该库用来记录在 acwing 的代码模板

第一章 基础算法(一)

排序

快速排序

```cpp
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int q[N];

void quick_sort(int q[], int l, int r) {
    if (l >= r) return ;

    int x = q[l + r >> 1], i = l - 1, j = r + 1; //如果超时的话建议修改一下中值 x

    while (i < j) {
        do i ++; while (q[i] < x);
        do j --; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j);
    quick_sort(q, j + 1, r);
}

int main() {
    int n;

    scanf("%d", &n);

    for (int i = 0; i < n; i ++) scanf("%d", &q[i]);

    quick_sort(q, 0, n - 1);

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

    return 0;
}
```

归并排序

```
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int n;

int q[N], temp[N];

void merge_sort(int q[], int l, int r)
{
    //递归的终止情况
    if(l >= r) return;

    //第一步：分成子问题
    int mid = l + r >> 1;

    //第二步：递归处理子问题
    merge_sort(q, l, mid ), merge_sort(q, mid + 1, r);

    //第三步：合并子问题
    int k = 0, i = l, j = mid + 1, tmp[r - l + 1];
    while(i <= mid && j <= r) {
        if(q[i] <= q[j]) tmp[k++] = q[i++];
        else tmp[k++] = q[j++];
    }
    while(i <= mid) tmp[k++] = q[i++];
    while(j <= r) tmp[k++] = q[j++];

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

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

    for (int i = 0; i < n; i ++)
        scanf("%d", &q[i]);

    merge_sort(q, 0, n - 1);

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

    return 0;
}
```

二分

数的范围(整数二分)

```
bool check(int x) {/* ... */} // 检查 x 是否满足某种性质

// 区间 [l, r] 被划分成 [l, mid] 和 [mid + 1, r] 时使用:
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check() 判断 mid 是否满足性质
        else l = mid + 1;
    }
    return l;
}
// 区间 [l, r] 被划分成 [l, mid - 1] 和 [mid, r] 时使用:
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}


#include <iostream>

using namespace std;

const int N = 100010;

int n, m;
int q[N];

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

    for (int i = 0; i < n; i ++) scanf("%d", &q[i]);

    while (m --) {
        int x;

        scanf("%d", &x);

        int l = 0, r = n - 1;

        while (l < r) {
            int mid = l  + r >> 1;

            if (q[mid] >= x) r = mid;
```

```
            else l = mid + 1;
        }

        if (q[l] != x) cout << "-1 -1" << endl;
        else {
            cout << l << ' ';

            int l = 0, r = n - 1;

            while (l > r) {
                int mid = l + r + 1 >> 1;

                if (q[mid] <= x) l = mid;
                else r = mid - 1;
            }

            cout << l << endl;
        }
    }

    return 0;
}
```

数的三次方根(浮点数二分)

```
bool check(double x) {/* ... */} // 检查 x 是否满足某种性质

double bsearch_3(double l, double r)
{
    const double eps = 1e-6;   // eps 表示精度 取决于题目对精度的要求
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}

#include <iostream>

using namespace std;

int main () {
    double x;

    cin >> x;

    double l = 0, r = x;

    while (r - l > 1e-8) {
        double mid = (l + r) / 2;
```

```
        if (mid * mid >= x) r = mid;
        else l = mid;
    }

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

    return 0;
}
```

第一章 基础算法(二)

高精度

高精度加法

```
#include <iostream>
#include <vector>

using namespace std;

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() {
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i -- ) B.push_back(b[i] - '0');

    auto C = add(A, B);

    for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];
    cout << endl;

    return 0;
}
```

高精度减法

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

using namespace std;

bool cmp(vector<int> &A, vector<int> &B) {
    if (A.size() != B.size()) return A.size() > B.size();

    for (int i = A.size() - 1; i >= 0; i -- )
        if (A[i] != B[i])
            return A[i] > B[i];

    return true;
}

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);
        if (t < 0) t = 1;
        else t = 0;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back(); // 去掉前导零
    return C;
}

int main() {
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i -- ) B.push_back(b[i] - '0');

    vector<int> C;

    if (cmp(A, B)) C = sub(A, B);
    else C = sub(B, A), cout << '-';

    for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];
    cout << endl;

    return 0;
}
```

高精度乘法

```
#include <iostream>
#include <vector>

using namespace std;


vector<int> mul(vector<int> &A, int b) {
    vector<int> C;

    int t = 0; // 进位
    for (int i = 0; i < A.size() || t; i ++ ) { // 当数字没有处理完或者还有进位没有处理完，就一直循环
        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(); // 用来考虑 b = 0 的情况的，如果 b != 0 是不是就不用加

    return C;
}


int main() {
    string a;
    int b;

    cin >> a >> b;

    vector<int> A;
    for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');

    auto C = mul(A, b);

    for (int i = C.size() - 1; i >= 0; i -- ) printf("%d", C[i]);

    return 0;
}
```

高精度除法

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// A / B 商是 C 余数是 r
vector<int> div(vector<int> &A, int b, int &r) { // r 是通过引用传回去的
    vector<int> C;
```

```
    r = 0;
    // 从最高位开始看
    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(); // 去前导零
    return C;
}

int main() {
    string a;
    vector<int> A;

    int B;
    cin >> a >> B;
    for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[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;

    return 0;
}
```

前缀和

一维前缀和

```
#include <iostream>

using namespace std;

const int N = 100010;

int n, m;
int a[N], s[N];

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

    for (int i = 1; i <= n; i ++ ) s[i] = s[i - 1] + a[i]; // 前缀和的初始化

    while (m -- )
    {
        int l, r;
```

```
        scanf("%d%d", &l, &r);
        printf("%d\n", s[r] - s[l - 1]); // 区间和的计算
    }

    return 0;
}
```

二维前缀和

```
#include <iostream>

using namespace std;

const int N = 1010;

int a[N][N], s[N][N];

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

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
            scanf("%d", &a[i][j]);
            s[i][j] = s[i][j - 1] + s[i - 1][j] - s[i - 1][j - 1] + a[i][j]; // 求前缀
和
        }

    while (q--) {
        int x1,y1,x2,y2;
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        // 算子矩阵的和
        printf("%d\n", s[x2][y2] - s[x2][y1 - 1] - s[x1 - 1][y2] + s[x1 - 1][y1 - 1]);
    }

    return 0;
}
```

差分

一维差分

```
#include <iostream>

using namespace std;

const int N = 100010;

int n, m;
int a[N], b[N];
```

```
void insert(int l, int r, int c)
{
    b[l] += c;
    b[r + 1] -= c;
}

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

    for (int i = 1; i <= n; i ++ ) insert(i, i, a[i]);

    while (m -- )
    {
        int l, r, c;
        scanf("%d%d%d", &l, &r, &c);
        insert(l, r, c);
    }

    for (int i = 1; i <= n; i ++ ) b[i] += b[i - 1];

    for (int i = 1; i <= n; i ++ ) printf("%d ", b[i]);

    return 0;
}
```

二维差分

```
#include <iostream>

using namespace std;

const int N = 1010;

int n, m, q;
int a[N][N], b[N][N];

void insert(int x1, int y1, int x2, int y2, int c)
{
    b[x1][y1] += c;
    b[x2 + 1][y1] -= c;
    b[x1][y2 + 1] -= c;
    b[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", &a[i][j]);

    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= m; j ++ )
            insert(i, j, i, j, a[i][j]);

    while (q -- )
    {
        int x1, y1, x2, y2, c;
        cin >> 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 ++ )
            b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1];

    for (int i = 1; i <= n; i ++ )
    {
        for (int j = 1; j <= m; j ++ ) printf("%d ", b[i][j]);
        puts("");
    }

    return 0;
}
```

第一章 基础算法(三)

双指针

最长连续不重复子序列

```
#include <iostream>

using namespace std;

const int N = 100010;

int n;
int q[N], s[N];

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

    int res = 0;
    for (int i = 0, j = 0; i < n; i ++ ) {
        s[q[i]] ++ ;
        while (j < i && s[q[i]] > 1) s[q[j ++ ]] --;
        res = max(res, i - j + 1);
    }
```

```
    cout << res << endl;

    return 0;
}
```

数组元素的目标和

```
#include <iostream>

using namespace std;

const int N = 1e5 + 10;

int n, m, x;
int a[N], b[N];

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

    for (int i = 0, j = m - 1; i < n; i ++ ) {
        while (j >= 0 && a[i] + b[j] > x) j -- ;
        if (j >= 0 && a[i] + b[j] == x) cout << i << ' ' << j << endl;
    }

    return 0;
}
```

位运算

二进制中 1 的个数

```
#include<iostream>

using namespace std;

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

int main() {
    int n;
    cin>>n;
    while (n --) {
        int x;

        cin>>x;

        int res = 0;
```

```
        while(x) x -= lowbit(x), res++;

        cout<<res<<' ';
    }

    return 0;
}
```

离散化

区间化

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef pair<int, int> PII;

const int N = 300010;

int n, m;
int a[N], s[N];

vector<int> alls;
vector<PII> add, query;

int find (int x) {
    int l = 0, r = alls.size();

    while (l < r) {
        int mid = l + r >> 1;

        if (alls[mid] >= x) r = mid;
        else l = mid + 1;
    }

    return r + 1;
}

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

    for (int i = 0; i < n; i ++) {
        int x, c;

        cin >> x >> c;

        add.push_back({x, c});
        alls.push_back(x);
    }
```

```
    for (int i = 0; i < m; i ++) {
        int l, r;

        cin >> l >> r;

        query.push_back({l, r});

        alls.push_back(l);
        alls.push_back(r);
    }

    // 去除 alls 中重复的元素
    sort(alls.begin(), alls.end());
    alls.erase(unique(alls.begin(), alls.end()), alls.end());

    // 处理插入
    for (auto item : add) {
        int x = find(item.first);
        a[x] += item.second;

    }

    // 预处理前缀和
    for (int i = 1; i <= alls.size(); i ++) s[i] = s[i - 1] + a[i];

    // 处理询问
    for (auto item : query) {
        int l = find(item.first), r = find(item.second);

        cout << s[r] - s[l + 1] << endl;
    }

    return 0;
}
```

区间合并

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

typedef pait<int, int> PII;

const int N = 100010;

int n;
vector<PII> segs;

void merge(vector<PII> &segs) {
    vector<PII> res;

    sort(segs.begin(), segs.end());
```

```
    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 () {
    cin >> n;

    for (int i = 0; i < n; i ++) {
        int l, r;

        cin >> l >> r;

        segs.push_back({l, r});
    }

    merge(segs);

    cout << segs.size() << endl;

    return 0;
}
```

第二章 数据结构(一)

单链表

```
#include <iostream>

using namespace std;

const int N = 1000010;

// head 表示头节点 e[i] 表示节点 i 的值 ne[i] 表示节点 i 的 next 指针是多少
// idx 存储当前已经用到哪个节点

int head, e[N], ne[N], idx;

// 初始化
init () {
```

```
    head = -1;
    idx = 0;
}

// 将x插到头节点
void add_to_head (int x) {
    e[idx] = x, ne[idx] = head, head = idx, idx ++;
}

// 将 x 插到下标是 k 的点后面
void add (int k, int x) {
    e[idx] = x;
    ne[idx] = ne[k];
    ne[k] = idx;
    indx ++;
}

// 将下标是 k 的点后面的点删掉
remove (int k) {
    ne[k] = ne[ne[k]]; // 删除的时候并没有关联到 idx
}

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

    init();

    while (m --) {
        int k, x;
        char op;

        cin >> op;

        if (op == 'H') {
            cin >> x;
            add_to_head(x);
        } else if (op == 'D') {
            cin >> k;
            if (!k) head = ne[head]; // 对头节点的一个特判
            remove(k - 1);
        } else {
            cin >> k >> x;
            add(k - 1, x);
        }
    }

    for (int i = head; i != -1; i = ne[i]) cout << e[i] << " ";

    cout << endl;

    return 0;
}
```

双链表

```
#include <iostream>

using namespace std;

const int N = 100010;

int m;
int e[N], l[N], r[N], idx;

// 初始化
void init () {
    // 0表示左端点 1表示右端点
    r[0] = 1, l[0] = 0;
    idx = 2;
}

// 在下标是 k 的点的右边插入 x (如果是在左边插入 其实也可以直接调用这个 但参数需要换一下)
void add (int k, int x) { //顺序别写反了
    e[idx] = x;
    r[idx] = r[k];
    l[idx] = k;
    l[r[k]] = idx;
    r[k] = idx;
}

// 删除操作
void remove (int k, int x) {
    r[l[k]] = r[k];
    l[r[k]] = l[k];

}
```

模拟栈

```
#include <iostream>

using namespace std;

const int N = 100010;

int stk[N], tt; // tt 表示栈顶元素

// 插入
stk[++ tt] = x;

// 弹出
tt --;

// 判断栈是否为空
if (tt > 0) not empty
```

```
esle empty

// 栈顶
stk[tt];
```

模拟队列

-普通队列

```
#include <iostream>

using namespace std;

const int N = 100010;

int q[N], hh, tt = -1; // hh 表示的是队头 tt 表示的是队尾 (注意这里栈初始的是 -1 而栈初始的是0)

// 插入一个元素
q[ ++ tt] = x;

// 弹出一个元素
hh ++;

// 判断是否为空
if (hh <= tt) not empty
else empty

// 取出队头元素
q[hh]
```

-循环队列

```
// hh 表示队头，tt表示队尾的后一个位置
int q[N], hh = 0, tt = 0;

// 向队尾插入一个数
q[tt ++ ] = x;
if (tt == N) tt = 0;

// 从队头弹出一个数
hh ++ ;
if (hh == N) hh = 0;

// 队头的值
q[hh];

// 判断队列是否为空
if (hh != tt) {

}
```

单调栈

```
#include <iostream>

using namespace std;

const int N = 100010;

int n;
int stk[N], tt;

int main () {
    cin >> n;

    for (int i = 0; i < n; i ++) {
        int x;
        cin >> x;

        while (tt && stk[tt] >= x) tt --;

        if (tt) cout << stk[tt] << " ";
        else cout << -1 << " ";

        stk[ ++ tt] = x;
    }
}
```

单调队列

```
#include <iostream>

using namespace std;

const int N = 1000010;

int n;
int a[N], q[N];

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

    for (int i = 0; i < n; i ++) scanf("%d", &a[i]);


    int hh = 0, tt = -1;

    for (int i = 0; i < n; i ++) {
        // 判断队头是否滑出窗口 q[hh] 里存的是数组下标
        if (hh <= tt && i - k + 1 > q[hh]) hh ++;

        while (hh < = tt && a[q[tt]] >= a[i]) tt --;
```

```
        q[ ++ tt] = i;

        if (i >= k - 1) print("%d", a[q[hh]]);
    }

    puts(" ");

    /*
    如果是求窗口里的最大值

    int hh = 0, tt = -1;

    for (int i = 0; i < n; i ++) {
        // 判断队头是否滑出窗口 q[hh] 里存的是数组下标
        if (hh <= tt && i - k + 1 > q[hh]) hh ++;

        while (hh < = tt && a[q[tt]] <= a[i]) tt --; // 就把这里的符号改一下

        q[ ++ tt] = i;

        if (i >= k - 1) print("%d", a[q[hh]]);
    }

    puts(" ");
    */

    return 0;
}
```

KMP字符串

```
#include <iostream>

using namespace std;

const int N = 10010, M = 1000010;

int n, m;

char P[N], s[M];
int ne[N]; // next 数组

int main () {
    cin >> n >> p + 1 >> m >> s + 1; // 下标从 1 开始

    // 求 next 过程
    for (int i = 2, j = 0; i <= n; i ++) {
        while (j && p[i] != p[j + 1]) j = ne[j];

        if (p[i] == p[j + 1]) j ++;

        ne[i] = j;
```

```
    }

    // kmp 匹配过程
    for (int i = 1, j = 0; i <= m; i ++) {

        while (j && s[i] != p[j + 1]) j = ne[j];

        if (s[i] == p[j + 1]) j ++;

        if (j == n) { // 匹配成功
            printf("%d", i - n);

            j = ne[j]; //
        }
    }
}
```

第二章 数据结构(二)

Trie数(字典树)

```
#include <iostream>

using namespace std;

const int N = 100010;

int son[N][26], cnt[N], idx; // son 是子节点 cnt 以当前这个字母结尾的单词有多个 idx 当前用到的下标 下标是 0 的点既是根节点 又是空节点
char str[N];

// 插入操作
void insert (char str[]) {
    int p = 0; // 从根节点开始

    for (int i = 0; str[i]; i ++) {
        int u = str[i] - 'a';

        if (!son[p][u]) { // 如果当前节点的子节点没有这个字母
            son[p][u] = ++ idx;
        }

        p = son[p][u];
    }

    cnt[p] ++;
}

// 查询操作
int query (char str[]) { // 返回的是这个字符串出现多少次
    int p = 0;
```

```
    for (int i = 0; str[i]; i ++) {
        int u = str[i] - 'a';

        if (!son[p][u]) return 0;

        p = son[p][u];
    }

    return cnt[p];
}

int main () {
    int n;

    scanf("%d", &n);
    while (n --) {
        char op[2];
        scanf("%s%s", op, str);

        if (op[0] == 'I') insert(str);
        else print("%d\n", query(str));
    }

    return 0;
}
```

并查集

```
#include <iostream>

using namespace std;

const int N = 100010;

int n, m;
int p[N]; // 每个元素的父节点是谁

int find (int x) { // 返回 x 所在集合的编号 x 的祖宗节点 + 路劲压缩
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

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

    for (int i = 1; i <= n; i ++) p[i] = i; // 初始的时候每个集合里只有一个元素 因此每个元素都是自己的父节点

    while (m --) {
        char op[2];
        int a, b;

        scanf("%s%d%d". op, &a, &b);
```

```
        if (op[0] == 'M') p[find(a)] == find(b); // 合并集合
        else {
            if (find(a) == find(b)) puts("Yes");
            else puts("No");
        }

    }
    return 0;
}
```

堆

```
#include <iostream>
#include <algorithm> // 导入额外的库

using namespace std;

const int N = 100010;

int n , m;

int h[N], size; // size 表示当前 h 有多少元素

void down (int u) { // down 操作
    int t = u;
    if (u * 2 <= size && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 <= size && h[u * 2 + 1] < h[t]) t = u * 2 + 1;

    if (u != t) { // 如果 u 不等于 t 说明根节点不是最小值
        swap(h[u], h[t]); // 交换一下最小值 继续执行 down 操作
        down(t);
    }
}

void up (int u) { // up 操作
    while (u / 2 && h[u / 2] > h[u]) {
        swap(h[u / 2], h[u]);
        u /= 2;
    }
}

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

    // 构建堆
    for (int i = n / 2; i; i --) down(i);

    while (m --) {
        printf("%d", h[1]); // 输出堆顶元素
        // 维护堆
```

```
        h[1] = h[size];
        size --;
        down(1);
    }

    return 0;
}
```

第二章 数据结构(三)

hash表

拉链法

```
#include <iostream>
#include <cstring> // memset 所在库

using namespace std;

int const N = 100003; // 取模时应该取质数 而且要离 2 的整数幂尽可能的远(这么取出错的概率最小)

int h[N], e[N], ne[], idx;

// 拉链法

void insert (int x) {
    int k = (x % N + N) % N // x % N 如果 x 是一个负数那么余数也是负数 所以 + N 让结果为正再取模

    e[idx] = x, ne[idx] = h[k], h[k] = idx ++;
}

bool find (int x) {
    int k = (x % N + N) % N;

    for (int i = h[k]; i != -1; i = ne[i]) {
        if (e[i] == x) return true;
    }

    return false;
}

int main () {
    int n;

    scanf("%d", &n);

    memset(h, -1, sizeof h); // 将数组清空

    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;
}
```

开放寻址法

```
#include <iostream>
#include <cstring> // memset 所在库

using namespace std;

int const N = 200003, null = 0x3f3f3f3f; // 开到两倍 开放定址法的数组长度一般需要开到题目数据的两到三倍

int h[N];

// 开放寻址法

int find (int x) {
    int k = (x % N + N) % N;

    while (h[k] != null && h[k] != x) {
        k ++;

        if (k == N) k = 0;
    }

    return k;
}

int main () {
    int n;

    scanf("%d", &n);

    memset(h, 0x3f, sizeof h); // 将数组清空 按字节来的 memset 而不是按数

    while (n --) {
        char op[2];
        int x;
        scanf("%s%d", op, &x);

        int k = find(x);
        if (op == 'I') h[k] = x;
```

```
        else {
            if (h[k] != null) puts("Yes");
            else puts("No");
        }
    }

    return 0;
}
```

字符串 hash(快速判断两个字符串是不是相等 O(1)的复杂度)

```
#include <iostream>

using namespace std;

typedef unsigned long long ULL; // 用 ULL 来表示 unsigned long long

const int N = 100010, P = 131; // P 常取 131 或者 13331

int n, m;
char str[N];
ULL h[N], p[N]; // h 数组表示某一前缀的 hash 值 p 数组表示 p 进制

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


int main () {
    scanf("%d%d%s", &n, &m, str + 1);

    p[0] = 1;

    for (int i = 0; i <= n; i ++) {
        p[i] = p[i + 1] * P;

        h[i] = h[i - 1] * P + str[i];
    }

    while (m --) {
        int l1, r1, l2, r2;

        scanf("%d%d%d%d", &l1, &r1, &l2, &r2);

        if (get(l1, r1) == get(l2, r2)) puts("Yes");
        else puts("No");
    }

    return 0;
}
```

STL

常用的 STL

- vector 变长数组 倍增的思想

```
    size()  -> 返回元素个数

    empyt()  -> 返回是否为空

    clear()  -> 清空

    front() / back() -> 返回第一个数 / 返回最后一个数

    push_back() / pop_back() -> 在最后插入一个数 / 把最后一个数删掉

    begin() / end() -> vector 的第 0 个数 / vector 的最后一个数的后面一个数
```

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

using namespace std;

int main () {
    vector<int> a;

    for (int i = 0; i < 10; i ++) a.push_back(i);

    for (int i = 0; i < a.size(); i ++) cout << a[i] << ' ';
    cout << endl;

    // vector 的迭代器来遍历
    for (vector<int> :: iterator i = a.begin(); i != a.end(); i ++) cout << *i << ' ';
// a.begin() 其实就是 a[0] a.end() 就是 a.size()
    cout << endl;

    for (auto i = a.begin(); i != a.end(); i ++) // 也可以把上一个这样写

    for (auto x : a) cout << x << ' ';
    cout << endl;

    // 支持比较运算
    vector<int> a(4, 3) b(3, 4);

    if (a < b) { // 可以比较 vector 之间的大小 按字典序来比

    }

    return 0;
}
```

- pair<int, int> 存储一个二元组

```
    first -> 第一个元素

    second -> 第二个元素

    支持比较运算 也是按字典序 以 first 为第一关键字 以 second 为第二关键字
```

```cpp
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

int main () {
    pair<int, string> p;

    // 初始化 pair 的两种方式
    // 第一种方式
    p = make_pair(10, "wjm");
    //第二种方式
    p = {20, "wjm"};

    // 也可以用 pair 存储三个属性
    pair<int, pair<int, int>> p;

    return 0;
}
```

- string 字符串

```
    substr() -> 返回某一个字串

    c_str() -> 返回 string 对应的字符数组的头指针

    size()

    empty()

    clear()
```

```cpp
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

int main () {
    string a = "wjm";

    // 在字符串后面添加字符
    a += "def";
```

```
    a += "c";

    cout << a << endl;

    cout << a.substr(1, 2) << endl; // 第一个参数是字串的起始位置 第二个参数是字串的长度 当长度超过字符串长度时 输出到末尾为止
    cout << a.substr(1) << endl; // 把第二个参数省略掉 就会返回从 1 开始的整个字符串

    printf("%s\n", a.c_str()) // 这样也可以输出整个字符串

    return 0;

}
```

- queue 队列

```
    push() -> 往队尾插入

    front() -> 返回队头元素

    back() -> 返回队尾元素

    pop() -> 把队头弹出

    size()

    empty()
```

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

using namespace std;

int main () {
    queue<int> q;

    // 如果想要清空 queue 那么重新构造一个就可以了
    q = queue<int>();
}
```

- priority_queue 优先队列(是一个堆 默认是大根堆)

```
    push() -> 往堆里插入一个元素

    top() -> 返回堆顶

    pop() -> 把堆顶弹出
```

```
#include <cstdio>
#include <cstring>
```

```cpp
#include <iostream>
#include <algorithm>
#include <queue>
#include <vector>

int main () {
    priority_queue<int> heap; // 默认是大根堆

    // 如果想是小根堆 那么插入负数 负数就是按从小到大排序
    heap.push(-x);

    // 如果想直接定义小根堆 定义的时候多加两个参数
    priority_queue<int, vector<int>, greater<int>> heap;

    return 0;
}
```

- stack

```
    size()

    empty()

    push() -> 往栈顶添加一个元素

    top() -> 返回栈顶元素

    pop() -> 弹出栈顶元素
```

```
    和队列的操作差不多
```

- deque 双端队列

```
    size()

    empty()

    clear()

    front() -> 返回第一个元素

    back() -> 返回最后一个元素

    push_back() / pop_back() -> 在最后插入一个元素 / 弹出最后一个元素

    push_front() / pop_front() -> 在队首插入一个元素 / 弹出队首元素

    begin() / end()

    []
```

- set map multiset multimap 基于平衡二叉树(红黑树) 动态维护有序数列

```
size()

empty()

clear()

begin() / end() -> ++ / -- 操作 返回前驱和后继  时间复杂度 O(logn)
```

- ** set / multiset

```
insert() -> 插入一个数

find() -> 查找一个数

count() -> 返回某一个数的个数

erase() 有两种参数
    (1) 输入是一个数 x 删除所有 x O(k + logn) k 是 x 的个数
    (2) 输入是迭代器 删除这个迭代器

lower_bound() / upper_bound() 最核心的两个操作
    lower_bound(x) -> 返回大于等于 x 的最小的数的迭代器
    upper_bound(x) -> 返回大于 x 的最小的数迭代器
```

- ** map / multimap

```
insert() -> 插入的数是一个 pair

erase() -> 输入的参数是一个 pair 或者是迭代器

find()

[] 时间复杂度是 O(logn)

lower_bound() / upper_bound()
```

- unordered_set unordered_map unordered_multiset unordered_multimap 没有顺序 基于 hash 表实现的

```
** 和上面类型 增删改查的时间复杂度是 O(1) 不支持 lower_bound() 和 upper_bound() 不支持迭代器的 ++ / -- 操作 和排序有关的操作都是不支持的**
```

- bitset 压位

```
bitset<10000> s; -> <> 里存的是个数

支持位运算操作
~s -> 取反

& | ^(异或) >> << == !=

[] -> 取出某一位
```

```
count() -> 返回有多少个 1

any() / none() -> 判断是否至少有一个 1 / 判断是否全为 0

set() -> 把所有位置改成 1

set(k, v) -> 将第 k 位变成 v

reset() -> 把所有变成 0

flip() -> 把所有位取反 等价于 ~

flip() -> 把第 k 位取反
```

第三章 搜索与图论(一)

DFS

```cpp
#include <iostream>

using namespace std;

const int N = 20;

int n;

int path[N];

char g[N][N];

bool col[n], dg[N], udg[N]; // 行 正对角线 反对角线

void dfs (int u) {
    if (u == n) { // 说明已经找到一组方案
        for (int i = 0; i < n; i ++) puts(g[i]);
        puts("");
        return ;
    }

    for (int i = 0; i < n; i ++) {
        if (!col[i] && !dg[u + i] && !udg[n - u + i]) {
            g[u][i] = 'Q';
            col[i] = dg[u + i] = udg[n - u + i] = true; // 表示放置在该位置

            dfs(i + 1);

            // 恢复现场
            col[i] = dg[u + i] = udg[n - u + i] = false;
            g[u][i] = '.';
        }
    }
```

```
}

int main () {
    cin >> n;

    dfs(0);

    return 0;
}
```

BFS

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <cstring>

// bfs 解决最短路问题前提是路径权重都是一样的

using namespace std;

typedef pair<int, int> PII;


int n, m;

int g[N][N]; // 存的是迷宫地图

int d[N][N]; // 每一个点到起点的距离

PII q[N * N];

int bfs () {
    int hh = 0, tt = 0; // 队头和队尾
    q[0] = {0, 0};

    memset(d, -1, sizeof d);

    d[0][0] = 0;

    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1}; // 表示四个方向
    while (hh <= tt) { // 队列不空
        auto t = q[hh ++];

        for (int i = 0; i < 4; i ++) {
            int x = t.first + dx[i], y = t.second + dy[i];

            if (x >= 0 && x < n && y >= 0 && y < m && g[x][y] == 0 && d[x][y] == -1) {
// 当前点可以走
                d[x][y] = d[t.first][t.second] + 1;
                q[ ++ t] = {x, y};
            }
        }
```

```
    }

    return d[n - 1][m - 1];
}

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

    for (int i = 0; i < n; i ++) {
        for (int j = 0; j < m; j ++) {
            cin >> g[i][j];
        }
    }

    cout << bfs() << endl;
}
```

图的存储(邻接表)

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010, M = N * 2;

int h[N], e[M], ne[M], idx; // h 存的是 N 个链表的链表头 e 存的每个节点的值是多少 ne 存的是每个节点的 next 指针是多少

void add (int a, int b) { // 插入一条 a 指向 b 的边
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

int main () {
    idx = 0;
    memset(h, -1, sizeof h); // 初始化所有链表的值位 -1

    return 0;
}
```

树和图的遍历

DFS

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;
```

```cpp
const int N = 100010, M = N * 2;

int h[N], e[M], ne[M], idx; // h 存的是 N 个链表的链表头 e 存的每个节点的值是多少 ne 存的是每个节点的 next 指针是多少

bool st[N]; // 记录哪些点已经被遍历过了

void add (int a, int b) { // 插入一条 a 指向 b 的边
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

void dfs (int u) {
    st[u] = true; // 标记一下 当前点已经被搜过了

    for (int i = h[u]; i != -1; i ++) {
        int j = e[i];

        if (!st[j]) dfs(j);
    }
}

int main () {
    idx = 0;
    memset(h, -1, sizeof h); // 初始化所有链表的值位 -1

    dfs(1);

    return 0;
}
```

BFS

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010;

int n, m; // 分别表示点和边

int h[N], e[N], ne[N], idx;

int d[N], q[N];

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

int bfs () {
    int hh = 0, tt = 0; // 队头和队尾
```

```
    q[0] = 1;

    memset(d, -1, sizeof d);

    d[1] = 0; // 刚开始只有第一个点被遍历过了

    while (hh <= tt) {
        int t = q[hh ++]; // 每次取一下队头元素

        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];

            if (d[j] == -1) { // 如果当前点没有被拓展过的话
                d[j] = d[t] + 1;
                q[++ tt] = j;
            }
        }
    }

    return d[n];
}

int main () {
    cin >> n >> m; // 读入点数和边数

    memset(h, -1, sizeof h); // 初始化所有表头

    for (int i = 0; i < m; i ++) { // 读入所有边
        int a, b;
        cin >> a >> b;

        add(a, b);
    }

    cout << bfs() << endl;

    return 0;
}
```

有向图的拓扑序(宽度优先搜索的运用)

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010;

int n, m;

int h[N], e[N], ne[N], idx;
```

```
int q[N], d[N]; // q 存的是队列 d 存的是入度

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

bool topsort () {
    int hh = 0, tt = -1;

    for (int i = 1; i <= n; i ++) { // 插入所有入度为 0 的点
        if (!d[i]) {
            q[++ tt] = i;
        }
    }

    while (hh <= tt) {
        int t = q[hh ++];

        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i]; // 找到出边

            d[j] --; // 让它的入度 --
            if (d[j] == 0) q[++ tt] = j;
        }
    }

    return tt == n -1; // 判断一下是不是所有点都已经入队了
}

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

    memset(h, -1, sizeof h);

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

        add(a, b);
    }

    // 拓扑排序
    if (topsort()) { // 判断一下是否存在拓扑排序 有向无环图都是存在拓扑排序的 但拓扑排序的结果不一定是唯一的
        for (int i = 0; i < n; i ++) printf("%d", q[i]); // 处理后队列里存的就是拓扑排序的结果

        puts("");
    } else puts("-1");

    return 0;
}
```

第三章 搜索与图论(二)

最短路

朴素 Dijkstra 算法 时间复杂是 O(n2+m)O(n2+m) nn 表示点数，mm 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 510;

int n, m;
int g[N][N];
int dist[N]; // dist 表示当前的最短路是多少
bool st[N]; // 哪个点已经是确定的


int dijkstra () {
    memset(dist, 0x3f, sizeof dist); // 将所有距离初始化为正无穷

    dist[1] = 0;

    for (int i = 0; i < n; i ++) {
        int t = -1;

        for (int j = 1; j <= n; j ++) {
            if (!st[j] && (t != -1 || dist[t] > dist[j]))
                t = j;
        }
        st[t] = true;

        for (int j = 1; j <= n; j ++) {
             dist[j] = min(dist[j], dist[t] + g[t][j]);
        }
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    else return dist[n];
}

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

    // 对图的初始化
    for (int i = 0; i <= n; i ++)
        for (int j = 0; i <= n; j ++)
            if (i == j) g[i][j] = 0;
            esle g[i][j] = INT;

    while (m --) {
        int a, b, c;

        scanf("%d%d%d", &a, &b, &c);
        g[a][b] = min(g[a][b], c); // 处理题目中的重边 每次保留最短的那条边
```

```
    }

    int t = dijkstra();

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

    return 0;
}
```

堆优化版 Dijkstra 算法 时间复杂度 O(mlogn)O(mlogn) n 表示点数，m 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>

using namespace std;

typedef pair<int, int> PII;

const int N = 100010;

int n, m;
int h[N], e[N], ne[N], w[N]; // w 记录的是权重
int dist[N]; // dist 表示当前的最短路是多少
bool st[N]; // 哪个点已经是确定的

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

int dijkstra () {
    memset(dist, 0x3f, sizeof dist); // 将所有距离初始化为正无穷

    dist[1] = 0;

    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, 1});

    while (heap.size()) { // 堆不空
        auto t = heap.top();
        heap.pop();

        int ver = t.second(), distance = t.first();
        if (st[ver]) continue;

        for (int i = h[ver]; i != -1; i = ne[i]) {
            int j = e[i];

            if (dist[j] > distance + w[j]) {
                dist[j] = distance + w[j];
                heap.push({dist[j], j});
```

```
            }
        }
    }


    if (dist[n] == 0x3f3f3f3f) return -1;
    else return dist[n];
}

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

    // 对图的初始化
    memset(h, -1, sizeof h); // 邻接表的初始化

    while (m --) {
        int a, b, c;

        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }

    int t = dijkstra();

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

    return 0;
}
```

Bellman-Ford 算法 适合变数有限制的最短路径 时间复杂度 O(nm) n 表示点数，m 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

int N = 510, M = 10010;

int n, m;
int dist[N], bakcup[N]; // dist 表示距离

struct Edge {
    int a, b , w;
} edge[M];

int bellman_ford () {
    // 初始化
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    for (int i = 0; i < k; i ++) { // 不超过 k 条边 所以迭代 k 次
```

```
        memcpy(bakcup, dist, sizeof dist); // 做一个备份

        for (int j = 0; j < m; j ++) {
            int a = edge[j].a, b = edge[j].b, w = edge[j].w;
            dist[b] = min(dist[b], bakcup[a] + w);
        }
    }

    if (dist[n] > 0x3f3f3f3f / 2) return -1; // 如果 dist[N] 大于一个比较大的数 返回 -1

    return dist[n];
}

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

    for (int i = 0; i < m; i ++) { // 读入每条边
        int a, b, w;
        scanf("%d%d%d", &a, &b, &w);
        edge[i] = {a, b, w};
    }

    int t = bellman_ford();

    if (t == -1) { // 说明最短路不存在
        puts("impossible");
    } else printf("%d\n", t);

    return 0;
}
```

Spfa 算法 (队列优化的 Bellman-Ford 算法) 时间复杂度 平均情况下 O(m) 最坏情况下 O(nm) n 表示点数 m 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>

using namespace std;

typedef pair<int, int> PII;

const int N = 100010;

int n, m;
int h[N], e[N], ne[N], w[N]; // w 记录的是权重
int dist[N]; // dist 表示当前的最短路是多少
bool st[N]; // 哪个点已经是确定的

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

```
int spfa () {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;

    queue<int> q;
    queue.push(1);
    st[1] = true; // 当前这个点是不是在队列当中

    while (q.size()) {
        int t = q.front();
        q.pop();
        st[t] = false;

        for (int i = h[t]; i != -1; i = ne[i]) { // 更新 t 的所有领边
            int j = e[i]

            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];

                if (!st[j]) {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }

    if (dist[n] == 0x3f3f3f3f) return -1;
    else return dist[n];
}

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

    // 对图的初始化
    memset(h, -1, sizeof h); // 邻接表的初始化

    while (m --) {
        int a, b, c;

        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }

    int t = spfa();

    if (t == -1) puts("impossible");
    else printf("%d\n", t);

    return 0;
}
```

Spfa 算法 判断图中是否存在负环 时间复杂度是 O(nm) n 表示点数 m 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>

using namespace std;

typedef pair<int, int> PII;

const int N = 100010;

int n, m;
int h[N], e[N], ne[N], w[N]; // w 记录的是权重
int dist[N], cnt[N]; // dist[x] 存储 1 号点到 x 的最短距离 cnt[x] 存储 1 到 x 的最短路中经过的点数
bool st[N]; // 哪个点已经是确定的

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

bool spfa () {
    // 不需要初始化 dist 数组
    // 原理：如果某条最短路径上有 n 个点(除了自己) 那么加上自己之后一共有 n + 1 个点，由抽屉原理一定有两个点相同 所以存在环。

    queue<int> q;
    for (int i = 1; i <= n; i ++ ) {
        q.push(i);
        st[i] = true;
    }

    while (q.size()) {
        int t = q.front();
        q.pop();
        st[t] = false;

        for (int i = h[t]; i != -1; i = ne[i]) { // 更新 t 的所有领边
            int j = e[i]

            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                cnt[j] = cnt[t] + 1;

                if (cnt[j] >= n) return true;

                if (!st[j]) {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }
```

```
    return false;
}

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

    // 对图的初始化
    memset(h, -1, sizeof h); // 邻接表的初始化

    while (m --) {
        int a, b, c;

        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }

    if (spfa()) puts("Yes");
    else puts("No");

    return 0;
}
```

Floyd 算法 求多源汇最短路 时间复杂度是 O(n3) n 表示点数

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 210, INF = 1e9;

int n, m, Q; // Q 表示的是询问个数

int d[N][N];

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);

    // 初始化
    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= n; j ++) {
```

```
            if (i == j) d[i][j] = 0;
            else d[i][j] = INF;
        }
    }

    while (m --) {
        int a, b, w;

        scanf("%d%d%d", &a, &b, &w);

        d[a][b] = min(d[a][b], w);
    }

    floyd();

    while (Q --) {
        int a, b;

        scanf("%d%d", &a, &b);

        if (d[a][b] > INF / 2) puts("impossible");
        else printf("%d\n", d[a][b]);
    }

    return 0;
}
```

第三章 搜索与图论(三)

最小生成树

朴素版的普利姆算法(prime) 适合稠密图 时间复杂度是 O(n2+m) n 表示点数 m 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 510, INF = 0x3f3f3f3f;

int n, m;
int g[N][N];
int dist[N];
bool st[N];

int prime () {
    memset(dist, 0x3f, sizeof dist);

    int res = 0; // 最小生成树的所有边的长度之和
    for (int i = 0; i < n; i ++) {
```

```
        int t = -1;

        for (int j = 1; j <= n; j ++) {
            if (!st[j] && (t == -1 || dist[t] > dist[j])) {
                t = j;
            }
        }

        if (i && dist[t] == INF) return INF;
        if (i) res += dist[t]; // 只要不是第一个点

        for (int j = 1; j <= n; j ++) { // 更新一下其他点到这个集合的距离
            dist[j] = min(dist[j], g[t][j]);
        }

        st[t] = true;
    }

    return res;
}

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

    memset(g, 0x3f, sizeof g);

    while (m --) { // 输入所有边
        int a, b, c;

        scanf("%d%d%d", &a, &b, &c);

        g[a][b] = g[b][a] = min(g[a][b], c);
    }

    int t = prime();

    if (t == INF) puts("No");
    else printf("%d\n", t);

    return 0;
}
```

堆优化的 Prime 算法 适合稀疏图 时间复杂度是 O(mlogn) n 表示点数 m 表示边数

克鲁斯卡尔算法(Kruskal) 适合稀疏图 时间复杂度是 O(mlogm) n 表示点数 m 表示边数

```
#include <iostream>
#include <algorith>

using namespace std;
```

```cpp
const int N = 200010;

int n, m;
int p[N];

struct Edge {
    int a, b, w;

    bool operator < (const Edge &W) const { // 运算符重载 方便排序 按权重来排序
        return w < W.w
    }
} edges[N];

int find (int x) {
    if (p[x] != x) p[x] = find(p[x]);

    return p[x];
}

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

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

        scanf("%d%d%d", &a, &b, &w);
        edges[i] = {a, b, w};
    }

    sort(edges, edges + m); // 把所有边排序

    for (int i = 0; i <= n; i ++) { // 初始化所有的并查集
        p[i] = i;
    }

    int res = 0, cnt = 0;
    for (int i = 0; i < m;  i ++) { // 从小到大枚举所有边
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        a = find(a), b = find(b); // 祖宗节点

        if (a != b) { // 判断一下两个点是不是连通的
            p[a] = b; // 两个集合合并

            // 如果不是连通的 就把这条边加进来
            res += w; // res 存的是最小生成树的所有权重之和

            cnt ++; // 存的是边数之和
        }
    }

    if (cnt < n -1) { // 说明图是不连通的
        puts("impossible");
    } else printf("%d\n", res);
```

```
    return 0;
}

/*
int kruskal() {
    sort(edges, edges + m);

    for (int i = 1; i <= n; i ++) p[i] = i; // 初始化并查集

    int res = 0, cnt = 0;
    for (int i = 0; i < m; i ++) {
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        a = find(a), b = find(b);
        if (a != b) { // 如果两个连通块不连通，则将这两个连通块合并
            p[a] = b;
            res += w;
            cnt ++ ;
        }
    }

    if (cnt < n - 1) return INF;
    return res;
}
*/
```

二分图

染色法 时间复杂度是 O(n+m) n 表示点数 m 表示边数

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010, M = 200010;

int n, m;
int h[N], e[M], ne[M], idx; // 邻接表存储图
int color[N]; // 表示每个点的颜色 -1表示未染色 0表示白色 1表示黑色

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

bool dfs (int u, int c) { // u表示当前节点 c表示当前点的颜色
    color[u] = c;

    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!color[j]) { // 如果当前这个点没有被染颜色
            if (!dfs(j, 3 - c)) { // 3 - c 可以把 1 变成 2 把 2 变成 1
```

```
                return false;
            } else if (color[j] == c) { // 如果已经染过颜色了 而且和当前颜色有矛盾
                return false;
            }
        }
    }

    return true;
}

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

    memset(h, -1, sizeof h);

    while (m --) {
        int a, b;

        scanf("%d%d", &a, &b);

        add(a, b), add(b, a);
    }

    // 开始染色
    bool flag = true; // 染的时候是否有矛盾发生
    for (int i = 0; i <= n; i ++ ) {
        if (!color[i]) { // 如果当前这个点没有被染色的话
            if (!dfs(i, 1)) { // 如果有矛盾发生
                flag = false;
                break;
            }
        }
    }

    if (flag) puts("Yes");
    else puts("No");

    return 0;
}

/*
bool check () {
    memset(color, -1, sizeof color);
    bool flag = true;
    for (int i = 1; i <= n; i ++)
        if (color[i] == -1)
            if (!dfs(i, 0)) {
                flag = false;
                break;
            }
    return flag;
}
*/
```

匈牙利算法 时间复杂度是 O(mn) 实际运行时间远小于 O(mn)

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 510, M = 1000010;

int n1, n2, m;
int h[N], e[M], ne[M], idx; // 邻接表
int match[N]; // 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
bool st[N]; // 表示第二个集合中的每个点是否已经被遍历过

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

bool find (int x) {
    for (int i = h[x]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!st[j]) { // 如果没有考虑过
            st[j] = true;

            if (match[j] == 0 || find(match[j])) {
                match[j] = x;
                return true;
            }
        }
    }

    return false;
}

int main () {
    scanf("%d%d%d", &n1, &n2, &m);

    memset(h, -1, sizeof h);

    while (m --) {
        int a, b;

        scanf("%d%d", &a, &b);
        add(a, b);
    }

    int res = 0; // 当前匹配的数量
    for (int i = 1; i <= n1; i ++) {
        memset(st, false, sizeof st);

        if (find(i)) res ++;
        else
    }
```

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

    return 0;
}
```

第四章 数学知识(一)

数论

试除法 时间复杂度 O(n) 优化后为 O(sqrt(n))

```
#include <iostream>
#include <algorihtm>

using namespace std;

bool is_prime (int n) { // 判断是否为质数
    if (n < 2) return false;

    /*for (int i = 2; i < n; i ++)
        if (n % i == 0) return false;*/

    // 对上面的算法进行优化
    for (int i = 2; i <= n / i; i ++)
        if (n % i == 0) return false

    return true;
}

int main () {

}
```

分解质因数 也是试除法

```
#include <iostream>
#include <algorithm>

using namespace std;

void divide (int n) {
    /*for (int i = 2; i <= n; i ++)  // 从小到大枚举 n 的所有质因数
        if (n % i == 0) { // i 一定是质数
            int s = 0;
            while (n % i == 0) {
                n /= i;
                s ++;
            }

            printf("%d%d\n", i, s);
```

```
        }*/

    for (int i =  2; i <= n / i; i ++) {
        if (n % i == 0) { // i 一定是质数
            int s = 0;
            while (n % i == 0) {
                n /= i;
                s ++;
            }

            printf("%d%d\n", i, s);
        }
    }
    if (n > 1) printf("%d%d\n", n, 1);
    puts("");
}

int main () {
    int n;

    scanf("%d", &n);

    while (n --) {
        int x;
        scanf("%d", &x);
        divide(x);
    }

    return 0;
}
```

筛质数 埃式筛法

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int primes[N], cnt;
bool st[N];

void get_primes (int n) {
    for (int i = 2; i <= n; i ++) {
        if (!st[i]) { // 如果没有被筛过的话说明是一个质数
            primes[cnt ++] = n;

            for (int j = i + 1; j <= n; j += i) { // 再把每个数的倍数删掉
                st[j] = true;
            }
        }
    }
```

```
}

int main () {
    int n;

    cin >> n;

    get_primes(n);

    cout << cnt << endl;

    return 0;
}
```

线性筛法求质数 n 只会被最小质因子筛掉

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int primes[N], cnt;
bool st[N];

void get_primes (int n) {
    for (int i = 2; i <= n; i ++) {
        if (!st[i]) primes[cnt ++] = i; // 如果不是质数的话就添加到列表中

        for (int j = 0; primes[j] <= n / i; j ++) {
            st[primes[j] * i] = true;

            if (i % primes[j] == 0) break; // primes[j] 一定是 i 的最小质因子
        }
    }
}

int main () {
    int n;

    cin >> n;

    get_primes(n);

    cout << cnt << endl;

    return 0;
}
```

试除法求所有约数

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

vector<int> get_divisors (int n) {
    vector<int> res;

    for (int i = 1; i <= n / i; i ++) {
        if (n % i == 0) {
            res.push_back(i);

            if (i != n / i) res.push_back(n / i);
        }
    }

    sort(res.begin(), res.end());

    return res;
}


int main () {
    int n;

    cin >> n;

    while (n --) {
        int x;

        cin >> x;

        auto res = get_divisors(x);

        for (auto t : res) cout << t << " ";
        cout << endl;
    }

    return 0;
}
```

约数个数和约数之和

```
#include <iostream>
#include <algorithm>
#include <unordered_map>

using namespace std;

typedef long long LL;
```

```cpp
const int mod = 1e9 + 7;

int main () {
    int n;

    cin >> n;

    unordered_map<int, int> primes;

    while (n --) {
        int x;

        cin >> x;

        for (int i = 2; i <= n / i; i ++) {
            while (x % i == 0) {
                x /= i;
                primes[i] ++;
            }
        }

        if (x > 1) primes[x] ++;
    }

    LL res = 1;

    for (auto prime : primes) res = res * (prime.second + 1) % mod;

    cout << res << endl;

    return 0;
}
```

```cpp
#include <iostream>
#include <algorithm>
#include <unordered_map>

using namespace std;

typedef long long LL;

const int mod = 1e9 + 7;

int main () {
    int n;

    cin >> n;

    unordered_map<int, int> primes;

    while (n --) {
        int x;
```

```
        cin >> x;

        for (int i = 2; i <= n / i; i ++) {
            while (x % i == 0) {
                x /= i;
                primes[i] ++;
            }
        }

        if (x > 1) primes[x] ++;
    }

    LL res = 1;

    for (auto prime : primes) { // 直接带入求和公式
        int p = prime.first, a = prime.second;

        LL t = 1;

        while (a --) t = (t * p + 1) % mod;

        res = res * t % mod;
    }

    cout << res << endl;

    return 0;
}
```

最大公约数 欧几里得算法(辗转相除法)

```
#include <iostream>

using namespace std;

int gcd (int a, int b) { // 最大公约数模板
    return b ? gcd(b, a % b) : a;
}

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

    while (n --) {
        int a, b;
        scanf("%d%d", &a, &b);

        printf("%d\n", gcd(a, b));
    }

    return 0;
}
```

第四章 数学知识(二)

欧拉函数

```
#include <iostream>
#include <algorithm>

using namespace std;

int main () {
    int n;

    cin >> n;

    while (n --) {
        int a;
        cin >> a;

        int res = a;

        for (int i = 2; i <= a / i; i ++)
            if (a % i == 0) {
                res = res / i * (i - 1);

                while (a % i == 0) a /= i;
            }

        if (a > 1) res = res / a * (a - 1);

        cout << res << endl;
    }

    return 0;
}
```

筛法求欧拉函数

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 1000010;

int primes[N], cnt; // primes 存的是每一个质数 cnt 存的是质数的下标
int phi[N];
bool st[N]; // 哪些数被筛掉了
```

```
LL get_eulers (int n) {
    phi[1] = 1;

    for (int i = 2; i <= n; i ++) {
        if (!st[i]) {
            primes[cnt ++] = i;

            phi[i] = i - 1;
        }

        for (int j = 0; primes[j] <= n / i; j ++) {
            st[primes[j] * i] = true;

            if (i % primes[j] == 0) {
                phi[primes[j] * i] = phi[i] * primes[j];

                break;
            }

            phi[primes[j] * i] = phi[i] * (primes[j] - 1);
        }
    }

    LL res = 0;

    for (int i = 1; i <= n; i ++) res += phi[i];

    return res;
}

int main () {
    int n;

    cin >> n;

    cout << get_eulers(n) << endl;

    return 0;
}
```

快速幂

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

// a ^ k % p
int qmi (int a, int k, int p) {
    int res = 1;
```

```
    while (k) {
        if (k % 1) res = (LL) res * a % p;

        k >>= 1;

        a = (LL) a * a % p;
    }

    return res;
}

int mian () {
    int n;

    scanf("%d", &n);

    while (n --) {
        int a, k, p;

        scanf("%d%d%d", &a, &k, &p);

        printf("%d\n", qmi(a, k, p));
    }

    return 0;
}
```

扩展欧几里得算法

```
#include <iostream>

using namespace std;

// 欧几里得算法
int exgcd (int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;

        return a;
    }

    int d = exgcd(b, a % b, y, x);

    y -= a / b * x;

    return d;
}

int main () {
    int n;

    scanf("%d", &n);
```

```
    while (n --) {
        int a, b, x, y;

        scanf("%d%d", &a, &b);

        exgcd(a, b, x, y);

        printf("%d %d\n", x, y);
    }

    return 0;
}
```

第四章 数学知识(三)

高斯消元求线性方程组

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 110;

const double eps = 1e-6;

int n;

double a[N][N]; // 系数矩阵

int guss () {
    int c ,r;

    for (c = 0, r = 0; c < n; c ++) {
        int t = r;

        for (int i = r; i < n; i ++)
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i <= n; i ++) swap(a[t][i], a[r][i]);
        for (int i = n; i >= c; i --) a[r][i] /= a[r][c];
        for (int i = r + 1; i < n; i ++)
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j --)
                    a[i][j] -= a[r][j] * a[i][c];

        r ++;
    }
```

```
    if (r < n) {
        for (int i = r; i < n; i ++)
            if (fabs(a[i][n]) > eps) return 2; // 无解

        return 1; // 有无穷多解
    }

    for (int i = n - 1; i >= 0; i --)
        for (int j = i + 1; j < n; j ++)
            a[j][n] -= a[i][j] * a[j][n];

    return 0; // 有唯一解
}

int main () {
    cin >> n;

    for (int i = 0; i < n; i ++) {
        for (int j  = 0; j < n + 1; j ++) {
            cin >> a[i][j];
        }
    }

    int guss();

    if (t == 0) {
        for (int i = 0; i < n; i ++) printf("%.2lf\n", a[i][n]); // 保留两位小数
    } else if (t == 1) puts("Infinite group solutions"); // 说明有无穷多组解
    else puts("No solution"); // 否则的话就是无解

    return 0;
}
```

求组合数

递推

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 2010, mod = 1e9 + 7;

int c[N][N];

void init () {
    for (int i = 0; i < N; i ++) {
        for (int j = 0; j <= i; j ++) {
            if (!j) c[i][j] = 0;

            else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
        }
```

```
    }
}

int mian () {
    init();

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

    while (n --) {
        int a, b;

        scanf("%d%d", &a, &b);

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

    return 0;
}
```

预处理

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 100010, mod = 1e9 + 7;

int fact[N], infact[N]; // 首先预处理出所有阶乘取模的余数fact[N] 以及所有阶乘取模的逆元
infact[N]

int qmi (int a, int k, int p) { // 快速幂
    int res = 1;

    while (k) {
        if (k & 1) res = (LL)res * a % p;

        a = (LL)a * a % p;

        k >>= 1;
    }

    return res;
}

int main () {
    fact[0] = infact[0] = 1;

    for (int i = 1; i < N; i ++) {
        fact[i] = (LL)fact[i - 1] * i % mod;
```

```
        infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
    }

    int n;

    scanf("%d", &n);

    while (n --) {
        int a, b;

        scanf("%d%d", &a, &b);

        printf("%d\n", (LL)fact[a] * infact[b] % mod * infact[a - b] % mod);
    }

    return 0;
}
```

卢卡斯定理

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

int p;

int qmi (int a, int k) {
    int res = 1;

    while (k) {
        if (k % 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }

    return res;
}

int C (int a, int b) {
    int res = 1;

    for (int i = 1, j = a; i <= b; i ++, j --) {
        res = (LL)res * j % p;

        res = (LL)res * qmi(i, p - 2) % p;
    }

    return res;
}
```

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

int main () {
    int n;

    cin >> n;

    while (n --) {
        LL a, b;

        cin >> a >> b >> p;

        cout << lucas(a, b) << endl;
    }

    return 0;
}
```

分解质因数法求组合数

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

const int N = 5010;

int primes[N], cnt;

int sum[N]; // 存储每个质数的次数

bool st[N];

void get_primes (int n) { // 线性筛法求素数
    for (int i = 2; i <= n; i ++) {
        if (!st[i]) primes[cnt ++] = i;

        for (int j = 0; primes[j] <= n / i; j ++ ) {
            st[primes[j] * i]  = true;

            if (i % primes[j] == 0) break;
        }
    }
}

int get (int n, int p) { // 求n! 中的次数
    int res = 0;
```

```
    while (n) {
        res += n / p;
        n /= p;
    }

    return res;
}

vector<int> mul (vector<int> a, int b) { // 高精度乘低精度模板
    vector<int> c;

    int t = 0;

    for (int i = 0; i < a.size(); i ++) {
        t += a[i] * b;
        c.push_back(t % 10);
        t / = 10;
    }

    while (t) {
        c.push_back(t % 10);

        t /= 10;
    }

    return c;
}

int main () {
    int a, b;

    cin >> a >> b;

    get_primes(a);

    for (int i = 0; i < cnt; i ++) {
        int p = primes[i];

        sum[i] = get(a, p) - get(b, p) - get(a - b, p);
    }

    vector<int> res;

    res.push_back(1);

    for (int i = 0; i < cnt; i ++) {
        for (int j = 0; j < sum[i]; j ++) {
            res = mul(res, primes[i]);
        }
    }

    for (int i = res.size() - 1; i >= 0; i --) printf("%d", res[i]);

    puts("");
```

```
    return 0;
}
```

卡特兰数

```
#include <iostream>

using namespace std;

typedef long long LL;

const int mod = 1e9 + 7;

int qmi (int a, int k, int p) {
    int res = 1;

    while (k) {
        if (k & 1) res = (LL)res * a % p;

        a = (LL)a * a % p;

        k >>= 1;
    }

    return res;
}

int main () {
    int n;

    cin >> n;

    int a = 2 * n, b = n;

    int res = 1;

    for (int i = a; i > a - b; i --) res = (LL)res * i % mod;
    for (int i = 1; i <= b; i ++) res = (LL)res * qmi(i, mod - 2, mod) % mod;

    res = (LL)res * qmi(n + 1, mod - 2, mod) & mod;

    cout << res << endl;

    return 0;
}
```

第四章 数学知识(四)

容斥原理

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 20;

typedef long long LL;

int n, m;

int p[N];

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

    for (int i = 0; i < m; i ++) cin >> p[i];

    int res = 0;

    for (int i = 1; i < 1 << m; i ++) {
        int t = 1, cnt = 0;

        for (int j = 0; j < m; j ++) {
            if (i >> j & 1) {
                cnt ++;

                if ((LL)t * p[j] > n) {
                    t = -1;
                    break;
                }

                t *= p[j];
            }
        }

        if (t != -1) {
            if (cnt % 2) res += n / t;

            else res -= n / t;
        }
    }

    cout << res << endl;

    return 0;
}
```

博弈论

Nim 游戏

```
/*
    先手必胜状态 -> 可以走到某一个必败状态
    先手必败状态 -> 走不到任何一个必败状态
*/

#include <iostream>
#include <algorithm>

using namespace std;

int main () {
    int n;

    int res = 0;

    scanf("%d", &n);

    while (n --) {
        int x;

        scanf("%d", &x);

        res ^= x;
    }

    if (res) puts("Yes");
    else puts("No");

    return 0;
}
```

集合 Nime 游戏

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <unordered_set>

using namespace std;

const int N = 110, M = 10010;

int n, m;

int s[N], f[M];

int sg (int x) {
    if (f[x] != -1) return f[x];
```

```
    unordered_set<int> S;

    for (int i = 0; i < m; i ++) {
        int sum = s[i];

        if (x >= sum) S.insert(sg(x - sum));
    }

    for (int i = 0; ; i ++) {
        if (!S.count(i)) return f[x] = i;
    }
}

int main () {
    cin >> m;

    for (int i = 0; i < m; i ++) cin >> s[i];

    cin >> n;

    memset(f, -1, sizeof f);

    int res = 0;

    for (int i = 0; i < n; i ++) {
        int x;

        cin >> x;

        res ^= sg(x);
    }

    if (res) puts("Yes");
    else puts("No");

    return 0;
}
```

第五章 动态规划(一)

背包问题

01 背包问题

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;
```

```cpp
int n, m;

int v[N], w[N];

int f[N][N];

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

    for (int i = 1; i <= n; i ++) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++) {
        for (int j = 0; j <= m; j ++) {
            f[i][j] = f[i - 1][j];
            if (j >= v[i]) f[i][j] = max(f[i][j], f[i - 1][j - v[i]] + w[i]);
        }
    }

    cout << f[n][m] << endl;

    return 0;
}

/*
    一维优化
*/

#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;

int v[N], w[N];

int f[N];

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

    for (int i = 1; i <= n; i ++) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++) {
        for (int j = m; j >= v[i]; j --) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);


        }
    }

    cout << f[m] << endl;
```

```
    return 0;
}
```

完全背包问题

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;

int v[N], w[N];

int f[N][N];

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

    for (int i = 1; i <= n; i ++) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++) {
        for (int j = 0; j <= m; j ++) {
            for (int k = 0; k * v[i] <= j; k ++) {
                f[i][j] = max(f[i][j], f[i - 1][j - v[i] * k] + w[i] * k);
            }
        }
    }

    cout << f[n][m] << endl;

    return 0;
}

/*
    优化为二维
*/
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;

int v[N], w[N];

int f[N][N];

int main () {
```

```
    cin >> n >> m;

    for (int i = 1; i <= n; i ++) cin >> v[i] >> w[i];

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

            if (j >= v[i]) f[i][j] = max(f[i][j], f[i][j - v[i]] + w[i]);
        }
    }

    cout << f[n][m] << endl;

    return 0;
}

/*
    一维优化
*/
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;

int v[N], w[N];

int f[N];

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

    for (int i = 1; i <= n; i ++) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++) {
        for (int j = v[i]; j <= m; j ++) {

            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }

    cout << f[m] << endl;

    return 0;
}
```

多重背包问题 I

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 110;

int n, m;

int v[N], w[N], s[N];

int f[N][N];

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

    for (int i = 1; i <= n; i ++) cin >> v[i] >> w[i] >> s[i];

    for (int i = 1; i <= n i ++) {
        for (int j = 0; j <= m; j ++) {
            for (int k = 0; k <= s[i] && k * v[i] <= j; k ++) {
                f[i][j] = max(f[i][j], f[i  -1][j - v[i] * k] + w[i] * k);
            }
        }
    }

    cout << f[n][m] << endl;

    return 0;
}
```

多重背包问题 II

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 25000, M = 2010;

int n, m;

int v[N], w[N];

int f[N];

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

    int cnt = 0;
```

```
    for (int i = 1; i <= n; i ++) {
        int a, b, s;

        cin >> a >> b >> s;

        int k = 1;

        while (k <= s) {
            cnt ++;
            v[cnt] = a * k;
            w[cnt] = b * k;
            s -= k;
            k *= 2;
        }

        if (s > 0) {
            cnt ++;
            v[cnt] = a * s;
            w[cnt] = b * s;
        }
    }


    n = cnt;

    for (int i = 1; i <= n; i ++) {
        for (int j = m; j >= v[i]; j --) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }

    cout << f[m] << endl;

    return 0;
}
```

分组背包问题

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 110;

int n, m;
int v[N][N], w[N][N], s[N][N];
int f[N];

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

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

```
        cin >> s[i];
        for (int j = 0; j <= n; j ++) {
            cin >> v[i][j] >> w[i][j];
        }
    }

    for (int i = 1; i <= n; i ++) {
        for (int j = m; j >= 0; j --) {
            for (int k = 0; k < s[i]; k ++) {
                if (v[i][j] <= j)
                    f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);
            }
        }
    }

    cout << f[m] << endl;

    return 0;
}
```

第五章 动态规划(二)

线性dp

数字三角形

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 510, INF = 1e9;

int n;
int a[N][N];
int f[N][N];

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

    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= i; j ++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (int i = 0; i <= n; i ++) {
        for (int j = 0; j <= i; j ++) {
            f[i][j] = -INF;
        }
    }
```

```
    f[1][1] = a[1][1];

    for (int i = 2; i <= n; i ++) {
        for (int j = 1; j <= i; j ++) {
            f[i][j] = max(f[i - 1][j - 1] + a[i][j], f[i - 1][j] + a[i][j]);
        }
    }

    int res = -INF;

    for (int i = 1; i <= n; i ++) res = max(res, f[n][i]);

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

    return 0;
}
```

最长上升子序列

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;

int n;
int a[N], f[N];

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

    for (int i = 1; i <= n; i ++) scanf("%d", &a[i]);

    for (int i = 1; i <= n; i ++) {
        f[i] = 1; // 只有 a[i] 一个数

        for (int j = 1; j < i; j ++) {
            if (a[j] < a[i])
                f[i] = max(f[i], f[j] + 1);
        }
    }

    int res = 0;

    for (int i = 1; i <= n; i ++) res = max(res, f[i]);

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

    return 0;
}
```

最长公共子序列

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;
char a[N], b[N];
int f[N][N];

int main () {
    scanf("%d%d", &n, &m);
    scanf("%s%s", a + 1, b + 1);

    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= m; j ++) {
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);

            if (a[i] == b[j]) f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
        }
    }

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

    return 0;
}
```

区间dp

石子合并

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 310;

int n;
int s[N];
int f[N][N];

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

    for (int i = 1; i <= n; i ++) scanf("%d", &s[i]);

    for (int i = 1; i <= n; i ++) s[i] += s[i + 1];
```

```
    for (int len = 2; len <= n; len ++) {
        for (int i = 1; i + len - 1 <= n; i ++) {
            int l = i, r = i + len - 1;
            f[l][r] = 1e8;
            for (int k = 1; k < r; k ++) {
                f[l][r] = min(f[l][r], f[l][k] + f[k + 1][r] + s[r] - s[l - 1]);
            }
        }
    }

    printf("%d\n", f[1][n]);

    return 0;
}
```

第三章 动态规划(三)

数位统计dp

计数问题

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int get (vector<int> num. int l, int r) {
    int res = 0;

    for (int i = l; i >= r; i --) {
        res = res * 10 + num[i];
    }

    return res;
}

int power10 (int x) { // 10 的 x 次方
    int res = 1;

    while (x --) {
        res *= 10;
    }

    return res;
}

int count (int n, int x) {
    if (!n) return 0;

    vector<int> num;
```

```
    while (n) {
        num.push_back(n % 10);
        n /= 10;
    }

    n = num.size();

    int res = 0;

    for (int i = n - 1 - !x; i >= 0; i ++) {
        if (i < n - 1) {
            res += get(num, n - 1, i + 1) * power10(i);
            if (!x) res -= power10(i);
        }

        if (num[i] == x) res += get(num, i -1, 0) + 1;
        else if (num[i] > x) res += power10(i);
    }

    return res;
}

int main () {
    int a, b;

    while (cin >> a >> b, a || b) {
        if (a > b) swap(a, b);

        for (int i = 0; i < 10; i ++) {
            cout << count(b, i) - count(a - 1, i) << ' ';
        }

        cout << endl;
    }

    return 0;
}
```

状态压缩dp

蒙德里安的梦想

```
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 12, M = 1 << N;

int n, m;
long long f[N][M];
bool st[M];
```

```
int main () {
    int n, m;

    while (cin >> n >> m, n || m) {
        memset(f, 0, sizeof f);

        for (int i = 0; i < 1 << n; i ++) {
            st[i] = true;

            int cnt = 0; // 当前 0 的个数

            for (int j = 0; j < n; j ++) {
                if (i >> j & 1) {
                    if (cnt & 1) st[i] = false;
                    cnt = 0;
                } else cnt ++;
            }

            if (cnt & 1) st[i] = false;
        }

        f[0][0] = 1;

        for (int i = 1; i <= m; i ++) {
            for (int j = 0; j < 1 << n; j ++) {
                for (int k = 0; k < 1 << n; k ++) {
                    if ((j & k) == 0 && st[j | k]) {
                        f[i][j] += f[i - 1][k];
                    }
                }
            }
        }

        cout << f[m][0] << endl;
    }

    return 0;
}
```

最短 Hamilton 路径

```
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 20, M = 1 << N;

int n;
int w[N][N];
int f[M][N];
```

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

    for (int i = 0; i < n; i ++) {
        for (int j = 0; j < n; j ++) {
            cin >> w[i][j];
        }
    }

    memset(f, 0x3f, sizeof f);
    f[1][0] = 0;

    for (int i = 0; i < 1 << n; i ++) {
        for (int j = 0; j < n; j ++) {
            if (i >> j & 1) {
                for (int k = 0; k < n; k ++) {
                    if ((i - (1 << j)) >> k & 1) {
                        f[i][j] = min(f[i][j], f[i - (1 << j)][k] + w[k][j]);
                    }
                }
            }
        }
    }

    cout << f[(1 << n) - 1][n - 1] << endl;

    return 0;
}
```

树形dp

没有上司的舞会

```
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 6010;

int n;
int happy[N];
int h[N], e[N], ne[N], idx;
int f[N][2];
bool has_father[N];

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

void dfs (int u) {
```

```
    f[u][1] = happy[u];

    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];

        dfs(j);

        f[u][0] += max(f[j][0], f[j][1]);
        f[u][1] += f[j][0];
    }
}

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

    for (int i = 0; i < = n; i ++) scanf("%d", &happy[i]);

    memset(h, -1, sizeof h);

    for (int i = 0; i < n - 1; i ++) {
        int a, b;

        scanf("%d%d", &a, &b);

        has_father[a] = true;
        add(b, a);
    }

    int root = 1;

    while (has_father[root]) root ++;

    dfs(root);

    printf("%d\n", max(f[root][0], f[root][1]));

    return 0;
}
```

记忆化搜索

滑雪

```
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 310;

int n, m;
int h[N][N];
```

```
int f[N][N];

int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};

int dp (int x, int y) {
    int &v = f[x][y];

    if (v != -1) return v;

    v = 1;

    for (int i = 0; i < 4; i ++) {
        int a = x + dx[i], b = y + dy[i];

        if (a >= 1 && a <= n && b >= 1 && b <= m && h[a][b] < h[x][y]) {
            v = max(v, dp(a, b) + 1);
        }
    }

    return v;
}

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

    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= m; j ++) {
            scanf("%d", &h[i][j]);
        }
    }

    memset(f, -1, sizeof f);

    int res = 0;

    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= m; j ++) {
            res = max(res, dp(i, j));
        }
    }

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

    return 0;
}
```

第六章 贪心算法(一)

区间问题

区间选点

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;

struct Range {
    int l, r;

    bool operator<const Range &w> const {
        return r < w.r;
    }
}rang[N];

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

    for (int i = 0; i < n; i ++) {
        int l, r;

        scanf("%d%d", &l, &r);
        range[i] = {l, r};
    }

    sort(range, range + n);

    int res = 0, ed = -2e9;

    for (int i = 0; i < n; i ++) {
        if (range[i].l > ed) {
            res ++;
            ed = range[i].r;
        }
    }

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

    return 0;
}
```

最大不相交区间数量

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;

struct Range {
    int l, r;

    bool operator<(const Range &w)> const {
        return r < w.r;
    }
}range[N];

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

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

        range[i] = {l, r};
    }

    sort(range, range + n);

    int res = 0, ed = -2e9;

    for (int i = 0; i < n; i ++) {
        if (ed < range[i].l) {
            res ++;

            ed = range[i].r;
        }
    }

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

    return 0;
}
```

区间分组

```cpp
#include <iostream>
#include <algorithm>
#include <queue>

using namespace std;
```

```
const int N = 100010;

int n;
struct Range {
    int l, r;
    bool operator<(const Range &w)> const {
        return l < w.l;
    }
}range[N];

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

    for (int i = 0; i < n; i ++) {
        int l, r;

        scanf("%d%d", &l, &r);
        range[i] = {l, r};
    }

    sort(range, range + n);

    priority_queue<int, vector<int>, greater<int>> heap;

    for (int i = 0; i < n; i ++) {
        auto r = range[i];

        if (heap.empty || heap.top() >= r.l) heap.push(r.r);
        else {
            int t = heap.top();

            heap.pop();
            heap.push(r.r);
        }
    }

    printf("%d\n", heap.size());

    return 0;

}
```

区间覆盖

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;
```

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

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

    scanf("%d", &n);

    for (int i = 0; i < n; i ++) {
        int l, r;

        scanf("%d%d", &l, &r);

        rang[i] = {l, r};
    }

    sort(range, range + n);

    int res =  0;
    bool success = false;

    for (int i = 0; i < n; i ++) {
        int j  = i, r = -2e9;
        while (j < n && range[j].l <= st) {
            r = max(r, range[j].r);
            j ++;
        }
        if (r < st) {
            res = -1;
            break;
        }

        res ++;

        if (r >= ed) {
            success = true;

            break;
        }

        st = r;
        i = j - 1;
    }

    if (!success) res = -1;

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

    return 0;
```

```
}
```

Huffman 树

合并果子

```
#include <iostream>
#include <algorithm>
#include <queue>

using namespace std;

int main () {
    int n;

    scanf("%d", &n);

    priority_queue<int, vector<int>, greater<int>> heap;

    while (n --) {
        int x;
        scanf("%d", &x);

        heap.push(x);
    }

    int res = 0;

    while (heap.size() > 1) {
        int a = heap.top(); heap.pop();

        int b = heap.top(); heap.pop();

        res += a + b;
        heap.push(a + b);


    }

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

    return 0;
}
```

第六章 贪心算法(二)

排序不等式

排队打水

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 100010;

int n;
int t[N];

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

    for (int i = 0; i < n; i ++) scanf("%d", &t[i]);

    sort(t, t + n);

    LL res = 0;

    for (int i = 0; i < n; i ++) res += t[i] * (n - i + 1);

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

    return 0;
}
```

绝对值不等式

货仓选址

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;
int a[N];

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

    for (int i = 0; i < n; i ++) scanf("%d", &a[i]);

    sort(a, a + n);
```

```
    int res = 0;

    for (int i = 0; i < n; i ++) res += abs(a[i] - a[n / 2]);

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

    return 0;
}
```

推公式

算杂技的牛

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef pair<int, int> PII;

const int N = 50010;

PII cow[N];

int n;

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

    for (int i = 0; i < n; i ++) {
        int w, s;
        scanf("%d%d", &w[i], &s[i]);

        cow[i] = {w + s, w};
    }

    sort(cow, cow + n);

    int res = -2e9, sum = 0;

    for (int i = 0; i < n; i ++) {
        int w = cow[i].second, s = cow[i].first - w;

        res = max(res, sum - s);

        sum += w;
    }

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

    return 0;
}
```