Reference of ICPC Algorithm Implementation

Yucheng Chen|Jifan Liang

代码 3
数据生成 4
一般 4
图 4
che.sh 5
输入输出 6
快速读入整数（即时） 6
快速读入整数（缓冲） 6
输出__int128 6
数据结构（静态分配内存） 7
优先队列 7
循环队列 7
邻接前向链表 7
可持久化并查集 8
可并堆 9
一次函数（等差数列区间赋值）线段树 10
线段树二分搜索 11
区间 Distinct 计数（在线）可持久化线段树 12
KD 树 13
Splay 16
卷积 21
FWT：快速沃尔什变换 21
NTT：快速数论变换 21
FFT：快速傅里叶变换 23
字符串 25
AC 自动机及其 FailTree 括号序 25
SAM：后缀自动机 26
SAM：后缀自动机（压空间） 26
SA：后缀数组 29
KMP 30
exKMP 31
回文树 32
网络流 34
Dinic 34
SAP 35
最小费用最大流 SPFA 36
最小费用最大流 Primary Dual 37
树剖分 40
树链剖分 40
点剖分 41
轻重链剖分 42
动态树 Link-Cut-Tree 44
图论 48
欧拉回路或路径 48
单源最短路 Dijkstra（堆优化） 49
SPFA（DFS）判断负环 49
SCC 强连通分量 50
二分图匹配 51
Tarjan 强连通分量、双连通分量 52
Tarjan 割边以及无向图缩块 53
Tarjan 割点及其控制范围 54
数论 57
快速幂和快速乘（非递归） 57
欧几里得相关算法 57
GCD（非递归） 57
exGCD 57
直线下方的整点数 57
两分数间的分母最小的分数 57
平面向量线性组合的最小模长 57
乘法逆元（基于 exGCD） 57
筛法 58
线性筛 58
杜教筛（莫比乌斯函数前缀和） 59
线性递推逆元、阶乘及其逆元 59
求原根 60
离散对数 60
BSGS（要求模数为素数） 60
exBSGS 60
同余方程组 61
CRT（要求模数两两互质） 61
exCRT 61
大整数分解 62
Miller Rabin 62
Pollard's rho 62
组合数学 63
整数分拆数 63
无限制的整数分拆数 63
要求分拆的每个数出现的次数严格小于 k 63
计算几何 64
点（向量）类 64
扫描线 64
K 圆覆盖区域面积并 66

二维凸包（上下凸壳法） 71
三维凸包相关 72
矩阵 79
高斯消元 79
标量类 81
有理数类 81
公式定理 82
数论 83
整值函数 83
组合数学 84
差分与不定和 84
二项式系数与特殊计数数列 84
通项公式 84
递推关系 85
求和关系 85
容斥原理 85
反演 85
错位排列 85
迪利克雷卷积 86
杜教筛 86
生成函数 86
（组合）生成函数 86
分拆数 86
排列生成函数 86
Ramsey 定理 87
Pólya 计数 87
连续数学 88
三角函数与双曲函数 88
基本性质 88
和差 88
和差化积 88
积化和差 89
倍角 89
半角 90
万能 90
辅助角 90
几何 90
正余弦定理 90
三角形各心 90
面积 90
表面积 91
体积 91
特殊函数的微分 91
积分 91
不定积分 91
定积分 98
坐标变换 98
级数 98
近似公式 98

代码

原则：为了在使用时能够快速誊抄，所有代码函数变量命名尽可能简短，并充分利用以下宏定义缩减代码长度。

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#define fill(a, x, n) memset(a, x, ((n) + 2) * sizeof *(a))
#define fo(i, l, r) for (auto i = l, _end = r; i <= _end; ++i)
#define fd(i, l, r) for (auto i = l, _end = r; i >= _end; --i)
#define lowbit(x) ((x) & -(x))
#define db(x) printf(x "Line: %d", __LINE__)
#define FI(x) freopen(x, "r", stdin)
#define FO(x) freopen(x, "w", stdout)
using ll = long long;
using namespace std;

int main() {
}
```

数据生成

一般

```
#include<ctime>
#include<random>

template <class T>
using rnd = uniform_int_distribution<T>;
//uniform_real_distribution<T>

int main(int arg, const char * args[]) {
    int n = arg>1 ? atoi(args[1]) : 10; //参数1: n
    default_random_engine e(arg>2 ? atoi(args[2]) : time(NULL)); //参数2: 随机种子
    printf("%d\n", n);
    fo(i, 1, n) printf("%d%c", rnd<int>(0, 10)(e), " \n"[i == n]); //范围[0..10]
}
```

图

```
#include<cstdlib>
#include<ctime>
#include<map>

using P = pair<int, int>;
map<P, bool> v;

int Range(int l, int r) {// Generate random number in section [l,r]
    return l + rand() % (r - l + 1);
}
void GraphGen(int l, int r, int m) { // Generate a Graph with vertex id of [l,r] and m edges
                                    // --------------------
    for (int i = l + 1; i <= r && m; i++) {
        int y = i, x = Range(l, i - 1);
        v[P(x, y)] = 1;
        printf("%d %d\n", x, y);
        m--;
    }
    // --------------------
    //若要 不保证联通的图，删除上面部分
    while (m--) {
        int x, y;
        // 所有边id均x小于y，若是有向边，则为DAG，若是无向，则是一般图
        do {
```

```
            x = Range(l, r - 1), y = Range(x + 1, r);
        } while (v[P(x, y)]);
        v[P(x, y)] = 1;
        printf("%d %d\n", x, y);//若要有向一般图，随机化x,y两者输出顺序
    }
    v.clear();
}

int main() {
    srand((unsigned)time(NULL));
    int n = 5, m = rand() % (n * 2) + 1;
    printf("%d %d\n", n, m);
    GraphGen(1, n, m);
}
```

che.sh

```
while true; do
. / mak>In
. / br>tmp.out
. / b>tmp2.out
if diff tmp.out tmp2.out; then
printf AC
else
echo WA
exit 0
fi
done
//Activate: sh che.sh

```

输入输出

快速读入整数（即时）

```
#include<cctype>
template <class T> void read(T& x) { //若要支持负数，解除下面的注释
    //bool sgn = 0;
    char c = 0;
    for (x = 0; !isdigit(c); c = getchar()) /*sgn ^= c == '-'*/;
    for (; isdigit(c); c = getchar()) x = x * 10 + c - '0';
    //if (sgn) x = -x;
}
```

快速读入整数（缓冲）

```
#include<cctype>
char Buf[BUF], *buf = Buf;
template <class T> void read(T& x) { //若要支持负数，解除下面的注释
    //bool sgn = 0;
    for (x = 0; !isdigit(*buf); buf++) /*sgn ^= *buf == '-'*/;
    while (isdigit(*buf)) x = x * 10 + *buf++ - '0';
    //if (sgn) x = -x;
}
int main() {
    fread(Buf, 1, BUF, stdin);
}
```

输出__int128

```
void print(__int128 x) {
    using ull = unsigned long long;
    const ull T = 1000000000000000000llu; //1e19
    if (x < 0) putchar('-'), x = -x; // 若不需要支持负数，去掉这行
    if (x < T) printf("%llu", ull(x));
    else printf("%llu%019llu", ull(x / T), ull(x % T));
}
```

数据结构（静态分配内存）

优先队列

```
template <class T, int N>
struct Pqueue {//默认为大根堆，解除下面注释即变为小根堆
    int n;
    T a[N];
    void clear() { n = 0; }
    bool empty() { return !n; }
    T top() { return *a; }
    void pop() { pop_heap(a, a + n-- /* ,greater<T>() */); }
    void push(T x) {
        a[n++] = x;
        push_heap(a, a + n /* ,greater<T>() */);
    }
};
```

循环队列

```
template <class T, int N>
struct Cqueue {
    int head, tail; //左闭右开
    T a[N];
    void clear() { head = tail = 0; }
    bool empty() { return head == tail; }
    T front() { return a[head]; }
    void pop() { ++head == N ? head = 0 : 0; }
    void push(T x) {
        a[tail] = x;
        ++tail == N ? tail = 0 : 0;
    }
};
```

邻接前向链表

```
#define ttn template <class T, int N >
ttn struct List { //支持C++11范围for循环
    static T a[N]; //N为公用空间大小，注意要+1
    static int nxt[N]; //nxt[0]用于存放已使用空间大小
    struct Iter {
        int p;
        void operator++() { p = nxt[p]; }
        T operator*() { return a[p]; }
        bool operator!=(Iter y) { return p != y.p; }
```

```
    };
    static void reset() { *nxt = 0; } //多组数据时一定要reset()
    int s;
    Iter begin() { return {s}; }
    Iter end() { return {0}; }
    void clear() { s = 0; }
    //bool empty() { return !s; }
    //T front() { return a[s]; }
    //void pop() { if (s) s = nxt[s]; }
    void push(T x) { a[++*nxt] = x, nxt[*nxt] = s, s = *nxt; }
};
ttn T List<T, N>::a[N];
ttn int List<T, N>::nxt[N];
```

可持久化并查集

```
const int N = 2e5 + 10, M = 50 * N;//M is dots set of perisi_SegmtTree , comsuming 160MB
                                   //M should as large as possible,Though 40*M is adequate
int n, m;
namespace Persi_dsu {
    int rot[N], dep[M], to[M], T, n;
    struct Node {
        int l, r;
    }tr[M];
    int Getid(int l, int r, int tar, int p) {
        //返回点tar在线段树的位置ps,且dep[ps],to[ps]分别为tar在id版本中的深度与父亲。
        if (l == r) return p;
        int m = l + r >> 1;
        if (tar <= m) return Getid(l, m, tar, tr[p].l);
        else return Getid(m + 1, r, tar, tr[p].r);
    }
    void Insert(int l, int r, int tar, int t, int de, int &p) {
        ++T;
        tr[T] = tr[p];
        p = T;
        if (l == r) {
            to[p] = t, dep[p] = de;
            return;
        }
        int m = l + r >> 1;
        if (tar <= m) Insert(l, m, tar, t, de, tr[p].l);
        else Insert(m + 1, r, tar, t, de, tr[p].r);
    }
    int dsu(int x, int id) {//id版本的x的祖先
        int ps = Getid(1, n, x, rot[id]);
```

```
        if (x == to[ps]) return x;
        return dsu(to[ps], id);
    }
    bool Connect(int a, int b, int id) {//在id版本联通a,b
        int A = dsu(a, id); int B = dsu(b, id);
        if (A == B) return 1; // Already connected
        int psA = Getid(1, n, A, rot[id]), psB = Getid(1, n, B, rot[id]);
        if (dep[psA] > dep[psB]) swap(psA, psB), swap(A, B);
        if (dep[psA] == dep[psB]) Insert(1, n, B, B, dep[psB] + 1, rot[id]);
        Insert(1, n, A, B, 0, rot[id]);
        return 0;//Wasn't connected
    }
    bool Same(int a, int b, int id) {//a，b在id版本是否联通
        return dsu(a, id) == dsu(b, id);
    }
    void intoHistory(int now, int k) {//将第now版本换成第k版本
        rot[now] = rot[k];
    }
    void Initiate(int cnt) {//并查集初始化，数据接口：cnt为点数
        n = cnt;
        for (int i = 1; i <= n; i++) Insert(1, n, i, i, 1, rot[0]);
    }
}
```

可并堆

```
int T, rot[N];
struct mark {//Lazy tag 对子树所有节点权值*mul+plu
    ll mul, plu;
};
struct node {
    mark v;
    int l, r, num;
    ll att;
}tr[N];
int dist[N];
void Marking(int ps, int a, ll v) {//Lazy tag
    if (a) {
        tr[ps].v.mul *= v;
        tr[ps].v.plu *= v;
    } else {
        tr[ps].v.plu += v;
    }
}
void Effect(int ps) {//Down tag
```

```
    if (ps == 0) return;
    tr[ps].att *= tr[ps].v.mul;
    tr[ps].att += tr[ps].v.plu;
    if (tr[ps].l != 0) {
        int l = tr[ps].l;
        tr[l].v.mul *= tr[ps].v.mul;
        tr[l].v.plu = tr[l].v.plu * tr[ps].v.mul + tr[ps].v.plu;
    }
    if (tr[ps].r != 0) {
        int l = tr[ps].r;
        tr[l].v.mul *= tr[ps].v.mul;
        tr[l].v.plu = tr[l].v.plu * tr[ps].v.mul + tr[ps].v.plu;
    }
    tr[ps].v.mul = 1, tr[ps].v.plu = 0;
}
int Merge(int a, int b) {// merge subtree a,b(node id)
    if (a == 0) return b; if (b == 0) return a;
    Effect(a), Effect(b);// Down included
    if (tr[a].att > tr[b].att) swap(a, b);
    tr[a].r = Merge(tr[a].r, b);
    if (dist[tr[a].l] < dist[tr[a].r]) swap(tr[a].l, tr[a].r);
    if (tr[a].r == 0) dist[a] = 0; else dist[a] = dist[tr[a].r] + 1;
    return a;

}
int pop(int ps) {//delete node ps , ps should be root,returing new rootid
    return Merge(tr[ps].l, tr[ps].r);
}
int main() {
    // new a node
    tr[++T].att = 123;
    rot[i] = T;
}
```

一次函数（等差数列区间赋值）线段树

```
const int N = 1e5 + 10;
struct Node {
    ll a, b, S;
    bool tag;
}tr[N * 4];

void SD(int l, int r, int p) {//标记下传
    if (tr[p].tag) {
        tr[p].S = (tr[p].a * (l + r) + 2 * tr[p].b) * (r - l + 1) / 2;
```

```
        if (l != r) {
            tr[p * 2] = tr[p], tr[p * 2 + 1] = tr[p];
        }
        tr[p].tag = 0;
    }
}
void Change(int l, int r, int _l, int _r, ll A, ll B, int p) {//将[l,r]每个点x赋值为Ax+B
    if (_l > _r) return;
    int m = (l + r) >> 1;
    if (l == _l && r == _r) {
        tr[p].a = A, tr[p].b = B;
        tr[p].tag = 1;
        SD(l, r, p);
        return;
    }
    SD(l, r, p);
    if (_r <= m) Change(l, m, _l, _r, A, B, p * 2);
    else if (_l > m) Change(m + 1, r, _l, _r, A, B, p * 2 + 1);
    else Change(l, m, _l, m, A, B, p * 2), Change(m + 1, r, m + 1, _r, A, B, p * 2 + 1);
    SD(l, m, p * 2), SD(m + 1, r, p * 2 + 1);
    tr[p].S = tr[p * 2].S + tr[p * 2 + 1].S;
}
ll Ask(int l, int r, int tar, int p) {//单点查询
    int m = (l + r) >> 1;
    SD(l, r, p);
    if (l == r) return tr[p].S;
    if (tar <= m) return  Ask(l, m, tar, p * 2);
    else   return Ask(m + 1, r, tar, p * 2 + 1);
}
ll Sum(int l, int r, int _l, int _r, int p) {//区间求和
    int m = (l + r) >> 1;
    SD(l, r, p);
    if (_l == l && r == _r) return tr[p].S;
    if (_r <= m) return Sum(l, m, _l, _r, p * 2);
    else if (_l > m) return Sum(m + 1, r, _l, _r, p * 2 + 1);
    else return Sum(l, m, _l, m, p * 2) + Sum(m + 1, r, m + 1, _r, p * 2 + 1);
}
```

线段树二分搜索

```
struct node {
    int l, r, sum;
}tr[N * 40];// Space with NlogN

int Ask(int l, int r, int _l, int _r, int p) {
```

```
    if (p == 0) return 0;
    int m = l + r >> 1;
    if (_l == l && _r == r) return tr[p].sum;
    if (_r <= m) return Ask(l, m, _l, _r, tr[p].l);
    else if (_l > m)  return Ask(m + 1, r, _l, _r, tr[p].r);
    else return Ask(l, m, _l, m, tr[p].l) + Ask(m + 1, r, m + 1, _r, tr[p].r);
}
//interface: find(1,n,L,R,k,p) 返回[L，R]中前缀和为k的最靠左位置
int find(int l, int r, int _l, int _r, int k, int p) {
    int m = l + r >> 1;
    if (l == r) return l;
    if (_r <= m) return find(l, m, _l, _r, k, tr[p].l);
    else if (_l > m) return find(m + 1, r, _l, _r, k, tr[p].r);
    else {
        int tmp = Ask(l, m, _l, m, tr[p].l);
        if (tmp >= k)
            return find(l, m, _l, m, k, tr[p].l);
        else return find(m + 1, r, m + 1, _r, k - tmp, tr[p].r);
    }
}
```

区间 Distinct 计数（在线）可持久化线段树

```
int a[N], n;

struct PersiSGT {
    struct node {
        int l, r, sum;
    }tr[N * 40];// Space with NlogN !! As large as can be;
    int rot[N], T;
    void succ(int fr, int to) {//to从fr处继承。
        rot[to] = rot[fr];
    }
    void Insert(int l, int r, int tar, int &p, int va) {
        tr[++T] = tr[p];
        p = T;
        if (l == r) {
            tr[p].sum += va;
            return;
        }
        int m = l + r >> 1;
        if (tar <= m) Insert(l, m, tar, tr[p].l, va);
        else   Insert(m + 1, r, tar, tr[p].r, va);
        tr[p].sum = tr[tr[p].l].sum + tr[tr[p].r].sum;
    }
```

```
    void f(int x, int y, int z) {//点修改,时间为x，线段树下表为y的值加上z，f(x,y)+=z
        Insert(1, n, y, rot[x], z);
    }
    int Ask(int l, int r, int _l, int _r, int p) {
        if (p == 0) return 0;
        int m = l + r >> 1;
        if (_l == l && _r == r) return tr[p].sum;
        if (_r <= m) return Ask(l, m, _l, _r, tr[p].l);
        else if (_l > m)  return Ask(m + 1, r, _l, _r, tr[p].r);
        else return Ask(l, m, _l, m, tr[p].l) + Ask(m + 1, r, m + 1, _r, tr[p].r);
    }
    int ac(int xa, int xb, int ya, int yb) {//区域和，时间轴[xa,xb]线段树下标[ya,yb]的闭区域和
        return -Ask(1, n, ya, yb, rot[xa - 1]) + Ask(1, n, ya, yb, rot[xb]);
    }
}D;
map< int, int > S;
int main() {
    //区间distinct。
    scanf("%d", &n);
    fo(i, 1, n) {
        scanf("%d", &a[i]);
        D.succ(i - 1, i);
        if (S[a[i]])
            D.f(i, S[a[i]], -1); //抵消掉上一个相同值的累加，询问[l,r]使用r处线段树即可
        S[a[i]] = i;
        D.f(i, S[a[i]], 1);
    }
    int q;
    scanf("%d", &q);
    fo(ii, 1, q) {
        int l, r;
        scanf("%d%d", &l, &r);
        printf("%d\n", D.ac(1, r, l, r)); //x轴y轴都是位置。真正位置轴是y，x只是为了在线而可持久化的多个版本线段树。
                                          //假如不在线，就只需要r排序后一颗线段树从头到尾。在线则需要可持久化多个版本。
    }
}

```

KD 树

```
const int N = 1e5 + 10, K = 6;
int n, k, cnt;
```

```cpp

struct Dot {
    int d[K], dim; bool lef; ll dis;
    Dot() {
        memset(d, 0, sizeof d);
        dim = lef = dis = 0;
    }
    bool operator <(Dot b) const {
        return dis < b.dis;
    }
}z[N], ths, use[N];

ll dis(Dot a, Dot b) {
    ll ret = 0;
    for (int i = 0; i<k; i++)
        ret += (a.d[i] - b.d[i]) * (a.d[i] - b.d[i]);
    return ret;
}

namespace KD_Tree {
    int cur, num;
    priority_queue< Dot > q;
    ll nowax;

    bool cmp(Dot a, Dot b) {
        return a.d[cur]  < b.d[cur];
    }
    void Construct_KDTree(int l, int r, int dim) {
        if (l>r) return;
        int m = l + r >> 1;
        cur = dim;
        nth_element(z + l, z + m, z + r + 1, cmp);
        z[m].dim = cur;
        if (l == r) {
            z[m].lef = 1;
            return;
        } else {
            Construct_KDTree(l, m - 1, (dim + 1) % k);
            Construct_KDTree(m + 1, r, (dim + 1) % k);
        }
    }
    void KD_Find(int l, int r, Dot &a) {
        int m = l + r >> 1;
        if (l > r) return;
```

```
        ll dj = dis(z[m], a);
        z[m].dis = dj;
        if (z[m].lef) {
            if ((int)q.size() < num) {
                q.push(z[m]);
                nowax = max(nowax, dj);
            } else if (dj < nowax) {
                q.pop(), q.push(z[m]);
                nowax = q.top().dis;
            }
            return;
        } else {
            ll t = a.d[z[m].dim] - z[m].d[z[m].dim];
            if (t < 0) {
                KD_Find(l, m - 1, a);
                if ((int)q.size() < num) {
                    q.push(z[m]), nowax = max(nowax, dj);
                    KD_Find(m + 1, r, a);
                } else {
                    if (dj < nowax)q.pop(), q.push(z[m]), nowax = q.top().dis;
                    if (t * t < nowax) KD_Find(m + 1, r, a);
                }
            } else {
                KD_Find(m + 1, r, a);
                if ((int)q.size() < num) {
                    q.push(z[m]), nowax = max(nowax, dj);
                    KD_Find(l, m - 1, a);
                } else {
                    if (dj < nowax) q.pop(), q.push(z[m]), nowax = q.top().dis;
                    if (t * t < nowax) KD_Find(l, m - 1, a);
                }
            }
        }
    }
    void Nearest(Dot X, int K) {//注意K的大小写
        num = K; nowax = 0;
        KD_Find(1, n, X);
        cnt = 0;
        while (q.empty() == 0) {
            use[++cnt] = q.top();
            q.pop();
        }
        sort(use + 1, use + 1 + cnt);
    }
```

```
}
int main() {
    // 数据接口:
    // n为点的个数，k为坐标维度
    // z[i].(lef,dis,dim)均应初始化为0
    // z[i].d[0...k-1]存放坐标
    Read(n, k, z[1]~z[n]);
    KD_Tree::Construct_KDTree(1, n, 0);
    //现在找出平面任意点x的m近点
    Read(Dot x  int m);
    KD_Tree::Nearest(x, m);
    //返回数组use，use的长度为cnt。
    //此时use[1],use[2]...use[cnt]为最近，第二近...第cnt近的点
}
```

Splay

```
const int N = 1e5 + 10;
int rt;
struct SplayTree {
    int cnt, tp[N], pa[N], ch[N][2], si[N], sta[N], rev[N];
    //tp[x]=2 while x is the rt of tree
    void Clear() {// 初始化请调用本函数
        rt = 0, si[0] = 0, cnt = 0;
    }
    int Malloc() {// 申请新节点，新节点自己是新树的根
        ++cnt;
        tp[cnt] = 2; ch[cnt][0] = ch[cnt][1] = 0;
        si[cnt] = 1; rev[cnt] = 0;
        return cnt;
    }
    void Update(int x) {
        si[x] = si[ch[x][0]] + si[ch[x][1]] + 1;
    }
    void Pass(int x) {
        if (rev[x]) {
            swap(ch[x][0], ch[x][1]);
            if (ch[x][0]) {
                tp[ch[x][0]] = 0;
                rev[ch[x][0]] ^= 1;
            }
            if (ch[x][1]) {
                tp[ch[x][1]] = 1;
                rev[ch[x][1]] ^= 1;
            }
```

```
            rev[x] = 0;
        }
    }
    void rotate(int x) {
        int t = tp[x], y = pa[x], z = ch[x][1 - t];
        tp[x] = tp[y];
        pa[x] = pa[y];
        if (tp[x] != 2) ch[pa[x]][tp[x]] = x;
        tp[y] = 1 - t;
        pa[y] = x;
        ch[x][1 - t] = y;
        if (z) {
            tp[z] = t; pa[z] = y;
        }
        ch[y][t] = z;
        Update(y);
    }
    void Splay(int x) {
        int st = 0;
        sta[st++] = x;
        for (int i = x; tp[i] != 2; i = pa[i]) sta[st++] = pa[i];
        for (int i = st - 1; i>-1; --i) Pass(sta[i]);
        while (tp[x] != 2) {
            int y = pa[x];
            if (tp[x] == tp[y]) {
                rotate(y);
            } else rotate(x);
            if (tp[x] == 2) {
                break;
            }
            rotate(x);
        }
        Update(x);

    }
    int find(int x, int rank) {
        while (1) {
            Pass(x);
            if (si[ch[x][0]] + 1 == rank) {
                break;
            }
            if (rank <= si[ch[x][0]]) {
                x = ch[x][0];
            } else {
```

```
                rank -= si[ch[x][0]] + 1;
                x = ch[x][1];
            }
        }

        return x;
    }
    void split(int &x, int &y, int a) {
        /*数据接口：x必须某树根。
        *输出：函数将前a个节点放到x树，剩余放到y树内，x，y均为根
        *若a过大，分裂不成功，x不变，y为0*/
        if (si[x] <= a) {
            y = 0;
            return;
        }
        y = find(x, a + 1);
        Splay(y);
        x = ch[y][0];
        tp[x] = 2;
        ch[y][0] = 0;
        Update(y);
    }
    void split3(int &x, int &y, int &z, int a, int b) {
        /*数据接口：x为树根
        * 输出：[1..a-1]->以x为根的树，[a,b]->以y为根的树，[b+1..剩余]->以z为根树
        * 分裂不成功的根返回0*/
        split(x, z, b);
        split(x, y, a - 1);
    }
    void join(int &x, int y) {
        /*数据接口：x,y均为树根
        * 输出x为新树根*/
        if (x == 0) {
            x = y;
            return;
        }
        x = find(x, si[x]);
        Splay(x);
        ch[x][1] = y;
        tp[y] = 1;
        pa[y] = x;
        Update(x);
    }
    void join3(int &x, int y, int z) {
```

```
        /*数据接口：,x,y,z均为树根
        * 输出x为新树根*/
        join(y, z);
        join(x, y);
    }
    int getRank(int x) {
        //查询x在树中排第几
        Splay(x);
        rt = x;
        return si[ch[x][0]] + 1;
    }
    void Reverse(int a, int b) {
        //翻转区间[a,b],注意，树标号[1..cnt]不是从0开始。
        int x, y;
        split3(rt, x, y, a, b);
        rev[x] ^= 1;
        join3(rt, x, y);
    }

}tr;
int main() {
    tr.Clear();
    tr.Malloc();
    rt = 1; //不妨设rt为主树的根
    for (int i = 2; i <= n; i++) {//连接
        tr.Malloc();
        tr.join(rt, i);
    }
    /*
    用例，区间[l,r]向左旋转k位
    eg1{
    int L = rt , M = 0 , R = 0;
    tt ++ ;
    tr.split3( L , M , R , l , r );
    tr.split( M , tmp , k ) ;
    tr.join( tmp , M ) ;
    tr.join3( L , tmp , R ) ;
    rt = L ;
    }
    eg2{
    tr.Reverse( l , l+k-1 ) ;
    tr.Reverse( l+k,r ) ;
    tr.Reverse( l , r ) ;
    }
```

```
    以上用例是良维护的。
    !!!注意，凡是调用splay函数无论是否修改树结构都要调用splay()使均摊时间不超时
    比如find()函数!!!
    */
}
```

卷积

FWT：快速沃尔什变换

```
ll  a[N], b[N];
void FWT(ll *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m) {
            for (int j = 0; j < d; j++) {
                ll x = a[i + j], y = a[i + j + d];
                a[i + j] = x + y; a[i + j + d] = x - y;
                //xor:a[i+j]=x+y,a[i+j+d]=x-y;
                //and:a[i+j]=x+y;
                //or:a[i+j+d]=x+y;
            }
        }
}
void IFWT(ll *a, int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int m = d << 1, i = 0; i < n; i += m) {
            for (int j = 0; j < d; j++) {
                ll x = a[i + j], y = a[i + j + d];
                a[i + j] = x + y >> 1, a[i + j + d] = x - y >> 1;
            }
        }
    }
}
int main() {
    S = (1 << 16) - 1; //设计算数组长度为N=(10001010)_2，则S=(11111111)_2
    memcpy(a, c, sizeof c);
    memcpy(b, c, sizeof c);
    FWT(a, S), FWT(b, S);
    for (int i = 0; i <= S; i++) a[i] *= b[i];
    IFWT(a, S);
    printf("%lld\n", (a[0] - n) / 2);
    for (int i = 1; i <= S; i++) printf("%lld\n", a[i] / 2);
}
```

NTT：快速数论变换

```
const int mo = 998244353;  // 卷积数组长度，更改mo注意原根变化
const int maxn = N * 4;  // 傅里叶变换所需长度
int n, ci; // n在main中应该是a,b较长者的数组长度[0,n-1]（不需要其他处理）。
int a[maxn], b[maxn], t[maxn];
int w[maxn];
```

```
void DFT(int *a, int sig) {
    for (int i = 0; i<n; i++) {
        int p = 0;
        for (int j = 0, tp = i; j<ci; j++, tp /= 2) p = (p << 1) + (tp & 1);
        t[p] = a[i];
    }
    for (int m = 2; m <= n; m *= 2) {
        int hal = m / 2, bei = n / m;
        for (int i = 0; i<hal; i++) {
            ll wi = (sig>0) ? w[i*bei] : w[n - i * bei];
            for (int j = i; j<n; j += m) {
                int u = t[j], v = (wi * t[j + hal]) % mo;
                t[j] = (u + v) % mo;
                t[j + hal] = ((u - v) % mo + mo) % mo;
            }
        }
    }
    for (int i = 0; i<n; i++) a[i] = t[i];
}
void exgcd(ll a, ll b, ll&x, ll&y) {
    if (b) exgcd(b, a % b, y, x), y -= a / b * x;
    else x = 1, y = 0;
}
int niyuan(ll a, ll b) {
    ll x, y;
    exgcd(a, b, x, y);
    return (x % mo + mo) % mo;
}
int found(int x) {
    int ret = 1;
    for (; ret<x; ret *= 2);
    return ret * 2; //maxn是N*4的原因
}
int calc(int x) { return (x == 1) ? 0 : calc(x / 2) + 1; }
int qpow(int x, int y) {
    if (y == 0) return 1;
    ll ret = qpow(x, y / 2);
    ret = (ret * ret) % mo;
    if (y & 1) ret = (ret * x) % mo;
    return int(ret);
}
//n传进应是那个全局变量n，DFT使用了全局变量
void Procedure(int * a, int * b, int &n) {
    n = found(n);
```

```
    ci = calc(n);

    int K = (mo - 1) / n;

    w[0] = 1;
    w[1] = qpow(3, K);//请在此处设置原根（将3改成其他）
    for (int i = 2; i <= n; i++) w[i] = (ll(w[i - 1]) * w[1]) % mo;

    DFT(a, 1); DFT(b, 1);
    for (int i = 0; i<n; i++) a[i] = (ll(a[i]) * b[i]) % mo;
    DFT(a, -1);

    int ni = niyuan(n, mo);
    for (int i = 0; i<n; i++) a[i] = (ll(a[i]) * ni) % mo;
    for (; a[n] == 0; --n);

    //for(int i=0; i<=n; i++) printf("%d ", a[i]); // OUTPUT  卷积结果
}//全局变量n此时已是卷积结果a数组长度，注意a[]定义域是a[0]~a[n]
```

质数	原根	质数	原根	质数	原根
40961	3	**998244353**	3	2748779069441	3
65537	3	**1004535809**	3	6597069766657	5
786433	10	2013265921	31	39582418599937	5
5767169	3	2281701377	3	79164837199873	5
7340033	3	3221225473	5	263882790666241	7
23068673	3	75161927681	3	1231453023109121	3
104857601	3	77309411329	7	1337006139375617	3
167772161	3	206158430209	22	3799912185593857	5
469762049	3	2061584302081	7	4222124650659841	19

NTT 常见质数及其原根

FFT：快速傅里叶变换

```
#include<cmath>
const double Pi = acos(-1.0);
const int N = 5e5 + 10; //开到卷积数组的4倍长度
struct Z {
    double x, y;
    Z(double _x = 0, double _y = 0) { x = _x; y = _y; }
};
typedef Z fft[N];
Z operator +(Z &a, Z &b) { return Z(a.x + b.x, a.y + b.y); }
Z operator -(Z &a, Z &b) { return Z(a.x - b.x, a.y - b.y); }
Z operator *(Z &a, Z &b) { return Z(a.x * b.x - a.y * b.y, a.x * b.y + b.x * a.y); }
fft w, tt;
```

```
void DFT(fft &a, int s, int t, int n) {
    if ((n >> t) == 1) return;
    DFT(a, s, t + 1, n), DFT(a, s + (1 << t), t + 1, n);
    for (int i = 0; i < (n >> (t + 1)); i++) {
        int j = s + (i << t + 1);
        Z W = w[i << t] * a[j + (1 << t)];
        tt[i] = a[j] + W;
        tt[i + (n >> t + 1)] = a[j] - W;
    }
    for (int i = 0; i<(n >> t); i++) a[s + (i << t)] = tt[i];
}
void Multi(fft &a, fft &b, int n) {//a，b为卷积数组，定义域为[0,n-1]
    int k;
    for (k = 1; k<2 * n; k <<= 1);
    n = k;
    for (int i = 0; i<n; i++) w[i].x = cos(2 * i * Pi / n), w[i].y = sin(2 * i * Pi / n);
    DFT(a, 0, 0, n); DFT(b, 0, 0, n);
    for (int i = 0; i<n; i++) w[i].y = -w[i].y, a[i] = a[i] * b[i];
    DFT(a, 0, 0, n);
    for (int i = 0; i<n; i++) a[i].x /= n;
}//a 数组的实数部分为卷积结果数组，定义域为[0,2n-2]
```

字符串

AC 自动机及其 FailTree 括号序

```
const int N = 2e4 + 10; //与模式串字符总数同阶
namespace AC_Automata {
    struct Node {
        int s[26], nx, fa, en; //nx is ths fail father , fa is trie father
    }tr[N];
    int T, q[N];
    void Addstring(const char * s, int n) {//字符串接口：s定义域[1,n]
        int now = 0;
        for (int i = 1; i <= n; i++) {
            if (tr[now].s[s[i] - 'a'] == 0)
                tr[now].s[s[i] - 'a'] = ++T, tr[T].fa = now;
            now = tr[now].s[s[i] - 'a'];
        }
        tr[now].en = 1;//当前字符串终止节点标志
    }
    //------------------Constructing Fail Tree----------------
    struct Edge {
        int y, l;
    }f[N];
    int t, g[N], l[N], r[N], ti;//l[po],r[po]为Fail Tree括号序左右端。
    void Ins(int x, int y) {
        f[++t].y = y, f[t].l = g[x], g[x] = t;
    }
    void Dfs(int po) {
        l[po] = ++ti;
        for (int k = g[po]; k; k = f[k].l) Dfs(f[k].y);
        r[po] = ++ti;
    }
    //---------------If needn't,delete this part--------------
    void Bfs() {
        int l = 0, r = 0;
        for (int i = 0; i<26; i++) if (tr[0].s[i]) q[++r] = tr[0].s[i];
        while (l != r) {
            if (++l == N) l = 1;
            Ins(tr[q[l]].nx, q[l]); // Fail tree linking
            for (int k = 0; k<26; k++) if (tr[q[l]].s[k]) {
                int tmp = tr[q[l]].s[k];
                tr[tmp].nx = tr[tr[q[l]].nx].s[k];
                q[++r] = tmp;
            } else tr[q[l]].s[k] = tr[tr[q[l]].nx].s[k];
```

```
        }
    }
    int go(int po, char ths) {//转移函数
        return tr[po].s[ths - 'a'];
    }
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        char s[101];
        scanf("%s", s + 1);
        AC_Automata::Addstring(s, strlen(s + 1));//Example
    }
    AC_Automata::Bfs(); AC_Automata::Dfs(0);
}

```

SAM：后缀自动机

```
int Delta(int now, int ch) {
    int i = 0, np = ++T;
    D[np].len = D[now].len + 1;
    for (i = now; i != -1 && D[i].s[ch] == 0; i = D[i].fa) D[i].s[ch] = np;
    if (i == -1) return np;
    int q = D[i].s[ch];
    if (D[i].len + 1 == D[q].len) {
        D[np].fa = q;
    } else {
        int nq = ++T;
        D[nq] = D[q];
        D[q].fa = D[np].fa = nq;
        D[nq].len = D[i].len + 1;
        for (; i != -1 && D[i].s[ch] == q; i = D[i].fa) D[i].s[ch] = nq;
    }
    return np;
}
int main() {
    D[0].fa = -1;
    for (i = 1; i <= n; i++) cur = Delta(cur, s[i] - 'a');
}
```

SAM：后缀自动机（压空间）

```
const int N = 1.3e5;//N is SAM vertex size. // N=1e5(6.2MB)
int a, b;
```

```
char s[N];
namespace SAM {
    ll ans;
    int q[N * 2], In[N * 2], cnt, g[N * 2], T, cur, n;
    struct Edge {
        int y, l;
    }f[N * 2];
    struct SAM {
        int  fa, len, ti;
    }D[N * 2];
    struct HASH {
        static const int mo = 2123461;
        //边集大小，能开多大开多大,必须是质数。
        //开小会超时。但存在极个别充分大质数也会超时
        //质数尽量奇怪点。mo=1e5(0.762MB)
        //Primelist:280001,312343,587579,721129,1021129,
        //1245649(10MB),1345649,1400017,2123461,3112237(23MB)
        struct NODE {
            int va, x;
        }d[mo + 5];
        void Assign(int ps, int va) {
            int x = ps % mo;
            while (d[x].va  && d[x].x != ps)if (++x == mo) x = 0;
            d[x].x = ps, d[x].va = va;
        }
        int Ask(int ps) {
            int x = ps % mo;
            while (d[x].va && d[x].x != ps) if (++x == mo) x = 0; ;
            return d[x].va;
        }
    }S;
    int F(int a, int b) {
        return (a * 1ll * N + b);
    }
    void Ins(int x, int y) {
        f[++cnt].y = y, f[cnt].l = g[x], g[x] = cnt; In[y] ++;
    }
    int Delta(int now, int ch) {
        int i = 0, np = ++T;
        int tmp;
        D[np].len = D[now].len + 1;
        D[np].ti++;
        for (i = now; i != -1 && S.Ask(F(ch, i)) == 0; i = D[i].fa)S.Assign(F(ch, i), np);
        if (i == -1) return np;
```

```
        int q = S.Ask(F(ch, i));
        if (D[i].len + 1 == D[q].len) {
            D[np].fa = q;
        } else {
            int nq = ++T;
            D[nq] = D[q];
            fo(k, 0, 25)if (S.Ask(F(k, q))) S.Assign(F(k, nq), S.Ask(F(k, q)));
            D[nq].ti = 0;
            D[q].fa = D[np].fa = nq;
            D[nq].len = D[i].len + 1;
            for (; i != -1 && S.Ask(F(ch, i)) == q; i = D[i].fa) S.Assign(F(ch, i), nq);
        }
        return np;
    }
    void Top() {
        int i, k, l = 0, r = 0;
        for (i = 0; i <= T; i++) if (In[i] == 0) q[++r] = i;
        while (l != r) {
            ++l;
            for (k = g[q[l]]; k; k = f[k].l)
                if (--In[f[k].y] == 0) q[++r] = f[k].y;
        }
        for (i = r; i >= 1; i--) {
            if (D[q[i]].ti >= a && D[q[i]].ti <= b)
                ans += D[q[i]].len - D[D[q[i]].fa].len;
            //ans为出现次数在[a,b]之间不同的子串总数
            D[D[q[i]].fa].ti += D[q[i]].ti;
            /*D[i],设fa_len为其fail父亲的len值,则节点i代表:
            {s[i-D[i].len+1,i-fa_len],
            s[i-D[i].len+2,i-fa_len],
            s[i-D[i].len+3,i-fa_len],
            ...
            s[i-fa_len,i-fa_len]}的字符串集合的共同属性*/
            //ti为这个集合每个元素（字符串）在s中出现了多少次
        }
    }
    void Run(const char * s) {
        n = strlen(s + 1);
        D[0].fa = -1;
        for (int i = 1; i <= n; i++) cur = Delta(cur, s[i] - 'A');
        for (int i = 1; i <= T; i++) Ins(D[i].fa, i);
        Top();
        printf("%lld\n", ans);
        memset(S.d, 0, sizeof S.d);
```

```
        fo(i, 0, T) D[i].len = D[i].fa = D[i].ti = 0;
        fill(g, 0, T);
        fill(In, 0, T);
        cnt = 0; ans = 0; T = 0;
    }
}
int main() {
    while (scanf("%s%d%d", s + 1, &a, &b) == 3)
        SAM::Run(s);
}
```

SA：后缀数组

```
int bin[N], ws[N], wv[N], sa[N], ra[N], yy[N];
char s[N];
bool comp(int * y, int a, int b, int len) {
    return y[a] == y[b] && y[a + len] == y[b + len];
}
void SA(int n, int m) {
    int * x = bin, *y = yy, i, j, p, *t;
    fo(i, 1, n) ws[x[i] = s[i]]++;
    fo(i, 1, m - 1) ws[i] += ws[i - 1];
    fd(i, n, 1) sa[ws[x[i]]--] = i;
    for (p = 1, j = 1; p < n; j = j * 2, m = p) {
        p = 0;
        for (i = n - j + 1; i <= n; i++) y[++p] = i;
        for (i = 1; i <= n; i++) if (sa[i] - j > 0) y[++p] = sa[i] - j;
        for (i = 0; i <= m; i++) ws[i] = 0;
        for (i = 1; i <= n; i++) wv[i] = x[y[i]];
        for (i = 1; i <= n; i++) ws[wv[i]] ++;
        for (i = 1; i <= m; i++) ws[i] += ws[i - 1];
        for (i = n; i; i--) sa[ws[wv[i]] --] = y[i];
        p = 1;
        for (t = y, y = x, x = t, x[sa[1]] = 1, i = 2; i <= n; i++) {
            x[sa[i]] = comp(y, sa[i], sa[i - 1], j) ?
                p : ++p;
        }
    }
}
int h[N];
void GH() {
    int i, j, k;
    fo(i, 1, n) ra[sa[i]] = i;
    k = 0;
    for (i = 1; i <= n; h[ra[i++]] = k) {
```

```
        if (ra[i] == 1) {
            k = 0;
            continue;
        }
        for (k = k ? k - 1 : 0; s[i + k] == s[sa[ra[i] - 1] + k]; k++);
    }
}
int l[22][N];
void Build_RMQ() {
    for (int j = 1; j <= n; j++) l[0][j] = h[j];
    for (int i = 1; i <= 20; i++) {
        for (int j = 1; j <= n; j++) {
            int jk = j + (1 << i - 1);
            l[i][j] = l[i - 1][j];
            if (jk <= n) l[i][j] = min(l[i - 1][j], l[i - 1][jk]);
        }
    }
}
int RMQ(int L, int R) {
    int i = log2(R - L + 1), len = (1 << i);
    return min(l[i][L], l[i][R - len + 1]);
}
int LCP(int fr, int to) {//In SA ps: LCP(fr,fr+1,...,to)
    if (fr == to) return n - sa[fr] + 1;
    return RMQ(fr + 1, to);
}
```

KMP

```
int nx[N];
char s[N], t[N];
int main() {
    scanf("%s%s", s + 1, t + 1);
    int AppearTimes = 0;
    int n = strlen(s + 1), m = strlen(t + 1);
    for (int i = 2, j = 0; i <= m; i++) {
        while (j && t[j + 1] != t[i]) j = nx[j];
        if (t[j + 1] == t[i]) nx[i] = ++j;
    }
    for (int i = 1, j = 0; i <= n; i++) {
        while (j && s[i] != t[j + 1]) j = nx[j];
        if (t[j + 1] == s[i]) ++j;
        if (j == m) {
            AppearTimes++;
            j = nx[j];
```

```
        }
    }
    printf("%d", AppearTimes);
}
```

exKMP

```
//ex[i]为t[ i..ex[i]+i-1 ] == s[ 1..ex[i] ] 即模板串第i位起和母串前缀的最长匹配长度
//nx[i]为s[ i..nx[i]+i-1 ] == s[ 1..nx[i] ] 即母串第i为与母串前置的最长匹配长度
int ex[N], nx[N];
char s[N], t[N];
//len(s)=n,len(t)=m,s[1..n],t[1..m]
void EX_KMP(int n, const char * s, int m, const char * t) {
    int a = 0;
    memset(nx, sizeof nx, 0);
    nx[1] = n;
    for (int i = 2; i <= n; i++) {
        int L = nx[i - a + 1], j = a + nx[a] - 1;
        if (i + L <= j) nx[i] = L; else {
            int h = j - i + 1;
            if (h < 0) h = 0;
            while (i + h <= n && s[1 + h] == s[i + h]) h++;
            nx[i] = h;
            a = i;
        }
    }
    a = 1;
    for (int i = 1; i <= m; i++) {
        int L = nx[i - a + 1], j = a + ex[a] - 1;
        if (i + L <= j) ex[i] = L; else {
            int h = j - i + 1;
            if (h < 0) h = 0;
            while (i + h <= m && t[i + h] == s[h + 1]) ++h;
            ex[i] = h;
            a = i;
        }
    }
}
int main() {
    scanf("%s%s", s + 1, t + 1);
    int n = strlen(s + 1), m = strlen(t + 1);
    EX_KMP(n, s, m, t);
}
```

回文树

```
int  n, m, T;
char s[N];

struct Pali {
    int len, cnt, s[27], nx;
}tr[N];

int la = 0, ps;
int FAIL(int now) {
    while (s[ps - tr[now].len - 1] != s[ps]) now = tr[now].nx;
    return now;
}

int ti[N];
void Add(char c) {
    ++ps;
    int cur = FAIL(la), ths = 0;
    if (tr[cur].s[c - 'a'] == 0) {
        ths = ++T;
        tr[T].nx = tr[FAIL(tr[cur].nx)].s[c - 'a'];
        tr[T].len = tr[cur].len + 2;
        tr[cur].s[c - 'a'] = ths;
    } else ths = tr[cur].s[c - 'a'];
    la = ths;
    tr[la].cnt++;
}

ll ans = 0;
void Calc() {
    for (int i = T; i; i--) {
        tr[tr[i].nx].cnt += tr[i].cnt;
        ans = max((ll)tr[i].cnt * tr[i].len, ans);// s的一个子串 t的“出现值”为 t在 s中的出现次数乘以t的长度。 求出s的所有回文子串中的最大出现值。
    }
}
int main() {
    //freopen( "palindrome.in","r",stdin ),freopen( "palindrome.out","w",stdout );
    scanf("%s", s + 1);
    n = strlen(s + 1);
    T = 1;
    tr[0].len = 0; tr[0].nx = 1;
    tr[1].len = -1;
```

```
    s[0] = -1;
    for (int i = 1; i <= n; i++) Add(s[i]);
    Calc();
    printf("%lld", ans);
}
```

网络流

Dinic

```
const int N = (500 + 500) * 5, M = 500 * 600 * 2;
struct Edge {
    int l, y, z;
}f[M * 2];
int T = 1;
int g[N], D[N], v[N], q[N], s, t;
void Ins(int x, int y, int z) {
    f[++T].y = y, f[T].l = g[x], g[x] = T; f[T].z = z;
    f[++T].y = x, f[T].l = g[y], g[y] = T; f[T].z = 0;
}
int Dinic(int po, int aug, int n) {
    if (po == t) return aug;
    int ret = 0;
    for (int k = g[po]; k; k = f[k].l)if (f[k].z>0 && D[f[k].y] == D[po] + 1) {
        int go = Dinic(f[k].y, min(f[k].z, aug - ret), n);
        f[k].z -= go;
        f[k ^ 1].z += go;
        ret += go;
    }
    if (ret == 0) D[po] = -1;
    return ret;
}

void Dbfs(int st, int n) {
    for (int i = 1; i <= n; i++) v[i] = 0;
    for (int i = 1; i <= n; i++) D[i] = -1;
    int l = 0, r = 0;
    q[++r] = st, v[st] = 1; D[st] = 0;
    while (l != r) {
        ++l;
        for (int k = g[q[l]]; k; k = f[k].l) if (f[k].z>0 && (v[f[k].y] == 0)) {
            v[f[k].y] = 1;
            D[f[k].y] = D[q[l]] + 1;
            q[++r] = f[k].y;
        }
    }
}
int Network(int n) {
    int flow = 0;
    while (1) {
```

```
        Dbfs(s, n);
        if (v[t] == 0) break;
        flow += Dinic(s, 0x7fffffff, n);

    }
    return flow;
}
int main() {
    s = 1, t = 2;
    T = 1;
    Ins(s, t, 1);
    int n = 2;
    int ans = Network(n);
    printf("%d\n", ans); //=1
}
```

SAP

```
int vh[N], D[N], di[N], g[N];
int sap(int po, int aug) {
    int use = 0;
    if (po == t) return aug;
    for (int k = di[po]; k; k = f[k].l) if (f[k].z > 0 && D[f[k].y] == D[po] - 1) {
        di[po] = k;
        int tmp = sap(f[k].y, min(aug - use, f[k].z));
        f[k].z -= tmp;
        f[k ^ 1].z += tmp;
        use += tmp;
        if (D[s] > n || use == aug) return use;
    }
    di[po] = g[po];
    if (--vh[D[po]] == 0) {
        D[s] = n + 1;
        return use;
    }
    ++vh[++D[po]];
    return use;
}
int main() {
    n = t;//网络点数
    for (int i = 1; i <= n; i++) di[i] = g[i];
    vh[0] = n;
    int flow = 0;
    while (D[s] <= n) flow += sap(s, 0x7fffffff);
    printf("%d", flow);
```

```
}
```

最小费用最大流 SPFA

```
//数据接口
//全局变量s，t源点汇点
//全局变量T初始化为1。F数组要求奇边偶边互相反向。
//N网络流最多点数，M网络流边数，flow_inf流量极值，cost_inf费用极值
int s, t, T, g[N];
struct Edge {
    int y, l, z;
    ll fee;
}f[M * 2];

const int flow_inf = 100000000;
const ll cost_inf = 10000000000000ll;

ll dis[N];
void Ins(int x, int y, int z, ll fee) {
    f[++T].y = y, f[T].l = g[x], g[x] = T, f[T].fee = fee, f[T].z = z;
    f[++T].y = x, f[T].l = g[y], g[y] = T, f[T].fee = -fee, f[T].z = 0;
}

bool v[N];
int q[N], pr[N];

ll Spfa(int n) {//传入网络点数
    int l = 0, r = 0;
    q[++r] = s;
    for (int i = 1; i <= n; i++) dis[i] = cost_inf;
    dis[s] = 0;
    v[s] = 1;
    while (l != r) {
        if (++l == N) l = 1;
        for (int k = g[q[l]]; k; k = f[k].l)
        if (dis[f[k].y] > dis[q[l]] + f[k].fee && f[k].z>0) {
            dis[f[k].y] = dis[q[l]] + f[k].fee;
            pr[f[k].y] = k;
            if (v[f[k].y] == 0) {
                v[f[k].y] = 1;
                if (++r == N) r = 1;
                q[r] = f[k].y;
            }
        }
        v[q[l]] = 0;
```

```
    }
    return dis[t];
}
ll MCF(int n) {//传入网络点数
    ll ret = 0;
    while (1) {
        ll tmp = Spfa(n);
        if (tmp == cost_inf) break;
        int aug = 0x7fffffff;
        for (int k = pr[t]; k; k = pr[f[k ^ 1].y]) {
            aug = min(f[k].z, aug);
        }
        ret += tmp * aug;
        for (int k = pr[t]; k; k = pr[f[k ^ 1].y]) {
            f[k].z -= aug, f[k ^ 1].z += aug;
        }
    }
    return ret;
}
```

最小费用最大流 Primary Dual

```
int n, m, s, t;
//-----------------------------------------
#include<queue>
const int N = 2e5 + 10, M = N; // N:vertex,M:edges
const ll inf_cost = 1e9, inf_flow = 1e9;
namespace Primary_Dual {
    ll dis[N];
    bool v[N];
    int n, T = 1, g[N], s, t;
    ll mincost = 0, flow;
    struct Edge {
        int y, l;
        ll z, fee;
    }f[M * 2];
    void Ins(int x, int y, ll z, ll fee) {
        f[++T].y = y, f[T].l = g[x], g[x] = T, f[T].fee = fee; f[T].z = z;
        f[++T].y = x, f[T].l = g[y], g[y] = T, f[T].fee = -fee; f[T].z = 0;

    }
    bool spfa(int s, int t) {
        for (int i = 0; i <= n; i++)dis[i] = inf_cost, v[i] = 0;
        dis[t] = 0;
        v[t] = 1;
```

```
        deque<int>q;
        q.push_back(t);
        while (!q.empty()) {
            int now = q.front();
            q.pop_front();
            for (int k = g[now]; k; k = f[k].l)if (f[k ^ 1].z && dis[f[k].y]>dis[now] - f[k].fee) {
                dis[f[k].y] = dis[now] - f[k].fee;
                if (!v[f[k].y]) {
                    v[f[k].y] = 1;
                    if (!q.empty() && dis[f[k].y]<dis[q.front()])q.push_front(f[k].y);
                    else q.push_back(f[k].y);
                }
            }
            v[now] = 0;
        }
        return dis[s]<inf_cost;
    }
    ll dfs(int po, ll low) {
        if (po == t) {
            v[t] = 1;
            return low;
        }
        ll used = 0, a;
        v[po] = 1;
        for (int k = g[po]; k; k = f[k].l)if (!v[f[k].y] && f[k].z && dis[po] - f[k].fee == dis[f[k].y]) {
            a = dfs(f[k].y, min(f[k].z, low - used));
            if (a) {
                mincost += a * f[k].fee;
                f[k].z -= a; f[k ^ 1].z += a;
                used += a;
            }
            if (used == low)break;
        }
        return used;
    }
    void Run(int _n, int _s, int _t) {// Start mincost flow here, n is vertex number,s:source,t:sink
        n = _n, s = _s, t = _t;
        flow = 0;
        while (spfa(s, t)) {
            v[t] = 1;
            while (v[t]) {
```

```
                for (int i = 1; i <= n; i++) v[i] = 0;
                flow += dfs(s, inf_flow);
            }
        }
    }
    void clear() {
        memset(g, 0, sizeof g);
        T = 1;
        mincost = 0;
    }
}
//-------------------------------------
int main() {
    scanf("%d%d%d%d", &n, &m, &s, &t);
    for (int i = 1; i <= m; i++) {
        int x, y;
        ll z, fee;
        scanf("%d%d%lld%lld", &x, &y, &z, &fee);
        Primary_Dual::Ins(x, y, z, fee);
    }
    Primary_Dual::Run(n, s, t);
    printf("%lld %lld", Primary_Dual::flow, Primary_Dual::mincost);
}
```

树剖分

树链剖分

```
//weigh[N]存树上点初值
int son[N], tid[N], dfn, top[N], si[N], l[22][N], de[N], fa[N];
void Find_Edge(int po) {
    si[po] = 1;
    int ax = 0;
    l[0][po] = fa[po];
    for (int i = 1; l[i - 1][l[i - 1][po]]; i++) l[i][po] = l[i - 1][l[i - 1][po]];
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y) {
        fa[f[k].y] = po;
        de[f[k].y] = de[po] + 1;
        Find_Edge(f[k].y);
        si[po] += si[f[k].y];
        if (si[f[k].y] > ax) {
            ax = f[k].y;
            son[po] = f[k].y;
        }
    }
}
void Mark_Edge(int po, int anc) {
    tid[po] = ++dfn;
    Add(1, n, tid[po], tid[po], weigh[po], 1);
    top[po] = anc;
    if (son[po]) Mark_Edge(son[po], anc);
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y && son[po] != f[k].y) {
        Mark_Edge(f[k].y, f[k].y);
    }
}
int LCA(int x, int y) {
    if (de[x] > de[y]) swap(x, y);
    for (int sub = de[y] - de[x], j = 0; sub; j++, sub >>= 1) if (sub & 1)
        y = l[j][y];
    if (x == y) return x;
    int i = 0;
    for (; i >= 0; i--)
        for (i = 20; i >= 0; i--) if (l[i][x] != l[i][y]) {
            x = l[i][x], y = l[i][y];
            break;
        }
    return fa[x];
}
```

```cpp

void ChainProce(int u, int v, int va) { // 链+=va的procedure
    int lca = LCA(u, v);
    int ti = 2;
    while (ti--) {
        while (top[u] != top[lca]) {
            Add(1, n, tid[top[u]], tid[u], va, 1); // Add( int l , int r , int _l , int _r , int va , int p )
            u = fa[top[u]];
        }
        if (tid[lca] + 1 <= tid[u]) Add(1, n, tid[lca] + 1, tid[u], va, 1); // Add except LCA
        u = v;
    }
    Add(1, n, tid[lca], tid[lca], va, 1); // fill back LCA
}
```

点剖分

```cpp
struct Edge {
    int y, l, z;
}f[N * 2];
int g[N], T;

void Ins(int x, int y, int z) {
    f[++T].y = y, f[T].l = g[x], g[x] = T, f[T].z = z;
    f[++T].y = x, f[T].l = g[y], g[y] = T, f[T].z = z;
}

long long ans;
int si[N], fa[N], Ts, cnt, tt[N];
bool apper[N], v[N];

void Dfs(int po) {
    // ANSWER process bruteforcer
    // BE VERY CAREFULLY , Without v[ f[k].y ] == 0 will go through the whole tree casing TLE or MLE or WA etc
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y && v[f[k].y] == 0) {
    }
}

void dfs(int po) { // calc size
    si[po] = 1;
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y && v[f[k].y] == 0) {
        fa[f[k].y] = po;
```

```
        dfs(f[k].y);
        si[po] += si[f[k].y];
    }
}

int  FindRt(int po) {//Find the heavy root
    int to = -1, ax = 0;
    ax = Ts - si[po];
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y && v[f[k].y] == 0) {
        if (ax < si[f[k].y]) {
            to = f[k].y;
            ax = si[f[k].y];
        }
    }
    if (ax <= Ts / 2) return po; else return FindRt(to);
}

void Dividual(int po) {
    fa[po] = 0;
    dfs(po);
    Ts = si[po]; // 本次处理子树总大小
    po = FindRt(po); //另一个替代方案，对每个点求mx[po]子树最大值，取mx[]最小po。
    fa[po] = 0;
    v[po] = 1; // 本次处理的子树根被从原树剔除
    for (int k = g[po]; k; k = f[k].l) if (v[f[k].y] == 0) {
        fa[f[k].y] = po;
        Dfs(f[k].y);
    }
    for (int i = 1; i <= cnt; i++) apper[tt[i]] = 0, ...;//CLEAR the used array
    cnt = 0;
    for (int k = g[po]; k; k = f[k].l)
        if (v[f[k].y] == 0)  Dividual(f[k].y);
}

int main() {
    Dividual(1);
}

```

轻重链剖分

```
int n;

struct Edge {
    int  y, l;
```

```
}f[N * 2];

int T, a[N], rt, g[N], satis[N];

vector< int > v[N];

int fa[N], si[N];

bool cmp(int a, int b) {
    return si[a] < si[b];
}

int cnt, q[N], F[N];

void preproce(int po) {
    si[po] = 1;
    for (int k = g[po]; k; k = f[k].l) if (f[k].y != fa[po]) {
        fa[f[k].y] = po;
        preproce(f[k].y);
        si[po] += si[f[k].y];
    }
}

void BruteVisit(int po) {
    q[++cnt] = po;
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y) {
        BruteVisit(f[k].y);
    }
}
void Dfs(int po) {
    v[po].clear();
    for (int k = g[po]; k; k = f[k].l) if (fa[po] != f[k].y) {
        v[po].push_back(f[k].y);
    }
    sort(v[po].begin(), v[po].end(), cmp);
    int len = v[po].size();
    for (int i = 0; i<len; i++) {
        Dfs(v[po][i]);
        if (i != len - 1) {//只留下重边子树的公用数组数据(i==len-1时才是重边)
            cnt = 0;
            BruteVisit(v[po][i]);
            for (int j = 1; j <= cnt; j++) {//擦除轻边子树（子树v[po][i]）留在公用数组的所有数据
                                                // q[j] is ths everypoint of v[po][i] subtree
```

```
                                        //satis[ a[ q[j] ] ] -- ;
            }
        }
    }
    //satis[ a[po] ] ++ ; //计当前点的数据入公用数组
    for (int i = 0; i<len - 1; i++) {
        cnt = 0;
        BruteVisit(v[po][i]);
        for (int j = 1; j <= cnt; j++) {//补回所有轻边子树数据
                                    //satis[ a[ q[j] ] ] ++ ;
        }
    }
}

int main() { //freopen("In","r",stdin ) ;
    fa[root] = 0;
    preproce(root);
    Dfs(root);
}
```

动态树 Link-Cut-Tree

```
const int INF = 0x3f3f3f3f;
int n, m;
namespace LCT {
    struct LCT {
        int l, r, id, fa, key;
        bool rev;
    }tr[N + M];
    bool root(int po) {
        LCT f = tr[tr[po].fa];
        return (f.l != po && f.r != po);
    }
    void Change(int po) {
        swap(tr[po].l, tr[po].r);
        tr[po].rev ^= 1;
    }
    void SendDown(int po) {
        if (tr[po].rev) {
            tr[po].rev = 0;
            if (tr[po].l) Change(tr[po].l);
            if (tr[po].r) Change(tr[po].r);
        }
    }
    void Fulset(int p, int key) {
```

```
        tr[p].key = key;
        tr[p].id = p;
        tr[p].l = tr[p].r = tr[p].fa = tr[p].rev = 0;
    }
    int Find(int a) {
        for (; tr[a].fa; a = tr[a].fa);
        return a;
    }
    void Update(int po) {
        int l = tr[po].l, r = tr[po].r;
        int &now = tr[po].id;
        now = po;
        if (l && tr[now].key > tr[tr[l].id].key) now = tr[l].id;
        if (r && tr[now].key > tr[tr[r].id].key) now = tr[r].id;
    }
    void Rotate(int po) {
        int ps = tr[po].fa;
        SendDown(ps), SendDown(po);
        if (tr[ps].l == po) {
            int tmp = tr[po].r;
            tr[po].r = ps;
            tr[po].fa = tr[ps].fa;
            if (root(ps) == 0) {
                LCT *f = &tr[tr[po].fa];
                if (f->r == ps) f->r = po; else f->l = po;
            }
            tr[ps].fa = po;
            tr[ps].l = tmp;
            tr[tmp].fa = ps;
        } else {
            int tmp = tr[po].l;
            tr[po].l = ps;
            tr[po].fa = tr[ps].fa;
            if (root(ps) == 0) {
                LCT *f = &tr[tr[po].fa];
                if (f->r == ps) f->r = po; else f->l = po;
            }
            tr[ps].fa = po;
            tr[ps].r = tmp;
            tr[tmp].fa = ps;
        }
        Update(ps), Update(po);
    }
    void splay(int x) {
```

```
        SendDown(x);
        for (int i = x; root(x) == 0; ) {
            int j = tr[i].fa, k = tr[j].fa;
            SendDown(i);
            if (root(j)) {
                Rotate(i);
                continue;
            }
            SendDown(k);
            SendDown(j);
            if ((tr[k].l == j && tr[j].l == i) || (tr[k].r == j && tr[j].r == i)) {
                Rotate(j);
                Rotate(i);
            } else Rotate(i), Rotate(i);
        }
        Update(x);
    }
    void access(int a) {
        int b = 0;
        for (; a; tr[a].r = b, b = a, a = tr[a].fa) splay(a);
    }
    void Evert(int a) {//Make root
        access(a);
        splay(a);
        Change(a);
    }
    void Link(int a, int b) {
        Evert(a);
        tr[a].fa = b;
    }
    void Cut(int a, int b) {
        Evert(a);
        access(b);
        splay(b);
        tr[b].l = 0;
        tr[a].fa = 0;
    }
    void Chainproce(int &sm, int &id, int u, int v) {
        Evert(u);
        access(v);
        splay(v);
        id = tr[v].id;
        sm = tr[id].key;
    }
```

```
}
struct edge {
    int a, u, v;
}e[M];
bool cmp(edge a, edge b) {
    return a.a<b.a;
}
void MST_Template(int n, int m) {//vertex n , edge m
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].a);
    }
    sort(e + 1, e + 1 + m, cmp);
    for (int i = 1; i <= m; i++) LCT::Fulset(i + n, e[i].a);// initialize , key is edge weigh
    for (int i = 1; i <= n; i++) LCT::Fulset(i, INF); // vertex has no weigh, so INF
    for (int i = 1; i <= m; i++) {
        if (LCT::Find(e[i].u) != LCT::Find(e[i].v)) {
            LCT::Link(e[i].u, i + n);
            LCT::Link(e[i].v, i + n);
        } else {
            int sm, num;
            sm = num = 0;
            sm = 0x7fffffff;
            LCT::Chainproce(sm, num, e[i].u, e[i].v);
            //sm return is min value of chain , num is id of it
            LCT::Cut(e[num - n].u, num);
            LCT::Cut(e[num - n].v, num);
            LCT::Link(e[i].u, i + n);
            LCT::Link(e[i].v, i + n);
        }
    }
}

```

图论

欧拉回路或路径

```
struct Edge {
    int y, l, id, en;
}f[N * 2];

int node[N], edge[N], sta[N], n, m, T, in[N], g[N], cnt;

void Ins(int x, int y, int i) {
    in[x] ++, in[y] ++;
    f[++T].y = y, f[T].l = g[x], g[x] = T; f[T].id = i; f[T].en = 0;//en删边
    f[++T].y = x, f[T].l = g[y], g[y] = T; f[T].id = -i; f[T].en = 0;
}
void dfs(int po, int dep) {
    for (int k = g[po]; k; k = f[k].l) if (f[k].en == 0) {
        f[k].en = 1;
        f[k ^ 1].en = 1;
        sta[dep + 1] = f[k].id;
        dfs(f[k].y, dep + 1);
    }
    edge[++cnt] = sta[dep];
    node[cnt] = po;
}
void Fleury(int n, int start_point) {
    sta[1] = 0; cnt = 0;
    dfs(start_point, 1);
    /* 欧拉点序列: node = { 0, 终点 , n-1 th , n-2 ... , 2 , 起点 }
     * 欧拉边序列: edge = { 0, n-1到终点的边 ,(n-2)to(n-1),..., 0 } ,
     * 因为起点无上一个点edge[cnt]=0*/
    for (int i = cnt; i>1; i--)
        printf("vertex %d->%d through edge %d\n", node[i], node[i - 1], edge[i - 1]);
        //回路有node[cnt]==node[1]
}
void Input() {
    freopen("In", "r", stdin);
    scanf("%d%d", &n, &m);
    T = 1; // 注意反边需要它
    for (int i = 1; i <= m; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        Ins(x, y, i);// 3th ele is id , negative or positive direction
    }
```

```
}
int main() {
    Input();
    /* Euler Path
    该图存在两个奇数点，选任意一个为起点，存在欧拉路径
    ..................................................................
    Euler circle
    所有节点偶数，存在欧拉回路*/
    bool extodd = 0;
    for (int i = 1; i <= n && extodd == 0; i++) if (in[i] & 1) {
        Fleury(n, i);//路径
        extodd = 1;
    }
    if (!extodd) Fleury(n, 1);//回路
}
```

单源最短路 Dijkstra（堆优化）

```
struct Node {
    int id; ll dis; //顺序不要写反
    bool operator<(const Node &t) const { return dis > t.dis; } //大根堆
};
struct Edge { int v; ll w; };

Pqueue<Node, M> Q; //手写的优先队列，注意堆的大小为边数
vector<Edge> G[N];
ll dis[N];
bool vis[N];
void Dijkstra(int s) {
    memset(dis, 0x1f, sizeof dis); // inf ~ max_ll/4;
    memset(vis, 0, sizeof vis);
    Q.clear();
    dis[s] = 0, Q.push({s, 0});
    while (!Q.empty()) {
        auto u = Q.top(); Q.pop();
        if (vis[u.id]++) continue;
        for (auto &e : G[u.id]) {
            auto t = u.dis + e.w;
            if (dis[e.v] > t) dis[e.v] = t, Q.push({e.v, t});
        }
    }
}
```

SPFA（DFS）判断负环

```
struct Edge {
```

```
    int v; double w;
};
int n, m;
vector<Edge> G[N];
bool vis[N];
double dis[N];

bool spfa(int u) {
    vis[u] = 1;
    for (auto &e : G[u]) {
        if (dis[e.v]>dis[u] + e.w) {
            dis[e.v] = dis[u] + e.w;
            if (vis[e.v] || spfa(e.v)) { vis[u] = 0; return 1; }
        }
    }
    vis[u] = 0;
    return 0;
}
int main() {
    scanf("%d%d", &n, &m);
    while (m--) {
        int u, v; double w;
        scanf("%d%d%lf", &u, &v, &w);
        G[u].push_back({v, w});
    }
    fo(i, 1, n) dis[i] = 1e10;
    bool ans = 0;
    fo(i, 1, n) ans = ans || spfa(i);
    puts(ans ? "Yes" : "No");
}
```

SCC 强连通分量

```
//有向图强联通分量，任意x, y∈分量S，则x可走到y，y可走到x。
//正向图跑一边dfs，反向图跑一边dfs。正向图f[k].en == 0，反向图f[k].en == 1。
void dfs1(int po) {
    v[po] = 1;
    for (int k = g[po]; k; k = f[k].l) if (f[k].en == 0 && v[f[k].y] == 0) dfs1(f[k].y);
    p[++ti] = po;
}

void dfs2(int po, int sour) {
    col[po] = sour; //col[po]是po属于的分量的标志点的编号，注意，编号不连续。
    v[po] = 0;
    for (int k = g[po]; k; k = f[k].l) if (f[k].en && v[f[k].y]) dfs2(f[k].y, sour);
```

```
}
void SCC(int n) {
    for (int i = 1; i <= n; i++) if (!v[i]) dfs1(i);
    for (int i = n; i; i--) if (v[p[i]]) dfs2(p[i], p[i]);
    for (int i = 1; i <= n; i++) {
        for (int k = g[i]; k; k = f[k].l) if (f[k].en == 0) {
            if (col[f[k].y] != col[i]) INS(col[i], col[f[k].y]); //构造新图
        }
    }
}
```

二分图匹配

```
// 数据形式描述
// 形式1:
// 编号 :  左集1到n，右集1到m。边集左集到右集有向边
// 形式2
// 编号 :  左集1到n，右集n+1到n+m.边集左集到右集有向边
//形式3 :
//左集1到n，右集n+1到n+m.边集左集到右集无向边
//以上3种形式均可以下代码处理。
struct Edge {
    int y, l;
}f[S * 2]; //边集的两倍（形式2，3）

int T;
int g[N];
int match[N], V[N], p[N];//N大小覆盖编号即可

void GreedyMatching(int n, int &ans) {
    for (int i = 1; i <= n; i++) {
        p[i] = 0;
        for (int k = g[i]; k; k = f[k].l) if (match[f[k].y] == 0) {
            match[f[k].y] = i;
            p[i] = 1;
            ans++;
            break;
        }
    }
}
bool Dfs(int po, int i) {
    for (int k = g[po]; k; k = f[k].l) if (V[f[k].y] != i) {
        V[f[k].y] = i;
        if (match[f[k].y] == 0 || Dfs(match[f[k].y], i)) {
            match[f[k].y] = po;
```

```
            return 1;
        }
    }
    return 0;
}
int Hungary(int n) {//传入左集大小
    int tot = 0;
    memset(V, 0, sizeof V);
    memset(match, 0, sizeof match);
    GreedyMatching(n, tot);
    for (int i = 1; i <= n; i++) {
        if (p[i] == 0) if (Dfs(i, i)) {
            ++tot;
        }
    }
    return tot;
}
```

Tarjan 强连通分量、双连通分量

```
const int N = 1E5;
int dfn[N], low[N], idx, stack[N], tot, cnt, belong[N], size[N];
bool in[N];
//有向图无需判断重复走边，无向图需要判断
void tarjan(int x/*,int fa*/) {
    dfn[x] = low[x] = ++idx;
    in[x] = 1; stack[++tot] = x;
    for (int i = fir[x]; i; i = net[i]) {
        //if(i==(fa^1)) continue;
        int y = to[i];
        if (dfn[y] == 0) {
            tarjan(y/*,i*/);
            low[x] = min(low[x], low[y]);
        } else if (in[y]) {
            low[x] = min(low[x], dfn[y]);
        }
    }
    if (dfn[x] == low[x]) {
        cnt++;
        do {
            belong[stack[tot]] = cnt;
            size[cnt]++;
            in[stack[tot--]] = 0;
        } while (stack[tot + 1] != x);
    }
```

```
}
```

Tarjan 割边以及无向图缩块

```
//数据接口: T初始化为1, E为边数
int n, e, T, g[N];
struct Edge {
    int y, l;
}f[E * 2];
void Ins(int x, int y) {
    f[++T].y = y, f[T].l = g[x], g[x] = T;
    f[++T].y = x, f[T].l = g[y], g[y] = T;
}
namespace Cut_Edge {
    int dfn[N], low[N], idx, sta[N], st, cnt, bel[N], si[N];
    //bel[原图的点]=联通块标号,  si[联通块标号]联通块大小
    bool in[N], v[N];
    void tarjan(int x, int fa) {
        dfn[x] = low[x] = ++idx;
        in[x] = 1;
        sta[++st] = x;
        for (int k = g[x]; k; k = f[k].l) {
            if (k == (fa ^ 1)) continue;
            int y = f[k].y;
            if (dfn[y] == 0) {
                tarjan(y, k);
                low[x] = min(low[x], low[y]);
            } else if (in[y]) {
                low[x] = min(low[x], dfn[y]);
            }

        }
        if (dfn[x] == low[x]) {
            cnt++;
            do {
                bel[sta[st]] = cnt;
                si[cnt]++;
                in[sta[st--]] = 0;
            } while (sta[st + 1] != x);
        }
    }

    int FA[N], ti, l[N], r[N];//FA为联通块树图的父亲标号
                              //l, r为联通块树的左右dfs括号序
    void Dfs(int po) {
```

```
        v[po] = 1;
        for (int k = g[po]; k; k = f[k].l)if (v[f[k].y] == 0) {
            v[f[k].y] = 1;
            int PO = bel[po], Y = bel[f[k].y];//大写PO，Y均为联通块编号
            if (PO != Y) {
                FA[Y] = PO;
                l[Y] = ++ti;
                //...进入新块前处理
                //(无需构新图，数组下标为联通块编号便等价新图DFS)
                //注意要维护的下标大写
            }
            Dfs(f[k].y);
            if (PO != Y) {
                r[Y] = ++ti;
                //...回溯上一个块处理
            }
        }
    }
    void Run() {//执行本函数调用双连通分量过程
        for (int i = 1; i <= n; i++) if (!dfn[i]) tarjan(i, 0);
        for (int i = 1; i <= n; i++) if (!v[i]) {
            FA[bel[i]] = 0;
            l[bel[i]] = ++ti;
            Dfs(i);
            r[bel[i]] = ++ti;
        }
    }
    bool Query(int x, int y, int g1, int g2) {//询问从x到y，断开(g1，g2)这条边，是否能走。
        if (bel[g1] == bel[g2]) return 1;
        x = bel[x], y = bel[y], g1 = bel[g1], g2 = bel[g2];
        if (FA[g1] == g2) swap(g1, g2);
        bool X = l[g2] <= l[x] && r[x] <= r[g2];
        bool Y = l[g2] <= l[y] && r[y] <= r[g2];
        return X == Y;
    }
}

```

Tarjan 割点及其控制范围

```
//数据接口: T初始化为1, E为边数
int n, e, T, g[N];
struct Edge {
    int y, l;
}f[E * 2];
```

```
void Ins(int x, int y) {
    f[++T].y = y, f[T].l = g[x], g[x] = T;
    f[++T].y = x, f[T].l = g[y], g[y] = T;
}
namespace Cut_Node {
    int dfn[N], low[N], idx, en[N];
    //en[原图的点x]=x子树的dfs序截至位置（本位置在子树内）
    map<int, int> dom[N];
    //dom[原图的割点][子树dfs序开始位置]=子树dfs截至位置
    //dom[x]是割点x的控制范围，里面的pair相当于存了若干个区间，每个区间都是某个子树的控制范围
    //在[dfn[x],en[x]]中不属于任意一个dom[x]区间的点，它存在返祖边，认为它处于上子树的控制范围
    //子树间（包括上子树）跨越必须经过割点x。
    bool cut[N];
    void tarjan(int x, int fa) {
        dfn[x] = low[x] = ++idx;
        int ch = 0;
        for (int k = g[x]; k; k = f[k].l) if ((k ^ 1) != fa) {
            int y = f[k].y;
            if (!dfn[y]) {
                tarjan(y, k);
                low[x] = min(low[x], low[y]);
                ch++;
                if ((fa == 0 && ch >= 2) || (fa && dfn[x] <= low[y])) cut[x] = 1;
                if (dfn[x] <= low[y]) dom[x][dfn[y]] = en[y];
                //dom[x]的y子树的控制范围确定

            } else low[x] = min(low[x], dfn[y]);
        }
        en[x] = idx;//x的截至位置确定
    }
    void Run() {//执行本过程求割点以及其控制范围
        for (int i = 1; i <= n; i++)if (!dfn[i])
            tarjan(i, 0);
    }
    bool Query(int x, int y, int c) {//从x走到y，断开c，能否走到
        if (cut[c] == 0) return 1;
        for (int step = 1; step <= 2; step++, swap(x, y)) {
            map<int, int>::iterator it = dom[c].upper_bound(dfn[x]);
            if (it == dom[c].begin()) continue;
            --it;
            int L = it->first, R = it->second;
            //c的某个可能控制x的子树的开始截至位置的确定
            bool X = (L <= dfn[x] && dfn[x] <= R);
            bool Y = (L <= dfn[y] && dfn[y] <= R);
```

```
            if (X != Y) return 0; //子树跨越

        }
        return 1;
    }
}

```

数论

快速幂和快速乘（非递归）

```
//当x*x>max_ll时，需要把x改成__int128，或者把同余乘法改成qmul
ll qpow(ll x, ll y, ll mod) { //y<0时死循环
    ll ret = 1;
    for (; y; y >>= 1, x = x * x % mod) if (y & 1) ret = ret * x % mod;
    return ret; //不保证返回值为正
}
ll qmul(ll x, ll y, ll mod) {
    if (y < 0) x = -x, y = -y; //避免y<0时死循环
    ll ret = 0;
    for (; y; y >>= 1, x = x * 2 % mod) if (y & 1) ret = (ret + x) % mod;
    return ret; //不保证返回值为正
}
```

欧几里得相关算法

GCD（非递归）

```
//g++自带__gcd()
ll gcd(ll a, ll b) {
    while (b) { ll t = a % b; a = b; b = t; }
    return a;
}
```

exGCD

```
//求整数x和y，ax+by=d，且|x|+|y|最小
void exgcd(ll a, ll b, ll&x, ll&y, ll&d) {
    if (b) exgcd(b, a % b, y, x, d), y -= a / b * x;
    else d = a, x = 1, y = 0;
}
```

直线下方的整点数

```
// 求f[a,b,c,n]:=Sum[Floor[(a*i+b)/c],{i,0,n}]
ll lkgcd_f(ll a, ll b, ll c, ll n) {
    ll ret = a / c * n*(n + 1) / 2 + b / c * (n + 1);
    a %= c, b %= c;
    ll m = (a*n + b) / c;
    return ret += a ? n * m - lkgcd_f(c, c - b - 1, a, m - 1) : b / c * (n + 1);
}
```

两分数间的分母最小的分数

平面向量线性组合的最小模长

乘法逆元（基于 exGCD）

```
ll inv(ll a, ll mod) {
```

```
    ll x, y, d;
    exgcd(a, mod, x, y, d); //若d!=1，则无逆元
    return (x % mod + mod) % mod;
}
```

筛法

线性筛

```
/* isprm[i]  i是质数              Is Prime
 * prm[i]    第i+1个质数          Prime
 * lpf[i]    最小质因子           Least Prime Factor
 * phi[i]    欧拉Phi函数
 * mu[i]     莫比乌斯Mu函数
 * elp[i]    最小质因子指数       Exponent of Least Prime factor
 * nof[i]    因子个数             Number Of Factors
 * sos[i]    最小质因子等比数列和 Sum Of least prime geometric Series
 * sof[i]    因子之和             Sum Of Factors
 */

bool isprm[N];
int prm[N], lpf[N], phi[N], mu[N], elp[N], nof[N], sof[N], sos[N];
int Sieve() {
    int cnt = 0;
    memset(isprm, 1, sizeof(isprm));
    isprm[0] = isprm[1] = 0;
    phi[1] = 1;
    mu[1] = 1;
    nof[1] = 1;
    sos[1] = 1;
    sof[1] = 1;
    for (int i = 2; i < N; i++) {
        if (isprm[i]) {
            prm[cnt++] = i;
            lpf[i] = i;
            phi[i] = i - 1;
            mu[i] = -1;
            elp[i] = 1;
            nof[i] = 2;
            sos[i] = i + 1;
            sof[i] = i + 1;
        }
        for (int j = 0; j < cnt; j++) {
            int u = i * prm[j];
            if (u >= N) break;
            bool v = i % prm[j];
```

```
            isprm[u] = 0;
            lpf[u] = prm[j];
            phi[u] = v ? phi[i] * (prm[j] - 1) : phi[i] * prm[j];
            mu[u] = v ? -mu[i] : 0;
            elp[u] = v ? 1 : elp[i] + 1;
            nof[u] = v ? nof[i] * 2 : nof[i] / (elp[i] + 1) * (elp[i] + 2);
            sos[u] = v ? prm[j] + 1 : sos[i] * prm[j] + 1;
            sof[u] = v ? (prm[j] + 1) * sof[i] : sof[i] / sos[i] * sos[u];
            if (!v) break;
        }
    }
    return cnt; //返回质数表大小
}
```

杜教筛（莫比乌斯函数前缀和）

```
unordered_map<ll, ll> mu_S(2 * N); //预开空间至少是线筛大小的两倍，碰撞概率与空间大小成反比
ll Mu_S(ll n) {
    if (n < N || mu_S.count(n)) return mu_S[n];
    ll ret = 1, ni;
    //phi:  ret = (n + 1) * n / 2;                //注意乘法溢出
    //xphi: ret = (2 * n + 1) * (n + 1) * n / 6;
    fo(i, 2ll, n) {
        ni = n / (n / i);
        ret -= (ni - i + 1) * Mu_S(n / i);
        //phi:  ret -= (ni - i + 1) * Phi_S(n / i);
        //xphi: ret -= (ni - i + 1) * (i + ni) / 2 * xPhi_S(n / i);
        i = ni;
    }
    return mu_S[n] = ret; //一定要记忆化
}
int main() {
    Sieve(); //线筛mu[1..N-1]，N越大则单次询问越快
    mu_S[0] = 0;
    fo(i, 1, N - 1) mu_S[i] = mu_S[i - 1] + mu[i];
}
```

线性递推逆元、阶乘及其逆元

```
int inv[N], fac[N], invfac[N];
//当mod>max_int时，需要把ll改成__int128，或者把同余乘法改成qmul
void get_inv(ll mod) {
    fac[0] = invfac[0] = inv[1] = 1;
    fo(i, 2, N - 1) inv[i] = ll(mod - (mod / i)) * inv[mod % i] % mod;
    fo(i, 1, N - 1) {
        fac[i] = ll(fac[i - 1]) * i % mod;
        invfac[i] = ll(invfac[i - 1]) * inv[i] % mod;
```

```
    }
}
```

求原根

离散对数

BSGS（要求模数为素数）

```
//求 a^x == b (% mod)，mod为素数
ll BSGS(ll a, ll b, ll mod) {
    a %= mod; b %= mod;
    map<ll, ll> h; //用自己的hash表代替map可防TLE
    ll m = ceil(sqrt(mod)), x, y, d, t = 1, v = 1;
    for (ll i = 0; i < m; i++) {
        if (h.count(t)) h[t] = min(h[t], i);
        else h[t] = i;
        t = (t * a) % mod;
    }
    for (ll i = 0; i < m; i++) {
        exgcd(v, mod, x, y, d);
        x = (x * b / d % mod + mod) % mod;
        if (h.count(x)) return i * m + h[x];
        v = v * t % mod;
    }
    return -1;
}
```

exBSGS

```
//求 a^x == b (% mod)，mod可为合数
ll exBSGS(ll a, ll b, ll mod) {
    a %= mod; b %= mod;
    ll ret = 1;
    for (ll i = 0; i <= 50; ++i) {
        if (ret == b) return i;
        ret = ret * a % mod;
    }//枚举比较小的i

    ll x, y, d, v = 1, cnt = 0;
    while ((d = gcd(a, mod)) != 1) {
        if (b % d) return -1;
        b /= d, mod /= d;
        v = v * (a / d) % mod;
        ++cnt;
    }//约分直到gcd(a, mod) == 1

    map<ll, ll> h; //用自己的hash表代替map可防TLE
```

```
    ll m = ceil(sqrt(mod)), t = 1;
    for (ll i = 0; i < m; ++i) {
        if (h.count(t)) h[t] = min(h[t], i);
        else h[t] = i;
        t = t * a % mod;
    }
    for (ll i = 0; i < m; ++i) {
        exgcd(v, mod, x, y, d);
        x = (x* (b / d) % mod + mod) % mod;
        if (h.count(x)) return i * m + h[x] + cnt;
        v = v * t % mod;
    }
    return -1;
}
```

同余方程组

CRT（要求模数两两互质）

```
//求{x==a[i] (% m[i]): i=1..n}，且m[i]两两互质
ll CRT(ll a[], ll m[], int n) {
    ll M = 1, ret = 0;
    for (int i = 0; i < n; i++) M *= m[i];
    for (int i = 0; i < n; i++) {
        ll Mi = M / m[i];
        ret = (ret + Mi * inv(Mi, m[i]) % M * a[i]) % M;
    }
    return ret < 0 ? ret + M : ret;
}
```

exCRT

```
bool CRTmerge(ll a1, ll m1, ll a2, ll m2, ll &a3, ll &m3) {
    ll d = gcd(m1, m2), c = a2 - a1;
    if (c % d) return 0;
    c = (c % m2 + m2) % m2;
    m1 /= d;
    m2 /= d;
    a3 = c / d * inv(m1, m2) % m2 * m1 * d + a1;
    m3 = m1 * m2 * d;
    return 1;
}
ll exCRT(ll a[], ll m[], int n) {
    ll a1 = a[0], m1 = m[0], a3, m3;
    for (int i = 1; i < n; i++) {
        if (!CRTmerge(a1, m1, a[i], m[i], a3, m3)) return -1;
        a1 = a3, m1 = m3;
    }
```

```
    return (a1 % m1 + m1) % m1;
}
```

大整数分解

```
void Factor(ll n, multiset<ll> &plist) {
    if (n <= 1) return;
    if (MR(n)) plist.insert(n);
    else {
        ll d = PR(n);
        Factor(d, plist), Factor(n / d, plist);
    }
}
```

Miller Rabin

```
bool MR(ll x) {
    if (x <= 1) return 0;
    //若max_x<max_ll，可以使用更短的素数表
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (x % a == 0) return x == a;
        ll s = x - 1, r = __builtin_ctzll(s), j;
        __int128 k = qpow(a, s >> r, x); //不支持__int128时，把同余乘法改成qmul
        if (k == 1) continue;
        for (j = 0; j < r; j++, k = k * k % x) if (k == s) break;
        if (j == r) return 0;
    }
    return 1;
}
```

判定范围	素数表	判定范围	素数表
2.046E+03	2	4.685E+12	2,3,7,61,24251
9.080E+06	31,73	3.416E+14	2,3,5,7,11,13,17
4.759E+09	**2,7,61**	3.825E+18	2,3,5,7,11,13,17,19,23
1.122E+12	2,13,23,1662803	3.186E+23	2,3,5,7,11,13,17,19,23,29,31,37

Miller Rabin 测试素数底数表

Pollard's rho

```
ll PR(ll n) { //若n不是合数，则死循环
    for (ll c = 1;; c = rand() % n) {
        //不支持__int128时，把同余乘法改成qmul
        auto f = [&](__int128 x) { return (x * x + c) % n; };
        ll d = 1;
        for (ll x = rand() % n, y = f(x); d == 1; x = f(x), y = f(f(y)))
            d = gcd(abs(x - y), n);
        if (d != n) return d;
    }
}
```

组合数学

整数分拆数

无限制的整数分拆数

```
// 预处理时间复杂度O(N*SqrtN)
// five: 广义五边形数 {1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70, 77, 92,...}
// f:    整数分拆数   {1, 1, 2, 3, 5,  7,  11, 15, 22, 30, 42, 56, 77, 101,135,...}
ll five[sqrtN * 2], f[N];
ll neg(ll x) { return x & 1 ? -1 : 1; }
void intpart_init() {
    fo(i, 0, sqrtN - 1) {
        five[2 * i] = i * (3 * i + 5) / 2 + 1;
        five[2 * i + 1] = five[2 * i] + i + 1;
    }
    memset(f, 0, sizeof f);
    f[0] = 1;
    fo(i, 1, N - 1) {
        for (int j = 0; five[j] <= i; j++) {
            f[i] += neg(j / 2) * f[i - five[j]];
            //f[i] %= mod; //若2*sqrtN*mod>max_ll，解除这句注释
        }
        f[i] = (f[i] % mod + mod) % mod;
    }
}
```

要求分拆的每个数出现的次数严格小于 k

```
//询问时间复杂度O(SqrtN)
ll intpart(ll n, ll k) {
    ll ret = f[n];
    for (int j = 0; five[j] * k <= n; j++) {
        ret -= neg(j / 2) * f[n - five[j] * k];
        //ret %= mod; //若2*sqrtN*mod>max_ll，解除这句注释
    }
    return (ret % mod + mod) % mod;
}
```

计算几何

```
// T是标量，在精度要求极高的情况下自定义分数类，但T是分数时有些运算不支持(易溢出)
#define T double //记得 #undef T
```

点（向量）类

```
const T eps = 1e-11;
int sgn(const T &t) { return t<-eps ? -1 : t>eps; }
struct Point {
    T x, y;
    Point operator+(const Point &t) const { return {x + t.x, y + t.y}; }
    Point operator-(const Point &t) const { return {x - t.x, y - t.y}; }
    Point operator*(const T &t) const { return {x * t, y * t}; }
    Point operator/(const T &t) const { return {x / t, y / t}; } //T是整型时避免除法取整
    bool operator==(const Point &t) const { return !sgn(x - t.x) && !sgn(y - t.y); }
    double len() const { return hypot(x, y); } //模长
    double arg() const { return atan2(y, x); } //辐角
    Point rotate(double a) const { //逆时针旋转角a
        return {x * cos(a) - y * sin(a), x * sin(a) + y * cos(a)};
    }
    Point unit() const { return *this / len(); } //长度单位化，整型错误
    Point normal() const { return Point({-y, x}).unit(); } //单位法向量(逆时针)，整型错误
};
T dot(const Point &u, const Point &v) { return u.x * v.x + u.y * v.y; }//点积，整型易溢出
T det(const Point &u, const Point &v) { return u.x * v.y - u.y * v.x; }//叉积，整型易溢出
double dist(const Point &u, const Point &v) { return (u - v).len(); } //距离
double angle(const Point &u, const Point &v) { //从向量u逆时针转到v的旋转角
    return atan2(det(u, v), dot(u, v)); //(-pi..pi]
}
using Vector = Point;
```

扫描线

```
const int N = 1e5 + 10;
struct Event {
    int x, a, b, dlt;
}z[N * 4];
int tot; int sum[N * 4], tag[N * 4];
void Add(int x, int y, int ex, int ey) {//The left below cood and the right upper cood , x < ex , y < ey
    z[++tot].x = x, z[tot].a = y, z[tot].b = ey; z[tot].dlt = 1;
    z[++tot].x = ex + 1, z[tot].a = y, z[tot].b = ey, z[tot].dlt = -1;
}
bool cmp(Event a, Event b) {
```

```
    return a.x < b.x;
}
void Update(int po, int l, int r) {
    if (tag[po] == 1) {
        sum[po] = r - l + 1; // 如果要对y坐标加权可以在这里改成前缀和相减形式。如果要对X坐标加权直接在proce处乘。
        return;
    }
    if (l != r) sum[po] = sum[po * 2] + sum[po * 2 + 1]; else sum[po] = 0;
}
void Change(int l, int r, int _l, int _r, int po, int dlt) {
    if (_l>_r) return;
    int m = l + r >> 1;
    if (_l == l && _r == r) {
        tag[po] += dlt;
        Update(po, l, r);
        return;
    }
    if (_r <= m) Change(l, m, _l, _r, po * 2, dlt);
    else if (_l>m) Change(m + 1, r, _l, _r, po * 2 + 1, dlt);
    else Change(l, m, _l, m, po * 2, dlt), Change(m + 1, r, m + 1, _r, po * 2 + 1, dlt);
    Update(po, l, r);
}
int proce(int X, int Y) { // X 是x坐标范围，Y是y范围
    int ret = 0;
    Add(1, 1, 3, 3);
    Add(2, 2, 4, 4);
    /* .xxx
     * xxxx
     * xxxx
     * xxx.
     */
    //返回的就是(x1,y1),(x2,y2)矩形的“面积”，一个格点的面积理解成其左下角方块面积
    //也可以说是算上边界，矩形[(x1,y1),(x2,y2)]有多少个格点
    z[++tot].x = X + 1;
    sort(z + 1, z + 1 + tot, cmp);
    int la = 0;
    int now = 0;
    for (int i = 1; i <= tot; i++) {
        if (z[i].x != la) {
            ret += (z[i].x - la) * 1ll * now;
            la = z[i].x;
        }
        if (z[i].x <= X) Change(1, Y, z[i].a, z[i].b, 1, z[i].dlt);
```

```
        now = sum[1];
    }
    return ret;
}
int main() {
    printf("%d", proce(4, 4));
}
```

K 圆覆盖区域面积并

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>
#define mp make_pair
using namespace std;
double eps = 1e-8;
const int MAXN = 500005;
const double pi = acos(-1.0);
struct vec {
    double x, y;
    vec(double _x = 0, double _y = 0) {
        x = _x;
        y = _y;
    }
    vec operator -(vec v) {
        return vec(x - v.x, y - v.y);
    }
    double len() {
        return sqrt(x * x + y * y);
    }
};
inline double cross(vec a, vec b) {
    return a.x * b.y - a.y * b.x;
}
inline double dist(vec a, vec b) {
    return (a - b).len();
}
struct circle {
    vec c;
    double r;
    circle() { c = vec(0, 0); r = 0; }
    circle(vec _c, double _r) {
        c = _c;
        r = _r;
```

```
    }
    vec Oncircle(double angle) {
        return vec(c.x + r * cos(angle), c.y + r * sin(angle));
    }
};
namespace K_Cir_Area {
    int n, m;
    inline int sgn(double x) {
        if (x < -eps) return -1;
        if (x > eps) return 1;
        return 0;
    }
    double angle(double a, double b, double c) { //余弦定理
        return acos((a * a + b * b - c * c) / (2 * a * b));
    }
    bool inter(circle a, circle b, double &a1, double &a2) {
        double d = dist(a.c, b.c);
        if (sgn(d - (a.r + b.r)) > 0) return false;
        if (!sgn(d) || sgn(d - fabs(a.r - b.r)) < 0) return false;
        double alpha = atan2(b.c.y - a.c.y, b.c.x - a.c.x), beta = angle(a.r, d, b.r);
        a1 = alpha - beta;
        a2 = alpha + beta;
        if (a1 < -pi) a1 += 2 * pi; if (a1 > pi) a1 -= 2 * pi;
        if (a2 < -pi) a2 += 2 * pi; if (a2 > pi) a2 -= 2 * pi;
        return true;
    }

    void unionarea(circle c[], int n, double tot[]) { //计算至少被i个圆覆盖的区域面积
        static pair<double, int>a[MAXN * 2];
        for (int i = 1; i <= n; i++) tot[i] = 0;
        for (int i = 0; i < n; i++) {
            int m = 0, k = 0;
            for (int j = 0; j < n; j++) {
                if (i == j) continue;
                double a1, a2;
                if (sgn(c[i].c.x - c[j].c.x) == 0
                    && sgn(c[i].c.y - c[j].c.y) == 0
                    && sgn(c[i].r - c[j].r) == 0 && i < j) continue;

                if (sgn(dist(c[i].c, c[j].c) + c[i].r - c[j].r) <= 0) {
                    ++k;
                    continue;
                }
                if (!inter(c[i], c[j], a1, a2)) continue;
```

```
                a[m++] = mp(a1, 1);
                a[m++] = mp(a2, -1);
                if (a1 > a2) ++k;
            }
            sort(a, a + m);
            double a1 = a[m - 1].first - 2 * pi, a2, rad;
            for (int j = 0; j < m; j++) {
                a2 = a[j].first, rad = a2 - a1;
                tot[k] += 0.5 * c[i].r * c[i].r * (rad - sin(rad));
                tot[k] += 0.5 * cross(c[i].Oncircle(a1), c[i].Oncircle(a2));
                k += a[j].second;
                a1 = a2;
            }
            if (!m) tot[k] += pi * c[i].r * c[i].r;
        }
    }
}
circle c[MAXN];
double tot[MAXN];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        double x, y, r;
        scanf("%lf%lf%lf", &x, &y, &r);
        c[i] = circle(vec(x, y), r);
    }
    K_Cir_Area::unionarea(c, n, tot);
    for (int i = 1; i <= n; ++i) {
        printf("[%d] = %.3lf\n", i, tot[i - 1] - tot[i]);//tot[i-1]-tot[i]为恰好被i个圆覆盖区域的面积并
    }
}
/*eg
Input:
3
0 0 1
1 0 1
1 1 1
Output:
[1] = 4.699
[2] = 1.699
[3] = 0.443
```

```
#define mp make_pair
double eps = 1e-8;
const int MAXN = 500005;
const double pi = acos(-1.0);
struct vec {
    double x, y;
    vec(double _x = 0, double _y = 0) {
        x = _x;
        y = _y;
    }
    vec operator -(vec v) {
        return vec(x - v.x, y - v.y);
    }
    double len() {
        return sqrt(x * x + y * y);
    }
};
inline double cross(vec a, vec b) {
    return a.x * b.y - a.y * b.x;
}
inline double dist(vec a, vec b) {
    return (a - b).len();
}
struct circle {
    vec c;
    double r;
    circle() { c = vec(0, 0); r = 0; }
    circle(vec _c, double _r) {
        c = _c;
        r = _r;
    }
    vec Oncircle(double angle) {
        return vec(c.x + r * cos(angle), c.y + r * sin(angle));
    }
};
namespace K_Cir_Area {
    int n, m;
    inline int sgn(double x) {
        if (x < -eps) return -1;
        if (x > eps) return 1;
        return 0;
    }
    double angle(double a, double b, double c) { //余弦定理
        return acos((a * a + b * b - c * c) / (2 * a * b));
```

```
}
bool inter(circle a, circle b, double &a1, double &a2) {
    double d = dist(a.c, b.c);
    if (sgn(d - (a.r + b.r)) > 0) return false;
    if (!sgn(d) || sgn(d - fabs(a.r - b.r)) < 0) return false;
    double alpha = atan2(b.c.y - a.c.y, b.c.x - a.c.x), beta = angle(a.r, d, b.r);
    a1 = alpha - beta;
    a2 = alpha + beta;
    if (a1 < -pi) a1 += 2 * pi; if (a1 > pi) a1 -= 2 * pi;
    if (a2 < -pi) a2 += 2 * pi; if (a2 > pi) a2 -= 2 * pi;
    return true;
}

void unionarea(circle c[], int n, double tot[]) { //计算至少被i个圆覆盖的区域面积
    static pair<double, int>a[MAXN * 2];
    for (int i = 1; i <= n; i++) tot[i] = 0;
    for (int i = 0; i < n; i++) {
        int m = 0, k = 0;
        for (int j = 0; j < n; j++) {
            if (i == j) continue;
            double a1, a2;
            if (sgn(c[i].c.x - c[j].c.x) == 0
                && sgn(c[i].c.y - c[j].c.y) == 0
                && sgn(c[i].r - c[j].r) == 0 && i < j) continue;

            if (sgn(dist(c[i].c, c[j].c) + c[i].r - c[j].r) <= 0) {
                ++k;
                continue;
            }
            if (!inter(c[i], c[j], a1, a2)) continue;
            a[m++] = mp(a1, 1);
            a[m++] = mp(a2, -1);
            if (a1 > a2) ++k;
        }
        sort(a, a + m);
        double a1 = a[m - 1].first - 2 * pi, a2, rad;
        for (int j = 0; j < m; j++) {
            a2 = a[j].first, rad = a2 - a1;
            tot[k] += 0.5 * c[i].r * c[i].r * (rad - sin(rad));
            tot[k] += 0.5 * cross(c[i].Oncircle(a1), c[i].Oncircle(a2));
            k += a[j].second;
            a1 = a2;
        }
        if (!m) tot[k] += pi * c[i].r * c[i].r;
```

```
        }
    }
}
circle c[MAXN];
double tot[MAXN];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        double x, y, r;
        scanf("%lf%lf%lf", &x, &y, &r);
        c[i] = circle(vec(x, y), r);
    }
    K_Cir_Area::unionarea(c, n, tot);
    for (int i = 1; i <= n; ++i) {
        printf("[%d] = %.3lf\n", i, tot[i - 1] - tot[i]);//tot[i-1]-tot[i]为恰好被i个圆覆盖区域的面积并
    }
}
/*eg
Input:
3
0 0 1
1 0 1
1 1 1
Output:
[1] = 4.699
[2] = 1.699
[3] = 0.443

```

二维凸包（上下凸壳法）

```
const int N = 1e5 + 10;
const double eps = 1e-7;
struct Dot {
    double x, y;
    Dot(double _x = 0, double _y = 0) { x = _x, y = _y; }
}z[N];
Dot operator +(Dot a, Dot b) { return Dot(a.x + b.x, a.y + b.y); }
Dot operator -(Dot a, Dot b) { return Dot(a.x - b.x, a.y - b.y); }
bool cmp(Dot a, Dot b) { return (b.x - a.x>eps) || (fabs(a.x - b.x) < eps &&  b.y - a.y > eps); }
double Cro(Dot a, Dot b) { return a.x * b.y - a.y * b.x; }
```

```
namespace Convex {
    int U, D, p[N], q[N], n;
    void Upconvex() {
        int r = 0;
        for (int i = 1; i <= n; i++) {
            while (r >= 2 && Cro(z[i] - z[q[r - 1]], z[q[r]] - z[q[r - 1]]) <= eps) r--;
            q[++r] = i;
        }
        U = r;
    }
    void Downconvex() {
        int r = 0;
        for (int i = 1; i <= n; i++) {
            while (r >= 2 && Cro(z[p[r]] - z[p[r - 1]], z[i] - z[p[r - 1]]) <= eps) r--;
            p[++r] = i;
        }
        D = r;
    }
    void Convex(int num) {
        n = num;
        sort(z + 1, z + 1 + n, cmp);
        Upconvex();
        Downconvex();
        for (int j = D - 1; j; j--) q[++U] = p[j];
        // z[q[1]],z[q[2]],...,z[q[U-1]],(z[q[U]]==z[q[1]]) is the convex hull clockwise
        for (int i = 1; i <= U; i++) printf("%lf %lf\n", z[q[i]].x, z[q[i]].y);
    }
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%lf%lf", &z[i].x, &z[i].y);
    Convex::Convex(n);
}
```

三维凸包相关

```
/*
HDU 4273 Rescue
给一个三维凸包，求重心到表面的最短距离
模板题：三维凸包+多边形重心+点面距离
*/
const int MAXN = 550;
const double eps = 1e-8;

```

```
struct Point {
    double x, y, z;
    Point() {}

    Point(double xx, double yy, double zz):x(xx), y(yy), z(zz) {}
    Point operator -(const Point p1) {
        return Point(x - p1.x, y - p1.y, z - p1.z);
    }
        Point operator +(const Point p1) {
        return Point(x + p1.x, y + p1.y, z + p1.z);
    }
    Point operator *(const Point p) {//叉乘
        return Point(y*p.z - z * p.y, z*p.x - x * p.z, x*p.y - y * p.x);
    }
    Point operator *(double d) {
        return Point(x*d, y*d, z*d);
    }
    Point operator / (double d) {
        return Point(x / d, y / d, z / d);
    }
    double  operator ^(Point p) {//点乘
        return (x*p.x + y * p.y + z * p.z);
    }
};

struct CH3D {
    struct face {
        int a, b, c; //表示凸包一个面上的三个点的编号
        bool ok; //表示该面是否属于最终凸包上的面
    };

    int n; //初始顶点数
    Point P[MAXN]; //初始顶点
    int num; //凸包表面的三角形数
    face F[8 * MAXN]; //凸包表面的三角形
    int g[MAXN][MAXN]; //凸包表面的三角形

    double vlen(Point a) {//向量长度
        return sqrt(a.x*a.x + a.y*a.y + a.z*a.z);
    }
    Point cross(const Point &a, const Point &b, const Point &c) {//叉乘
        return Point((b.y - a.y)*(c.z - a.z) - (b.z - a.z)*(c.y - a.y),
            (b.z - a.z)*(c.x - a.x) - (b.x - a.x)*(c.z - a.z),
            (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x)
```

```
        );
    }

    double area(Point a, Point b, Point c) {//三角形面积*2
        return vlen((b - a)*(c - a));
    }

    double volume(Point a, Point b, Point c, Point d) {//四面体有向体积*6
        return (b - a)*(c - a) ^ (d - a);
    }

    double dblcmp(Point &p, face &f) {//正：点在面同向
        Point m = P[f.b] - P[f.a];
        Point n = P[f.c] - P[f.a];
        Point t = p - P[f.a];
        return (m*n) ^ t;
    }
    void deal(int p, int a, int b) {
        int f = g[a][b];//搜索与该边相邻的另一个平面
        face add;
        if (F[f].ok) {
            if (dblcmp(P[p], F[f]) > eps)
                dfs(p, f);
            else {
                add.a = b;
                add.b = a;
                add.c = p;//这里注意顺序，要成右手系
                add.ok = true;
                g[p][b] = g[a][p] = g[b][a] = num;
                F[num++] = add;
            }
        }
    }
    void dfs(int p, int now)//递归搜索所有应该从凸包内删除的面
    {
        F[now].ok = 0;
        deal(p, F[now].b, F[now].a);
        deal(p, F[now].c, F[now].b);
        deal(p, F[now].a, F[now].c);
    }
    bool same(int s, int t) {
        Point &a = P[F[s].a];
        Point &b = P[F[s].b];
        Point &c = P[F[s].c];
```

```
        return fabs(volume(a, b, c, P[F[t].a])) < eps &&
            fabs(volume(a, b, c, P[F[t].b])) < eps &&
            fabs(volume(a, b, c, P[F[t].c])) < eps;
    }
    //构建三维凸包
    void create() {
        int i, j, tmp;
        face add;

        num = 0;
        if (n < 4)return;
        //**********************************************
        //此段是为了保证前四个点不共面
        bool flag = true;
        for (i = 1; i < n; i++) {
            if (vlen(P[0] - P[i]) > eps) {
                swap(P[1], P[i]);
                flag = false;
                break;
            }
        }
        if (flag)return;
        flag = true;
        //使前三个点不共线
        for (i = 2; i < n; i++) {
            if (vlen((P[0] - P[1])*(P[1] - P[i])) > eps) {
                swap(P[2], P[i]);
                flag = false;
                break;
            }
        }
        if (flag)return;
        flag = true;
        //使前四个点不共面
        for (int i = 3; i < n; i++) {
            if (fabs((P[0] - P[1])*(P[1] - P[2]) ^ (P[0] - P[i])) > eps) {
                swap(P[3], P[i]);
                flag = false;
                break;
            }
        }
        if (flag)return;
        //*****************************************
        for (i = 0; i < 4; i++) {
```

```
            add.a = (i + 1) % 4;
            add.b = (i + 2) % 4;
            add.c = (i + 3) % 4;
            add.ok = true;
            if (dblcmp(P[i], add) > 0)swap(add.b, add.c);
            g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
            F[num++] = add;
        }
        for (i = 4; i < n; i++) {
            for (j = 0; j < num; j++) {
                if (F[j].ok&&dblcmp(P[i], F[j]) > eps) {
                    dfs(i, j);
                    break;
                }
            }
        }
        tmp = num;
        for (i = num = 0; i < tmp; i++)
            if (F[i].ok)
                F[num++] = F[i];

    }
    //表面积
    double area() {
        double res = 0;
        if (n == 3) {
            Point p = cross(P[0], P[1], P[2]);
            res = vlen(p) / 2.0;
            return res;
        }
        for (int i = 0; i < num; i++)
            res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
        return res / 2.0;
    }
    double volume() {
        double res = 0;
        Point tmp(0, 0, 0);
        for (int i = 0; i < num; i++)
            res += volume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
        return fabs(res / 6.0);
    }

    int triangle() {//表面三角形个数
        return num;
```

```
    }

    int polygon() {//表面多边形个数
        int i, j, res, flag;
        for (i = res = 0; i < num; i++) {
            flag = 1;
            for (j = 0; j < i; j++)
                if (same(i, j)) {
                    flag = 0;
                    break;
                }
            res += flag;
        }
        return res;
    }

    Point barycenter() {//三维凸包重心
        Point ans(0, 0, 0), o(0, 0, 0);
        double all = 0;
        for (int i = 0; i < num; i++) {
            double vol = volume(o, P[F[i].a], P[F[i].b], P[F[i].c]);
            ans = ans + (o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4.0*vol;
            all += vol;
        }
        ans = ans / all;
        return ans;
    }
    double ptoface(Point p, int i) {//点到面的距离
        return fabs(volume(P[F[i].a], P[F[i].b], P[F[i].c], p) / vlen((P[F[i].b] - P[F[i].a])*(P[F[i].c] - P[F[i].a])));
    }
};
CH3D hull;
int main() {
    while (scanf("%d", &hull.n) == 1) {
        for (int i = 0; i < hull.n; i++) {
            scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);
        }
        hull.create();
        Point p = hull.barycenter();
        double ans1 = 1e20;
        for (int i = 0; i < hull.num; i++) {
            ans1 = min(ans1, hull.ptoface(p, i));
        }
```

```


        scanf("%d", &hull.n);
        for (int i = 0; i < hull.n; i++) {
            scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);
        }
        hull.create();
        p = hull.barycenter();
        double ans2 = 1e20;
        for (int i = 0; i < hull.num; i++) {
            ans2 = min(ans2, hull.ptoface(p, i));
        }

        printf("%.5f\n", ans1 + ans2);
    }
    return 0;
}
```

矩阵

高斯消元

```
typedef double Mat[N][N];
typedef double vec[N];
const double eps = 1e-7;
Mat A;
vec x, b;
namespace Matrix {//Ax=b
    int piv[N];
    bool equ0(double x) {
        return fabs(x) < eps;
    }
    bool Gauss(Mat & A, vec & x, vec & b, int n) {
        for (int i = 1; i <= n; i++) {
            piv[i] = 0;
            for (int j = 1; j <= n; j++) if (equ0(A[i][j]) == 0) {
                piv[i] = j;
                break;
            }
            if (piv[i] == 0) return 0;
            for (int j = i + 1; j <= n; j++) {
                double tp = A[j][piv[i]] / A[i][piv[i]];
                for (int k = 1; k <= n; k++) A[j][k] -= tp * A[i][k];
                b[j] -= b[i] * tp;
                A[j][piv[i]] = 0;
            }
        }
        for (int i = n; i; i--) {
            double cst = 0;
            for (int j = 1; j <= n; j++) if (equ0(A[i][j]) == 0 && j != piv[i])
                cst += x[j] * A[i][j];
            cst = b[i] - cst;
            x[piv[i]] = cst / A[i][piv[i]];
        }
        return 1;
    }
}

int main() {
    freopen("ic", "r", stdin);
    int n;
    scanf("%d", &n);
```

```
    for (int i = 1; i <= n; i++) {//Input ith equation
        for (int j = 1; j <= n; j++) scanf("%lf", &A[i][j]);
        scanf("%lf", &b[i]);
    }
    // det(A)!=0 return 1, have certain solution , else No solution
    if (Matrix::Gauss(A, x, b, n))
        for (int i = 1; i <= n; i++) printf("%.2lf\n", x[i]);
    else puts("No Solution");
    /*
    3
    1 3 4 5
    1 4 7 3
    9 3 2 2
    ------------------------
    -0.97
    5.18
    -2.39
    */
}

```

标量类

有理数类

```
template <class T>
struct Frac {//支持与ll混合运算
    T a, b;
    Frac(const T &a_ = 0, const T &b_ = 1): a(a_), b(b_) {
        T d = gcd(a, b); //gcd支持负数，无需特判
        a /= d, b /= d;
        if (b < 0) a = -a, b = -b; //若无负数，去掉该行
    }
    const Frac operator+(const Frac &y) const { return {a * y.b + b * y.a, b * y.b}; }
    const Frac operator-(const Frac &y) const { return {a * y.b - b * y.a, b * y.b}; }
    const Frac operator*(const Frac &y) const { return {a * y.a, b * y.b}; }
    const Frac operator/(const Frac &y) const { return {a * y.b, b * y.a}; }
};
```

公式定理

$$[\text{stament}] \coloneqq \begin{cases} 1,\text{stament} \\ 0,\text{else} \end{cases}$$

$$\Gamma(x) \coloneqq \int_0^\infty t^{z-1} e^{-t}\, dt$$

$$n! = \Gamma(n+1)$$

数论

$a \mid b \coloneqq \exists k \in \mathbb{Z}(ak = b)$

$a^k \parallel b \coloneqq (a^k \mid b) \wedge (a^{k+1} \nmid b)$

$\lfloor \alpha \rfloor \coloneqq \max\limits_{k \in \mathbb{Z}, k \le \alpha} k$

$\gcd(a, b) \coloneqq \max\limits_{k \mid a, k \mid b} k$

$$\chi(d) = \begin{cases} -1, & d\%4 = 1 \\ 1, & d\%4 = 3 \\ 0, & d\%2 = 0 \end{cases}$$

$$\mu(d) = \begin{cases} 0, & d \text{ has a square factor} \\ 1, & d \text{ has an even number of prime facotrs} \\ -1, & d \text{ has an odd number of prime facotrs} \end{cases}$$

整值函数

$p^k \parallel n!$, where $k = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor$

$n! = \prod_{p \le n} \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor$

$\left\lfloor \frac{\lfloor n\alpha \rfloor}{n} \right\rfloor = \lfloor \alpha \rfloor$

$\lfloor \alpha \rfloor + \left\lfloor \alpha + \frac{1}{n} \right\rfloor + \cdots + \left\lfloor \alpha + \frac{n-1}{n} \right\rfloor = \lfloor n\alpha \rfloor$

圆内整点数 $T = 1 + 4\lfloor R \rfloor + 8 \sum_{0 < i \le \frac{R}{\sqrt{2}}} \left\lfloor \sqrt{R^2 - i^2} \right\rfloor - 4 \left\lfloor \frac{R}{\sqrt{2}} \right\rfloor = 1 + 4 \sum_{i=0}^{\infty} (-1)^i \left\lfloor \frac{R^2}{2i+1} \right\rfloor, \mathbb{D} = \{(x, y): x^2 + y^2 \le R^2\}$

双曲线内整点数 $T = 2 \sum_{0 < i \le \sqrt{n}} \left\lfloor \frac{n}{i} \right\rfloor - 4 \left\lfloor \sqrt{n} \right\rfloor^2, \mathbb{D} = \{(x, y): x, y \ge 0, xy \le n\}$

组合数学

16 排列数 $\mathrm{P}(n,k) \coloneqq \frac{n!}{(n-k)!}$

17 降阶函数 $x^{\underline{n}} \coloneqq \mathrm{P}(x,n) = \frac{x!}{(x-n)!}$

18 斐波那契数 $F_n \coloneqq \begin{cases} F_{n-1}+F_{n-2}, & n \geq 2 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$

19 调和数 $H_n \coloneqq 1 + \frac{1}{2} + \cdots + \frac{1}{n}$

20 移位算子 $\mathrm{E}\, f(x) \coloneqq \mathrm{f}(x+1)$

21 差分算子 $\Delta\, \mathrm{f}(x) \coloneqq \mathrm{f}(x+1) - \mathrm{f}(x)$

22 定和 $\sum_a^b \mathrm{f}(x)\, \Delta\, x \coloneqq \begin{cases} \sum_{i=a}^{b-1} \mathrm{f}(i), & a \leq b \\ -\sum_b^a \mathrm{f}(x)\, \Delta\, x, & a > b \end{cases}$

差分与不定和

23 $x^{\underline{n+m}} = x^{\underline{n}}(x-n)^{\underline{m}} = (x-m)^{\underline{n}} x^{\underline{m}}$

24 $\Delta\, x^{\underline{n}} = n x^{\underline{n-1}}$

25 $\Delta\, n^x = (n-1) n^x$

26 $\Delta\, H_x = x^{\underline{-1}}$

27 $\Delta(uv) = u\, \Delta\, v + \mathrm{E}\, v\, \Delta\, u$（见公式 20）

28 $\sum x^{\underline{n}}\, \Delta\, x = \begin{cases} \frac{1}{n+1} x^{\underline{n+1}}, & n \neq -1 \\ H_n, & n = -1 \end{cases}$

29 $\sum n^x\, \Delta\, x = \begin{cases} \frac{n^x}{n-1}, & n \neq 1 \\ x, & n = 1 \end{cases}$

30 $\sum u\, \Delta\, v = uv - \sum \mathrm{E}\, v\, \Delta\, u$

二项式系数与特殊计数数列

31 组合数 $\binom{n}{k}$:=n 个元素选出 k 个的方法数

32 第一类有符号斯特林数 $\left[{n \atop k}\right]_s \coloneqq$ 降阶函数的各项系数

33 第一类斯特林数 $\left[{n \atop k}\right] \coloneqq$ 升阶函数的各项系数，n 个元素排成 k 个非空循环排列的方法数

34 第二类特林数 $\left\{{n \atop k}\right\} \coloneqq$ n 个元素划分成 k 个（不可辨别的）非空子集的方法数

35 卡特兰数 $C_n \coloneqq$ n 个元素的出入栈的方法数，n 对括号匹配的方法数，n 个节点的有根二叉树的形态数，n 边形的三角形划分数

通项公式

36 $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n^{\underline{k}}}{k!}$

37 $\left[{n \atop k}\right]_s =$

38 $\left[{n \atop k}\right] = \left|\left[{n \atop k}\right]_s\right|$

39 $\left\{{n \atop k}\right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$

$C_n = \frac{1}{n+1}\binom{2n}{n}$

递推关系

$\binom{n}{k} = \begin{cases} \binom{n-1}{k} + \binom{n-1}{k-1}, & k > 0 \\ 1, & k = 0 \\ 0, & \text{else} \end{cases}$

$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$

$\binom{n}{k} = \frac{n+1-k}{k}\binom{n}{k-1}$

$\binom{n-1}{k} - \binom{n-1}{k-1} = \frac{n-2k}{n}\binom{n}{k}$

$\binom{n}{h}\binom{n-h}{k} = \binom{n}{k}\binom{n-k}{h}$

$\begin{bmatrix} n \\ k \end{bmatrix}_s = \begin{cases} -(n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}_s + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}_s, & n > 0 \wedge k > 0 \\ 1, & n = 0 \wedge k = 0 \\ 0, & \text{else} \end{cases}$

$\begin{bmatrix} n \\ k \end{bmatrix}_u = \begin{cases} (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}, & n > 0 \wedge k > 0 \\ 1, & n = 0 \wedge k = 0 \\ 0, & \text{else} \end{cases}$

$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{cases} k\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}, & n > 0 \wedge k > 0 \\ 1, & n = 0 \wedge k = 0 \\ 0, & \text{else} \end{cases}$

$C_n = \begin{cases} \frac{4n-2}{n+1} C_{n-1}, & n > 0 \\ 1, & n = 0 \end{cases}$

求和关系

$\binom{n+1}{k+1} = \sum_{i=0}^{n}\binom{i}{k} = \sum_{i=k}^{n}\binom{i}{k}$

$(1+x)^{\alpha} = \sum_{i=0}^{\infty}\binom{\alpha}{i}x^i$

$\sum_{i=0}^{n} i^{\underline{k}}\binom{n}{i} = n^{\underline{k}}2^{n-k}$

$\sum_{i=0}^{n}\binom{n}{i}^2 = \sum_{i=0}^{n}\binom{n}{i}\binom{n}{n-i} = \binom{2n}{n}$

容斥原理

$|A_1 \cup A_2 \cup \ldots \cup A_n| = \sum |A_i| - \sum |A_i \cap A_j| + \sum |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1}|A_1 \cap A_2 \cap \ldots \cap A_n|$

$|\overline{A_1} \cap \overline{A_2} \cap \ldots \cap \overline{A_n}| = |S| - \sum |A_i| + \sum |A_i \cap A_j| - \sum |A_i \cap A_j \cap A_k| \ldots + (-1)^n |A_1 \cap A_2 \cap \ldots \cap A_n|$

反演

$\mathrm{F}(n) = \sum_{d|n} \mathrm{f}(d) \Leftrightarrow \mathrm{f}(n) = \sum_{d|n} \mu(d)\,\mathrm{F}\left(\frac{n}{d}\right)$

$\mathrm{F}(n) = \sum_{n|d} \mathrm{f}(d) \Leftrightarrow \mathrm{f}(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right)\mathrm{F}(d)$

$\mathrm{F}(n) = \sum_{i=0}^{n}\binom{n}{i}\mathrm{f}(i) \Leftrightarrow \mathrm{f}(n) = \sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}\mathrm{F}(i)$

$\mathrm{F}(n) = \sum_{i=0}^{n}\begin{Bmatrix} n \\ i \end{Bmatrix}\mathrm{f}(i) \Leftrightarrow \mathrm{f}(n) = \sum_{i=0}^{n}(-1)^{n-i}\begin{bmatrix} n \\ i \end{bmatrix}\mathrm{F}(i)$

$\mathrm{F}(n) = \sum_{i=0}^{n}\begin{bmatrix} n \\ i \end{bmatrix}\mathrm{f}(i) \Leftrightarrow \mathrm{f}(n) = \sum_{i=0}^{n}(-1)^{n-i}\begin{Bmatrix} n \\ i \end{Bmatrix}\mathrm{F}(i)$

错位排列

$D_n := n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$

$D_n = nD_{n-1} + (-1)^n, D_0 = 1$

$n! = \sum_{i=0}^{n} \binom{n}{i} D_i$

迪利克雷卷积

$(\mathrm{f} * \mathrm{g})(n) \coloneqq \sum_{d|n} \mathrm{f}(d)\, \mathrm{g}\left(\frac{n}{d}\right)$

杜教筛

$\mathrm{S}(n) = \sum_{i=1}^{n} \mathrm{f}(i) \Leftrightarrow \mathrm{g}(1)\,\mathrm{S}(n) = \sum_{i=1}^{n} (\mathrm{g} * \mathrm{f})(i) - \sum_{i=2}^{n} \mathrm{g}(i)\,\mathrm{S}\left(\frac{n}{i}\right)$

其中$\mathrm{f}(x)$，$\mathrm{g}(x)$为积性函数，且选择的$\mathrm{g}(x)$应令$\sum_{i=1}^{n} (g * f)(i)$易于求解

$\mu(x)$ 的前缀和 $\mathrm{S}_\mu(n) \coloneqq \sum_{i=1}^{n} \mu(i) = 1 - \sum_{i=2}^{n} \mathrm{S}_\mu\left(\frac{n}{i}\right)$ （选$\mathrm{g}(x) = 1$）

$\varphi(x)$ 前缀和 $\mathrm{S}_\varphi(n) \coloneqq \sum_{i=1}^{n} \varphi(i) = \frac{n(n+1)}{2} - \sum_{i=2}^{n} \mathrm{S}_\varphi\left(\frac{n}{i}\right)$ （选$\mathrm{g}(x) = x$）

$x\varphi(x)$ 前缀和 $\mathrm{S}_{x\varphi}(n) \coloneqq \sum_{i=1}^{n} i\varphi(i) = \frac{n(n+1)(2n+1)}{6} - \sum_{i=2}^{n} i\,\mathrm{S}_{x\varphi}\left(\frac{n}{i}\right)$ （选$\mathrm{g}(x) = x$）

生成函数

（组合）生成函数

（组合）生成函数 $\{a_n\}(x) \coloneqq \sum_{n=0}^{\infty} a_n x^n$

$\{1\}(x) \coloneqq \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$

$\left\{\binom{k}{n}\right\}(x) \coloneqq \sum_{n=0}^{\infty} \binom{k}{n} x^n = (1+x)^k$

分拆数

五边形数 $Five_n \coloneqq \frac{n(3n-1)}{2}$

五边形数定理 $\Phi(x) = \prod_{n=1}^{\infty}(1-x^n) = \sum_{n=-\infty}^{\infty}(-1)^n x^{\frac{n(3n-1)}{2}} = 1 + \sum_{n=1}^{\infty}(-1)^n x^{\frac{n(3n\mp1)}{2}} = 1 - x - x^2 + x^5 + x^7 \ldots$

分拆数 $p_n \coloneqq$把 n 分拆成若干个正整数之和的方案数

$\{p_n\}(x) = (1 + x + x^2 + \cdots)(1 + x^2 + x^4 + \cdots)(1 + x^3 + x^6 + \cdots)\cdots = \prod_{n=1}^{\infty} \frac{1}{1-x^n} = \frac{1}{\Phi(x)}$

$p_n = p_{n-Five_1} + p_{n-Five_{-1}} - p_{n-Five_2} - p_{n-Five_{-2}} + p_{n-Five_3} + \cdots = p_{n-1} + p_{n-2} - p_{n-5} - p_{n-7} + p_{n-12} + \cdots$

限制分拆数 $p_n^{(k)} \coloneqq$把 n 分拆成若干个正整数之和，且每个分拆数出现次数严格小于 k

$\left\{p_n^{(k)}\right\}(x) \coloneqq (1 + x + \cdots + x^{k-1})\left(1 + x^2 + \cdots + x^{2(k-1)}\right)\left(1 + x^3 + \cdots + x^{3(k-1)}\right)\cdots = \prod_{n=1}^{\infty} \frac{1-x^{kn}}{1-x^n} = \{p_n\}(x)\,\Phi(x^k)$

$p_n^{(k)} = p_n - p_{n-k} - p_{n-2k} + p_{n-5k} + p_{n-7k} - p_{n-12k} - \cdots$

排列生成函数

排列生成函数 $\{a_n\}_P(x) \coloneqq \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}$

$\{1\}_P(x) \coloneqq \sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x$

$\{n\%2\}_P(x) \coloneqq \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = \frac{e^x - e^{-x}}{2} = \sinh x$

83 $\{(n+1)\%2\}_P(x) \coloneqq \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = \frac{e^x+e^{-x}}{2} = \cosh x$

84 $\{[n\%a=b]\}_P(x) \coloneqq \sum_{n=0}^{\infty} \frac{x^{an+b}}{(an+b)!} = \frac{1}{a}\sum_{k=0}^{a-1} \frac{\exp(\omega_a^k x)}{\omega_a^{bk}}$, 其中$\omega_a \coloneqq \exp\left(\frac{2\pi}{a} i\right)$

Ramsey 定理

85 对于一个给定的两个整数$n, m \geq 2$，则一定存在一个最小整数$\mathrm{R}(n,m)$，使得用两种颜色（例如红蓝）无论给K_R的每条边如何染色，总能找到一个红色的K_n或者蓝色的K_m。（K_p是指p个结点的完全图）

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	2	3	4	5	6	7	8	9
3		6	9	14	18	23	28	36
4			18	25	36~41	49~61	59~84	73~115
5				43~48	58~87	80~143	101~216	133~316
6					102~165	115~298	134~495	183~780
7						205~540	217~1031	252~1713
8							282~1870	329~3583
9								565~6588

Ramsey 定理的 R(n,m)表

Pólya 计数

连续数学

三角函数与双曲函数

1 $\sin x = \frac{e^{ix}-e^{-ix}}{2i} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$

2 $\cos x = \frac{e^{ix}+e^{-ix}}{2} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$

3 $\tan x = \frac{\sin x}{\cos x} = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \frac{62}{2835}x^9 + \cdots$

4 $\sinh x = \frac{e^{x}-e^{-x}}{2} = \frac{x}{1!} + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots$

5 $\cosh x = \frac{e^{x}+e^{-x}}{2} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \cdots$

6 $\tanh x = \frac{\sinh x}{\cosh x} = x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7 + \frac{62}{2835}x^9 - \cdots$

基本性质

7 $\sin^2 x + \cos^2 x = 1$

8 $1 + \tan^2 x = \sec^2 x$

9 $1 + \cot^2 x = \csc^2 x$

10 $\cosh^2 x - \sinh^2 x = 1$

11 $1 - \tanh^2 x = \operatorname{sech}^2 x$

12 $1 - \coth^2 x = -\operatorname{csch}^2 x$

和差

13 $\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y$

14 $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y$

15 $\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y}$

16 $\sinh(x \pm y) = \sinh x \cos y \pm \cosh x \sinh y$

17 $\cosh(x \pm y) = \cosh x \cos y \pm \sinh x \sinh y$

18 $\tanh(x \pm y) = \frac{\tanh x \pm \tanh y}{1 \pm \tanh x \tanh y}$

19 $\sin(x + y + z) = \sin x \cos y \cos z + \cos x \sin y \cos z + \cos x \cos y \sin z - \sin x \sin y \sin z$

20 $\cos(x + y + z) = \cos x \cos y \cos z - \cos x \sin y \sin z - \sin x \cos y \sin z - \sin x \sin y \cos z$

21 $\tan(x + y + z) = (\tan x + \tan y + \tan z - \tan x \tan y \tan z)/(1 - \tan x \tan y - \tan y \tan z - \tan z \tan x)$

和差化积

22 $\sin x + \sin y = 2 \sin \frac{x+y}{2} \cos \frac{x-y}{2}$

23 $\sin x - \sin y = 2 \cos \frac{x+y}{2} \sin \frac{x-y}{2}$

24 $\cos x + \cos y = 2 \cos \frac{x+y}{2} \cos \frac{x-y}{2}$

25 $\cos x - \cos y = -2 \sin \frac{x+y}{2} \sin \frac{x-y}{2}$

26 $\tan x \pm \tan y = \frac{\sin(x \pm y)}{\cos x \cos y}$

27 $\sinh x + \sinh y = 2 \sinh \frac{x+y}{2} \cosh \frac{x-y}{2}$

$\sinh x - \sinh y = 2\cosh\frac{x+y}{2}\sinh\frac{x-y}{2}$

$\cosh x + \cosh y = 2\cosh\frac{x+y}{2}\cosh\frac{x-y}{2}$

$\cosh x - \cosh y = 2\sinh\frac{x+y}{2}\sinh\frac{x-y}{2}$

积化和差

$\cos x\sin y = \frac{1}{2}(\sin(x+y) - \sin(x-y))$

$\sin x\cos y = \frac{1}{2}(\sin(x+y) + \sin(x-y))$

$\cos x\cos y = \frac{1}{2}(\cos(x+y) + \cos(x-y))$

$\sin x\sin y = -\frac{1}{2}(\cos(x+y) - \cos(x-y))$

$\cosh x\sinh y = \frac{1}{2}(\sinh(x+y) - \sinh(x-y))$

$\sinh x\cosh y = \frac{1}{2}(\sinh(x+y) + \sinh(x-y))$

$\cosh x\cosh y = \frac{1}{2}(\cosh(x+y) + \cosh(x-y))$

$\sinh x\sinh y = \frac{1}{2}(\cosh(x+y) - \cosh(x-y))$

倍角

$\sin 2x = 2\sin x\cos x$

$\cos 2x = \cos^2 x - \sin^2 x = 2\cos^2 x - 1 = 1 - 2\sin^2 x$

$\tan 2x = \frac{2\tan x}{1-\tan^2 x}$

$\sinh 2x = 2\sinh x\cosh x$

$\cosh 2x = \cosh^2 x + \sinh^2 x = 2\cosh^2 x - 1 = 1 + 2\sinh^2 x$

$\tanh 2x = \frac{2\tanh x}{1+\tanh^2 x}$

$\sin 3x = 4\sin x\sin\left(\frac{\pi}{3} - x\right)\sin\left(\frac{\pi}{3} + x\right) = 3\sin x - 4\sin^3 x$

$\cos 3x = 4\cos x\cos\left(\frac{\pi}{3} - x\right)\cos\left(\frac{\pi}{3} + x\right) = 4\cos^3 x - 3\cos x$

$\tan 3x = 4\tan x\tan\left(\frac{\pi}{3} - x\right)\tan\left(\frac{\pi}{3} + x\right)$

$\sinh 3x = 3\sinh x + 4\sinh^3 x$

$\cosh 3x = 4\cosh^3 x - 3\cosh x$

$\sin nx = \sum_{i=0}^{\infty}(-1)^i\binom{n}{2i-1}\cos^{n-2i+1} x\sin^{2i-1} x = \sin x\sum_{i=0}^{\infty}((-1)^i\cos^{n-2i+1} x\sum_{j=i}^{\infty}\binom{n}{2j+1}\binom{j}{i})$

$\cos nx = \sum_{i=0}^{\infty}(-1)^i\binom{n}{2i}\cos^{n-2i} x\sin^{2i} x = \sum_{i=0}^{\infty}((-1)^i\cos^{n-2i} x\sum_{j=i}^{\infty}\binom{n}{2j}\binom{j}{i})$

半角

52 $\sin\frac{x}{2}=\pm\sqrt{\frac{1-\cos x}{2}}$

53 $\cos\frac{x}{2}=\pm\sqrt{\frac{1+\cos x}{2}}$

54 $\tan\frac{x}{2}=\pm\sqrt{\frac{1-\cos x}{1+\cos x}}=\frac{\sin x}{1+\cos x}=\frac{1-\cos x}{\sin x}$

55 $\cot\frac{x}{2}=\pm\sqrt{\frac{1+\cos x}{1-\cos x}}=\frac{1+\cos x}{\sin x}=\frac{\sin x}{1-\cos x}$

万能

56 $\sin x=\frac{2\tan\frac{x}{2}}{1+\tan^2\frac{a}{2}}$

57 $\cos x=\frac{1-\tan^2\frac{a}{2}}{1+\tan^2\frac{a}{2}}$

58 $\tan x=\frac{2\tan\frac{x}{2}}{1-\tan^2\frac{a}{2}}$

辅助角

59 $a\sin x+b\cos x=A\sin(x+\varphi)$, where $A=\sqrt{a^2+b^2},\ \tan\varphi=\frac{b}{a}$

60 $\sum_{k=0}^{\infty}A_k\sin(x+\varphi_k)=A\sin(x+\varphi)$, where $A=|Z|,\ \varphi=\arg Z,\ Z=\sum_{k=0}^{\infty}A_k\mathrm{e}^{\mathrm{i}\varphi_k}$

几何

正余弦定理

1 $\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}=2$

2 $a^2=b^2+c^2-2bc\cos A$

三角形各心

3 质心 $\overrightarrow{O_{\text{mass}}}=\frac{1}{3}\left(\vec{A}+\vec{B}+\vec{C}\right)$

4 内心 $\overrightarrow{O_{\text{inner}}}=\frac{1}{a+b+c}\left(a\vec{A}+b\vec{B}+c\vec{C}\right)$

5 外心 $\overrightarrow{O_{\text{outer}}}=$

6 垂心 $\overrightarrow{O_{\text{ortho}}}=$

7 旁心 $\overrightarrow{O_{\text{escri}}}=$

面积

8 $S_3=\frac{1}{2}lh=\frac{1}{2}ab\sin C=\frac{1}{2}rp=\frac{1}{4R}abc=\frac{1}{2}\begin{vmatrix}1 & x_A & y_A\\1 & x_B & y_B\\1 & x_C & y_C\end{vmatrix}$, where $p=\frac{1}{2}(a+b+c)$

9 $S_3=\sqrt{p(p-a)(p-b)(p-c)}$

10 圆内接四边形 $S_4=\sqrt{(p-a)(p-b)(p-c)(p-d)}$, where $p=\frac{1}{2}(a+b+c+d)$

11 圆内接正多边形 $S_n=nR^2\sin\frac{\pi}{n}\cos\frac{\pi}{n}=\frac{1}{2}nR^2\sin\frac{2\pi}{n}$

12 圆外接正多边形 $S_n = nR^2 \tan\frac{\pi}{n}$

表面积

13 球冠(球缺) $S = 2\pi RH$

14 高维球体 $S_n = \frac{dV_n}{dR} = \frac{2\pi^{\frac{n}{2}}}{\left(\frac{n}{2}-1\right)!}R^{n-1}$, $\left(-\frac{1}{2}\right)! = \sqrt{\pi}$

体积

15 球缺(球冠) $V = \frac{1}{3}\pi H^2(3R - H) = \frac{1}{6}\pi H(3r^2 + H^2)$

16 台体 $\mathrm{V} = \frac{1}{3}h\left(S_1 + \sqrt{S_1 S_2} + S_2\right)$

17 高维球体 $V_n = \frac{\pi^{\frac{n}{2}}}{\left(\frac{n}{2}\right)!}R^n$, $\left(\frac{1}{2}\right)! = \frac{\sqrt{\pi}}{2}$

特殊函数的微分

18 $\frac{\mathrm{d}}{\mathrm{d}x}(|x|) = \operatorname{sgn} x$

19 $\frac{\mathrm{d}}{\mathrm{d}x}(\operatorname{sgn} x) = 2\,\delta(x)$

20 $\frac{\mathrm{d}}{\mathrm{d}x}(\arcsin x) = \frac{1}{\sqrt{1-x^2}}$

21 $\frac{\mathrm{d}}{\mathrm{d}x}(\arctan x) = \frac{1}{1+x^2}$

积分

不定积分

含 $\boldsymbol{ax+b}$ 的积分

1 $\frac{1}{ax+b} \rightsquigarrow \frac{1}{a}\ln|ax+b|$

2 $(ax+b)^{\mu} \rightsquigarrow \frac{1}{a(\mu+1)}(ax+b)^{\mu+1}\ (\mu \neq -1)$

3 $\frac{x}{ax+b} \rightsquigarrow \frac{1}{a^2}(ax+b-b\ln|ax+b|)$

4 $\frac{x^2}{ax+b} \rightsquigarrow \frac{1}{a^3}\left(\frac{1}{2}(ax+b)^2 - 2b(ax+b) + b^2\ln|ax+b|\right)$

5 $\frac{1}{x(ax+b)} \rightsquigarrow -\frac{1}{b}\ln\left|\frac{ax+b}{x}\right|$

6 $\frac{1}{x^2(ax+b)} \rightsquigarrow -\frac{1}{bx} + \frac{a}{b^2}\ln\left|\frac{ax+b}{x}\right|$

7 $\frac{x}{(ax+b)^2} \rightsquigarrow \frac{1}{a^2}\left(\ln|ax+b| + \frac{b}{ax+b}\right)$

8 $\frac{x^2}{(ax+b)^2} \rightsquigarrow \frac{1}{a^3}\left(ax+b-2b\ln|ax+b| - \frac{b^2}{ax+b}\right)$

9 $\frac{1}{x(ax+b)^2} \rightsquigarrow \frac{1}{b(ax+b)} - \frac{1}{b^2}\ln\left|\frac{ax+b}{x}\right|$

含 $\sqrt{ax+b}$ 的积分

10 $\sqrt{ax+b} \rightsquigarrow \frac{2}{3a}\sqrt{(ax+b)^3}$

11 $x\sqrt{ax+b} \rightsquigarrow \frac{2}{15a^2}(3ax-2b)\sqrt{(ax+b)^3}$

12 $x^2\sqrt{ax+b} \rightsquigarrow \frac{2}{105a^3}(15a^2x^2-12abx+8b^2)\sqrt{(ax+b)^3}$

13 $\frac{x}{\sqrt{ax+b}} \rightsquigarrow \frac{2}{3a^2}(ax-2b)\sqrt{ax+b}$

14 $\frac{x^2}{\sqrt{ax+b}} \rightsquigarrow \frac{2}{15a^3}(3a^2x^2-4abx+8b^2)\sqrt{ax+b}$

15 $\frac{1}{x\sqrt{ax+b}} \rightsquigarrow \begin{cases} \frac{1}{\sqrt{b}}\ln\left|\frac{\sqrt{ax+b}-\sqrt{b}}{\sqrt{ax+b}+\sqrt{b}}\right|, & b>0 \\ \frac{2}{\sqrt{-b}}\arctan\sqrt{\frac{ax+b}{-b}}, & b<0 \end{cases}$

16 $\frac{1}{x^2\sqrt{ax+b}} \rightsquigarrow -\frac{\sqrt{ax+b}}{bx} - \frac{a}{2b}\int\frac{1}{x\sqrt{ax+b}}$

17 $\frac{\sqrt{ax+b}}{x} \rightsquigarrow 2\sqrt{ax+b} + b\int\frac{1}{x\sqrt{ax+b}}$

18 $\frac{\sqrt{ax+b}}{x^2} \rightsquigarrow -\frac{\sqrt{ax+b}}{x} + \frac{a}{2}\int\frac{1}{x\sqrt{ax+b}}$

含 $x \pm a^2$ 的积分

19 $\frac{1}{x^2+a^2} \rightsquigarrow \frac{1}{a}\arctan\frac{x}{a}$

20 $\frac{1}{(x^2+a^2)^n} \rightsquigarrow \frac{x}{2(n-1)a^2(x^2+a^2)^{n-1}} + \frac{2n-3}{2(n-1)a^2}\int\frac{1}{(x^2+a^2)^{n-1}}$

21 $\frac{1}{x^2-a^2} \rightsquigarrow \frac{1}{2a}\ln\left|\frac{x-a}{x+a}\right|$

含 $ax^2+b\ (a>0)$ 的积分

22 $\frac{1}{ax^2+b} \rightsquigarrow \begin{cases} \frac{1}{\sqrt{ab}}\arctan\sqrt{\frac{a}{b}}x, & b>0 \\ \frac{1}{2\sqrt{-ab}}\ln\left|\frac{\sqrt{a}x-\sqrt{-b}}{\sqrt{a}x+\sqrt{-b}}\right|, & b<0 \end{cases}$

23 $\frac{x}{ax^2+b} \rightsquigarrow \frac{1}{2a}\ln|ax^2+b|$

24 $\frac{x^2}{ax^2+b} \rightsquigarrow \frac{x}{a} - \frac{b}{a}\int\frac{1}{ax^2+b}$

25 $\frac{1}{x(ax^2+b)} \rightsquigarrow \frac{1}{2b}\ln\frac{x^2}{|ax^2+b|}$

26 $\frac{1}{x^2(ax^2+b)} \rightsquigarrow -\frac{1}{bx} - \frac{a}{b}\int\frac{1}{ax^2+b}$

27 $\frac{1}{x^3(ax^2+b)} \rightsquigarrow \frac{a}{2b^2}\ln\frac{|ax^2+b|}{x^2} - \frac{1}{2bx^2}$

28 $\frac{1}{(ax^2+b)^2} \rightsquigarrow \frac{x}{2b(ax^2+b)} + \frac{1}{2b}\int \frac{1}{ax^2+b}$

含 $\boldsymbol{ax^2+bx+c\ (a>0)}$ 的积分

29 $\frac{1}{ax^2+bx+c} \rightsquigarrow \begin{cases} \frac{2}{\sqrt{-\Delta}}\arctan\frac{2ax+b}{\sqrt{-\Delta}}, & \Delta<0 \\ \frac{1}{\sqrt{\Delta}}\ln\left|\frac{2ax+b-\sqrt{\Delta}}{2ax+b+\sqrt{\Delta}}\right|, & \Delta>0 \end{cases}$, 其中 $\Delta = b^2-4ac$

30 $\frac{x}{ax^2+bx+c} \rightsquigarrow \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int\frac{1}{ax^2+bx+c}$

含 $\boldsymbol{\sqrt{x^2+a^2}\ (a>0)}$ 的积分

31 $\frac{1}{\sqrt{x^2+a^2}} \rightsquigarrow \operatorname{arcsinh}\frac{x}{a} = \ln\left(x+\sqrt{x^2+a^2}\right)$

32 $\frac{1}{\left(\sqrt{x^2+a^2}\right)^3} \rightsquigarrow \frac{x}{a^2\sqrt{x^2+a^2}}$

33 $\frac{x}{\sqrt{x^2+a^2}} \rightsquigarrow \sqrt{x^2+a^2}$

34 $\frac{x}{\left(\sqrt{x^2+a^2}\right)^3} \rightsquigarrow -\frac{1}{\sqrt{x^2+a^2}}$

35 $\frac{x^2}{\sqrt{x^2+a^2}} \rightsquigarrow \frac{x}{2}\sqrt{x^2+a^2} - \frac{a^2}{2}\ln\left(x+\sqrt{x^2+a^2}\right)$

36 $\frac{x^2}{\left(\sqrt{x^2+a^2}\right)^3} \rightsquigarrow -\frac{x}{\sqrt{x^2+a^2}} + \ln\left(x+\sqrt{x^2+a^2}\right)$

37 $\frac{1}{x\sqrt{x^2+a^2}} \rightsquigarrow \frac{1}{a}\ln\frac{\sqrt{x^2+a^2}-a}{|x|}$

38 $\frac{1}{x^2\sqrt{x^2+a^2}} \rightsquigarrow -\frac{\sqrt{x^2+a^2}}{a^2x}$

39 $\sqrt{x^2+a^2} \rightsquigarrow \frac{x}{2}\sqrt{x^2+a^2} + \frac{a^2}{2}\ln\left(x+\sqrt{x^2+a^2}\right)$

40 $\left(\sqrt{x^2+a^2}\right)^3 \rightsquigarrow \frac{x}{8}(2x^2+5a^2)\sqrt{x^2+a^2} + \frac{3}{8}a^4\ln\left(x+\sqrt{x^2+a^2}\right)$

41 $x\sqrt{x^2+a^2} \rightsquigarrow \frac{1}{3}\left(\sqrt{x^2+a^2}\right)^3$

42 $x^2\sqrt{x^2+a^2} \rightsquigarrow \frac{x}{8}(2x^2+a^2)\sqrt{x^2+a^2} - \frac{1}{8}a^4\ln\left(x+\sqrt{x^2+a^2}\right)$

43 $\frac{\sqrt{x^2+a^2}}{x} \rightsquigarrow \sqrt{x^2+a^2} + a\ln\frac{\sqrt{x^2+a^2}-a}{|x|}$

44 $\frac{\sqrt{x^2+a^2}}{x^2} \rightsquigarrow -\frac{\sqrt{x^2+a^2}}{x} + \ln\left(x+\sqrt{x^2+a^2}\right)$

含 $\boldsymbol{\sqrt{x^2-a^2}\ (a>0)}$ 的积分

45 $\frac{1}{\sqrt{x^2-a^2}} \rightsquigarrow \operatorname{sgn} x \operatorname{arccosh}\frac{|x|}{a} = \ln\left|x+\sqrt{x^2-a^2}\right|$

46 $\frac{1}{\left(\sqrt{x^2-a^2}\right)^3} \rightsquigarrow -\frac{x}{a^2\sqrt{x^2-a^2}}$

47 $\frac{x}{\sqrt{x^2-a^2}} \rightsquigarrow \sqrt{x^2-a^2}$

48 $\frac{x}{\left(\sqrt{x^2-a^2}\right)^3} \rightsquigarrow -\frac{1}{\sqrt{x^2-a^2}}$

49 $\frac{x^2}{\sqrt{x^2-a^2}} \rightsquigarrow \frac{x}{2}\sqrt{x^2-a^2}+\frac{a^2}{2}\ln\left|x+\sqrt{x^2-a^2}\right|$

50 $\frac{x^2}{\left(\sqrt{x^2-a^2}\right)^3} \rightsquigarrow -\frac{x}{\sqrt{x^2-a^2}}+\ln\left|x+\sqrt{x^2-a^2}\right|$

51 $\frac{1}{x\sqrt{x^2-a^2}} \rightsquigarrow \frac{1}{a}\arccos\frac{a}{|x|}$

52 $\frac{1}{x^2\sqrt{x^2-a^2}} \rightsquigarrow \frac{\sqrt{x^2-a^2}}{a^2x}$

53 $\sqrt{x^2-a^2} \rightsquigarrow \frac{x}{2}\sqrt{x^2-a^2}-\frac{a^2}{2}\ln\left|x+\sqrt{x^2-a^2}\right|$

54 $\left(\sqrt{x^2-a^2}\right)^3 \rightsquigarrow \frac{x}{8}(2x^2-5a^2)\sqrt{x^2-a^2}+\frac{3}{8}a^4\ln\left|x+\sqrt{x^2-a^2}\right|$

55 $x\sqrt{x^2-a^2} \rightsquigarrow \frac{1}{3}\left(\sqrt{x^2-a^2}\right)^3$

56 $x^2\sqrt{x^2-a^2} \rightsquigarrow \frac{x}{8}(2x^2-a^2)\sqrt{x^2-a^2}+\frac{1}{8}a^4\ln\left|x+\sqrt{x^2-a^2}\right|$

57 $\frac{\sqrt{x^2-a^2}}{x} \rightsquigarrow \sqrt{x^2-a^2}-a\arccos\frac{a}{|x|}$

58 $\frac{\sqrt{x^2-a^2}}{x^2} \rightsquigarrow -\frac{\sqrt{x^2-a^2}}{x}+\ln\left|x+\sqrt{x^2-a^2}\right|$

含 $\sqrt{a^2-x^2}$ $(a>0)$ 的积分

59 $\frac{1}{\sqrt{a^2-x^2}} \rightsquigarrow \arcsin\frac{x}{a}$

60 $\frac{1}{\left(\sqrt{a^2-x^2}\right)^3} \rightsquigarrow \frac{x}{a^2\sqrt{a^2-x^2}}$

61 $\frac{x}{\sqrt{a^2-x^2}} \rightsquigarrow -\sqrt{a^2-x^2}$

62 $\frac{x}{\left(\sqrt{a^2-x^2}\right)^3} \rightsquigarrow \frac{1}{\sqrt{a^2-x^2}}$

63 $\frac{x^2}{\sqrt{a^2-x^2}} \rightsquigarrow -\frac{x}{2}\sqrt{a^2-x^2}+\frac{a^2}{2}\arcsin\frac{x}{a}$

64 $\frac{x^2}{\left(\sqrt{a^2-x^2}\right)^3} \rightsquigarrow \frac{x}{\sqrt{a^2-x^2}}-\arcsin\frac{x}{a}$

65 $\frac{1}{x\sqrt{a^2-x^2}} \rightsquigarrow \frac{1}{a}\ln\frac{a-\sqrt{a^2-x^2}}{|x|}$

66 $\frac{1}{x^2\sqrt{a^2-x^2}} \rightsquigarrow -\frac{\sqrt{a^2-x^2}}{a^2x}$

67 $\sqrt{a^2-x^2} \rightsquigarrow \frac{x}{2}\sqrt{a^2-x^2}+\frac{a^2}{2}\arcsin\frac{x}{a}$

68 $\left(\sqrt{a^2-x^2}\right)^3=\frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2}+\frac{3}{8}a^4\arcsin\frac{x}{a}$

69 $x\sqrt{a^2-x^2} \rightsquigarrow -\frac{1}{3}\left(\sqrt{a^2-x^2}\right)^3$

70 $x^2\sqrt{a^2-x^2} \rightsquigarrow \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2}+\frac{a^4}{8}\arcsin\frac{x}{a}$

71 $\frac{\sqrt{a^2-x^2}}{x} \rightsquigarrow \sqrt{a^2-x^2}+a\ln\frac{a-\sqrt{a^2-x^2}}{|x|}$

72 $\frac{\sqrt{a^2-x^2}}{x^2} \rightsquigarrow -\frac{\sqrt{a^2-x^2}}{x}-\arcsin\frac{x}{a}$

含 $\sqrt{\pm\boldsymbol{ax^2+bx+c}}\ (\boldsymbol{a>0})$ 的积分

73 $\frac{1}{\sqrt{ax^2+bx+c}} \rightsquigarrow \frac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|$

74 $\sqrt{ax^2+bx+c} \rightsquigarrow \frac{2ax+b}{4a}\sqrt{ax^2+bx+c}+\frac{4ac-b^2}{8\sqrt{a^3}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|$

75 $\frac{x}{\sqrt{ax^2+bx+c}} \rightsquigarrow \frac{1}{a}\sqrt{ax^2+bx+c}-\frac{b}{2\sqrt{a^3}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|$

76 $\frac{1}{\sqrt{-ax^2+bx+c}} \rightsquigarrow -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}$

77 $\sqrt{-ax^2+bx+c} \rightsquigarrow \frac{2ax-b}{4a}\sqrt{-ax^2+bx+c}+\frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}$

78 $\frac{x}{\sqrt{-ax^2+bx+c}} \rightsquigarrow -\frac{1}{a}\sqrt{-ax^2+bx+c}+\frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}$

含 $\sqrt{\pm\frac{\boldsymbol{x-a}}{\boldsymbol{x-b}}}$ 或 $\sqrt{(\boldsymbol{x-a})(\boldsymbol{b-x})}$ 的积分

79 $\sqrt{\frac{x-a}{x-b}} \rightsquigarrow (x-b)\sqrt{\frac{x-a}{x-b}}+(b-a)\ln\left(\sqrt{|x-a|}+\sqrt{|x-b|}\right)$

80 $\sqrt{\frac{x-a}{b-x}} \rightsquigarrow (x-b)\sqrt{\frac{x-a}{b-x}}+(b-a)\arcsin\sqrt{\frac{x-a}{b-a}}$

81 $\frac{1}{\sqrt{(x-a)(b-x)}} \rightsquigarrow 2\arcsin\sqrt{\frac{x-a}{b-a}}$

82 $\sqrt{(x-a)(b-x)} \rightsquigarrow \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)}+\frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-a}}$

含三角函数的积分

83 $\sin x \rightsquigarrow -\cos x$

84 $\cos x \rightsquigarrow \sin x$

85 $\tan x \rightsquigarrow -\ln|\cos x|$

86 $\cot x \rightsquigarrow \ln|\sin x|$

87 $\sec x \rightsquigarrow \ln\left|\tan\left(\frac{x}{2}+\frac{\pi}{4}\right)\right|=\ln|\sec x+\tan x|$

88 $\csc x \rightsquigarrow \ln\left|\tan\frac{x}{2}\right| = \ln|\csc x - \cot x|$

89 $\sec^2 x \rightsquigarrow \tan x$

90 $\csc^2 x \rightsquigarrow -\cot x$

91 $\sec x \tan x \rightsquigarrow \sec x$

92 $\csc x \cot x \rightsquigarrow -\csc x$

93 $\sin^2 x \rightsquigarrow \frac{x}{2} - \frac{1}{4}\sin 2x$

94 $\cos^2 x \rightsquigarrow \frac{x}{2} + \frac{1}{4}\sin 2x$

95 $\sin^n x \rightsquigarrow -\frac{1}{n}\sin^{n-1} x \cos x + \frac{n-1}{n}\int \sin^{n-2} x$

96 $\cos^n x \rightsquigarrow \frac{1}{n}\cos^{n-1} x \sin x + \frac{n-1}{n}\int \cos^{n-2} x$

97 $\frac{1}{\sin^n x} \rightsquigarrow -\frac{1}{n-1}\cdot\frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1}\int \frac{1}{\sin^{n-2} x}$

98 $\frac{1}{\cos^n x} \rightsquigarrow \frac{1}{n-1}\cdot\frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1}\int \frac{1}{\cos^{n-2} x}$

99 $\cos^m x \sin^n x \rightsquigarrow \frac{1}{m+n}\cos^{m-1} x \sin^{n+1} x + \frac{m-1}{m+n}\int \cos^{m-2} x \sin^n x = -\frac{1}{m+n}\cos^{m+1} x \sin^{n-1} x + \frac{n-1}{m+n}\int \cos^m x \sin^{n-2} x$

100 $\sin ax \cos bx \rightsquigarrow -\frac{1}{2(a+b)}\cos(a+b)x - \frac{1}{2(a-b)}\cos(a-b)x$

101 $\sin ax \sin bx \rightsquigarrow -\frac{1}{2(a+b)}\sin(a+b)x + \frac{1}{2(a-b)}\sin(a-b)x$

102 $\cos ax \cos bx \rightsquigarrow \frac{1}{2(a+b)}\sin(a+b)x + \frac{1}{2(a-b)}\sin(a-b)x$

103 $\frac{1}{a+b\sin x} \rightsquigarrow \frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}}\ (a^2 > b^2)$

104 $\frac{1}{a+b\sin x} \rightsquigarrow \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right|\ (a^2 < b^2)$

105 $\frac{1}{a+b\cos x} \rightsquigarrow \frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right)\ (a^2 > b^2)$

106 $\frac{1}{a+b\cos x} \rightsquigarrow \frac{1}{a+b}\sqrt{\frac{a+b}{b-a}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right|\ (a^2 < b^2)$

107 $\frac{1}{a^2\cos^2 x+b^2\sin^2 x} \rightsquigarrow \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right)$

108 $\frac{1}{a^2\cos^2 x-b^2\sin^2 x} \rightsquigarrow \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right|$

109 $x\sin ax \rightsquigarrow \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax$

110 $x^2\sin ax \rightsquigarrow -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax$

111 $x\cos ax \rightsquigarrow \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax$

112 $x^2\cos ax \rightsquigarrow \frac{1}{a}x^2\sin ax+\frac{2}{a^2}x\cos ax-\frac{2}{a^3}\sin ax$

含反三角函数的积分

113 $\arcsin\frac{x}{a} \rightsquigarrow x\arcsin\frac{x}{a}+\sqrt{a^2-x^2}$

114 $x\arcsin\frac{x}{a} \rightsquigarrow \left(\frac{x^2}{2}-\frac{a^2}{4}\right)\arcsin\frac{x}{a}+\frac{x}{4}\sqrt{a^2-x^2}$

115 $x^2\arcsin\frac{x}{a} \rightsquigarrow \frac{x^3}{3}\arcsin\frac{x}{a}+\frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2}$

116 $\arccos\frac{x}{a} \rightsquigarrow x\arccos\frac{x}{a}-\sqrt{a^2-x^2}$

117 $x\arccos\frac{x}{a} \rightsquigarrow \left(\frac{x^2}{2}-\frac{a^2}{4}\right)\arccos\frac{x}{a}-\frac{x}{4}\sqrt{a^2-x^2}$

118 $x^2\arccos\frac{x}{a} \rightsquigarrow \frac{x^3}{3}\arccos\frac{x}{a}-\frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2}$

119 $\arctan\frac{x}{a} \rightsquigarrow x\arctan\frac{x}{a}-\frac{a}{2}\ln(a^2+x^2)$

120 $x\arctan\frac{x}{a} \rightsquigarrow \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a}-\frac{a}{2}x$

121 $x^2\arctan\frac{x}{a} \rightsquigarrow \frac{x^3}{3}\arctan\frac{x}{a}-\frac{a}{6}x^2+\frac{a^3}{6}\ln(a^2+x^2)$

含指数函数的积分

122 $a^x \rightsquigarrow \frac{1}{\ln a}a^x$

123 $e^{ax} \rightsquigarrow \frac{1}{a}e^{ax}$

124 $xe^{ax} \rightsquigarrow \frac{1}{a^2}(ax-1)e^{ax}$

125 $x^ne^{ax} \rightsquigarrow \frac{1}{a}x^ne^{ax}-\frac{n}{a}\int x^{n-1}e^{ax}$

126 $xa^x \rightsquigarrow \left(\frac{x}{\ln a}-\frac{1}{\ln^2 a}\right)a^x$

127 $x^na^x \rightsquigarrow \frac{1}{\ln a}x^na^x-\frac{n}{\ln a}\int x^{n-1}a^x$

128 $e^{ax}\sin bx \rightsquigarrow \frac{1}{a^2+b^2}e^{ax}(a\sin bx-b\cos bx)$

129 $e^{ax}\cos bx \rightsquigarrow \frac{1}{a^2+b^2}e^{ax}(b\sin bx+a\cos bx)$

130 $e^{ax}\sin^n bx \rightsquigarrow \frac{1}{a^2+b^2n^2}e^{ax}\sin^{n-1}bx\,(a\sin bx-nb\cos bx)+\frac{n(n-1)b^2}{a^2+b^2n^2}\int e^{ax}\sin^{n-2}bx$

131 $e^{ax}\cos^n bx \rightsquigarrow \frac{1}{a^2+b^2n^2}e^{ax}\cos^{n-1}bx\,(a\cos bx+nb\sin bx)+\frac{n(n-1)b^2}{a^2+b^2n^2}\int e^{ax}\cos^{n-2}bx$

含对数函数的积分

132 $\ln x \rightsquigarrow x(\ln x-1)$

133 $\frac{1}{x\ln x} \rightsquigarrow \ln|\ln x|$

134 $x^n \ln x \rightsquigarrow \frac{1}{n+1} x^{n+1}\left(\ln x - \frac{1}{n+1}\right)$

135 $\ln^n x \rightsquigarrow x \ln^n x - n\int \ln^{n-1} x$

136 $x^m \ln^n x \rightsquigarrow \frac{1}{m+1} x^{m+1} \ln^n x - \frac{n}{m+1}\int x^m \ln^{n-1} x$

含双曲函数的积分

137 $\sinh x \rightsquigarrow \cosh x$

138 $\cosh x \rightsquigarrow \sinh x$

139 $\tanh x \rightsquigarrow \ln\cosh x$

140 $\sinh^2 x \rightsquigarrow -\frac{x}{2} + \frac{1}{4}\sinh 2x$

141 $\cosh^2 x \rightsquigarrow \frac{x}{2} + \frac{1}{4}\sinh 2x$

含绝对值函数的积分

142 $\mathrm{sgn}(ax+b) \rightsquigarrow \frac{1}{a}|ax+b|$

143 $|ax+b|^\mu \rightsquigarrow \begin{cases} \frac{\mathrm{sgn}(ax+b)}{a(\mu+1)}|ax+b|^{\mu+1}, & \mu \neq -1 \\ \frac{\mathrm{sgn}(ax+b)}{a}\ln|ax+b|, & \mu = -1 \end{cases}$

144 $|\mathrm{f}(x)| \rightsquigarrow \mathrm{sgn}\,\mathrm{f}(x) \cdot \int \mathrm{f}(x)$ （在 $\{x \mid \mathrm{f}(x) = 0$ 且 $\mathrm{f}'(x)\int \mathrm{f}(x)\,dx \neq 0\}$公式失效，需要分段平移使得结果函数连续）

定积分

145 $\int_0^{\frac{\pi}{2}} \sin^n x = \int_0^{\frac{\pi}{2}} \cos^n x = \begin{cases} \frac{(n-1)!!}{n!!} \cdot \frac{\pi}{2}, & n\%2 = 0 \\ \frac{(n-1)!!}{n!!}, & n\%2 = 1 \end{cases}$

坐标变换

146 二维旋转变换 $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$

147 二维极坐标 $\begin{cases} x = \rho\cos\theta \\ y = \rho\sin\theta \end{cases}$，其中 $\begin{cases} \rho \in [0,+\infty) \\ \theta \in [0,2\pi) \end{cases}$，$|J| = \rho$

148 三维球坐标 $\begin{cases} x = \rho\sin\varphi\cos\theta \\ y = \rho\sin\varphi\sin\theta \\ z = \rho\cos\varphi \end{cases}$，其中 $\begin{cases} \rho \in [0,+\infty) \\ \varphi \in [0,\pi] \\ \theta \in [0,2\pi) \end{cases}$，$|J| = \rho^2\sin\varphi$

149 高维球坐标 $\begin{cases} x_1 = \rho\cos\theta_1 \\ x_2 = \rho\sin\theta_1\cos\theta_2 \\ x_3 = \rho\sin\theta_1\sin\theta_2\cos\theta_3 \\ \dots \\ x_{n-1} = \rho\sin\theta_1 \dots \sin\theta_{n-2}\cos\theta_{n-1} \\ x_n = \rho\sin\theta_1 \dots \sin\theta_{n-2}\sin\theta_{n-1} \end{cases}$，其中 $\begin{cases} \rho \in [0,+\infty) \\ \theta_1 \in [0,\pi] \\ \theta_2 \in [0,\pi] \\ \dots \\ \theta_{n-2} \in [0,\pi] \\ \theta_{n-1} \in [0,2\pi) \end{cases}$，$|J| = \rho^{n-1}\sin^{n-2}\theta_1\sin^{n-3}\theta_2 \dots \sin\theta_{n-2}$

级数

近似公式

150 $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} - \frac{571}{2488320n^4}\right)$