






叽叽喳喳动物园模板_2

-    叽叽喳喳动物园模板_2
 - 搜索技术
 - BFS
 - 双向广搜
 - 双向广搜 模板
 - DFS
 - IDAstar
 - IDAstar 模板
 - 基础算法思想
 - 分治法
 - 归并排序
 - 归并排序 模板
 - 归并排序求逆序对 模板
 - 快速排序
 - 快速排序模板
 - 动态规划
 - 基础 DP
 - 硬币问题
 - 不限定硬币数量求每个金额最小硬币数 模板
 - 限定硬币数量求一种金额的所有组合方案 模板
 - 最长公共子序列
 - 最长公共子序列 $O(mn)$ 模板
 - 最长公共子序列 滚动 模板
 - 递推与记忆化搜索
 - 递推
 - 数塔最大和问题递推 模板
 - 递归+记忆化搜索
 - 数塔最大和问题递归+记忆化搜索 模板
 - 区间 DP
 - 回文串
 - 得到回文串的最小花费 模板
 - 数学

- 数论
 - 快速幂
 - 矩阵快速幂 模板
 - 扩展欧几里得算法与二元一次方程的整数解
 - 扩展欧几里得算法求解二元一次方程 模板
- 组合数学
 - 杨辉三角和二项式系数
 - 求杨辉三角的某项或某行 模板
- 公平组合游戏
 - 巴什游戏与 P-position, N-position
 - 巴什游戏 模板
 - 尼姆游戏
 - 尼姆游戏 模板
 - 威佐夫游戏
 - 威佐夫游戏 模板
- 计算几何
 - 二维几何基础
 - 最近点对
 - 求最近点距离 模板
 - 几何模板
 - 公共部分
 - 平面几何：点和线 模板
 - 平面几何：多边形 模板
 - 平面几何：圆 模板
 - 三维几何 模板
 - 使用示例

搜索技术

BFS

双向广搜

双向广搜 模板

```

// 双向广搜
void TBFS(){
    if(/*起点等于终点*/){
        found=true;
        return;
    }
    bool found=false;
    memset(visited,0,sizeof(visited)); // 判重数组
    visited[s1.x][s1.y]=1; // 初始状态标记为1
    visited[s2.x][s2.y]=2; // 结束状态标记为2
    Q1.push(s1); // 初始状态入正向队列
    Q2.push(s2); // 结束状态入反向队列
    while(!Q1.empty() || !Q2.empty()){
        if(!Q1.empty())
            BFS_expand(Q1,true); // 在正向队列中搜索
        if(found) // 搜索结束
            return ;
        if(!Q2.empty())
            BFS_expand(Q2,false); // 在反向队列中搜索
        if(found) // 搜索结束
            return ;
    }
}

void BFS_expand(queue<Status> &Q,bool flag){// 这里可以去掉queue的引用传递改为全局变量方便一点
    s=Q.front(); // 从队列中得到头结点s
    Q.pop()
    for(/*每个s 的子节点 t*/){
        t.state=Gethash(t.temp) // 获取子节点的状态
        if(flag){// 在正向队列中判断
            if (visited[t.state]!=1){// 没在正向队列出现过
                if(visited[t.state]==2) {// 该状态在反向队列中出现过
                    /*各种操作*/
                    found=true;
                    return;
                }
                visited[t.state]=1; // 标记为在正向队列中
                Q.push(t); // 入队
            }
        }else{// 在正向队列中判断
            if (visited[t.state]!=2){// 没在反向队列出现过
                if(visited[t.state]==1){// 该状态在正向队列中出现过
                    /*各种操作*/
                    found=true;
                    return;
                }
                visited[t.state]=2; // 标记为在反向队列中
                Q.push(t); // 入队
            }
        }
    }
}
}

```

DFS

IDAstar

IDAstar 模板

2020-3-15 🐱 整理

```
#include<iostream>
#include<cmath>

using namespace std;

const int BEGINX = 0, BEGINY = 0, ENDX = 5, ENDY = 5, MAXH = 5;// 限制最大深度

int manhattanDistance(int x1, int y1, int x2, int y2) {
    return (abs(x1 - x2) + abs(y1 - y2));
}

bool idastar(int x, int y, int l, int maxL){
    if(l > maxL) return false;
    if(/*终点*/){
        return true;
    }
    if((l + manhattanDistance(x, y, ENDX, ENDY)) >= MAXH) return false;// 这是核心剪枝
    /*
    DFS操作
    */
    return false;
}

int main(){
    int maxl = 1;
    while(!idastar(BEGINX, BEGINY, 0, maxl)){
        maxl++;
    }
    return 0;
}
```

基础算法思想

分治法

归并排序

归并排序 模板

2020-2-20  整理

```

#include<bits/stdc++.h>

using namespace std;
void mergearray(int a[], int l , int mid, int r, int temp[]){ //将两个有序数组合并排序

    int i = l, j = mid + 1;
    int m = mid, n = r;
    int k = 0;
    while(i <= m && j <= n) {
        if(a[i] < a[j])
            temp[k++] = a[i++];
        else
            temp[k++] = a[j++];
    }
    //将剩余的元素存放到临时数组中
    while(i <= m)
        temp[k++] = a[i++];
    while( j <= n)
        temp[k++] = a[j++];
    for(int i = 0; i < k; i++) //临时存放数组的元素存放到原数组中
        a[l + i] = temp[i];

}

void mergesort(int a[], int l, int r, int temp[]){ //将任意两个数组合并排序
    if(l < r) {
        int mid = (l + r) / 2;
        mergesort(a,l,mid,temp); //左边有序
        mergesort(a,mid+1,r,temp); // 右边有序
        mergearray(a,l,mid,r,temp); //将两个有序的数组合并
    }
}

int main(){
    int a[1005];
    int temp[1005];
    int n;
    scanf("%d",&n);
    for(int i = 0; i < n; i++) {
        scanf("%d",&a[i]);
    }
    mergesort(a,0,n-1,temp);
    for(int i = 0 ;i < n; i++) {
        printf("%d ", a[i]);
    }
    return 0;
}

```

归并排序求逆序对 模板

```

#include<iostream>

using namespace std;
const int MAXN = 100005;
typedef long long ll;
ll a[MAXN], b[MAXN], cnt;
void mergearray(ll l, ll mid, ll r){ //将两个有序数组合并排序
    ll i = l, j = mid + 1, t = 0;
    while(i <= mid && j <= r) {
        if(a[i] > a[j]) {
            b[t++] = a[j++];
            cnt += mid - i + 1; //记录逆序对数量
        }
        else
            b[t++] = a[i++];
    }
    //将剩余的元素存放到临时数组中
    while(i <= mid)
        b[t++] = a[i++];
    while(j <= r)
        b[t++] = a[j++];
    for(ll i = 0; i < t; i++) //临时存放数组的元素存放到原数组中
        a[l + i] = b[i];
}

void mergesort(ll l, ll r){ //将任意两个数组合并排序
    if(l < r) {
        ll mid = (l + r) / 2;
        mergesort(l,mid); //左边有序
        mergesort(mid+1,r); // 右边有序
        mergearray(l,mid,r); //将两个有序的数组合并
    }
}

int main(){
    ll n,k;
    while(~scanf("%lld %lld", &n, &k)){
        for(ll i = 0; i < n; i++) {
            scanf("%lld",&a[i]);
        }
        cnt = 0;
        mergesort(0,n-1);
        if(cnt <= k) printf("0\n");
        else printf("%I64d\n", cnt-k);
    }
    return 0;
}

```

快速排序

快速排序模板

2020-2-23  整理


```

#include<bits/stdc++.h>

using namespace std;

//数组打印
void print(int a[], int n){
    for(int i = 0; i < n; i++) {
        cout<<a[i]<<" ";
    }
    cout<<endl;
}

//找到每次的基数位置
int quickposition(int s[], int l, int r){
    int i = l, j = r, x = s[l]; // 以最左元素为基数
    while(i < j) {
        //从右向左找到第一个小于x的数
        while(i < j && s[j] >= x) {
            j--;
        }
        if(i < j) {
            s[i] = s[j]; // 直接替换掉最左元素 最左元素已经备份于x
            i++;
        }
        //从左向右找第一个大于x的数
        while(i < j && s[i] <= x) {
            i++;
        }
        if(i < j) {
            s[j] = s[i]; // 替换掉最右元素, 最左元素已替换
            j--;
        }
    }
    s[i] = x; // i的位置放x, 其左侧元素都小于x, 右侧元素都大于x
    return i;
}

void quickSort(int s[], int l, int r){
    //数组左界小于右界才有意义, 否则说明都已排好, 直接返回即可。
    if(l >= r) {
        return;
    }

    //划分 找到基数位置
    int i = quickposition(s,l,r);

    //递归处理左右两部分 i为分界点
    quickSort(s,l,i-1);
    quickSort(s,i+1,r);
}

```

```
int main(){  
    int arr[] = {72,6,57,88,60,42,83,73,48,85};  
    print(arr,10);  
    quickSort(arr,0,9); //最后一个参数为n-1  
    print(arr,10);  
    return 0;  
}
```

动态规划

基础 DP

硬币问题

不限定硬币数量求每个金额最小硬币数 模板

2020-3-12 🍷 整理

```

/*
    以下为不限定硬币数量 求每个金额的最小硬币数量
*/
const int MONEY = 251; //定义最大金额
const int value = 5; //5种硬币
int type[value] = {1, 5, 10, 25, 50}; //5种面值
int Min[MONEY]; //每个金额对应最少的硬币数量
int Min_path[MONEY] = {0}; //记录最少硬币的路径
void solve(){
    for(int i = 0; i < MONEY; i++) { //初始值为无穷大
        Min[i] = INT_MAX;
    }
    Min[0] = 0;
    for(int i = 0; i < value; i++) {
        for(int j = type[i]; j < MONEY; j++) {
            if(Min[j] > Min[j - type[i]] + 1) {
                Min_path[j] = type[i]; // 在每个金额上记录路径, 即某个硬币的面值
                Min[j] = Min[j - type[i]] + 1;
            }
            Min[j] = min(Min[j], Min[j - type[i]] + 1);
        }
    }
}

void print(int *Min_path, int s) { //打印硬币组合
    while(s){
        cout<<Min_path[s]<<" ";
        s = s - Min_path[s];
    }
}

```

限定硬币数量求一种金额的所有组合方案 模板

2020-3-12 🍷 整理

```

/*
    以下为限定硬币数量 求该金额所对应的所有组合方案数
*/
const int COIN = 101; //要求不超过的硬币数量
const int MONEY = 251; //给定的钱数不超过的金额
int dp[MONEY][COIN] = {0};
int type[5] = {1, 5, 10, 25, 50};
void solve(){
    dp[0][0] = 1;
    for(int i = 0; i < 5; i++) {
        for(int j = 1; j < COIN; j++) {
            for(int k = type[i]; k < MONEY; k++) {
                dp[k][j] += dp[k - type[i]][j-1];
            }
        }
    }
}

int main(){
    int ans[MONEY] = {0}; //记录对应金额的方案
    solve();
    for(int i = 0; i < MONEY; i++) {
        for(int j = 0; j < COIN; j++) {
            ans[i] += dp[i][j];
        }
    }
}

```

最长公共子序列

最长公共子序列 O(mn) 模板

2020-2-20 🐼 整理

```

int dp[1009][1009] = {0};
string strA, strB;

// 最长公共序列 非子串
int lcs(){
    memset(dp, 0, sizeof(dp));
    for(int i = 1; i <= strA.length(); i++){// 注意从1开始
        for(int j = 1; j <= strB.length(); j++){
            if(strA[i - 1] == strB[j - 1]) dp[i][j] = dp[i - 1][j - 1] + 1;
            else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    return dp[strA.length()][strB.length()];
}

```

最长公共子序列 滚动 模板

2020-2-20 🐱 整理

```
// 最长公共子序列 非子串 滚动
int lcs(){
    memset(dp, 0, sizeof(dp));
    bool nowR = 1, preR = 0;
    for(int i = 0; i < strA.length(); i++){
        int j = 0;
        for(swap(nowR, preR), j = 0; j < strB.length(); j++){
            if(strA[i] == strB[j]) dp[nowR][j + 1] = dp[preR][j] + 1;
            else dp[nowR][j + 1] = dp[preR][j + 1] > dp[nowR][j] ? dp[preR][j + 1] : dp[nowR][j]
        }
    }
    return dp[nowR][strB.length()];
}
```

递推与记忆化搜索

递推

数塔最大和问题递推 模板

2020-2-23 🐱 整理

```
int a[150][150]; // a[i][j]是数塔第 i 层的第 j 个数
int dp[150][150]; // dp[i][j]记录从第 i 层第 j 个数开始向下可以得到的最大值
for (int j = 1; j <= n; j++) {
    dp[n][j] = a[n][j];
}
for (int i = n - 1; i > 0; i--) {
    for (int j = 1; j <= i; j++) {
        dp[i][j] = a[i][j] + max(dp[i+1][j], dp[i+1][j+1]);
    }
}
return dp[1][1];
```

递归+记忆化搜索

数塔最大和问题递归+记忆化搜索 模板

2020-2-23 🐱 整理

```

int a[150][150]; // a[i][j]是数塔第 i 层的第 j 个数
int dp[150][150]; // dp[i][j]记录从第 i 层第 j 个数开始向下可以得到的最大值，初始化为 -1
int dfs(int i, int j) {
    if (i == n) return a[i][j];
    if (dp[i][j] >= 0) return dp[i][j];
    dp[i][j] = max(dfs(i+1, j), dfs(i+1, j+1)) + a[i][j];
    return dp[i][j];
}

```

区间 DP

回文串

得到回文串的最小花费 模板

2020-3-12 🐼 整理

```

int m; // 给定字符串的长度
char s[m]; // 字符串
int n; // 字符串中出现的小写字母个数
int w[26]; // 每个小写字母对应的改变价值（给定的增加或删除价值中的较小者）
int dp[m][m]; // 区间 i, j 变成回文的最小花费
for (int i = m - 1; i >= 0; i--) {
    // i 是子区间起点
    for (int j = i + 1; j < m; j++) {
        // j 是子区间终点
        // 如果区间两端相同，说明已经是回文串
        if (s[i] == s[j]) dp[i][j] = dp[i+1][j-1];
        // 否则改变其中一个位置，变成回文串
        else dp[i][j] = min(dp[i+1][j] + w[s[i]-'a'], dp[i][j-1] + w[s[j]-'a']);
    }
}
return dp[0][m-1];

```

数学

数论

快速幂

矩阵快速幂 模板

2020-3-15 🐼 整理

斐波那契数列的另一个公式是：

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{\text{n times}}$$

```

#include<iostream>
#include<string>
#include<cstring>
using namespace std;

const int MAXN = 2; //矩阵的阶
const int MOD = 10000; //题目要求模

struct Matrix{
    int m[MAXN][MAXN];
    Matrix(){
        memset(m, 0, sizeof(m));
    }
    Matrix operator *(Matrix a){ // 矩阵乘法
        Matrix res;
        for(int i = 0; i < MAXN; i++){
            for(int j = 0; j < MAXN; j++){
                for(int k = 0; k < MAXN; k++){
                    res.m[i][j] = (res.m[i][j] + m[i][k] * a.m[k][j]) % MOD;
                }
            }
        }
        return res;
    }
};

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

int main(){
    int n;
    while(cin>>n){
        if(n == -1) return 0;
        Matrix a;
        a.m[0][0] = 1;
        a.m[0][1] = 1;
        a.m[1][0] = 1;
        a.m[1][1] = 0;
        cout<<MPow(a, n)<<endl;
    }
    return 0;
}

```


扩展欧几里得算法与二元一次方程的整数解

扩展欧几里得算法求解二元一次方程 模板

2020-3-16 🐼 整理

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

using namespace std;
//满足 $ax + by = \gcd(a,b)$  可求得一个点 $(x_0, y_0)$ ;
int extend_gcd(int a, int b, int &x, int &y){
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }

    int d = extend_gcd(b, a%b, x, y);
    int tmp = x;
    x = y;
    y = tmp - (a / b) * y;
    return d; //得到a, b的最大公因数
}

int main(){
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c); //ax + by = c 满足 $ax + by = \gcd(a,b)$ 
    int x, y, d;
    d = extend_gcd(a, b, x, y); //得到a, b的最大公因数
    if(c % d != 0) printf("No solution!\n");
    else {
        a /= d, b /= d, c /= d;
        x *= c, y *= c;
        printf("特解: %d %d\n", x, y);
        printf("一个通解: %d %d\n", x+b, y-a);
    }
    return 0;
}
```

组合数学

杨辉三角和二项式系数

求杨辉三角的某项或某行 模板

2020-3-15 🐼 整理

```

#include<vector>

typedef long long ll;

/**
 * 得到杨辉三角第 n 行, 第 m 项,  $0 \leq m \leq n$ 
 * 塔尖为第 0 行, 第 0 项
 * re *= (n - prev + 1) 此处可能溢出, 所以需确保  $n \leq 61$ 
 * 利用二项式系数与杨辉三角中项的关系进行计算
 */
ll getGen(int n, int m) {
    if (m == 0 || m == n) return 1;
    ll re = 1;
    for (int prev = 1; prev <= m; prev++) {
        re *= (n - prev + 1);
        re /= prev;
    }
    return re;
}

/**
 * 得到杨辉三角第 n 行, 结果放在 re[] 中
 * 约定 re 的长度为 n+1, 塔尖为第 0 行
 * ans[prev] = ans[prev-1] * (n - prev + 1) 此处可能溢出, 所以需确保  $n \leq 61$ 
 * 利用二项式系数与杨辉三角中项的关系进行计算
 */
void getGens(int n, vector<ll> &ans) {
    ans[0] = 1;
    ans[n] = 1;
    for (int prev = 1; prev < n; prev++) {
        ans[prev] = ans[prev-1] * (n - prev + 1);
        ans[prev] /= prev;
    }
}

```

公平组合游戏

巴什游戏与 P-position, N-position

巴什游戏 模板

2020-2-23 🐼 整理

```

/*
巴什游戏
n个石子 甲先取 乙后取 每次可以拿1-m个石子 轮流拿 拿到最后一个石子的人获胜
*/
void solve(){
    int n, m;
    cin>>n>>m;
    if(n % (m + 1) == 0) cout<<"second"<<endl;
    else cout<<"first"<<endl;
}

```

尼姆游戏

尼姆游戏 模板

2020-3-18 🐱 整理

```

#include<iostream>
#include<cmath>

using namespace std;
/*
    尼姆游戏
    地上有n堆石子（每堆石子数量小于10000），
    每人每次可从任意一堆石子里取出任意多枚石子扔掉，可以取完，不能不取。
    每次只能从一堆里取。最后没石子可取的人就输了。
    假如甲是先手，且告诉你这n堆石子的数量，他想知道是否存在先手必胜的策略。
    第一行一个整数n，表示有n堆石子。
    第二行有n个数，表示每一堆石子的数量
*/
int main(){
    int n = 0, temp=0, ret=0;
    cin>>n;
    for(int i = 1; i <= n; i++){
        cin>>temp;
        ret ^= temp;
    }
    if(ret) cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
    return 0;
}

```

威佐夫游戏

威佐夫游戏 模板

2020-3-18 🐱 整理

```

#include<iostream>
#include<cmath>

using namespace std;
/*
    威佐夫游戏
    游戏规定，每次有两种不同的取法，一是可以在任意的一堆中取走任意多的石子；
    二是可以在两堆中同时取走相同数量的石子。最后把石子全部取完者为胜者。
    现在给出初始的两堆石子的数目，如果轮到你先取，假设双方都采取最好的策略，问最后你是胜者还是败者。
    1表示自己是胜者
*/
int main(){
    int n, m;
    double gold = (1 + sqrt(5)) / 2;
    while(cin>>n>>m){
        int a = min(n, m);
        int b = max(n, m);
        double k = (double)(b - a);
        int test = (int)(k * gold);
        if(test == a) cout<<0<<endl;
        else cout<<1<<endl;
    }
    return 0;
}

```

计算几何

二维几何基础

最近点对

求最近点距离 模板

2020-3-18  整理

```

/**
 * 计算最近点距离之前要先排序一次 sort(p, p+n, cmpxy)
 */
const double eps = 1e-8;
const int MAXN = 100010;
const double INF = 1e20;

int sgn(double x) {
    if(fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}

struct Point{
    double x,y;
};

double Distance(Point A, Point B) {
    return hypot(A.x-B.x,A.y-B.y);
}

// 先对横坐标 x 排序, 再对 y 排序
bool cmpxy(Point A,Point B) {
    return sgn(A.x-B.x) < 0 || (sgn(A.x-B.x) == 0 && sgn(A.y-B.y) < 0);
}

// 只对 y 坐标排序
bool cmpy (Point A,Point B) {
    return sgn(A.y-B.y)<0;
}

Point p[MAXN],tmp_p[MAXN];

double Closest_Pair(int left, int right) {
    double dis = INF;
    // 只剩1个点
    if(left == right) return dis;
    // 只剩2个点
    if(left + 1 == right) {
        return Distance(p[left], p[right]);
    }
    // 分治
    int mid = (left + right) / 2;
    // 求s1内的最近点对
    double d1 = Closest_Pair(left, mid);
    // 求s2内的最近点对
    double d2 = Closest_Pair(mid+1, right);
    dis = min(d1, d2);
    int k = 0;

    // 在s1和s2中间附近找可能的最小点对
    for(int i = left; i <= right; i++){

```

```

        // 按x坐标来找
        if(fabs(p[mid].x - p[i].x) <= dis) {
            tmp_p[k++] = p[i];
        }
    }

    // 按y坐标排序，用于剪枝。这里不能按x坐标排序
    sort(tmp_p, tmp_p+k, cmpy);
    for(int i = 0; i < k; i++) {
        for(int j = i + 1; j < k; j++) {
            // 剪枝
            if(tmp_p[j].y - tmp_p[i].y >= dis) break;
            dis = min(dis, Distance(tmp_p[i], tmp_p[j]));
        }
    }
    // 返回最小距离
    return dis;
}

```

几何模板

2020-2-20  整理

公共部分

```

#include <bits/stdc++.h>
using namespace std;
const double pi = acos(-1.0); //高精度圆周率
const double eps = 1e-8;      //偏差值
const int maxp = 1010;        //点的数量
int sgn(double x){ //判断x是否等于0
    if(fabs(x) < eps) return 0;
    else return x<0?-1:1;
}
int Dcmp(double x, double y){ //比较两个浮点数: 0 相等; -1 小于; 1 大于
    if(fabs(x - y) < eps) return 0;
    else return x<y ?-1:1;
}

```

平面几何：点和线 模板

//-----平面几何：点和线-----

struct Point{ //定义点和基本运算

double x,y;

Point(){}

Point(double x,double y):x(x),y(y){}

Point operator + (Point B){return Point(x+B.x,y+B.y);}

Point operator - (Point B){return Point(x-B.x,y-B.y);}

Point operator * (double k){return Point(x*k,y*k);} //长度增大k倍

Point operator / (double k){return Point(x/k,y/k);} //长度缩小k倍

bool operator == (Point B){return sgn(x-B.x)==0 && sgn(y-B.y)==0;}

bool operator < (Point B){return sgn(x-B.x)<0 || (sgn(x-B.x)==0 && sgn(y-B.y)<0);} //用于凸包

};

typedef Point Vector; //定义向量

double Dot(Vector A,Vector B){return A.x*B.x + A.y*B.y;} //点积

double Len(Vector A){return sqrt(Dot(A,A));} //向量的长度

double Len2(Vector A){return Dot(A,A);} //向量长度的平方

double Angle(Vector A,Vector B){return acos(Dot(A,B)/Len(A)/Len(B));} //A与B的夹角

double Cross(Vector A,Vector B){return A.x*B.y - A.y*B.x;} //叉积

double Area2(Point A, Point B, Point C){return Cross(B-A, C-A);} //三角形ABC面积的2倍

double Distance(Point A, Point B){return hypot(A.x-B.x,A.y-B.y);} //两点的距离

double Dist(Point A,Point B){return sqrt((A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y));}

Vector Normal(Vector A){return Vector(-A.y/ Len(A), A.x/ Len(A));} //向量A的单位法向量

bool Parallel(Vector A, Vector B){return sgn(Cross(A,B)) == 0;}//向量平行或重合)

Vector Rotate(Vector A, double rad){ //向量A逆时针旋转rad度

return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));

}

struct Line{

Point p1,p2;//线上的两个点

Line(){}

Line(Point p1,Point p2):p1(p1),p2(p2){}

// Line(Point x,Point y){p1 = x;p2 = y;}

// Point(double x,double y):x(x),y(y){}

//根据一个点和倾斜角 angle 确定直线,0<=angle<pi

Line(Point p,double angle){

p1 = p;

if(sgn(angle - pi/2) == 0){p2 = (p1 + Point(0,1));}

else{p2 = (p1 + Point(1,tan(angle)));}

}

//ax+by+c=0

Line(double a,double b,double c){

if(sgn(a) == 0){

p1 = Point(0,-c/b);

p2 = Point(1,-c/b);

}

else if(sgn(b) == 0){

p1 = Point(-c/a,0);

p2 = Point(-c/a,1);

}

```

        else{
            p1 = Point(0,-c/b);
            p2 = Point(1,(-c-a)/b);
        }
    }
};

```

```

typedef Line Segment;    //定义线段, 两端点是Point p1,p2

```

```

//返回直线倾斜角  $0 \leq \text{angle} < \pi$ 

```

```

double Line_angle(Line v){
    double k = atan2(v.p2.y-v.p1.y, v.p2.x-v.p1.x);
    if(sgn(k) < 0)k += pi;
    if(sgn(k-pi) == 0)k -= pi;
    return k;
}

```

```

//点和直线关系:1 在左侧;2 在右侧;0 在直线上

```

```

int Point_line_relation(Point p,Line v){
    int c = sgn(Cross(p-v.p1,v.p2-v.p1));
    if(c < 0)return 1;    //1: p在v的左边
    if(c > 0)return 2;    //2: p在v的右边
    return 0;            //0: p在v上
}

```

```

// 点和线段的关系: 0 点p不在线段v上; 1 点p在线段v上。

```

```

bool Point_on_seg(Point p, Line v){
    return sgn(Cross(p-v.p1, v.p2-v.p1)) == 0 && sgn(Dot(p - v.p1,p- v.p2)) <= 0;
}

```

```

//两直线关系:0 平行,1 重合,2 相交

```

```

int Line_relation(Line v1, Line v2){
    if(sgn(Cross(v1.p2-v1.p1,v2.p2-v2.p1)) == 0){
        if(Point_line_relation(v1.p1,v2)==0) return 1;//1 重合
        else return 0;//0 平行
    }
    return 2; //2 相交
}

```

```

//点到直线的距离

```

```

double Dis_point_line(Point p, Line v){
    return fabs(Cross(p-v.p1,v.p2-v.p1))/Distance(v.p1,v.p2);
}

```

```

//点在直线上的投影

```

```

Point Point_line_proj(Point p, Line v){
    double k=Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
}

```

```

//点p对直线v的对称点

```

```

Point Point_line_symmetry(Point p, Line v){
    Point q = Point_line_proj(p,v);
}

```



```

    return Point(2*q.x-p.x,2*q.y-p.y);
}

//点到线段的距离
double Dis_point_seg(Point p, Segment v){
    if(sgn(Dot(p- v.p1,v.p2-v.p1))<0 || sgn(Dot(p- v.p2,v.p1-v.p2))<0) //点的投影不在线段上
        return min(Distance(p,v.p1),Distance(p,v.p2));
    return Dis_point_line(p,v); //点的投影在线段上
}

//求两直线ab和cd的交点
//调用前要保证两直线不平行或重合
Point Cross_point(Point a,Point b,Point c,Point d){ //Line1:ab, Line2:cd
    double s1 = Cross(b-a,c-a);
    double s2 = Cross(b-a,d-a); //叉积有正负
    return Point(c.x*s2-d.x*s1,c.y*s2-d.y*s1)/(s2-s1);
}

//两线段是否相交：1 相交, 0不相交
bool Cross_segment(Point a,Point b,Point c,Point d){//Line1:ab, Line2:cd
    double c1=Cross(b-a,c-a),c2=Cross(b-a,d-a);
    double d1=Cross(d-c,a-c),d2=Cross(d-c,b-c);
    return sgn(c1)*sgn(c2)<0 && sgn(d1)*sgn(d2)<0; //注意交点是端点的情况不算在内
}

```

平面几何：多边形 模板

```

//-----平面几何：多边形-----
struct Polygon{
    int n;    //多边形的顶点数
    Point p[maxp];    //多边形的点
    Line v[maxp];    //多边形的边
};
//判断点和任意多边形的关系：3 点上；2 边上；1 内部；0 外部
int Point_in_polygon(Point pt,Point *p,int n){ //点pt, 多边形Point *p
    for(int i = 0;i < n;i++){ //点在多边形的顶点上
        if(p[i] == pt)return 3;
    }
    for(int i = 0;i < n;i++){//点在多边形的边上
        Line v=Line(p[i],p[(i+1)%n]);
        if(Point_on_seg(pt,v)) return 2;
    }
    int num = 0;
    for(int i = 0;i < n;i++){
        int j = (i+1)% n;
        int c = sgn(Cross(pt-p[j],p[i]-p[j]));
        int u = sgn(p[i].y - pt.y);
        int v = sgn(p[j].y - pt.y);
        if(c > 0 && u < 0 && v >=0) num++;
        if(c < 0 && u >=0 && v < 0) num--;
    }
    return num != 0; //1 内部；0 外部
}

double Polygon_area(Point *p, int n){ //Point *p表示多边形。从原点开始划分三角形
    double area = 0;
    for(int i = 0;i < n;i++){
        area += Cross(p[i],p[(i+1)%n]);
    }
    return area/2;    //面积有正负，不能简单地取绝对值
}

Point Polygon_center(Point *p, int n){ //求多边形重心。Point *p表示多边形。
    Point ans(0,0);
    if(Polygon_area(p,n)==0) return ans;
    for(int i = 0;i < n;i++){
        ans = ans + (p[i]+p[(i+1)%n]) * Cross(p[i],p[(i+1)%n]); //面积有正负
    }
    return ans/Polygon_area(p,n)/6.;
}

//Convex_hull()求凸包。凸包顶点放在ch中，返回值是凸包的顶点数
int Convex_hull(Point *p,int n,Point *ch){
    sort(p,p+n);    //对点排序：按x从小到大排序，如果x相同，按y排序
    n=unique(p,p+n)-p;    //去除重复点
    int v=0;
    //求下凸包。如果p[i]是右拐弯的，这个点不在凸包上，往回退
    for(int i=0;i<n;i++){
        while(v>1 && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2]))<=0)
            v--;
    }
}

```

```

        ch[v++] = p[i];
    }
    int j = v;
    //求上凸包
    for(int i = n - 2; i >= 0; i--){
        while(v > j && sgn(Cross(ch[v - 1] - ch[v - 2], p[i] - ch[v - 2])) <= 0)
            v--;
        ch[v++] = p[i];
    }
    if(n > 1) v--;
    return v;    //返回值v是凸包的顶点数
}

```

平面几何：圆 模板

//-----平面几何：圆-----

```
struct Circle{
    Point c;//圆心
    double r;//半径
    Circle(){}
    Circle(Point c,double r):c(c),r(r){}
    Circle(double x,double y,double _r){c=Point(x,y);r = _r;}
};
```

//点和圆的关系：0 点在圆内，1 圆上，2 圆外。

```
int Point_circle_relation(Point p, Circle C){
    double dst = Distance(p,C.c);
    if(sgn(dst - C.r) < 0) return 0; //点在圆内
    if(sgn(dst - C.r) ==0) return 1; //圆上
    return 2; //圆外
}
```

//直线和圆的关系：0 直线在圆内，1 直线和圆相切，2 直线在圆外

```
int Line_circle_relation(Line v,Circle C){
    double dst = Dis_point_line(C.c,v);
    if(sgn(dst-C.r) < 0) return 0; //直线在圆内
    if(sgn(dst-C.r) ==0) return 1; //直线和圆相切
    return 2; //直线在圆外
}
```

//线段和圆的关系：0 线段在圆内，1 线段和圆相切，2 线段在圆外

```
int Seg_circle_relation(Segment v,Circle C){
    double dst = Dis_point_seg(C.c,v);
    if(sgn(dst-C.r) < 0) return 0; //线段在圆内
    if(sgn(dst-C.r) ==0) return 1; //线段和圆相切
    return 2; //线段在圆外
}
```

//直线和圆的交点 hdu 5572

```
int Line_cross_circle(Line v,Circle C,Point &pa,Point &pb){//pa, pb是交点。返回值是交点个数
    if(Line_circle_relation(v, C)==2) return 0;//无交点
    Point q = Point_line_proj(C.c,v); //圆心在直线上的投影点
    double d = Dis_point_line(C.c,v); //圆心到直线的距离
    double k = sqrt(C.r*C.r-d*d); //
    if(sgn(k) == 0){ //1个交点，直线和圆相切
        pa = q;
        pb = q;
        return 1;
    }
    Point n=(v.p2-v.p1)/ Len(v.p2-v.p1); //单位向量
    pa = q + n*k;
    pb = q - n*k;
    return 2;//2个交点
}
```

三维几何 模板

```

//-----三维几何-----
//三维: 点
struct Point3{
    double x,y,z;
    Point3(){}
    Point3(double x,double y,double z):x(x),y(y),z(z){}
    Point3 operator + (Point3 B){return Point3(x+B.x,y+B.y,z+B.z);}
    Point3 operator - (Point3 B){return Point3(x-B.x,y-B.y,z-B.z);}
    Point3 operator * (double k){return Point3(x*k,y*k,z*k);}
    Point3 operator / (double k){return Point3(x/k,y/k,z/k);}
    bool operator == (Point3 B){return sgn(x-B.x)==0 && sgn(y-B.y)==0 && sgn(z-B.z)==0;}
};
typedef Point3 Vector3;
//点积。和二维点积函数同名。C++允许函数同名。
double Dot(Vector3 A,Vector3 B){return A.x*B.x+A.y*B.y+A.z*B.z;}
//叉积
Vector3 Cross(Vector3 A,Vector3 B){return Point3(A.y*B.z-A.z*B.y,A.z*B.x-A.x*B.z,A.x*B.y-A.y*B.x)}
double Len(Vector3 A){return sqrt(Dot(A,A));} //向量的长度
double Len2(Vector3 A){return Dot(A,A);} //向量长度的平方
double Distance(Point3 A,Point3 B){
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y)+(A.z-B.z)*(A.z-B.z));
}
double Angle(Vector3 A,Vector3 B){return acos(Dot(A,B)/Len(A)/Len(B));} //A与B的夹角
//三维: 线
struct Line3{
    Point3 p1,p2;
    Line3(){}
    Line3(Point3 p1,Point3 p2):p1(p1),p2(p2){}
};
typedef Line3 Segment3; //定义线段, 两端点是Point p1,p2

//三角形面积的2倍
double Area2(Point3 A,Point3 B,Point3 C){return Len(Cross(B-A, C-A));}

//三维: 点到直线距离
double Dis_point_line(Point3 p, Line3 v){
    return Len(Cross(v.p2-v.p1,p-v.p1))/Distance(v.p1,v.p2);
}

//三维: 点在直线上
bool Point_line_relation(Point3 p,Line3 v){
    return sgn( Len(Cross(v.p1-p,v.p2-p))) == 0 && sgn(Dot(v.p1-p,v.p2-p))== 0;
}
//三维: 点到线段距离。
double Dis_point_seg(Point3 p, Segment3 v){
    if(sgn(Dot(p- v.p1,v.p2-v.p1)) < 0 || sgn(Dot(p- v.p2,v.p1-v.p2)) < 0)
        return min(Distance(p,v.p1),Distance(p,v.p2));
    return Dis_point_line(p,v);
}
//三维: 点 p 在直线上的投影
Point3 Point_line_proj(Point3 p, Line3 v){

```

```

    double k=Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
}
//三维: 平面
struct Plane{
    Point3 p1,p2,p3;//平面上的三个点
    Plane(){}
    Plane(Point3 p1,Point3 p2,Point3 p3):p1(p1),p2(p2),p3(p3){}
};
//平面法向量
Point3 Pvec(Point3 A,Point3 B,Point3 C){ return Cross(B-A,C-A);}
Point3 Pvec(Plane f){ return Cross(f.p2-f.p1,f.p3-f.p1);}
//四点共平面
bool Point_on_plane(Point3 A,Point3 B,Point3 C,Point3 D){
    return sgn(Dot(Pvec(A,B,C),D-A)) == 0;
}
//两平面平行
int Parallel(Plane f1, Plane f2){
    return Len(Cross(Pvec(f1),Pvec(f2))) < eps;
}
//两平面垂直
int Vertical (Plane f1, Plane f2){
    return sgn(Dot(Pvec(f1),Pvec(f2)))==0;
}
//直线与平面的交点p, 返回值是交点个数
int Line_cross_plane(Line3 u,Plane f,Point3 &p){
    Point3 v = Pvec(f);
    double x = Dot(v, u.p2-f.p1);
    double y = Dot(v, u.p1-f.p1);
    double d = x-y;
    if(sgn(x) == 0 && sgn(y) == 0) return -1;//-1: v在f上
    if(sgn(d) == 0)return 0;                //0: v与f平行
    p = ((u.p1 * x)-(u.p2 * y))/d;          //v与f相交
    return 1;
}

//四面体有向体积*6
double volume4(Point3 A,Point3 B,Point3 C,Point3 D){return Dot(Cross(B-A,C-A),D-A);}

```

使用示例

```

int main(){
    Point a(0,1),b(0,0),c(1,1),d(1,2),p(1.5,1);
    double a1=5,b1=6,c1=1;
    Line k(a,b),k2(c,d);
    Point pr(1,1),cr(1,1); double r=1; Circle C(cr,r);

    cout<<endl<<"Line_circle_relation="<<Line_circle_relation(k,C);
    cout<<endl<<"Seg_circle_relation="<<Seg_circle_relation(k,C);
    cout<<endl<<"Point_circle_relation="<<Point_circle_relation(pr,C);
    cout<<endl<<"parallel="<<Parallel(a,b)<<endl;
    cout<<"dot="<<Dot(a,b)<<endl<<" angle="<<Angle(a,b)<<endl;
    cout<<"90:"<<sgn(Rotate(a, -pi/2).x)<<endl<<Rotate(a, -pi/2).y;
    cout<<endl<<"line angle="<<Line_angle(k)*4;
    cout<<endl<<"line place="<<Point_line_relation(p,k);
    cout<<endl<<"point_on_seg="<<Point_on_seg(p,k);
    cout<<endl<<"dis_point_line="<<Dis_point_line(p,k);
    cout<<endl<<"dis_point_line="<<Dis_point_seg(p,k);
    Point pp=Cross_point(a,b,c,d);
    cout<<endl<<"crosspoint="<<pp.x<<" "<<pp.y;
    cout<<endl<<"cross_seg="<<Cross_segment(a,b,c,d);
    cout<<endl<<"distance="<<Distance(a,b);
    cout<<endl<<"line_relation="<<Line_relation(k,k2);
    Point g[4];
    g[0]=a;g[1]=b;g[2]=c;g[3]=d;
    cout<<endl<<"Point_in_polygon="<<Point_in_polygon(p,g,4);
    cout<<endl<<"Polygon_area="<<Polygon_area(g,4);
    cout<<endl<<endl;
    return 0;
}

```