基础算法

一维前缀和

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

const int N = 1e5 + 10;
int sum[N];

int main() {
    int n, k, x;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> x;
        sum[i] = sum[i - 1] + x;  // 前缀和数组
    }

    cin >> k;
    while (k--) {
        int l, r;
        cin >> l >> r;
        cout << sum[r] - sum[l - 1] << endl;
    }

    return 0;
}
```

二维前缀和

```
#include <iostream>
#include <vector>
using namespace std;

const int N = 1010;
int s[N][N];

int main() {
    int n, m, q;  // n行m列, q次查询
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> s[i][j];  // 此时 s[i][j] = a[i][j];  (a[i]
[j]就是原数组值，被省略掉了)
            s[i][j] += s[i-1][j] + s[i][j-1] - s[i-1][j-1];  //
构建前缀和矩阵
        }
    }

    // 询问
    while (q--) {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        // 因为多减了一次 s[x1-1][y1-1]，所以要加回去
        cout << (s[x2][y2] - s[x1-1][y2] - s[x2][y1-1] + s[x1-
1][y1-1]) << endl;
    }

    return 0;
}
```

一维差分

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

```cpp
const int N = 1e5 + 10;

int a[N]; //原数组
int d[N]; //差分数组
int s[N]; //原数组（修改后的）

int main()
{
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i];

    //求差分数组
    for(int i = 1; i <= n; i++) d[i] = a[i] - a[i-1];

    //m次修改差分数组
    for(int i = 1; i <= m; i++)
    {
        int l, r, x;
        cin >> l >> r >> x;
        d[l] += x, d[r + 1] -= x;
    }

    //对差分数组求前缀和，得到修改后的原数组
    for(int i = 1; i <= n; i++)
    {
        s[i] = s[i - 1] + d[i];
        cout << s[i] << " ";
    }
    return 0;
}
```

二维差分

```
#include<iostream>
#include<cstdio>
using namespace std;
const int N = 1e3+5;
int a[N][N], s[N][N];

// (x1, y1) 到 (x2, y2) 增加 v
void insert(int x1,int y1,int x2,int y2,int v){
    a[x1][y1] += v;
    a[x2+1][y1] -= v;     // 不该加，减回去
    a[x1][y2+1] -= v;     // 不该加，减回去
    a[x2+1][y2+1] += v;   // 多减一次，加v进行抵消操作
}
int main(){
    int n,m,q;
    scanf("%d%d%d",&n,&m,&q);
    for(int i = 1; i <= n; i++){
        for(int j = 1;j <= m; j++){
            int t;
            scanf("%d",&t);
            insert(i,j,i,j,t);
        }
    }
    while(q--){
        int x1,y1,x2,y2,t;
        scanf("%d%d%d%d%d",&x1,&y1,&x2,&y2,&t);
        insert(x1,y1,x2,y2,t);
    }
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] +
a[i][j];
            printf("%d ",s[i][j]);
        }
        printf("\n");
    }
    return 0;
```

```
}
```

双指针

最长不重复子序列

给定一个长度为 n 的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。

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

int a[100010];  //原数组
int s[100010];  //桶计数数组

int main()
{
    int n, ans = 0;
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];

    for(int i = 1, j = 1; j <= n; j++) //j枚举新的数字
    {
        s[a[j]] ++;  //a[j]桶计数
        while(s[a[j]] > 1)  //若a[j]出现的次数>=2，则需要i向后移
动
        {
            s[a[i]] --;  //删除第i个数字
            i++;  //i向后移动
        }
        ans = max(ans, j - i + 1);
    }
    cout << ans;

    return 0;
}
```

二分

```
bool check(int x)
{
// 进行某些操作
}
// 二分查找函数
int binarySearch()
{
    int l = 1, r = n; // 初始化左右边界
    while (r - l > 1) // 当右边界与左边界相差大于1时
    {
        int mid = (l + r) >> 1; // 取中间位置
        if (check(mid)) // 如果满足条件
            r = mid; // 更新右边界为mid
        else
            l = mid; // 否则更新左边界为mid
    }
    if (check(l)) // 如果满足条件
        return l; // 返回左边界值
    else if (check(r)) // 如果满足条件
        return r; // 返回右边界值
    return -1; // 否则返回-1
}
```

数据结构

队列

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

using namespace std;
```

```cpp
queue<int> q;

int main()
{
    string s;
    int n;
    cin >> n;
    for(int i = 1;i <= n;i++) {
        cin >> s;
        if(s == "push") {
            int x;
            cin >> x;
            q.push(x);
        }
        else if(s == "pop") {
            if(q.size()) {
                q.pop();
            }
        }
        else if(s == "empty") {
            if(q.empty()) {
                cout << "YES" << endl;
            }
            else {
                cout << "NO" << endl;
            }
        }
        else {
            if(!q.empty()) {
                cout << q.front() << endl;
            }
            else {
                cout << "ERR" << endl;
            }
        }
    }
}
```

栈

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

int main() {
    int n;
    while (cin >> n && n != 0) {
        stack<int> s;
        char a;
        int t;
        for (int i = 1; i <= n; ++i) {
            cin >> a;
            if (a == 'P') {
                cin >> t;
                s.push(t);
            } else if (a == 'O') {
                if (!s.empty()) {
                    s.pop();
                }
            } else if (a == 'A') {
                if (s.empty()) {
                    cout << "E" << endl;
                } else {
                    cout << s.top() << endl;
                }
            }
        }
        cout << endl;
    }
    return 0;
}
```

单调栈

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

const int N = 100010;

int stk[N], n, tt;

int main()
{
    cin>>n;
    for(int i = 1;i <= n;i++)
    {
        int x;
        cin >> x;
        while(tt && stk[tt] >= x) tt--;//出单调栈

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

        stk[++tt] = x;//入单调栈
    }
}
```

单调队列

```
#include <iostream>
using namespace std;

const int N = 1e6+10;
int n, k;
int a[N];

int que[N], head = 1, tail = 0;
```

```
void min_queue()
{
    // min queue
    head = 1;
    tail = 0;
    for(int i = 1; i <= n; i++)
    {
        while(head <= tail && que[head] + k <= i)   //如果队首元
素已经不在区间内，弹出
            head++;
        while(head <= tail && a[i] > a[que[tail]])  //如果队尾元
素小于新元素，弹出
             tail--;
        que[++tail] = i;       // 当前元素入队
        // 输出
        if(i >= k) cout << a[que[head]] << ' ';
    }
}

void max_queue()
{
    // min queue
    head = 1;
    tail = 0;
    for(int i = 1; i <= n; i++)
    {
        while(head <= tail && que[head] + k <= i)   //如果队首元
素已经不在区间内，弹出
            head++;
        while(head <= tail && a[i] < a[que[tail]])  //如果队尾元
素大于新元素，弹出
             tail--;
        que[++tail] = i;       // 当前元素入队
        // 输出
        if(i >= k) cout << a[que[head]] << ' ';
    }
}

int main()
```

```
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> a[i];
    max_queue();
    cout << endl;
    min_queue();


    return 0;
}
```

字符串哈希

```
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 1e5 + 10,P = 131;

typedef unsigned long long ull;

int n,m;
char str[N];
ull h[N],p[N];

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

int main() {
    cin >> n >> m;
    scanf("%s", str + 1);
    p[0] = 1;
    for(int i = 1;i <= n;i++) {
        h[i] = h[i - 1] * P + str[i];
        p[i] = p[i - 1] * P;
```

```
    }
    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");
    }
}
```

并查集

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

const int N = 100010;

//并查集
int n, m;
int p[N];//存放节点的父节点

int find(int x) // 返回x的祖先节点 + 路径压缩
{
    if(x != p[x]) p[x] = find(p[x]);
    return p[x];
}

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

    // 初始化把每个节点的父都设置为自己
    for(int i = 1; i <= n; i++) p[i] = i;

    while(m--)
    {
        int a, b;
        char c;
        cin >> c >> a >> b;
```

```
        if(c == 'M')
            p[find(a)] = find(b); // 合并，修改a祖宗节点的父节点为b的祖宗节点
        else
        {
            if(find(a) == find(b)) puts("Yes"); //查询
            else puts("No");
        }
    }
}
```

树状数组

区间修改单点查询

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
long long n, q, a[N], c[N];

long long lowbit(int x){
    return x&-x;
}
void add(int x, int y) {   // D[x] += y
    while(x<=n) {
        c[x] += y;
        x += lowbit(x);
    }
}
long long sum(int x) {   // 求D[1 ~ i]的和，即 a[i] 值
    long long res = 0;
    while(x) {
        res += c[x];
        x -= lowbit(x);
    }
    return res;
}
int main(){
```

```
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> q;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
        add(i, a[i]-a[i-1]);  //输入初值的时候，也相当于更新了值，同时构建的是差分数组上的树状数组
    }
    while(q--){
        int op, x, y, z;
        cin >> op >> x;
        if(op == 1) {
            cin >> y >> z;
            add(x, z);       //a[x] - a[x-1] 增加 k
            add(y+1, -z);    //a[y+1] - a[y] 减少 k
        } else {
            cout << sum(x) << '\n';   // 差分数组上的前缀和就是原数组 a[x]
        }
    }
}
```

单点修改区间查询

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

const int N = 1e6+5;
long long n, q, a[N], c[N];
int lowbit(int x){
    return x&(-x);
}
void add(int x, int y) {
    while(x <= n) {
        c[x] += y;
        x += lowbit(x);
    }
}
long long sum(int x) {
```

```
    long long res = 0;
    while(x) {
        res += c[x];
        x -= lowbit(x);
    }
    return res;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> q;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
        add(i, a[i]);
    }
    // 初始 c[i] = 0
    for(int i = 1, op, x, y; i <= q; i++) {
        cin >> op >> x >> y;
        if(op == 1) {
            add(x, y);
        } else {
            cout << sum(y) - sum(x-1) << '\n';
        }
    }
    return 0;
}
```

区间修改区间查询

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

const int N = 1e6+5;
int n, q, a[N];
long long c1[N];   // 维护 (D[1] + D[2] + ... + D[n])
long long c2[N];   // 维护 (0*D[1] + 1*D[2] + ... + (n-1)*D[n])

// 获取 x 的最低位 1 的位置
```

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

// 在树状数组中进行单点更新
void add(int x, int y) {
    long long p = x;  // 因为 x 不变，所以要先保存
    while(x <= n) {
        c1[x] += y;
        c2[x] += (p - 1) * y;
        x += lowbit(x);
    }
}

// 计算前缀和
long long sum(int x) {
    long long res = 0, p = x;
    while(x) {
        res += p * c1[x] - c2[x];
        x -= lowbit(x);
    }
    return res;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    // 输入数组长度和查询次数
    cin >> n >> q;

    // 输入数组元素
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
        add(i, a[i] - a[i - 1]);
    }

    while(q--) {
        int op, l, r, x;
        cin >> op >> l >> r;
```

```
        if(op == 1) {
            cin >> x;
            add(l, x);
            add(r + 1, -x);
        } else {
            cout << sum(r) - sum(l - 1) << '\n';
        }
    }

    return 0;
}
```

线段树

单点修改区间查询

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

const int MAXN = 5000010;
long long a[MAXN];

struct SegmentTree {
    long long l, r, n;
} tree[MAXN];

void push_up(long long p) {
    tree[p].n = tree[2 * p].n + tree[2 * p + 1].n;
}

void build(long long p, long long l, long long r) {
    tree[p].l = l;
    tree[p].r = r;
    if (l == r) {
        tree[p].n = a[l];
        return;
    }
    long long mid = (l + r) >> 1;
```

```
    build(p << 1, l, mid);
    build(p << 1 | 1, mid + 1, r);
    push_up(p);
}

long long query(long long p, long long l, long long r) {
    if (tree[p].l >= l && tree[p].r <= r) {
        return tree[p].n;
    }
    long long mid = (tree[p].r + tree[p].l) >> 1;
    long long s = 0;
    if (l <= mid) {
        s += query(p << 1, l, r);
    }
    if (r > mid) {
        s += query(p << 1 | 1, l, r);
    }
    return s;
}

void modify(long long p, long long l, long long k) {
    if (tree[p].l == tree[p].r) {
        tree[p].n += k;
        return;
    }
    long long mid = (tree[p].l + tree[p].r) >> 1;
    if (l <= mid) {
        modify(p * 2, l, k);
    } else {
        modify(2 * p + 1, l, k);
    }
    push_up(p);
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    long long n, k;
```

```
    cin >> n >> k;
    for (long long i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    build(1, 1, n);

    for (long long i = 1; i <= k; ++i) {
        long long t, l, x, r;
        cin >> t >> l;
        if (t == 1) {
            cin >> x;
            modify(1, l, x);
        } else {
            cin >> r;
            cout << query(1, l, r) << '\n';
        }
    }

    return 0;
}
```

区间修改单点查询

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

const int N = 1e6 + 1;
long long tree[4 * N];
long long tr[4 * N];
int sz[N];

void push_up(int p) {
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
}

void build(int p, int l, int r) {
    if (l == r) {
        tr[p] = sz[l];
```

```
        return;
    }
    int mid = (l + r) / 2;
    build(p * 2, l, mid);
    build(p * 2 + 1, mid + 1, r);
    // push_up(p);
}

void modify(int ql, int qr, int z, int p, int l, int r) {
    if (ql <= l && r <= qr) {
        tr[p] += z;
        return;
    }
    int mid = (l + r) / 2;
    if (ql <= mid) {
        modify(ql, qr, z, p * 2, l, mid);
    }
    if (qr > mid) {
        modify(ql, qr, z, p * 2 + 1, mid + 1, r);
    }
}

long long query(int x, int p, int l, int r) {
    long long ans = tr[p];
    if (l == r) {
        return ans;
    }
    int mid = (l + r) / 2;
    if (x <= mid) {
        ans += query(x, p * 2, l, mid);
    } else {
        ans += query(x, p * 2 + 1, mid + 1, r);
    }
    return ans;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
```

```
    int n, q;
    cin >> n >> q;
    for (int i = 1; i <= n; ++i) {
        cin >> sz[i];
    }
    build(1, 1, n);
    for (int i = 1; i <= q; ++i) {
        int t, a, b, c;
        cin >> t;
        if (t == 1) {
            cin >> a >> b >> c;
            modify(a, b, c, 1, 1, n);
        } else if (t == 2) {
            cin >> a;
            long long ans = query(a, 1, 1, n);
            cout << ans << '\n';
        }
    }
    return 0;
}
```

区间修改区间查询

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

#define ll long long
const int N = 1e6 + 10;
ll a[N];          // 记录数列的元素，从 a[1] 开始
ll tree[N << 2]; // tree[i]: 第 i 个结点的值，表示一个线段区间的值，例如最值、区间和
ll tag[N << 2];  // tag[i]: 第 i 个结点的 lazy-tag，统一记录这个区间的修改

void push_up(ll p) { // 从下往上传递区间值
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
    // 本题是区间和。如果求最小值，改为: tree[p] = min(tree[ls(p)], tree[rs(p)]);
}
```

```
void build(ll p, ll l, ll r) { // 建树。p 是结点编号，它指向区间 [pl, pr]
    tag[p] = 0;                // lazy-tag 标记
    if (l == r) {
        tree[p] = a[l];        // 最底层的叶子，赋值
        return;
    }
    ll mid = (l + r) >> 1;     // 分治：折半
    build(p * 2, l, mid);      // 左儿子
    build(p * 2 + 1, mid + 1, r); // 右儿子
    push_up(p);                // 从下往上传递区间值
}

inline void addtag(ll p, ll l, ll r, ll d) { // 给结点 p 打 tag 标记，并更新 tree
    tag[p] += d;                // 打上 tag 标记
    tree[p] += d * (r - l + 1); // 计算新的 tree
}

inline void push_down(ll p, ll l, ll r) { // 不能覆盖时，把 tag 传给子树
    if (tag[p]) {                         // 有 tag 标记，这是以前做区间修改时留下的
        ll mid = (l + r) >> 1;
        addtag(p * 2, l, mid, tag[p]);           // 把 tag 标记传给左子树
        addtag(p * 2 + 1, mid + 1, r, tag[p]);   // 把 tag 标记传给右子树
        tag[p] = 0;                              // p 自己的 tag 被传走了，归 0
    }
}

void modify(ll ql, ll qr, ll p, ll l, ll r, ll d) { // 区间修改：把 [L, R] 内每个元素加上 d
    if (ql <= l && r <= qr) {                       // 完全覆盖，直接返回这个结点，它的子树不用再深入了
        addtag(p, l, r, d);                         // 给结点 p 打 tag 标记，下一次区间修改到 p 这个结点时会用到
```

```
        return;
    }
    push_down(p, l, r);                                  // 如果不能
覆盖，把 tag 传给子树
    ll mid = (l + r) >> 1;
    if (ql <= mid) modify(ql, qr, p * 2, l, mid, d); // 递归左
子树
    if (qr > mid)  modify(ql, qr, p * 2 + 1, mid + 1, r, d); //
递归右子树
    push_up(p);                                          // 更新
}

ll query(ll ql, ll qr, ll p, ll l, ll r) {
    // 查询区间 [L, R]；p 是当前结点（线段）的编号，[pl, pr] 是结点
p 表示的线段区间
    if (ql <= l && r <= qr) return tree[p];           // 完全覆
盖，直接返回
    push_down(p, l, r);                                  // 不能覆
盖，把 tag 传给子树
    ll res = 0;
    ll mid = (l + r) >> 1;
    if (ql <= mid) res += query(ql, qr, p * 2, l, mid); // 左子
节点有重叠
    if (qr > mid)  res += query(ql, qr, p * 2 + 1, mid + 1, r);
// 右子节点有重叠
    return res;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    ll n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) cin >> a[i];
    build(1, 1, n); // 建树
    while (m--) {
        ll op, l, r, x;
        cin >> op;
        if (op == 1) { // 区间修改：把 [L, R] 的每个元素加上 x
```

```
            cin >> l >> r >> x;
            modify(l, r, 1, 1, n, x);
        } else { // 区间询问：[L, R] 的区间和
            cin >> l >> r;
            cout << query(l, r, 1, 1, n) << '\n';
        }
    }
    return 0;
}
```

树与图

树的bfs

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

int n, x, y;
vector<int> v[10010];
queue<int> q;

void bfs() {
    q.push(1);
    while (!q.empty()) {
        int node = q.front();
        cout << node << " ";
        for (int i = 0; i < v[node].size(); ++i) {
            q.push(v[node][i]);
        }
        q.pop();
    }
}

int main() {
```

```
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    cin >> n;
    while (cin >> x >> y) {
        v[x].push_back(y);
    }

    bfs();

    return 0;
}
```

树的dfs

优先遍历节点编号较小的子树

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

int n, x, y;
vector<int> son[101];

void dfs(int k) {
    cout << k << " ";
    sort(son[k].begin(), son[k].end());
    for (int i = 0; i < son[k].size(); ++i) {
        dfs(son[k][i]);
    }
}

int main() {
    cin >> n;
    for (int i = 1; i < n; ++i) { // 这里千万不能写成<=n
        cin >> x >> y;
        son[x].push_back(y);
    }
    dfs(1);
```

```
    return 0;
}
```

堆优化dij

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

const int N = 1e6 + 10;

typedef pair<int, int> PII; //距离，编号

int n, m;
struct edge
{
    int ne; //next表示下一个点的下标
    int w; //权值
};
vector<edge> g[N];

int dist[N]; //1号点到其他点的最短距离
int st[N]; //是否更新最短距离

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

    // 创建小根堆，以距离排序
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, 1});

    // 循环，以最小距离更新其他点的距离
    while(heap.size())
    {
        //获取堆顶
        PII t = heap.top();
```

```
        heap.pop();

        //确定最小距离的编号id
        int id = t.second;
        if(st[id]) continue;  //若更新过点id，就跳过
        st[id] = 1;

        //用id更新其他相邻点的最短距离
        for(unsigned int i = 0; i < g[id].size(); i++)
        {
            int j = g[id][i].ne, w = g[id][i].w; //确定下一个点j、从id到j的距离
            if(dist[j] > dist[id] + w) //1到j点的距离 > 1到id点的距离 + 从id到达j的距离
            {
                dist[j] = dist[id] + w;
                heap.push({dist[j], j});
            }
        }
    }

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

int main()
{
    cin >> n >> m;
    while(m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a].push_back({b, c});
    }

    cout << dijkstra();
    return 0;
}
```

spfa求负环

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

const int N = 2e3 + 10, INF = 0x3f3f3f3f;

struct edge{int v, w;};
vector<edge> e[N];
int dis[N]; //距离
int cnt[N]; //到i点的边数
int vis[N]; //i点是否访问
int n, m;
queue<int> q; //队列

bool SPFA()
{
    memset(dis, INF, sizeof dis);

    for(int i = 1; i <= n; i++) //初始化，把所有点放入队列，并标记
    {
        vis[i] = 1;
        q.push(i);
    }

    while(q.size())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for(int j = 0; j < e[u].size(); j++) //枚举u点的所有出边
        {
            int v = e[u][j].v, w = e[u][j].w;
            if(dis[u] + w < dis[v]) //有松弛操作
            {
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u] + 1; //更新到达v点的边数

                //若到达v点的边数>=n，说明有负环
```

```
                if(cnt[v] >= n) return 1;

                //v点被更新，且不在队列内
                if(!vis[v]) q.push(v), vis[v] = 1;
            }
        }
    }
    return 0; //无负环
}

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= m; i++)
    {
        int x, y, z;
        cin >> x >> y >> z;
        e[x].push_back({y, z});
    }

    if(SPFA()) cout << "Yes";
    else cout << "No";
    return 0;
}
```

floyd多源最短路

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

const int N = 3e2+5;
int n, m, t;
int g[N][N];

int main(){
    memset(g, 0x3f, sizeof g);
    cin >> n >> m >> t;
    for(int i = 1; i <= m; i++) {
```

```
        int s, e, h;
        cin >> s >> e >> h;
        g[s][e] = h;
    }
    // floyd
    for(int k = 1; k <= n; k++)
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                g[i][j] = min(g[i][j], max(g[i][k],g[k][j]));
    while(t--) {
        int a, b;
        cin >> a >> b;
        if(g[a][b] != 0x3f3f3f3f)
            cout << g[a][b] << endl;
        else
            cout << -1 << endl;
    }
    return 0;
}
```

kruskal求最小生成树

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

const int N = 2e5 + 10;

int p[N]; //并查集数组，p[i]存储i的祖宗节点

struct Edge
{
    int u, v, w;

    bool operator<(const Edge &rhs) const
    {
        return w < rhs.w;
    }
```

```
} e[N];

int find(int x) //并查集查找x的祖宗节点
{
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int main()
{
    int n, m, u, v, w, ans = 0, cnt = 0; //cnt表示已加入最小生成树的边的个数
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        cin >> u >> v >> w;
        e[i] = {u, v, w};
    }
    sort(e, e + m); //对所有边权从小到大排序

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

    for (int i = 0; i < m; i++) //从小到大枚举所有边
    {
        u = e[i].u, v = e[i].v, w = e[i].w;
        u = find(u), v = find(v); //分别查找u和v的祖宗节点
        if (u != v) //两个点不在一个集合中
        {
            p[u] = v; //合并集合
            ans += w;
            cnt++;
        }
    }
    if (cnt < n - 1) cout << "impossible"; //边数不够，则不连通
    else cout << ans;
}
```

拓扑排序

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

const int N = 100010;

int n, m, idx;
vector<int> g[N];//邻接表存放图
queue<int> q;//队列
int d[N];//d数组存放每个点的入度
int ans[N];//存放拓扑排序

//返回是否有拓扑排序
bool toposort()
{
    for(int i = 1; i <= n; i++)//将所有初始入度为0的点放入队列
        if(!d[i]) q.push(i);


    while(!q.empty())//队列不为空
    {
        int t = q.front(); q.pop();//获取队头，并出队

        ans[++idx] = t;//放入拓扑排序

        for(int i = 0; i < g[t].size(); i++)//扩展t所有的点
        {
            int j = g[t][i];//扩展的点j
            d[j] --;//删除i到j的边，j的入度--

            if(d[j] == 0) q.push(j);//若j的入度减为0，则把j入队
        }
    }

    //idx从1~n。若idx最终变为n，则所有点都入了队，说明有拓扑排序
    return idx == n;
}
```

```
int main()
{
    cin >> n >> m;//n个点，m条边

    for(int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        g[a].push_back(b); //建立a到b的边
        d[b] ++; //b的入度++
    }

    if(toposort())//若有拓扑排序
    {
        cout << "loop not exist.\n";
        for(int i = 1; i <= n; i++) cout << ans[i] << " ";//输
出ans数组即可
    }
    else cout << "loop exist.";
}
```

动态规划

最长不下降子序列

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1010;
int a[N], dp[N];
int main()
{
```

```
    int n, maxn = 0;
    cin>>n;
    for(int i = 1; i <= n; i++)
    {
        cin>>a[i];
        dp[i] = 1;
    }
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j < i; j++)//找一遍所有的可能性
        {
            if(a[i] >= a[j])//如果是非严格递增的
            {
                dp[i] = max(dp[j] + 1, dp[i]);//更新最大值
            }
        }
        if(dp[i] > maxn)
        {
            maxn = dp[i];
        }
    }
    cout << maxn;
}
```

最长公共子序列

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

string x, y;
int f[210][210];
//f[i][j]表示从x的前i位选，y的前j位选最长的公共子序列的最大长度

int main()
{
    cin >> x >> y;
    x = " " + x; //从下标1开始比较
    y = " " + y;
```

```
    int lx = x.size() - 1;
    int ly = y.size() - 1;

    //状态计算
    for(int i = 1; i <= lx; i++)
    {
        for(int j = 1; j <= ly; j++)
        {
            //若对应位相同，那么最长公共子序列=f[i-1][j-1] + 1
            if(x[i] == y[j]) f[i][j] = f[i-1][j-1] + 1;

            //若对应位不同，就考虑和之前的位置元素是不是相同
            else f[i][j] = max(f[i-1][j], f[i][j-1]);
        }
    }
    cout << f[lx][ly];
    return 0;
}
```

编辑距离

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

const int N = 2010;
int f[N][N]; //f[i][j]表示字符串a的1~i变成b的1~j需要的最短编辑距离

int main()
{
    string a, b;
    cin >> a >> b;
    int la = a.size(), lb = b.size();

    //初始化
    for(int i = 0; i <= la; i++) f[i][0] = i;  //a的长度为i，b的长度为0，全删除
    for(int i = 0; i <= lb; i++) f[0][i] = i;  //a的长度为0，b的长度为i，全插入
```

```
    //状态转移
    for(int i = 1; i <= la; i++)
    {
        for(int j = 1; j <= lb; j++)
        {
            f[i][j] = min(f[i][j-1], f[i-1][j]) + 1; // 添加or
删除
            f[i][j] = min(f[i][j], f[i-1][j-1] + (a[i-1] !=
b[j-1])); //替换or不动
        }
    }
    cout << f[la][lb];

    return 0;
}
```

01背包

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

int n, m;
int v[40], w[40];
int f[210];  // f[j]表示背包容量为j的最大价值

int main()
{
    cin >> m >> n;  //m的容量，n件物品
    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--) // 注意是倒序，否则会先修
改后使用
        {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
```

```
        }
    }
    cout << f[m];  // 注意是m不是n
}
```

完全背包

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

int n, m;
int v[1010], w[1010];
int f[1010];  // f[j]表示考虑背包容量为j条件下的最大价值

int main()
{
    cin >> n >> m;  //n件物品，m的容量
    for(int i = 1; i <= n; i++)
        cin >> v[i] >> w[i];

    for(int i = 1; i <= n; i++)
    {
        // 因为f[i][j]需要用到第i层f[i][j-v[i]]的结果，所以从前往后计算
        for(int j = v[i]; j <= m; j++)
            f[j] = max(f[j], f[j-v[i]] + w[i]);
    }
    cout << f[m];
    return 0;
}
```

区间dp

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

const int N = 410;
```

```
int n;
int a[N], s[N];
int f[N][N]; //f[i][j]表示合并[i,j]区间的最小总分
int g[N][N]; //g[i][j]表示合并[i,j]区间的最大总分

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i];
        a[i + n] = a[i];
    }
    for(int i = 1; i <= 2*n; i++)
    {
        s[i] = s[i - 1] + a[i];
    }

    for(int len = 2; len <= n; len++) //枚举区间长度
    {
        for(int i = 1; i + len - 1 < 2*n; i++) //枚举左端点
        {
            int j = i + len - 1; //右端点
            f[i][j] = 1e9;
            for(int k = i; k < j; k++)
            {
                f[i][j] = min(f[i][j], f[i][k] + f[k+1][j] +
s[j] - s[i-1]);
                g[i][j] = max(g[i][j], g[i][k] + g[k+1][j] +
s[j] - s[i-1]);
            }
        }
    }
    int maxv = 0, minv = 1e9;
    for(int i = 1; i <= n; i++) //枚举长度为n的所有合并答案
    {
        minv = min(minv, f[i][i+n-1]);
        maxv = max(maxv, g[i][i+n-1]);
    }
```

```
    cout << minv << endl << maxv;
}
```

树形dp

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

/*
状态表示:
    f[u][0]表示以u为根的子树不选u的最大欢乐值
    f[u][1]表示以u为根的子树选u的最大欢乐值
状态计算:
    f[u][0] = Σmax(f[si][0], f[si][1])  其中si表示u点的第i个子节点
    f[u][1] = Σf[si][0]
*/

const int N = 6010;

int n;
int happy[N];
vector<int> g[N];
int f[N][2];

bool has_father[N]; //是否有父节点

void dfs(int u)
{
    f[u][1] = happy[u];
    for(int i = 0; i < g[u].size(); i++)
    {
        int v = g[u][i];
        dfs(v);
        f[u][0] += max(f[v][0], f[v][1]);
        f[u][1] += f[v][0];
    }
}
```

```
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin>> happy[i];

    int u, v;
    while(cin >> u >> v && u && v) //u是v的子节点
    {
        g[v].push_back(u);
        has_father[u] = 1;
    }

    int root = 1;
    while(has_father[root]) root++;

    dfs(root); //递归求解f[root][0]和f[root][1]

    cout << max(f[root][0], f[root][1]);

    return 0;
}
```

数学

十进制转k进制

```
#include <bits/stdc++.h> // 包含标准头文件
using namespace std;

long long s, base; // 定义两个长整型变量，s 为要转换的数，base 为进制
string p = "0123456789ABCDEF"; // 定义字符串 p，存储 16 进制数的所有可能字符
string ans; // 定义字符串 ans，用于存储转换后的结果
```

```
int main() {
    cin >> s >> base; // 输入要转换的数 s 和进制 base
    while (s) { // 当 s 不为 0 时循环
        ans.push_back(p[s % base]); // 将 s 对 base 取模的结果作为索引，将对应的字符添加到 ans 末尾
        s /= base; // 将 s 除以 base，更新 s 的值
    }
    reverse(ans.begin(), ans.end()); // 将 ans 反转，因为从末尾开始计算的结果需要反转才能得到正确的结果
    cout << ans; // 输出转换后的结果
    return 0; // 返回0，表示程序正常结束
}
```

k进制转十进制

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    string s;
    int r, w = 1, ans = 0; //w表示每次计算的权值
    cin >> s >> r;
    for (int i = s.size() - 1; i >= 0 ; i--) {
        if (s[i] >= 'A') ans += (s[i] - 'A' + 10) * w;
        else ans += (s[i] - '0') * w;
        w *= r; //权值每次要翻r倍
    }
    cout << ans;
}
```

埃氏筛

```
const int N = 100000;   // N 的大小取决于问题中 n 的范围
bool a[N];   // 标记数组  a[i] = 0(false):素数，a[i]=1(true): 非素数
```

```
// 埃氏筛法 函数执行完后 a[] 数组即为筛出的素数结果 a[i]=false 表示 i 是素数
void prime()
{
    a[0]=a[1]=true;          // 0 和 1 不是素数
    for(int i = 2; i <= n; i++)
    {
        if(a[i] == false)    // i 是素数
        {
            for(int j = i+i; j <= n; j += i)
                a[j] = true;
        }
    }
}
```

线性筛

```
#include<iostream>
using namespace std;
bool f[100000010];
int prime[6000010], cnt;
int n,q,c;
void get_prime(){
    for(int i = 2 ; i <= n ; i ++)
    {
        if(!f[i]) prime[++cnt] = i;

        for(int j = 1; prime[j] * i <= n; j++)
        {
            f[prime[j] * i] = 1; //标记不是质数
            if(i % prime[j] == 0) break; //优化成O(n)
        }
    }
}
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    cin>>n>>q;
```

```
    get_prime();
    while(q--){
        cin>>c;
        cout<<prime[c]<<"\n";
    }

    return 0;
}
```

最大公约数与最小公倍数

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

int gcd(int a, int b) //最大公约数
{
    if(a % b == 0) return b;
    return gcd(b, a % b);
}

int lcm(int a, int b) //最小公倍数
{
    return a * b / gcd(a, b);
}

int main()
{
    int x, y;
    cin >> x >> y;
    cout << gcd(x, y);
}
```

快速幂

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

typedef long long ll;

ll qmi(ll a, ll k, ll p)
{
    ll res = 1; //初始为1
    while(k)  // 对k进行二进制化，从低位到高位
    {
        if(k & 1) res = res * a % p; //如果k的二进制末位为1，则乘上当前的a
        k >>= 1;
        a = a * a % p; //更新a。a依次为a^{2^0},a^{2^1},a^{2^2},...,a^{2^logb}
    }
    return res;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll n, a, k, p;
    cin >> n;
    while(n--)
    {
        cin >> a >> k >> p;
        cout << qmi(a, k, p) << "\n";
    }
    return 0;
}
```

线性求逆元

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e6 + 10;
long long n, p, inv[N];
int main(int argc, char const *argv[]) {
    inv[1] = 1;
    scanf("%lld%lld", &n, &p);

    for (int i = 2; i <= n; i++) {
        inv[i] = (p - p / i) * inv[p % i] % p;
    }

    for (int i = 1; i <= n; i++) {
        printf("%lld\n", inv[i]);
    }

    return 0;
}
```

求组合数

```
#include<iostream>
using namespace std;
const int mod = 1e9 + 7;
long long f[2010][2010];
int main()
{
    //预处理
    for (int i = 0; i <= 2000; i++)
    {
        for (int j = 0; j <= i; j++)
        {
            if (!j) f[i][j] = 1;
            else f[i][j] = (f[i - 1][j - 1] + f[i - 1][j]) %
mod;
```

```
        }
    }
    int n;
    cin >> n;
    while (n--)
    {
        int a, b;
        cin >> a >> b;
        printf("%lld\n", f[a][b]);
    }
}
```