULL;
    n4.left  = NULL;
    n4.right = NULL;

    preorder(&n1);
    inorder(&n1);
    postorder(&n1);

    return 0;
}
```

7.10.2 二叉树

7.10.2.1 二叉树静态描述.cpp

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

const int maxn = 1e3;
int index = 0;
bool flag;
struct Node {
    int data;
    int left;
    int right;
}node[maxn];

int newNode(int value) {
    node[index].data  = value;
    node[index].left  = -1;
    node[index].right  = -1;
    return index++;
}

```

```
//找到数据域为x的结点，把它修改为newData
void search(int root, int x, int newData) {
    if (root == -1)
        return ;

    if (node[root].data == x)
        node[root].data = newData;

    search(node[root].left,  x, newData);
    search(node[root].right, x, newData);
}

void insert(int &root, int x) {
    if (root == -1) {
        root = newNode(x);
        return ;
    }
    if (/*由二叉树的性质，x需要插入到左子树*/flag)
        insert(node[root].left,  x);
    else
        insert(node[root].right, x);
}

int creatTree(int data[], int n) {
    int root = -1;
    for (int i = 0; i < n; i++)
        insert(root, data[i]);

    return root;
}

void preorder(int root) {
    if (root == -1)
        return ;

    printf("%d\n", node[root].data);
    preorder(node[root].left);
    preorder(node[root].right);
}

void inorder(int root) {
    if (root == -1)
        return ;

    printf("%d\n", node[root].data);
    preorder(node[root].left);
    preorder(node[root].right);
}

void postorder(int root) {
    if (root == -1)
        return ;

    printf("%d\n", node[root].data);
    preorder(node[root].left);
    preorder(node[root].right);
}

void layerorder(int root) {
    queue<int> quu;
    quu.push(root);
    while (quu.empty() == false) {
        int now = quu.front();
        quu.pop();
        printf("%d ", node[now].data);
        if (node[now].left != -1)
            quu.push(node[now].left);

        if (node[now].right != -1)
            quu.push(node[now].left);
```

```
    }
}
```

7.10.2.2 创建二叉树.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：1-创建二叉树.cpp
 *   创建日期：2020年09月22日 ---- 17时06分
 *   算    法：二叉树
 *   描    述：学习二叉搜索树的创建
 *
-----------------------------------------------------------------*/

#include <cstdio>

bool flag = false;
struct Node {
    int data;
    Node* left;
    Node* right;
};

/*新建结点*/
Node* newNode(int value) {
    Node* node = new Node;
    node -> data = value;
    node -> left = node -> right = NULL;
    return node;
}

/*二叉树结点的查找与修改*/
void search(Node* root, int x, int newdata) {
    if (root == NULL)
        return ;
    if (root -> data == x)
        root -> data = newdata;

    search(root -> left,  x, newdata);
    search(root -> right, x, newdata);
}

/*二叉树结点的插入*/
/*必须使用引用*/
void insert(Node* &root, int x) {
    if (root == NULL) {
        root = newNode(x);
        return ;
    }
    if (/*根据题目，x插在左子树*/flag)
        insert(root -> left, x);
    else
        insert(root -> right, x);
}

/*二叉树的创建*/
Node* creatTree(int data[], int n) {
    Node* root = NULL;
    for (int i = 0; i < n; i++)
        insert(root, data[i]);
    return root;
}

int main() {
    int data[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    Node* root = creatTree(data, 10);
    search(root, 4, 5);
    /*遍历*/
    return 0;
```

```
}
```

7.10.2.3 层序遍历.cpp

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

struct Node {
    int data;
    Node* left;
    Node* right;
};

struct Tree {
    Node* root;
};

void layerOrder(Node* root) {
    queue<Node*> quu;
    quu.push(root);
    while (quu.empty() == false) {
        Node* first = quu.front();
        quu.pop();
        printf("%d\n", first -> data);
        if (first -> left  != NULL)
            quu.push(first -> left);
        if (first -> right != NULL)
            quu.push(first -> right);
    }
}

int main() {
    return 0;
}
```

7.10.2.4 静态二叉树.cpp

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

const int maxn = 1e3;
int index = 0;
bool flag;
struct Node {
    int data;
    int left;
    int right;
}node[maxn];

int newNode(int value) {
    node[index].data  = value;
    node[index].left  = -1;
    node[index].right  = -1;
    return index++;
}

//找到数据域为x的结点，把它修改为newData
void search(int root, int x, int newData) {
    if (root == -1)
        return ;

    if (node[root].data == x)
        node[root].data = newData;

    search(node[root].left,  x, newData);
    search(node[root].right, x, newData);
```

```
}

void insert(int &root, int x) {
    if (root == -1) {
        root = newNode(x);
        return ;
    }
    if (/*由二叉树的性质，x需要插入到左子树*/flag)
        insert(node[root].left,  x);
    else
        insert(node[root].right, x);
}

int creatTree(int data[], int n) {
    int root = -1;
    for (int i = 0; i < n; i++)
        insert(root, data[i]);

    return root;
}

void preorder(int root) {
    if (root == -1)
        return ;

    printf("%d\n", node[root].data);
    preorder(node[root].left);
    preorder(node[root].right);
}

void inorder(int root) {
    if (root == -1)
        return ;

    printf("%d\n", node[root].data);
    preorder(node[root].left);
    preorder(node[root].right);
}

void postorder(int root) {
    if (root == -1)
        return ;

    printf("%d\n", node[root].data);
    preorder(node[root].left);
    preorder(node[root].right);
}

void layerorder(int root) {
    queue<int> quu;
    quu.push(root);
    while (quu.empty() == false) {
        int now = quu.front();
        quu.pop();
        printf("%d ", node[now].data);
        if (node[now].left != -1)
            quu.push(node[now].left);

        if (node[now].right != -1)
            quu.push(node[now].left);
    }
}
```

7.10.3 二叉搜索树

7.10.3.1 二叉搜索树.cpp

```
/*-----------------------------------------------------------------
 *
 *   文件名称：1-二叉查找树.cpp
```

```
 *   创建日期: 2020年09月22日 ---- 15时50分
 *   算    法: BST Binary Search Tree
 *   描    述: 正月打灯笼
 *             未考虑负数
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstdlib>

struct Node {
    int data;
    Node* left;
    Node* right;
};
struct Tree {
    Node* root;
};

void insert(Tree* tree, int value) {
    Node* node = (Node *)malloc(sizeof(Node));
    node -> data  = value;
    node -> left  = NULL;
    node -> right = NULL;
    if (tree -> root == NULL)
        tree -> root = node;
    else {
        Node* temp = tree -> root;
        while (temp != NULL) {
            if (value < temp -> data) {
                if (temp -> left == NULL) {
                   temp -> left = node;
                    return ;
                }
                else
                    temp = temp -> left;
            }
            else {
               if (temp -> right == NULL) {
                    temp -> right = node;
                    return ;
                }
                else
                    temp = temp -> right;
            }
        }
    }
}

int get_height(Node* node) {
    if (node == NULL)
        return 0;
    else {
        int left_h  = get_height(node -> left);
        int right_h = get_height(node -> right);
        int max = left_h;
        if (right_h > max)
            max = right_h;
        return max + 1;
    }
}

int get_maximum(Node* node) {
    if (node == NULL)
        return -1;
    else {
        int m1 = get_maximum(node -> left);
        int m2 = get_maximum(node -> right);
        int m3 = node -> data;
        int max = m1;
```

```
        if (m2 > max) { max = m2; }
        if (m3 > max) { max = m3; }
        return max;
    }
}

void preorder(Node* node) {
    if (node != NULL) {
        printf("%d\n", /* node.data */ node -> data);
        preorder(node -> left );
        preorder(node -> right);
    }
}

void inorder(Node* node) {
    if (node != NULL) {
        inorder(node -> left );
        printf("%d\n", node -> data);
        inorder(node -> right);
    }
}

void postorder(Node* node) {
    if (node != NULL) {
        postorder(node -> left );
        postorder(node -> right);
        printf("%d\n", node -> data);
    }
}

int main() {
    int arr[7] = {6, 3, 8, 2, 5, 1, 7};
    Tree tree;
    tree.root = NULL;

    for (int i = 0; i < 7; ++i)
        insert(&tree, arr[i]);

    preorder(tree.root);
    printf("\n");
    // inorder(tree.root);

    int h = get_height(tree.root);
    printf("h = %d\n", h);

    int m= get_maximum(tree.root);
    printf("max = %d\n", m);
    return 0;
}
```

7.10.4 AVL 树

7.10.4.1 平衡二叉树.cpp

```
/*------------------------------------------------------------------
 *
 *   文件名称：平衡二叉树.cpp
 *   创建日期：2021年03月04日 ---- 21时39分
 *   算    法：平衡二叉树
 *   描    述：感觉是个半成品
 *   但是我不想改了，在那两个unsed地方，注释了更新height的代码
 *   这两处代码应该是没问题的，但是报错了，注释之后，好像也没影响
 *
------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cassert>
using namespace std;
struct Node {
```

```
    int data;
    int height;
    Node* left;
    Node* right;
};
struct Tree {
    Node* root;
};
#define bug printf("<------>\n");

/*得到当前结点的高度*/
int getHeight(Node* node) {
    if (node)
        return max(getHeight(node -> left), getHeight(node -> right)) + 1;
    return 0;
}

/*得到结点的平衡因子*/
int balanceFactor(Node* node) {
    int R = 0;
    int L = 0;
    if (node -> left)
        L = getHeight(node->left);
    if (node -> right)
        R = getHeight(node->right);

    return L - R;  //因为调用此函数之前，一定使node指向非空的结点，所以不用下面注释的一句
    // return node ? L - R : 0;
}


/*
 *     4           4           2
 *    / \          |\         / \
 *   2   5       2 | 5      1   4
 *  / \         / \|           / \
 * 1   3       1   3          3   5
 */

/*右旋操作*/
Node* rotateRight(Node* node) {
    Node* left  = node -> left;
    Node* right = node -> right;

    node->left  = left->right;
    left->right = node;

    // 感觉需要下面两句，但是会报错
    // left->height  = max(getHeight(left->left), getHeight(left->right)) + 1;
    // right->height = max(getHeight(right->left), getHeight(right->right)) + 1;

    return left;
}


/*
 *   2           2            4
 *  / \         /|           / \
 * 1   4       1 | 4        2   5
 *    / \        |/ \      / \
 *   3   5       3   5    1   4
 */

/*左旋操作*/
Node* rotateLeft(Node* node) {
    Node* left  = node->left;
    Node* right = node->right;

    node->right = right->left;
    right->left = node;
```

```
87
88      // 感觉需要下面两句，但是会报错
89      // left->height  = max(getHeight(left->left), getHeight(left->right)) + 1;
90      // right->height = max(getHeight(right->left), getHeight(right->right)) + 1;
91
92      return right;
93  }
94
95  /*平衡化操作*/
96  Node* rebalance(Node* &node) {
97      int factor = balanceFactor(node);
98      if (factor > 1 && balanceFactor(node->left) > 0)           // LL
99          return rotateRight(node);
100     else if (factor < -1 && balanceFactor(node->right) <= 0)   // RR
101         return rotateLeft(node);
102     else if (factor >  1 && balanceFactor(node->left) <= 0) {  // LR
103         node->left = rotateLeft(node->left);
104         return rotateRight(node);
105     }
106     else if (factor < -1 && balanceFactor(node->right) > 0) {  // RL
107         node->right = rotateRight(node->right);
108         return rotateLeft(node);
109     }
110     else  // Nothing happened.
111         return node;
112 }
113
114 /*使用了递归，二叉搜索树那里使用的是迭代，结果相同，只是这里使用了rebalance进行平衡化*/
115 void treeInsert(Node* &root, int value) {
116     if (root == NULL) {
117         Node* newNode = (Node*)malloc(sizeof(Node));
118         newNode->data = value;
119         newNode->left = newNode->right = NULL;
120         newNode->height = 1;
121         root = newNode;
122     }
123     else if (root->data == value)
124         return;
125     else {
126         if (root->data < value)
127             treeInsert(root->right, value);
128         else
129             treeInsert(root->left, value);
130     }
131     if (root)
132         root -> height = getHeight(root);
133     root = rebalance(root); // 为什么要使用root =，root实际上是递归的root -> left，这里更新了root，也就是使root
    -> left指向新的结点
134 }
135
136 /*删除操作很难，这部分代码错误*/
137 /*如果当前结点需要删除，而它又有两个子结点，那么就无法删除当前结点*/
138 void treeDelete(Node* root, int value) {
139     Node* toFree;
140     if (root) {
141         if (root->data == value) {
142             if (root->right)
143                 treeDelete(root->right, value);
144             else {
145                 toFree = root;
146                 root = toFree->left;
147                 free(toFree);
148             }
149         }
150         else {
151             if(root->data < value)
152                 treeDelete(root->right, value);
153             else
154                 treeDelete(root->left, value);
155         }
```

```
        rebalance(root);
    }
}

void preorder(Node* node) {
    if (node != NULL) {
        printf("%d ", /* node.data */ node -> data);
        // printf("%d\n", [> node.data <] node -> height);
        preorder(node -> left );
        preorder(node -> right);
    }
}

void inorder(Node* node) {
    if (node != NULL) {
        inorder(node -> left );
        printf("%d ", node -> data);
        inorder(node -> right);
    }
}

void postorder(Node* node) {
    if (node != NULL) {
        postorder(node -> left );
        postorder(node -> right);
        printf("%d ", node -> data);
    }
}

int main() {
    Tree tree;
    tree.root = NULL;
    int arr[] = {8, 3, 5, 1, 0, 2, 9, 4, 7, 6};
    for (int i = 0; i < 10; ++i)
        treeInsert(tree.root, arr[i]);
    preorder(tree.root);  printf("\n");
    inorder(tree.root);   printf("\n");
    postorder(tree.root); printf("\n");
    return 0;
}
```

7.10.5 Treap

7.10.5.1 treap.cpp

```
/*------------------------------------------------------------------
 *
 *   文件名称: treap.cpp
 *   创建日期: 2021年03月06日 ---- 19时58分
 *   题    目: hdu4585
 *   算    法: treap
 *   描    述: treap与名次树问题
 *
------------------------------------------------------------------*/

#include <cstdio>
#include <cstdlib>
#include <ctime>
const int maxn = 5e6 + 5;
int id[maxn];

struct Node{
    int size;  //以这个结点为根的子树的结点数，用于名次树
    int rank;  //priority
    int data;  //键值
    Node* son[2];
    bool operator < (const Node &x)const {return rank < x.rank;}
    int cmp(int x)const {
        if (x == data)
            return -1;
```

```
        return x < data ? 0 : 1;
    }
    void update() {  //更新size
        size = 1;
        if (son[0] != NULL)
            size += son[0] -> size;
        if (son[1] != NULL)
            size += son[1] -> size;
    }
};

/*d = 0左旋, d = 1右旋*/
void rotate(Node* &o, int d) {
    Node* k = o -> son[d^1];
    o -> son[d^1] = k -> son[d];
    k -> son[d] = o;
    o -> update();
    k -> update();
    o = k;
}

/*插入x*/
void insert(Node* &o, int x) {
    if (o == NULL) {
        o = new Node();
        o -> son[0] = o -> son[1] = NULL;
        o -> rank = rand();
        o -> data = x;
        o -> size = 1;
    }
    else {
        int d = o->cmp(x);
        insert(o->son[d], x);
        o -> update();
        if (o < o->son[d])
            rotate(o, d^1);
    }
}

/*第k大的数*/
int kth(Node* o, int k) {
    if (o == NULL || k <= 0 || k > o->size)
        return 01;
    int s = (o->son[1] == NULL) ? 0 : o->son[1]->size;
    if (k == s+1)
        return o -> data;
    else if (k <= s)
        return kth(o->son[1], k);
    else
        return kth(o->son[0], k-s-1);
}

/*返回k的名次*/
int find(Node* o, int k) {
    if (o == NULL)
        return -1;
    int d = o->cmp(k);
    if (d == -1)
        return o->son[1] == NULL ? 1 : o->son[1]->size + 1;
    else if (d == 1)
        return find(o->son[d], k);
    else {
        int tmp = find(o->son[d], k);
        if (tmp == -1)
            return -1;
        else
            return o->son[1] == NULL ? tmp+1 : tmp+1+o->son[1]->size;
    }
}

```

```
int main() {
    int n;
    while (~scanf("%d", &n) && n) {
        srand(time(NULL));
        int k, g;
        scanf("%d %d", &k, &g);
        Node* root = (Node *)malloc(sizeof(Node));
        // Node* root = new Node();
        root -> son[0] = root -> son[1] = NULL;
        root -> rank = rand();
        root -> data = g;
        root -> size = 1;
        id[g] = k;
        printf("%d %d\n", k, 1);
        for (int i = 2; i <= n; ++i) {
            scanf("%d %d", &k, &g);
            id[g] = k;
            insert(root, g);
            int t = find(root, g);
            int ans1, ans2, ans;
            ans1 = kth(root, t-1);
            ans2 = kth(root, t+1);
            if (ans1 != -1 && ans2 != -1)
                ans = (ans1 - g >= g - ans2) ? ans2 : ans1;
            else if (ans1 == -1)
                ans = ans2;
            else
                ans = ans1;
            printf("%d %d\n", k, id[ans]);
        }
    }
    return 0;
}
```

7.10.5.2 treap-OIwiki.cpp

```
/*----------------------------------------------------------------
 *
 *   From OIwiki
 *   文件名称：treap.cpp
 *   创建日期：2021年03月06日 ---- 12时32分
 *   算    法：treap
 *   描    述：treap是一种简单的平衡二叉搜索树
 *   treap除了要满足二叉搜索树的性质之外，还需满足父节点的 priority大于等于两个儿子的priority
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <utility>
#include <cstdlib>
using namespace std;
struct Node{
    int data;
    int priority;
    Node* left;
    Node* right;
};

struct Tree{
    Node* root;
};

/**
 * 分裂
 * pair: 第一个元素存放分裂后的右子树，第二个元素存放分裂后的左子树
 */
pair<Node*, Node*> split(Node *root, int val) {
    if (root == nullptr)
        return make_pair(nullptr, nullptr);
```

```
34
35      if (val < root->data) {
36          pair<Node*, Node*> pir = split(root->left, val);
37          root->left = pir.second;
38          return make_pair(pir.first, root);
39      }
40      else {
41          pair<Node*, Node*> pir = split(root->right, val);
42          root->right = pir.first;
43          return make_pair(root, pir.second);
44      }
45  }
46
47  /**
48   * 合并
49   * 首先要满足rootR上结点值都大于rootL上的结点值
50   * 其次，默认条件是每棵树上的根结点优先级最高
51   * 通过比较priority合并两棵树
52   * 所以合并时，rootR一定放在右边
53   * 因为也要满足右结点 > 根结点 > 左结点，两者都要满足
54   */
55  Node* merge(Node* rootL, Node* rootR) {
56      if (rootR == nullptr)
57          return rootL;
58      if (rootL == nullptr)
59          return rootR;
60
61      if (rootR->priority > rootL->priority) {
62          rootR->left = merge(rootL, rootR->left);
63          return rootL;
64      }
65      else {
66          rootL->right = merge(rootL->right, rootR);
67          return rootR;
68      }
69  }
70
71  void preorder(Node* node) {
72      if (node != NULL) {
73          printf("%d ", /* node.data */ node -> data);
74          preorder(node -> left );
75          preorder(node -> right);
76      }
77  }
78
79  void inorder(Node* node) {
80      if (node != NULL) {
81          inorder(node -> left );
82          printf("%d ", node -> data);
83          inorder(node -> right);
84      }
85  }
86
87  void postorder(Node* node) {
88      if (node != NULL) {
89          postorder(node -> left );
90          postorder(node -> right);
91          printf("%d ", node -> data);
92      }
93  }
94
95  int main() {
96      Tree tree;
97      Node* node5 = (Node *)malloc(sizeof(Node));
98      node5 -> data = 5;
99      tree.root = node5;
100
101     Node* node2 = (Node *)malloc(sizeof(Node));
102     node2 -> data = 2;
103     tree.root -> left = node2;
```

```

    Node* node1 = (Node *)malloc(sizeof(Node));
    node1 -> data = 1;
    tree.root -> left -> left = node1;

    Node* node4 = (Node *)malloc(sizeof(Node));
    node4 -> data = 4;
    tree.root -> left -> right = node4;

    Node* node7 = (Node *)malloc(sizeof(Node));
    node7 -> data = 7;
    tree.root -> right = node7;

    Node* node6 = (Node *)malloc(sizeof(Node));
    node6 -> data = 6;
    tree.root -> right -> left = node6;

    Node* node9 = (Node *)malloc(sizeof(Node));
    node9 -> data = 9;
    tree.root -> right -> right = node9;

    Node* node8 = (Node *)malloc(sizeof(Node));
    node8 -> data = 8;
    tree.root -> right -> right -> left = node8;

    pair<Node*, Node*> pir = split(tree.root, 3);

    preorder(pir.first);
    printf("\n");
    inorder(pir.first);
    printf("\n");

    preorder(pir.second);
    printf("\n");
    inorder(pir.second);
    printf("\n");
    return 0;
}
```

7.10.6 Splay

7.10.6.1 splay.cpp

```
/*--------------------------------------------------------------------
 *
 *   文件名称：splay.cpp
 *   创建日期：2021年03月06日 ---- 21时47分
 *   题    目：hdu1890
 *   算    法：splay
 *   描    述：<++>
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int root;
int rev[maxn];  //标记i被翻转
int pre[maxn];  //i的父结点
int size[maxn]; //i的子树上结点的个数
int tree[maxn][2]; //tree[i][o],i的左儿子; tree[i][1],i的右儿子
struct Node{
    int data;
    int id;
    bool operator <(const Node &x)const {
        if (data == x.data)
            return id < x.id;
        return data < x.data;
    }
```

```
}node[maxn];

/*计算以x为根的子树包含了多少个子结点*/
void pushup(int x) {
    size[x] = size[tree[x][0]] + size[tree[x][1]] + 1;
}

void update_rev(int x) {
    if (!x)
        return;
    swap(tree[x][0], tree[x][1]);
    rev[x] ^= 1; //标记x被翻转
}

/*本题需要，处理机械臂的翻转*/
void pushdown(int x) {
    if (rev[x]) {
        update_rev(tree[x][0]);
        update_rev(tree[x][1]);
        rev[x] = 0;
    }
}

/*旋转，c = 0左旋; c = 1右旋*/
void rotate(int x, int c) {
    int y = pre[x];
    pushdown(y);
    pushdown(x);
    tree[y][!c] = tree[x][c];
    pre[tree[x][c]] = y;
    if (pre[y])
        tree[pre[y]][tree[pre[y]][1] == y] = x;
    pre[x] = pre[y];
    tree[x][c] = y;
    pre[y] = x;
    pushup(y);
}

/*把结点x旋转为goal的儿子，如果goal是0，则旋转到根*/
void splay(int x, int goal) {
    pushdown(x);
    while (pre[x] != goal) {        //一直旋转，直到x成为goal的儿子
        if (pre[pre[x]] == goal) {  //情况1: x的父结点是根，单旋一次就行
            pushdown(pre[x]);
            pushdown(x);
            rotate(x, tree[pre[x]][0] == x);
        }
        else {                      //x的父结点不是根
            pushdown(pre[pre[x]]);
            pushdown(pre[x]);
            pushdown(x);
            int y = pre[x];
            int c = (tree[pre[y]][0] == y);
            if (tree[y][c] == x) {  //情况2: x、x的父、x父的父不共线
                rotate(x, !c);
                rotate(x, c);
            }
            else {                  //情况3: x、x的父、x父的父共线
                rotate(y, c);
                rotate(x, c);
            }
        }
    }
    pushup(x);
    if (goal == 0)                  //如果goal是0，则将根结点更新为x
        root = x;
}

int get_max(int x) {
    pushdown(x);
```

```
    while (tree[x][1]) {
        x = tree[x][1];
        pushdown(x);
    }
    return x;
}

void del_root() {
    if (tree[root][0] == 0) {
        root = tree[root][1];
        pre[root] = 0;
    }
    else {
        int m = get_max(tree[root][0]);
        splay(m, root);
        tree[m][1] = tree[root][1];
        pre[tree[root][1]] = m;
        root = m;
        pre[root] = 0;
        pushup(root);
    }
}

void newnode(int &x, int fa, int val) {
    x = val;
    pre[x] = fa;
    size[x] = 1;
    rev[x] = 0;
    tree[x][0] = tree[x][1] = 0;
}

void buildtree(int &x, int L, int R, int fa) {
    if (L > R)
        return;
    int mid = (L + R) >> 1;
    newnode(x, fa, mid);
    buildtree(tree[x][0], L, mid-1, x);
    buildtree(tree[x][1], mid+1, R, x);
    pushup(x);
}

void init(int n) {
    root = 0;
    tree[root][0] = tree[root][1] = pre[root] = size[root] = 0;
    buildtree(root, 1, n, 0);
}

int main() {
    int n;
    while (~scanf("%d", &n) && n) {
        init(n);
        for (int i = 1; i <= n; ++i)
            scanf("%d", &node[i].data);
        sort(node+1, node+n+1);
        for (int i = 0; i < n; ++i) {
            splay(node[i].id, 0);
            update_rev(tree[root][0]);
            printf("%d", i + size[tree[root][0]]);
            del_root();
        }
        printf("%d\n", n);
    }
    return 0;
}
```

7.10.7 替罪羊树

7.10.7.1 替罪羊树.cpp

7.11 K-D-Tree

7.11.1 K-D-Tree.cpp

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

#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;
}
```

7.11.2 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 DFS-邻接矩阵.cpp

```
#include <cstdio>
#include <cstring>
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;
}
```

8.2.2 DFS-邻接表.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;
}
```

8.2.3 DFS-链式前向星.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() {
    freopen("in/in-02.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);
    }
    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

```
#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

```

```cpp
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

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

```
/*-----------------------------------------------------------------
 *
 *   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>
#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

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

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

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

```
 *
 *    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

```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>
#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

```
/*----------------------------------------------------------------
 *
 *   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>
#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

```
/*-----------------------------------------------------------------
 *
 *   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]);
}

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;
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

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

#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 匈牙利算法.cpp

```
/*----------------------------------------------------------------
 *
 *   文件名称：01-匈牙利算法.cpp
 *   创建日期：2021年03月29日 ---- 18时51分
 *   题    目：hdu2063
 *   算    法：匈牙利算法
 *   描    述：<++>
 *
 ----------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
using namespace std;
int g[510][510];
int match[510];
int reserve_boy[510];     //匹配结果在match[]中
int k, m_girl, n_boy;

//找一个增广路径，即给女孩x找一个配对男孩
bool dfs(int x) {
    for (int i = 1; i<=n_boy; i++)
        if (!reserve_boy[i] && g[x][i]) {
            reserve_boy[i] = 1;      // 预定男孩i，准备分给女孩x
            if (!match[i] || dfs(match[i])) {
                /*
                 * 有两种情况：(1)如果男孩i还没配对，就分给女孩x；
                 *             (2)如果男孩i已经配对，尝试用dfs()更换原配女孩，以腾出位置给女孩x
                 */
                match[i] = x;
                //配对成功；如果原来有配对，更换成功。现在男孩i属于女孩x
                return true;
            }
        }
    return false;      //女孩x没有喜欢的男孩，或者更换不成功
}

int main() {
    while (~scanf("%d",&k) && k) {
        scanf("%d %d", &m_girl, &n_boy);
        memset(g, 0, sizeof(g));
        memset(match, 0, sizeof(match));
        for (int i = 0; i < k; ++i) {
            int a, b;
            scanf("%d %d", &a, &b);
            g[a][b] = 1;
        }
        int sum = 0;
```

```
        for (int i = 1; i <= m_girl; ++i) {          //为每个女孩找配对
            memset(reserve_boy, 0, sizeof(reserve_boy));
            if(dfs(i))
                sum++;
            //第i个女孩配对成功，这个配对后面可能更换，但是保证她能配对
        }
        printf("%d\n",sum);
    }
    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>
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

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

```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

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

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

```cpp
#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

```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

```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>
```

```cpp
#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

```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];
```

```cpp
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);
        }
    }
};
```

```
80
81  typedef Line Segment;   //定义线段，两端点是Point p1,p2
82
83  /*返回直线倾斜角 0 <= angle < pi*/
84  double Line_angle(Line v) {
85      double k = atan2(v.p2.y-v.p1.y, v.p2.x-v.p1.x);
86      if (sgn(k) < 0) k += pi;
87      if (sgn(k-pi) == 0) k -= pi;
88      return k;
89  }
90
91  /*点和直线关系:1 在左侧;2 在右侧;0 在直线上*/
92  int Point_line_relation(Point p, Line v) {
93      int c = sgn(Cross(p-v.p1, v.p2-v.p1));
94      if (c < 0) return 1;      //1：p在v的左边
95      if (c > 0) return 2;      //2：p在v的右边
96      return 0;                 //0：p在v上
97  }
98  /*点和线段的关系：0 点p不在线段v上；1 点p在线段v上*/
99  bool Point_on_seg(Point p, Line v) {
100     return sgn(Cross(p-v.p1, v.p2-v.p1)) == 0 && sgn(Dot(p - v.p1, p- v.p2)) <= 0;
101 }
102
103 /*两直线关系:0 平行,1 重合,2 相交*/
104 int Line_relation(Line v1, Line v2) {
105     if (sgn(Cross(v1.p2-v1.p1, v2.p2-v2.p1)) == 0) {
106         if (Point_line_relation(v1.p1, v2) == 0)  return 1; //1 重合
107         else return 0;  //0 平行
108     }
109     return 2;           //2 相交
110 }
111
112 /*点到直线的距离*/
113 double Dis_point_line(Point p, Line v) {
114     return fabs(Cross(p-v.p1, v.p2-v.p1))/Distance(v.p1, v.p2);
115 }
116
117 /*点在直线上的投影*/
118 Point Point_line_proj(Point p, Line v) {
119     double k = Dot(v.p2-v.p1, p-v.p1)/Len2(v.p2-v.p1);
120     return v.p1+(v.p2-v.p1)*k;
121 }
122
123 /*点p对直线v的对称点*/
124 Point Point_line_symmetry(Point p, Line v) {
125     Point q = Point_line_proj(p, v);
126     return Point(2*q.x-p.x, 2*q.y-p.y);
127 }
128
129 /*点到线段的距离*/
130 double Dis_point_seg(Point p, Segment v) {
131     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
132         return min(Distance(p, v.p1), Distance(p, v.p2));
133     return Dis_point_line(p, v); //点的投影在线段上
134 }
135
136 /*求两直线ab和cd的交点*/
137 /*调用前要保证两直线不平行或重合*/
138 Point Cross_point(Point a, Point b, Point c, Point d) { //Line1:ab,  Line2:cd
139     double s1 = Cross(b-a, c-a);
140     double s2 = Cross(b-a, d-a);  //叉积有正负
141     return Point(c.x*s2-d.x*s1, c.y*s2-d.y*s1)/(s2-s1);
142 }
143 /*两线段是否相交：1 相交，0不相交*/
144 bool Cross_segment(Point a, Point b, Point c, Point d) { //Line1:ab,  Line2:cd
145     double c1 = Cross(b-a, c-a), c2 = Cross(b-a, d-a);
146     double d1 = Cross(d-c, a-c), d2 = Cross(d-c, b-c);
147     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
148 }
149
```

```
//---------------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) {
```

```
290     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));
291 }
292 double Angle(Vector3 A, Vector3 B) {return acos(Dot(A, B)/Len(A)/Len(B));}   //A与B的夹角
293 /*三维：线*/
294 struct Line3 {
295     Point3 p1, p2;
296     Line3() {}
297     Line3(Point3 p1, Point3 p2) : p1(p1), p2(p2) {}
298 };
299 typedef Line3 Segment3;   //定义线段，两端点是Point p1,p2
300
301 /*三角形面积的2倍*/
302 double Area2(Point3 A, Point3 B, Point3 C) {return Len(Cross(B-A, C-A));}
303
304 /*三维：点到直线距离*/
305 double Dis_point_line(Point3 p, Line3 v) {
306     return Len(Cross(v.p2-v.p1, p-v.p1))/Distance(v.p1, v.p2);
307 }
308
309 /*三维：点在直线上*/
310 bool Point_line_relation(Point3 p, Line3 v) {
311     return sgn( Len(Cross(v.p1-p, v.p2-p))) == 0 && sgn(Dot(v.p1-p, v.p2-p)) == 0;
312 }
313 /*三维：点到线段距离*/
314 double Dis_point_seg(Point3 p, Segment3 v) {
315     if (sgn(Dot(p- v.p1, v.p2-v.p1)) < 0 || sgn(Dot(p- v.p2, v.p1-v.p2)) < 0)
316         return min(Distance(p, v.p1), Distance(p, v.p2));
317     return Dis_point_line(p, v);
318 }
319 /*三维：点 p 在直线上的投影*/
320 Point3 Point_line_proj(Point3 p, Line3 v) {
321     double k = Dot(v.p2-v.p1, p-v.p1)/Len2(v.p2-v.p1);
322     return v.p1+(v.p2-v.p1)*k;
323 }
324 /*三维：平面*/
325 struct Plane {
326     Point3 p1, p2, p3;//平面上的三个点
327     Plane() {}
328     Plane(Point3 p1, Point3 p2, Point3 p3) : p1(p1), p2(p2), p3(p3) {}
329 };
330 /*平面法向量*/
331 Point3 Pvec(Point3 A, Point3 B, Point3 C) {return Cross(B-A,C-A);}
332 Point3 Pvec(Plane f) {return Cross(f.p2-f.p1, f.p3-f.p1);}
333 /*四点共平面*/
334 bool Point_on_plane(Point3 A, Point3 B, Point3 C, Point3 D) {
335     return sgn(Dot(Pvec(A, B, C), D-A)) == 0;
336 }
337 /*两平面平行*/
338 int Parallel(Plane f1, Plane f2) {
339     return Len(Cross(Pvec(f1), Pvec(f2))) < eps;
340 }
341 /*两平面垂直*/
342 int Vertical (Plane f1, Plane f2) {
343     return sgn(Dot(Pvec(f1), Pvec(f2)))==0;
344 }
345 /*直线与平面的交点p，返回值是交点个数*/
346 int Line_cross_plane(Line3 u, Plane f, Point3 &p) {
347     Point3 v = Pvec(f);
348     double x = Dot(v, u.p2-f.p1);
349     double y = Dot(v, u.p1-f.p1);
350     double d = x-y;
351     if (sgn(x) == 0 && sgn(y) == 0) return -1;//-1：v在f上
352     if (sgn(d) == 0) return 0;                  //0：v与f平行
353     p = ((u.p1*x) - (u.p2*y))/d;             //v与f相交
354     return 1;
355 }
356
357 /*四面体有向体积*6*/
358 double volume4(Point3 A, Point3 B, Point3 C, Point3 D) {return Dot(Cross(B-A, C-A), D-A);}
359
```

```
360
361 int main() {
362     Point a(0, 1), b(0, 0), c(1, 1), d(1, 2), p(1.5, 1);
363     Line k(a, b), k2(c, d);
364     Point pr(1, 1), cr(1, 1); double r = 1;
365     Circle C(cr, r);
366
367     cout << endl << "Line_circle_relation=" << Line_circle_relation(k, C);
368     cout << endl << "Seg_circle_relation=" << Seg_circle_relation(k, C);
369     cout << endl << "Point_circle_relation=" << Point_circle_relation(pr, C);
370     cout << endl << "parallel=" << Parallel(a, b) << endl;
371     cout << "dot=" << Dot(a, b) << endl << " angle=" << Angle(a, b) << endl;
372     cout << "90:" << sgn(Rotate(a, -pi/2).x) << endl << Rotate(a, -pi/2).y;
373     cout << endl << "line angle=" << Line_angle(k)*4;
374     cout << endl << "line place=" << Point_line_relation(p, k);
375     cout << endl << "point_on_seg=" << Point_on_seg(p, k);
376     cout << endl << "dis_point_line=" << Dis_point_line(p, k);
377     cout << endl << "dis_point_line=" << Dis_point_seg(p, k);
378     Point pp = Cross_point(a, b, c, d);
379     cout << endl << "crosspoint=" << pp.x << "  " << pp.y;
380     cout << endl << "cross seg=" << Cross_segment(a, b, c, d);
381     cout << endl << "distance=" << Distance(a, b);
382     cout << endl << "line_relation=" << Line_relation(k, k2);
383     Point g[4];
384     g[0] = a; g[1] = b; g[2] = c; g[3] = d;
385     cout << endl << "Point_in_polygon=" << Point_in_polygon(p, g, 4);
386     cout << endl << "Polygon_area=" << Polygon_area(g, 4);
387     cout << endl << endl;
388     return 0;
389 }
```