程序设计板子集锦

1 Python 头文件

```python
import gc
import sys

# 快速输入
input = sys.stdin.readline  # 替代 input()

# 快速输出
def print(x):
    sys.stdout.write(str(x) + '\n')
sys.stdout.write("\n")  # 替代 print()

# 解除递归深度限制
sys.setrecursionlimit(1000000)  # 默认通常是1000

# 设置大整数转字符串的最大位数
sys.set_int_max_str_digits(50000)

# 设置浮点数字符串转换的最大位数
# sys.float_info  # 查看浮点数信息

# 手动垃圾回收
gc.collect()

# 禁用垃圾回收（谨慎使用）
gc.disable()

# 设置输出缓冲区
sys.stdout.flush()  # 强制刷新输出缓冲区
```

Listing 1: Python 常用头文件

2 C++ 头文件

```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
using ll = long long;
using ull = unsigned long long;
using ld = long double;
const ll INF = 1e18;            // long long的无穷大
const int MOD = 1e9 + 7;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int a;
    cin >> a;
    cout << a << endl;
    return 0;
}
```

Listing 2: C++ 常用头文件

3 二叉树遍历

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

struct Node
{
    int left, right;
}tree[1000005];

vector<int> leftroot, midroot, rightroot;

void leftans(int root)
{
    if (root == 0) return;
    else
    {
        leftroot.push_back(root);
        leftans(tree[root].left);
        leftans(tree[root].right);
    }
}

void midans(int root)
{
    if (root == 0) return;
    else
    {
        midans(tree[root].left);
        midroot.push_back(root);
        midans(tree[root].right);
    }
}

void rightans(int root)
{
    if (root == 0) return;
    else
    {
        rightans(tree[root].left);
        rightans(tree[root].right);
        rightroot.push_back(root);
    }
}

int main()
{
    int n;
```

```
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int l, r;
        cin >> tree[i].left >> tree[i].right;
    }
    leftans(1);
    midans(1);
    rightans(1);
    for (int i = 0; i < n; i++)
    {
        cout << leftroot[i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << midroot[i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << rightroot[i] << " ";
    }
    return 0;
}
```

Listing 3: 二叉树前序、中序、后序遍历

4 01 迷宫 DFS

```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'

int migong[1145][1145];
bool zouguo[1145][1145];
int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0, 0};
int ans[100002], f[1145][1145];

int dfs(int x, int y, int z, int lll, int n)
{
    if (x < 0 || x >= n || y < 0 || y >= n || f[x][y] != -1 || migong[x][y] != z)
        return 0;
    f[x][y] = lll;
    ans[lll]++;
    int count = 1;
    for (int i = 0; i < 4; i++)
    {
        int nx = x + dx[i];
        int ny = y + dy[i];
        count += dfs(nx, ny, !z, lll, n);
    }
    return count;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            char c;
            cin >> c;
            migong[i][j] = c - '0';
        }
    }
    memset(f, -1, sizeof(f));
    memset(ans, 0, sizeof(ans));
    for (int i = 0; i < m; i++)
    {
```

```
47          int a, b;
48          cin >> a >> b;
49          a--; b--;
50          if (f[a][b] == -1)
51              dfs(a, b, migong[a][b], i, n);
52          else
53              ans[i] = ans[f[a][b]];
54      }
55      for (int i = 0; i < m; i++)
56          cout << ans[i] << endl;
57      return 0;
58  }
```

Listing 4: 01 迷宫 DFS 解法

5 BFS 迷宫路径查找

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

char a[110][110]; // 存储迷宫地图
bool vis[110][110]; // 记录访问状态
int n, m; // 迷宫尺寸
struct node {
int x, y;
}; // 定义坐标结构体
int dx[] = {0, 0, 1, -1}, dy[] = {1, -1, 0, 0}; // 方向数组（右左上下）

bool chk(int x, int y) {
    return (x >= 1 && x <= n && y >= 1 && y <= m && !vis[x][y] && a[x][y] != '#');
}

bool bfs() {
    queue<node> q;
    q.push({1, 1}); // 起点入队
    vis[1][1] = 1; // 标记起点已访问
    while (!q.empty()) {
        node p = q.front(); // 取出队首坐标
        q.pop();
        int px = p.x, py = p.y;
        if (px == n && py == m) return true; // 到达终点立即返回
        for (int i = 0; i < 4; ++i) {
            int nx = px + dx[i], ny = py + dy[i];
            if (chk(nx, ny)) { // 合法性检查
                q.push({nx, ny}); // 新坐标入队
                vis[nx][ny] = 1; // 标记已访问
            }
        }
    }
    return false;
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) cin >> a[i][j];
    cout << (bfs() ? "Yes" : "No") << endl;
    return 0;
}
```

Listing 5: BFS 判断迷宫是否可达

6 欧拉函数

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

int oula[114514];

int main()
{
    int n;
    cin >> n;
    int count = 0;
    if (n == 1)
    {
        cout << 0 << endl;
        return 0;
    }
    for (int i = 1; i <= n; i++)
    {
        oula[i] = i;
    }
    for (int i = 2; i <= n; i++)
    {
        if (oula[i] == i)
        {
            for (int j = i; j <= n; j += i)
            {
                oula[j] = oula[j] / i * (i - 1);
            }
        }
    }
    for (int i = 1; i < n; i++)
    {
        count += oula[i];
    }
    cout << count * 2 + 1 << endl;
    return 0;
}
```

Listing 6: 欧拉函数计算

7 并查集

```python
import sys
sys.setrecursionlimit(1000000)
input = lambda:sys.stdin.readline().strip()

def find(n):
    if n != s[n]:
        s[n] = find(s[n])
        return s[n]
    return s[n]

def uni(a,b):
    a_root = find(a)
    b_root = find(b)
    if a_root != b_root:
        s[a_root] = s[b_root]

n,m,k = map(int,input().split())
s = list(range(n+1))
zuo=[[0 for _ in range (n+1)] for _ in range (n+1)]

for _ in range (m):
    a = list(map(int,input().split()))
    if a[2] ==  1:
        uni(a[0],a[1])
    else:
        zuo[a[0]][a[1]] = zuo[a[1]][a[0]] = -1

for _ in range (k):
    a,b = map(int,input().split())
    if find(a)==find(b) and zuo[a][b]!=-1:
        print("No problem")
    elif find(a)!=find(b) and zuo[a][b]!=-1:
        print("OK")
    elif find(a)==find(b) and zuo[a][b]==-1:
        print("OK but...")
    elif find(a)!=find(b) and zuo[a][b]==-1:
        print("No way")
```

Listing 7: Python 并查集实现

8 C++ 并查集实现

```cpp
const int N=1005; // 指定并查集所能包含元素的个数（由题意决定）
int pre[N]; // 存储每个结点的前驱结点
int rank[N]; // 树的高度

void init(int n) // 初始化函数，对录入的 n 个结点进行初始化
{
    for(int i = 0; i < n; i++){
        pre[i] = i; // 每个结点的上级都是自己
        rank[i] = 1; // 每个结点构成的树的高度为 1
    }
}

int find(int x) // 查找结点 x 的根结点
{
    if(pre[x] == x) return x; // 递归出口：x 的上级为 x 本身，则 x 为根结点
    return find(pre[x]); // 递归查找
}

int find(int x) // 改进查找算法：完成路径压缩，将 x 的上级直接变为根结点
{
    if(pre[x] == x) return x;
    return pre[x] = find(pre[x]);
}

bool isSame(int x, int y) // 判断两个结点是否连通
{
    return find(x) == find(y);
}

bool join(int x,int y)
{
    x = find(x);
    y = find(y);
    if(x == y) return false;
    if(rank[x] > rank[y]) pre[y]=x;
    else
    {
        if(rank[x]==rank[y]) rank[y]++;
        pre[x]=y;
    }
    return true;
}
```

Listing 8: C++ 并查集实现

9 树状数组

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

void updata(int x,int k)
{
    for(;x<=n;x+=lowbit(x))
        tree[x]+=k;
}

long long query(int x){
    int ans;
    for(;x;x-=lowbit(x))
        ans=ans+tree[x];
    return ans;
}

inline long long my_union(int x,int y)
{
    return query(x)-query(y-1);
}
```

Listing 9: 树状数组基本操作

10 树状数组完整示例

```cpp
#include<bits/stdc++.h>
using namespace std;
const int Maxn=500005;
int n,s[Maxn],m;

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

struct node
{
    int c[Maxn];
    void add(int x,int d)//修改
    {
        s[x]+=d;
        for(int i=x;i<=n;i+=lowbit(i))
            c[i]+=d;
    }
    int sum(int x)//查询
    {
        int ans=0;
        for(int i=x;i>0;i-=lowbit(i))
            ans+=c[i];
        return ans;
    }
    void init()//初始化
    {
        memset(c,0,sizeof(c));
        for(int i=1;i<=n;i++)
            for(int j=i-lowbit(i)+1;j<=i;j++)
                c[i]=c[i]+s[j];
    }
    void print()//用来调试的输出
    {
        for(int i=1;i<=n;i++)
            printf("%d ",s[i]);
        cout <<" ";
        for(int i=1;i<=n;i++)
            printf("%d ",c[i]);
        cout << endl;
    }
}a;

int main()
{
```

```
47      cin >> n >> m;
48      for(int i=1;i<=n;i++)//输入
49          scanf("%d",&s[i]);
50      a.init();//初始化
51      for(int i=1;i<=m;i++)
52      {
53          int b,x,y;
54          scanf("%d%d%d",&b,&x,&y);
55          if(b==1)//操作1
56              a.add(x,y);//把x为加上y
57          else//操作2
58              printf("%d\n",a.sum(y)-a.sum(x-1));
59      }
60      return 0;
61  }
```

Listing 10: 树状数组完整示例

11 后缀表达式求和

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

stack<int> n;
char ch;
int s,x,y;

int main()
{
    while(ch!='@')
    {
        ch=getchar();
        switch(ch)
        {
            case '+':x=n.top();n.pop();y=n.top();n.pop();n.push(x+y);break;
            case '-':x=n.top();n.pop();y=n.top();n.pop();n.push(y-x);break;
            case '*':x=n.top();n.pop();y=n.top();n.pop();n.push(x*y);break;
            case '/':x=n.top();n.pop();y=n.top();n.pop();n.push(y/x);break;
            case '.':n.push(s);s=0;break;
            default :s=s*10+ch-'0';break;
        }
    }
    printf("%d\n",n.top());
    return 0;
}
```

Listing 11: 后缀表达式求值

12 C++ 常用知识点

```cpp
// NOTE : 1, 创建动态数组:
vector<int> b;
// 使用迭代器遍历
for(auto it = b.begin(); it != b.end(); ++it)

// NOTE: 2.c++排序:
int num[10] = {6,5,9,1,2,8,7,3,4,0};
sort(num, num+10, greater<int>()); // 降序

bool cmp(int x,int y){
    return x % 10 > y % 10;
}
sort(num, num+10, cmp); // 自定义排序

// 3, 用空格隔开输出:
for(int i=0; i<n; i++){
    cout << a[i];
    if(i != n-1) cout << " ";
}

// 4, map 实现字典功能:
map<char, int> mp;
mp['a'] = 1;

// 5, 字符遍历:
for(char i='a'; i<='z'; i++)

// 6, 范围 for 循环:
for(int i : a) { }

// 7, 快读:
ios::sync_with_stdio(false);
cin.tie(0);

// 8, 队列与双端队列:
#include <queue>
#include <deque>
queue<int> q;
deque<int> dq;
```

Listing 12: C++ 常用技巧

13 快速幂与逆元

```cpp
// 快速幂 (a^b % mod)
long long qpow(long long a, long long b, long long mod = MOD) {
    long long res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

// 快速乘 (防溢出)
long long qmul(long long a, long long b, long long mod = MOD) {
    long long res = 0;
    while (b) {
        if (b & 1) res = (res + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return res;
}

// 费马小定理求逆元 (mod 必须为质数)
long long inv(long long a, long long mod = MOD) {
    return qpow(a, mod - 2, mod);
}

// 扩展欧几里得求逆元 (不要求mod为质数)
long long exgcd(long long a, long long b, long long &x, long long &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    long long d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

long long inv_exgcd(long long a, long long mod = MOD) {
    long long x, y;
    exgcd(a, mod, x, y);
    return (x % mod + mod) % mod;
}
```

Listing 13: 快速幂、快速乘、逆元

14 素数筛与质因数分解

```cpp
// 埃氏筛
const int MAXN = 1e6 + 5;
bool is_prime[MAXN];
vector<int> primes;

void eratosthenes(int n) {
    memset(is_prime, true, sizeof(is_prime));
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (long long j = 1LL * i * i; j <= n; j += i) {
                is_prime[j] = false;
            }
        }
    }
}

// 欧拉筛（线性筛）
void euler_sieve(int n) {
    memset(is_prime, true, sizeof(is_prime));
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) primes.push_back(i);
        for (int p : primes) {
            if (1LL * i * p > n) break;
            is_prime[i * p] = false;
            if (i % p == 0) break;
        }
    }
}

// 质因数分解
vector<pair<int, int>> prime_factorize(int n) {
    vector<pair<int, int>> factors;
    for (int p : primes) {
        if (p * p > n) break;
        if (n % p == 0) {
            int cnt = 0;
            while (n % p == 0) {
                n /= p;
                cnt++;
            }
            factors.push_back({p, cnt});
        }
    }
```

```
    if (n > 1) factors.push_back({n, 1});
    return factors;
}
```

Listing 14: 埃氏筛、欧拉筛、质因数分解

15 最短路算法

```cpp
// Dijkstra (优先队列优化)
const long long INF = 1e18;
vector<pair<int, int>> G[MAXN]; // 邻接表: {to, weight}

vector<long long> dijkstra(int start, int n) {
    vector<long long> dist(n + 1, INF);
    vector<bool> visited(n + 1, false);
    priority_queue<pair<long long, int>,
                   vector<pair<long long, int>>,
                   greater<pair<long long, int>>> pq;

    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        if (visited[u]) continue;
        visited[u] = true;

        for (auto [v, w] : G[u]) {
            if (dist[v] > d + w) {
                dist[v] = d + w;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}

// Floyd 多源最短路
void floyd(int n, vector<vector<long long>>& dist) {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}

// SPFA (可处理负权边, 但效率不稳定)
vector<long long> spfa(int start, int n) {
```

```cpp
    vector<long long> dist(n + 1, INF);
    vector<bool> in_queue(n + 1, false);
    vector<int> cnt(n + 1, 0);
    queue<int> q;

    dist[start] = 0;
    q.push(start);
    in_queue[start] = true;

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        in_queue[u] = false;

        for (auto [v, w] : G[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                if (!in_queue[v]) {
                    q.push(v);
                    in_queue[v] = true;
                    cnt[v]++;
                    if (cnt[v] > n) {
                        // 存在负环
                        return vector<long long>();
                    }
                }
            }
        }
    }
    return dist;
}
```

Listing 15: Dijkstra、Floyd、SPFA

16 最小生成树

```cpp
// Kruskal 最小生成树
struct Edge {
    int u, v, w;
    bool operator<(const Edge& other) const {
        return w < other.w;
    }
};

vector<Edge> edges;
int parent[MAXN];

int find(int x) {
    return parent[x] == x ? x : parent[x] = find(parent[x]);
}

bool unite(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return false;
    parent[y] = x;
    return true;
}

long long kruskal(int n) {
    sort(edges.begin(), edges.end());
    for (int i = 1; i <= n; i++) parent[i] = i;

    long long total_weight = 0;
    int edges_used = 0;

    for (const auto& e : edges) {
        if (unite(e.u, e.v)) {
            total_weight += e.w;
            edges_used++;
            if (edges_used == n - 1) break;
        }
    }

    if (edges_used != n - 1) return -1; // 图不连通
    return total_weight;
}
```

Listing 16: Kruskal 算法

17 拓扑排序

```cpp
// Kahn 算法 (BFS实现)
vector<int> topological_sort_kahn(int n, vector<int> indegree, vector<vector<int>> adj) {
    vector<int> result;
    queue<int> q;

    for (int i = 1; i <= n; i++) {
        if (indegree[i] == 0) {
            q.push(i);
        }
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        result.push_back(u);

        for (int v : adj[u]) {
            indegree[v]--;
            if (indegree[v] == 0) {
                q.push(v);
            }
        }
    }

    if (result.size() != n) {
        // 存在环
        return vector<int>();
    }
    return result;
}

// DFS 实现拓扑排序
vector<int> visited, result;

bool dfs_topo(int u, vector<vector<int>>& adj) {
    visited[u] = 1; // 正在访问
    for (int v : adj[u]) {
        if (visited[v] == 1) return false; // 存在环
        if (visited[v] == 0 && !dfs_topo(v, adj)) return false;
    }
    visited[u] = 2; // 访问完成
    result.push_back(u);
    return true;
}

```

```cpp
vector<int> topological_sort_dfs(int n, vector<vector<int>>& adj) {
    visited.assign(n + 1, 0);
    result.clear();

    for (int i = 1; i <= n; i++) {
        if (visited[i] == 0 && !dfs_topo(i, adj)) {
            return vector<int>(); // 存在环
        }
    }
    reverse(result.begin(), result.end());
    return result;
}
```

Listing 17: 拓扑排序 (DFS 和 Kahn 算法)

18 动态规划模板

```cpp
// 01背包 (一维优化)
vector<long long> dp(MAXW, 0);
for (int i = 0; i < n; i++) {
    for (int w = W; w >= weight[i]; w--) {
        dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
    }
}

// 完全背包 (一维优化)
vector<long long> dp(MAXW, 0);
for (int i = 0; i < n; i++) {
    for (int w = weight[i]; w <= W; w++) {
        dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
    }
}

// 最长上升子序列 (LIS) - O(nlogn)
vector<int> lis(const vector<int>& nums) {
    vector<int> tails;
    for (int num : nums) {
        auto it = lower_bound(tails.begin(), tails.end(), num);
        if (it == tails.end()) {
            tails.push_back(num);
        } else {
            *it = num;
        }
    }
    return tails; // tails.size() 即为 LIS 长度
}

// 最长公共子序列 (LCS)
string lcs(const string& s1, const string& s2) {
    int n = s1.size(), m = s2.size();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s1[i-1] == s2[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            } else {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        }
    }

    // 回溯构造LCS
```

```
    string result;
    int i = n, j = m;
    while (i > 0 && j > 0) {
        if (s1[i-1] == s2[j-1]) {
            result += s1[i-1];
            i--, j--;
        } else if (dp[i-1][j] > dp[i][j-1]) {
            i--;
        } else {
            j--;
        }
    }
    reverse(result.begin(), result.end());
    return result;
}
```

Listing 18: 常用 DP 模板

19 字符串算法

```cpp
// KMP 字符串匹配
vector<int> build_next(const string& pattern) {
    int m = pattern.size();
    vector<int> next(m, 0);
    for (int i = 1, j = 0; i < m; i++) {
        while (j > 0 && pattern[i] != pattern[j]) {
            j = next[j - 1];
        }
        if (pattern[i] == pattern[j]) {
            j++;
        }
        next[i] = j;
    }
    return next;
}

vector<int> kmp_search(const string& text, const string& pattern) {
    vector<int> next = build_next(pattern);
    vector<int> positions;
    int n = text.size(), m = pattern.size();

    for (int i = 0, j = 0; i < n; i++) {
        while (j > 0 && text[i] != pattern[j]) {
            j = next[j - 1];
        }
        if (text[i] == pattern[j]) {
            j++;
        }
        if (j == m) {
            positions.push_back(i - m + 1);
            j = next[j - 1];
        }
    }
    return positions;
}

// 字典树 (Trie)
struct TrieNode {
    TrieNode* children[26];
    bool is_end;
    int count;

    TrieNode() : is_end(false), count(0) {
        memset(children, 0, sizeof(children));
    }
};
```

```
47
48  class Trie {
49  private:
50      TrieNode* root;
51
52  public:
53      Trie() {
54          root = new TrieNode();
55      }
56
57      void insert(const string& word) {
58          TrieNode* node = root;
59          for (char ch : word) {
60              int idx = ch - 'a';
61              if (!node->children[idx]) {
62                  node->children[idx] = new TrieNode();
63              }
64              node = node->children[idx];
65              node->count++;
66          }
67          node->is_end = true;
68      }
69
70      bool search(const string& word) {
71          TrieNode* node = root;
72          for (char ch : word) {
73              int idx = ch - 'a';
74              if (!node->children[idx]) {
75                  return false;
76              }
77              node = node->children[idx];
78          }
79          return node->is_end;
80      }
81
82      bool startsWith(const string& prefix) {
83          TrieNode* node = root;
84          for (char ch : prefix) {
85              int idx = ch - 'a';
86              if (!node->children[idx]) {
87                  return false;
88              }
89              node = node->children[idx];
90          }
91          return true;
92      }
93
94      int countPrefix(const string& prefix) {
```

```
        TrieNode* node = root;
        for (char ch : prefix) {
            int idx = ch - 'a';
            if (!node->children[idx]) {
                return 0;
            }
            node = node->children[idx];
        }
        return node->count;
    }
};
```

Listing 19: KMP、字典树

20 数论进阶

```cpp
// 组合数取模（预处理阶乘和逆元）
vector<long long> fact, inv_fact;

void init_comb(int n, long long mod = MOD) {
    fact.resize(n + 1);
    inv_fact.resize(n + 1);

    fact[0] = 1;
    for (int i = 1; i <= n; i++) {
        fact[i] = fact[i - 1] * i % mod;
    }

    inv_fact[n] = inv(fact[n], mod);
    for (int i = n - 1; i >= 0; i--) {
        inv_fact[i] = inv_fact[i + 1] * (i + 1) % mod;
    }
}

long long comb(int n, int k, long long mod = MOD) {
    if (k < 0 || k > n) return 0;
    return fact[n] * inv_fact[k] % mod * inv_fact[n - k] % mod;
}

// 扩展欧几里得（求解 ax + by = gcd(a,b)）
tuple<long long, long long, long long> exgcd_full(long long a, long long b) {
    if (b == 0) return {a, 1, 0};
    auto [d, x1, y1] = exgcd_full(b, a % b);
    return {d, y1, x1 - a / b * y1};
}

// 中国剩余定理（CRT）
long long crt(const vector<long long>& rem, const vector<long long>& mod) {
    long long M = 1;
    for (long long m : mod) M *= m;

    long long result = 0;
    int n = rem.size();
    for (int i = 0; i < n; i++) {
        long long Mi = M / mod[i];
        auto [d, inv, _] = exgcd_full(Mi, mod[i]);
        result = (result + rem[i] * Mi % M * inv % M) % M;
    }
    return (result + M) % M;
}

// 线性同余方程求解 ax   b (mod m)
```

```cpp
vector<long long> linear_congruence(long long a, long long b, long long m) {
    long long d = gcd(a, m);
    if (b % d != 0) return {}; // 无解

    vector<long long> solutions;
    long long a0 = a / d, b0 = b / d, m0 = m / d;
    long long x0, y0;
    exgcd(a0, m0, x0, y0);
    x0 = (x0 % m0 + m0) % m0;
    x0 = (x0 * b0) % m0;

    for (int k = 0; k < d; k++) {
        solutions.push_back((x0 + k * m0) % m);
    }
    return solutions;
}
```

Listing 20: 组合数、CRT、扩展欧几里得

21 计算几何基础

```cpp
const double EPS = 1e-9;
const double PI = acos(-1.0);

// 判断浮点数相等
inline bool eq(double a, double b) { return fabs(a - b) < EPS; }

// 点/向量类
struct Point {
    double x, y;
    Point(double x = 0, double y = 0) : x(x), y(y) {}

    Point operator+(const Point& p) const { return Point(x + p.x, y + p.y); }
    Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
    Point operator*(double k) const { return Point(x * k, y * k); }
    Point operator/(double k) const { return Point(x / k, y / k); }
    bool operator==(const Point& p) const { return eq(x, p.x) && eq(y, p.y); }

    // 点积
    double dot(const Point& p) const { return x * p.x + y * p.y; }

    // 叉积
    double cross(const Point& p) const { return x * p.y - y * p.x; }

    // 模长
    double norm() const { return sqrt(x * x + y * y); }

    // 模长平方
    double norm2() const { return x * x + y * y; }

    // 单位向量
    Point unit() const { return *this / norm(); }

    // 旋转角度（弧度）
    Point rotate(double theta) const {
        return Point(x * cos(theta) - y * sin(theta),
                     x * sin(theta) + y * cos(theta));
    }
};

// 判断点是否在线段上
bool on_segment(Point p, Point a, Point b) {
    return (p - a).cross(p - b) == 0 &&
           (p - a).dot(p - b) <= 0;
}

// 判断两线段是否相交
```

```cpp
bool segments_intersect(Point a1, Point a2, Point b1, Point b2) {
    auto cross = [](Point p1, Point p2, Point p3) {
        return (p2 - p1).cross(p3 - p1);
    };

    double c1 = cross(a1, a2, b1);
    double c2 = cross(a1, a2, b2);
    double c3 = cross(b1, b2, a1);
    double c4 = cross(b1, b2, a2);

    if (c1 * c2 < 0 && c3 * c4 < 0) return true;
    if (c1 == 0 && on_segment(b1, a1, a2)) return true;
    if (c2 == 0 && on_segment(b2, a1, a2)) return true;
    if (c3 == 0 && on_segment(a1, b1, b2)) return true;
    if (c4 == 0 && on_segment(a2, b1, b2)) return true;
    return false;
}

// 多边形面积（点按顺时针或逆时针顺序给出）
double polygon_area(const vector<Point>& poly) {
    double area = 0;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
        area += poly[i].cross(poly[(i + 1) % n]);
    }
    return fabs(area) / 2.0;
}

// 判断点是否在多边形内（射线法）
bool point_in_polygon(Point p, const vector<Point>& poly) {
    int n = poly.size();
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        Point a = poly[i], b = poly[(i + 1) % n];
        if (on_segment(p, a, b)) return true; // 在边界上

        if (a.y > b.y) swap(a, b);
        if (p.y <= a.y || p.y > b.y) continue;
        if ((b - a).cross(p - a) > 0) cnt++;
    }
    return cnt % 2 == 1;
}
```

Listing 21: 点、向量、多边形

22 竞赛注意事项

```python
# 1. 输入输出优化
# Python 使用 sys.stdin.read() 一次读取所有数据
import sys
data = sys.stdin.read().split()
# 然后按需取用

# C++ 使用 ios::sync_with_stdio(false); cin.tie(0);

# 2. 常用库导入
import sys, math, collections, itertools, heapq, bisect, functools

# 3. 调试技巧
def debug(*args):
    import sys
    if __debug__:  # 只有在非优化模式下才输出
        print(*args, file=sys.stderr)

# 4. 随机数生成 (用于哈希或随机算法)
import random, time
random.seed(time.time())

# 5. 常用数据范围
# int 范围: ±2.1e9
# long long 范围: ±9.2e18
# Python int 无限制

# 6. 浮点数精度处理
def equal(a, b, eps=1e-9):
    return abs(a - b) < eps

# 7. 无穷大设置
INF = float('inf')  # Python
const long long INF = 1e18;  // C++

# 8. 常见复杂度参考
# n   10: O(n!) 可能可行
# n   20: O(2^n) 可能可行
# n   100: O(n^3) 可能可行
# n   1000: O(n^2) 可能可行
# n   10^5: O(nlogn) 通常可行
# n   10^6: O(n) 通常可行
# n   10^7: O(n) 需要小心常数
# n > 10^7: O(logn) 或 O(1)

# 9. 模运算技巧
MOD = 10**9 + 7
```

```python
def mod_add(a, b): return (a + b) % MOD
def mod_sub(a, b): return (a - b + MOD) % MOD
def mod_mul(a, b): return (a * b) % MOD
def mod_pow(a, b): return pow(a, b, MOD)

# 10. 位运算技巧
# 判断奇偶: x & 1
# 取最低位的1: x & -x
# 消去最低位的1: x & (x - 1)
# 判断2的幂: x & (x - 1) == 0
# 交换两数: a ^= b; b ^= a; a ^= b;
```

Listing 22: 竞赛技巧与注意事项