# Standard Code Library

Your TeamName

Your School

August 20, 2022

# Contents

# 开始

## 定义

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

typedef long long LL;
typedef __int128 LLL;
typedef unsigned u32;
typedef unsigned long long u64;
typedef long double LD;
typedef pair<int,int> pii;
typedef pair<LL,LL> pll;

#define pln putchar('\n')
#define For(i,a,b)  for(int i=(a),(i##i)=(b);i<=(i##i);++i)
#define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)

const int M=1000000007,INF=0x3f3f3f3f;
const long long INFLL=0x3f3f3f3f3f3f3f3fLL;
const int N=1000009;
```

## 取消流同步

此时混用 `iostream` 和 `cstdio` 会寄

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

## 整数读写

`read()` 读有符号数极限数据可能溢出未定义行为 `write()` 最长能写 `__int128`

```cpp
template <typename T>
bool read(T &x) {
    x = 0; char c = getchar(); int f = 1;
    while (!isdigit(c) && (c != '-') && (c != EOF)) c = getchar();
    if (c == EOF) return 0;
    if (c == '-') f = -1, c = getchar();
    while (isdigit(c)) { x = x * 10 + (c & 15); c = getchar();}
    x *= f; return 1;
}

template <typename T, typename... Args>
bool read(T &x, Args &...args) {
    bool res = 1;
    res &= read(x);
    res &= read(args...);
    return res;
}

template <typename T>
void write(T x) {
    if (x < 0) x = -x, putchar('-');
    static char sta[55];
    int top = 0;
    do sta[top++] = x % 10 + '0', x /= 10; while (x);
    while (top) putchar(sta[--top]);
}
```

# 二维计算几何

- `Point` 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 `1..n`

## 点向量基本运算

```cpp
template <typename T>
struct Point {
    T x, y;
    Point() {}
    Point(T u, T v) : x(u), y(v) {}
    Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
    Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
    Point operator*(const T &a) const { return Point(x * a, y * a); }
    T operator*(const Point &a) const { return x * a.x + y * a.y; }
    T operator%(const Point &a) const { return x * a.y - y * a.x; }
    double len() const { return hypot(x, y); }
    double operator^(const Point &a) const { return (a - (*this)).len(); }
    double angle() const { return atan2(y, x); }
    bool id() const { return y < 0 || (y == 0 && x < 0); }
    bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
};
typedef Point<double> point;

#define sqr(x) ((x) * (x))
const point O(0, 0);
const double PI(acos(-1.0)), EPS(1e-8);
inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }
```

## 位置关系

```cpp
inline bool in_same_seg(point p, point a, point b) {
    if (fabs(mul(p, a, b)) < EPS) {
        if (a.x > b.x) swap(a, b);
        return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
    } else return 0;
}

inline bool is_right(point st, point ed, point a) {
    return ((ed - st) % (a - st)) < 0;
}

inline point intersection(point s1, point t1, point s2, point t2) {
    return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
}

inline bool parallel(point a, point b, point c, point d) {
    return dcmp((b - a) % (d - c), 0);
}

inline double point2line(point p, point s, point t) {
    return fabs(mul(p, s, t) / (t - s).len());
}

inline double point2seg(point p, point s, point t) {
    return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
}
```

## 多边形

### 求多边形面积

```cpp
inline double area(int n, point s[]) {
    double res = 0;
    s[n + 1] = s[1];
    for (int i = 1; i <= n; ++i)
        res += s[i] % s[i + 1];
    return fabs(res / 2);
}
```

### 判断点是否在多边形内

- 特判边上的点
- 使用了 a[1]...a[n+1] 的数组

```
inline bool in_the_area(point p, int n, point area[]) {
    bool ans = 0; double x;
    area[n + 1] = area[1];
    for (int i = 1; i <= n; ++i) {
        point p1 = area[i], p2 = area[i + 1];
        if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
        if (p1.y == p2.y) continue;
        if (p.y < min(p1.y, p2.y)) continue;
        if (p.y >= max(p1.y, p2.y)) continue;
        ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
    }
    return ans;
}
```

## 凸包

- *Andrew* 算法
- O(n log n)
- 可以应对凸包退化成直线/单点的情况但后续旋转卡壳时应注意特判
- 注意是否应该统计凸包边上的点

```
inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }

inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
    sort(p + 1, p + 1 + n, pcmp1);
    int m = 0;
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    int k = m;
    for (int i = n - 1; i >= 1; --i) {
        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    return m - (n > 1); //返回凸包有多少个点
}
```

## 凸包直径·平面最远点对

- 旋转卡壳算法
- O(n)
- 凸包的边上只能有端点，否则不满足严格单峰
- 凸包不能退化成直线，调用前务必检查 n>=3
- 使用了 a[1]...a[n+1] 的数组

```
inline double Rotating_Caliper(int n, point a[]) {
    a[n + 1] = a[1];
    double ans = 0;
    int j = 2;
    for (int i = 1; i <= n; ++i) {
        while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
        ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
    }
    return ans;
}
```

## 平面最近点对

- 分治 + 归并

- O(n log n)

```cpp
namespace find_the_closest_pair_of_points {
    const int N = 200010; //maxn
    inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
    inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }

    point a[N], b[N];
    double ans;
    inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }

    void find(int l, int r) {
        if (l == r) return;
        if (l + 1 == r) {
            if (a[l] > a[r]) swap(a[l], a[r]);
            upd(a[l], a[r]); return;
        }
        int mid = (l + r) >> 1;
        double mx = (a[mid + 1].x + a[mid].x) / 2;
        find(l, mid); find(mid + 1, r);
        int i = l, j = mid + 1;
        for (int k = l; k <= r; ++k) b[k] = a[((j > r) || (i <= mid && a[j] > a[i])) ? (i++) : (j++)];
        for (int k = l; k <= r; ++k) a[k] = b[k];
        int tot = 0;
        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);
            b[++tot] = a[k];
        }
    }

    //接口
    inline double solve(int n, point ipt[]){
        ans = 0x3f3f3f3f3f3f3f3fll; //max distance
        for (int i = 1; i <= n; ++i) a[i] = ipt[i];
        sort(a + 1, a + 1 + n, cmp1);
        find(1, n);
        return ans;
    }
}
```

## 圆

### 三点垂心

```cpp
inline point geto(point p1, point p2, point p3) {
    double a = p2.x - p1.x;
    double b = p2.y - p1.y;
    double c = p3.x - p2.x;
    double d = p3.y - p2.y;
    double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
    double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
    return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
}
```

### 最小覆盖圆

- 随机增量 O(n)

```cpp
inline void min_circlefill(point &o, double &r, int n, point a[]) {
    mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
    o = a[1];
    r = 0;
    for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
        o = a[i];
        r = 0;
        for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
            o = (a[i] + a[j]) * 0.5;
            r = (a[i] ^ a[j]) * 0.5;
            for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
                o = geto(a[i], a[j], a[k]);
                r = (o ^ a[i]);
```

```
            }
        }
    }
}
```

# 图论

## 存图

- 前向星
- 注意边数开够

```
int Head[N], Ver[N*2], Next[N*2], Ew[N*2], Gtot=1;
inline void graphinit(int n) {Gtot=1; for(int i=1; i<=n; ++i) Head[i]=0;}
inline void edge(int u, int v, int w=1) {Ver[++Gtot]=v; Next[Gtot]=Head[u]; Ew[Gtot]=w; Head[u]=Gtot;}
#define go(i,st,to) for (int i=Head[st], to=Ver[i]; i; i=Next[i], to=Ver[i])
```

## 最短路

### Dijkstra

- 非负权图

```
namespace DIJK{//适用非负权图 满足当前 dist 最小的点一定不会再被松弛
    typedef pair<long long,int> pii;
    long long dist[N];//存最短路长度
    bool vis[N];//记录每个点是否被从队列中取出  每个点只需第一次取出时扩展
    priority_queue<pii,vector<pii>,greater<pii> >pq;//维护当前 dist[] 最小值及对应下标 小根堆

    inline void dijk(int s,int n){//s 是源点 n 是点数
        while(pq.size())pq.pop();for(int i=1;i<=n;++i)dist[i]=INFLL,vis[i]=0;//所有变量初始化
        dist[s]=0;pq.push(make_pair(0,s));
        while(pq.size()){
            int now=pq.top().second;pq.pop();
            if(vis[now])continue;vis[now]=1;
            go(i,now,to){
                const long long relx(dist[now]+Ew[i]);
                if(dist[to]>relx){dist[to]=relx;pq.push(make_pair(dist[to],to));}//松弛
            }
        }
    }
}
```

## LCA

- 倍增求 lca
- 数组开够

```
namespace LCA_Log{
    int fa[N][22],dep[N];
    int t,now;
    void dfs(int x){
        dep[x]=dep[fa[x][0]]+1;
        go(i,x,to){
            if(dep[to])continue;
            fa[to][0]=x;for(int j=1;j<=t;++j)fa[to][j]=fa[fa[to][j-1]][j-1];
            dfs(to);
        }
    }

    //初始化接口
    inline void lcainit(int n,int rt){//记得初始化全部变量
        now=1;t=0;while(now<n)++t,now<<=1;
        for(int i=1;i<=n;++i)dep[i]=0,fa[i][0]=0;
        for(int i=1;i<=t;++i)fa[rt][i]=0;
        dfs(rt);
    }

```

```
    //求 lca 接口
    inline int lca(int u,int v){
        if(dep[u]>dep[v])swap(u,v);
        for(int i=t;~i;--i)if(dep[fa[v][i]]>=dep[u])v=fa[v][i];
        if(u==v)return u;
        for(int i=t;~i;--i)if(fa[u][i]!=fa[v][i])u=fa[u][i],v=fa[v][i];
        return fa[u][0];
    }
}
```

## 连通性

### 有向图强联通分量

- tarjan $O(n)$

```
namespace SCC{
    int dfn[N],clk,low[N];
    bool ins[N];int sta[N],tot; //栈  存正在构建的强连通块
    vector<int>scc[N];int c[N],cnt;//cnt 为强联通块数 scc[i] 存放每个块内点 c[i] 为原图每个结点属于的块
    void dfs(int x){
        dfn[x]=low[x]=(++clk);//low[] 在这里初始化
        ins[x]=1;sta[++tot]=x;
        go(i,x,to){
            if(!dfn[to]){dfs(to);low[x]=min(low[x],low[to]);}//走树边
            else if(ins[to])low[x]=min(low[x],dfn[to]);//走返祖边
        }
        if(dfn[x]==low[x]){//该结点为块的代表元
            ++cnt;int u;
            do{u=sta[tot--];ins[u]=0;c[u]=cnt;scc[cnt].push_back(u);}while(x!=u);
        }
    }
    inline void tarjan(int n){//n 是点数
        for(int i=1;i<=cnt;++i)scc[i].clear();//清除上次的 scc 防止被卡 MLE
        for(int i=1;i<=n;++i)dfn[i]=ins[i]=0;tot=clk=cnt=0;//全部变量初始化
        for(int i=1;i<=n;++i)if(!dfn[i])dfs(i);
        for(int i=1;i<=n;++i)c[i]+=n;//此行 (可以省略) 便于原图上加点建新图 加新点前要初始化 Head[]=0
    }
}
```

## 二分图匹配

### 匈牙利算法求二分图无权最大匹配

- 复杂度 O(nm)

```
#include<bits/stdc++.h>
using namespace std;
namespace Hungary{
    const int N=1000009;
    //图中应储存左部到右部的边  左点编号 1..n  右点编号 1..m
    int Head[N],Ver[N*2],Next[N*2],Gtot=1;  //注意边数开够
    inline void graphinit(int n){Gtot=1;for(int i=1;i<=n;++i)Head[i]=0;}
    inline void edge(int u,int v){Ver[++Gtot]=v,Next[Gtot]=Head[u],Head[u]=Gtot;}
    #define go(i,st,to) for(int i=Head[st],to=Ver[i];i;i=Next[i],to=Ver[i]) //st 是左点, to 是 st 能到达的右点

    int match[N],vis[N];//右点的的匹配点和访问标记

    bool dfs(int x){//x 是左点
        go(i,x,to)if(!vis[to]){
            vis[to]=1;
            if(!match[to]||dfs(match[to])){
                match[to]=x;return 1;
            }
        }
        return 0;
    }

    inline int ask(int n,int m){//左点数和右点数 返回最大匹配数
        for(int i=1;i<=m;++i)match[i]=0;int res=0;
```

```
        for(int i=1;i<=n;++i){
            for(int j=1;j<=m;++j)vis[j]=0;
            res+=dfs(i);
        }
        return res;
    }
}
```

**KM 算法求二分图带权最大匹配**

- 要求该图存在完美匹配该算法将最大化完美匹配的权值和
- 复杂度 O(n^3)
- naive 的写法复杂度为 O((n^2)m) 在完全图上会退化至 O(n^4) luogu P6577

```
#include<bits/stdc++.h>
using namespace std;
namespace KM{
    const int N=509;
    const long long INFLL=0x3f3f3f3f3f3f3f3fll;//注意 INF 应足够大 至少远大于 n*n*Max{W}

    int n;//左右点数均为 n 标号均为 1..n
    int w[N][N];//邻接矩阵存边权
    bool r[N][N];//记录 i j 之间是否有边连接 完全图/非 0 权图则不必要

    inline void init(){//记得重写初始化
        int m;scanf("%d%d",&n,&m);
        for(int i=1;i<=m;++i){
            int u,v,wt;scanf("%d%d%d",&u,&v,&wt);
            w[u][v]=wt;r[u][v]=1;
        }
    }

    long long la[N],lb[N],upd[N];//左右顶标 每个右点对应的最小 delta 要开 longlong
    int last[N];//每个右点对应的回溯右点
    bool va[N],vb[N];
    int match[N];//每个右点对应的左匹配点
    bool dfs(int x,int fa){
        va[x]=1;
        for(int y=1;y<=n;++y)if(r[x][y]&&!vb[y]){
            const long long dt=la[x]+lb[y]-w[x][y];
            if(dt==0){//相等子图
                vb[y]=1;last[y]=fa;
                if(!match[y]||dfs(match[y],y)){
                    match[y]=x;
                    return 1;
                }
            }else if(upd[y]>dt){//下次 dfs 直接从最小 delta 处开始
                upd[y]=dt;
                last[y]=fa;//用 last 回溯该右点的上一个右点以更新增广路
            }
        }
        return 0;
    }

    inline void KM(){
        for(int i=1;i<=n;++i){//初始化顶标
            la[i]=-INFLL;lb[i]=0;
            for(int j=1;j<=n;++j)if(r[i][j])la[i]=max(la[i],(long long)w[i][j]);
        }
        for(int i=1;i<=n;++i){//尝试给每一个左点匹配上右点 匹配失败则扩展相等子图重试至成功
            memset(va,0,sizeof va);//注意复杂度
            memset(vb,0,sizeof vb);
            memset(last,0,sizeof last);
            memset(upd,0x3f,sizeof upd);
            int st=0;match[0]=i;//给起点 i 连一个虚右点 标号为 0
            //不断尝试将右点 st 从已有的匹配中解放 以获得增广路
            while(match[st]){//当 st 到达非匹配右点直接退出
                long long delta=INFLL;
                if(dfs(match[st],st))break;//st 的左点匹配到了新的右点则退出
```

```
                for(int j=1;j<=n;++j)
                    if(!vb[j]&&upd[j]<delta){//下次从最小的 delta 处开始 DFS
                        delta=upd[j];
                        st=j;
                    }
                for(int j=1;j<=n;++j){//将交错树上的左顶标加 delta 右顶标减 delta 使更多的边转为相等边
                    if(va[j])la[j]-=delta;
                    if(vb[j])lb[j]+=delta;
                        else upd[j]-=delta;//小问题: 这里在干啥 每次修改顶标后重新计算 upd 是否可行
                }
                vb[st]=1;
            }
            while(st){//更新增广路
                match[st]=match[last[st]];
                st=last[st];
            }
        }

        long long ans=0;
        for(int i=1;i<=n;++i)ans+=w[match[i]][i];
        printf("%lld\n",ans);
        for(int i=1;i<=n;++i)printf("%d%c",match[i]," \n"[i==n]);
    }
    //signed main(){init();KM();return 0;}
}
```

# 数据结构

## STL

### 小跟堆

```
priority_queue<vector<int>, int, greater<int> > q;
```

### 整数哈希

```
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
using namespace std;
typedef unsigned long long u64;

struct neal {
    static u64 A(u64 x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    u64 operator()(u64 x) const{
        static const u64 C = chrono::steady_clock::now().time_since_epoch().count();
        return A(x + C);
    }
};

unordered_map<int, int, neal> HASH1;

__gnu_pbds::gp_hash_table<int, int, neal> HASH2;
```

## 二维树状数组

- LOJ135 区间修改 + 区间查询单次操作复杂度 log^2 空间复杂度 N^2
- 对原数组差分 S(n,m) = ∑∑A_ij 用树状数组维护差分数组 A_ij 转化为单点修改
- 区间查询 SS(n,m) = ∑∑S(i,j) 推式子转化为单点查询问题
- SS(n,m) = ∑∑∑∑A_ij = (n+1)*(m+1)*∑∑A_ij - (n+1)*∑∑jA_ij - (m+1)*∑∑iA_ij + ∑∑ijA_ij

```cpp
//省略了文件头
const int N=2051;
int n,m;
struct dat{
    LL c,cx,cy,cxy;
    void operator+=(const dat&a){c+=a.c;cx+=a.cx;cy+=a.cy;cxy+=a.cxy;}
}c[N][N];

inline void add(int a,int b,int k){
    const dat d={k,k*a,k*b,k*a*b};
    for(int i=a;i<=n;i+=(i&(-i)))
    for(int j=b;j<=m;j+=(j&(-j)))c[i][j]+=d;
}

inline LL f(int a,int b){
    dat r={0,0,0,0};
    for(int i=a;i>0;i-=(i&(-i)))
    for(int j=b;j>0;j-=(j&(-j)))r+=c[i][j];
    return (a+1)*(b+1)*r.c-(b+1)*r.cx-(a+1)*r.cy+r.cxy;
}

signed main(){
    read(n,m);
    int op;while(read(op))if(op==1){
        int a,b,c,d,k;read(a,b,c,d,k);
        add(a,b,k);add(c+1,b,-k);add(a,d+1,-k);add(c+1,d+1,k);
    }else{
        int a,b,c,d;read(a,b,c,d);
        printf("%lld\n",f(c,d)-f(a-1,d)-f(c,b-1)+f(a-1,b-1));
    }
    return 0;
}
```

### 堆式线段树

- 区间求和区间修改
- 空间估算
- 所有数组务必初始化

```cpp
struct SegmentTree_Heap{
    #define TreeLen (N<<2)          //N
    #define lc(x)   ((x)<<1)
    #define rc(x)   ((x)<<1|1)
    #define sum(x)  (tr[x].sum)     //
    #define t(x)    (t[x])          //

    struct dat{
        LL sum;
        /* 满足结合律的基本运算 用于合并区间信息 */
        dat operator+(const dat&brother){
            dat result;
            result.sum=sum+brother.sum;
            return result;
        }
    }tr[TreeLen];
    LL t[TreeLen];  //lazy tag

    /* 单区间修改 */
    inline void change(const int&x,const int&l,const int&r,const LL&d){
        tr[x].sum=tr[x].sum+d*(r-l+1);
        t[x]=t[x]+d;
    }

    inline void pushup(int x){tr[x]=tr[lc(x)]+tr[rc(x)];}

    inline void pushdown(int x,const int&l,const int&r,const int&mid){
        if(t(x)){ // 区间修改注意细节
            change(lc(x),l,mid,t(x));
            change(rc(x),mid+1,r,t(x));
            t(x)=0;
```

```
        }
    }

    void build(int x,int l,int r){
        t(x)=0;   // 记得初始化
        if(l==r){
            sum(x)=0;
            return;
        }
        int mid=(l+r)>>1;
        build(lc(x),l,mid);
        build(rc(x),mid+1,r);
        pushup(x);
    }

    void add(int x,int l,int r,const int&L,const int&R,const LL&d){
        if(L<=l&&r<=R){
            change(x,l,r,d);
            return;
        }
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        if(L<=mid)add(lc(x),l,mid,L,R,d);
        if(R>mid)add(rc(x),mid+1,r,L,R,d);
        pushup(x);
    }

    LL ask(int x,int l,int r,const int&L,const int&R){
        if(L<=l&&r<=R)return sum(x);
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        LL res=0;
        if(L<=mid)res=(res+ask(lc(x),l,mid,L,R));
        if(mid<R)res=(res+ask(rc(x),mid+1,r,L,R));
        return res;
    }
};
```

## 小根堆

```
namespace MyPQ{
    typedef int pqdat;    ////
    pqdat q[N];
    int tot;
    void up(int x){
        while(x>1)
        if(q[x]<q[x/2]){
            swap(q[x],q[x/2]);
            x/=2;
        }else return;
    }
    void down(int x){
        int ls=x*2;
        while(ls<=tot){
            if(ls<tot&&q[ls+1]<q[ls])++ls;
            if(q[ls]<q[x]){
                swap(q[x],q[ls]);x=ls;ls=x*2;
            }else return;
        }
    }
    void push(pqdat x){q[++tot]=x;up(tot);}
    pqdat top(){return q[1];}
    void pop(){if(!tot)return;q[1]=q[tot--];down(1);}
    void pop(int k){if(!tot)return;q[k]=q[tot--];up(k);down(k);}
}
```

## Treap

```
#include<bits/stdc++.h>
using namespace std;
const int N=2000007,INF=(1ll<<30)+7;

```

```cpp
//Treap 维护升序多重集
//支持操作: 数 <-> 排名 查询某数前驱后继
//操作数 x 可以不在集合中
//x 的排名: 集合中 <x 的数的个数 +1
//排名 x 的数: 集合中排名 <=x 的数中的最大数
//x 的前驱 比 x 小的最大数

struct treap{//所有点值不同 用副本数实现多重集
    int l,r;
    int v,w;//v 是数据 w 是维护堆的随机值
    int num,sz;//num 是该点副本数 sz 是该子树副本总数
}tr[N];int tot,rt;//tr[0] 始终全 0 使用范围 tr[1..n]
#define lc(x) tr[x].l
#define rc(x) tr[x].r
#define sz(x) tr[x].sz
#define num(x) tr[x].num
#define val(x) tr[x].v
#define wt(x) tr[x].w

inline int New(int x){
    val(++tot)=x; wt(tot)=rand();
    num(tot)=sz(tot)=1; return tot;
}

inline void upd(int p){sz(p)=sz(lc(p))+sz(rc(p))+num(p);}

inline void build(){//初始化 INF 和-INF 两个点
    srand(time(0));
    rt=1;tot=2;
    rc(1)=2;val(1)=-INF;wt(1)=rand();num(1)=1;sz(1)=2;
    val(2)=INF;wt(2)=rand();num(2)=1;sz(2)=1;
}

//调用时记得减一 askrk(rt,x)-1
int askrk(int p,int x){//当前子树中查询 x 的排名
    if(p==0)return 1;//说明某子树所有数均比 x 大
    if(x==val(p))return sz(lc(p))+1;
    return x<val(p)?askrk(lc(p),x):askrk(rc(p),x)+sz(lc(p))+num(p);
}

//调用时记得加一 kth(rt,++rank)
int kth(int p,int rk){//当前子树中查询排名 rk 的数
    if(p==0)return INF;//说明集合大小 <rk
    if(sz(lc(p))>=rk)return kth(lc(p),rk);
    rk-=sz(lc(p))+num(p);
    return (rk>0)?kth(rc(p),rk):val(p);
}

inline void zig(int &p){//与左子节点交换位置
    int q=lc(p);lc(p)=rc(q);rc(q)=p;
    upd(p);p=q;upd(p);
}

inline void zag(int &p){//与右子节点交换位置
    int q=rc(p);rc(p)=lc(q);lc(q)=p;
    upd(p);p=q;upd(p);
}

//insert(rt,x)
void insert(int &p,int x){//当前子树中插入 x
    if(p==0){p=New(x);return;}//x 首次插入
    if(x==val(p)){++num(p);++sz(p);return;}
    if(x<val(p)){
        insert(lc(p),x);
        if(wt(p)<wt(lc(p)))zig(p);//维护大根堆
    }else{
        insert(rc(p),x);
        if(wt(p)<wt(rc(p)))zag(p);//维护大根堆
    }
    upd(p);
}
```

```cpp

//erase(rt,x)
void erase(int &p,int x){//当前子树中删除一个 x
    if(p==0)return;//已经无需删除
    if(val(p)==x){//如果找到了 x 的位置
        if(num(p)>1){//无需删点
            --num(p);--sz(p);return;//如果有多个 x 维护副本数即可
        }
        if(lc(p)||rc(p)){//该点不是叶子节点 则不断向下调整至叶子节点
            if(rc(p)==0||wt(lc(p))>wt(rc(p)))zig(p),erase(rc(p),x);//由于 rand() 的值域 & 大根堆的实现 故省略左子树为空的判断
            else zag(p),erase(lc(p),x);
            upd(p);
        }else p=0;//是叶子节点则直接删除
        return;
    }
    x<val(p)?erase(lc(p),x):erase(rc(p),x);upd(p);
}

int askpre(int x){
    int id=1;//-INF 若没有前驱则返回-INF
    //尝试自顶向下寻找 x 则 x 的前驱有两种情况
    //1) 未找到 x 或 x 没有左子树 则前驱在搜索路径上
    //2) 前驱是 x 的左子树中最大值 即 x 的左子树一直向右走
    int p=rt;
    while(p){
        if(x==val(p)){//找到 x
            if(lc(p)){p=lc(p);while(rc(p))p=rc(p);id=p;}
            break;
        }
        if(val(p)<x&&val(p)>val(id))id=p;//每经过一个点尝试更新前驱
        p=(val(p)>x?lc(p):rc(p));//找 x
    }
    return val(id);
}

int asknxt(int x){
    int id=2;//INF
    int p=rt;
    while(p){
        if(x==val(p)){
            if(rc(p)){p=rc(p);while(lc(p))p=lc(p);id=p;}
            break;
        }
        if(val(p)>x&&val(p)<val(id))id=p;
        p=(val(p)>x?lc(p):rc(p));
    }
    return val(id);
}
```

# 字符串

## KMP

```cpp
//luogu P3375
#include <bits/stdc++.h>
using namespace std;
const int N = 1000009;

char s1[N], s2[N];
int fail[N], n, m;

signed main() {
    scanf("%s%s", s1 + 1, s2 + 1);
    n = strlen(s1 + 1);
    m = strlen(s2 + 1);

    //fail[1] = 0;
    for (int i = 2, j = 0; i <= m; ++i) {
        while (j != 0 && s2[j + 1] != s2[i]) j = fail[j];
        if (s2[j + 1] == s2[i]) ++j;
```

```
        fail[i] = j;
    }

    int p = 0;
    for (int i = 1; i <= n; ++i) {
        while (p != 0 && s2[p + 1] != s1[i]) p = fail[p];
        if (s2[p + 1] == s1[i]) ++p;

        if (p == m) {
            printf("%d\n", i - p + 1);
            p = fail[p];
        }
    }

    for (int i = 1; i <= m; ++i) printf("%d ", fail[i]);
    return 0;
}
```

## 扩展 KMP(Z 函数)

```
//luogu 5410
#include <bits/stdc++.h>
using namespace std;
const int N = 20000009;

char t[N], p[N];
int n, m, z[N], mc[N];

signed main() {
    scanf("%s%s", t + 1, p + 1);
    n = strlen(t + 1);
    m = strlen(p + 1);

    z[1] = m;
    for (int i = 2, l = 0, r = 0; i <= m; ++i) {
        z[i] = ((r < i) ? 0 : min(r - i + 1, z[i - l + 1]));
        while (z[i] <= m - i && p[i + z[i]] == p[z[i] + 1]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    for (int i = 1, l = 0, r = 0; i <= n; ++i) {
        mc[i] = ((r < i) ? 0 : min(r - i + 1, z[i - l + 1]));
        while (mc[i] <= n - i && mc[i] < m && t[i + mc[i]] == p[mc[i] + 1])
            ++mc[i];
        if (i + mc[i] - 1 > r) l = i, r = i + mc[i] - 1;
    }

    unsigned long long u = 0, v = 0;
    for (int i = 1; i <= m; ++i) u ^= (i * (z[i] + 1ull));
    for (int i = 1; i <= n; ++i) v ^= (i * (mc[i] + 1ull));
    printf("%llu\n%llu\n", u, v);
    return 0;
}
```

## AC 自动机

```
#include<bits/stdc++.h>
using namespace std;
const int N=1000009;

//洛谷 P3808 AC 自动机 (简单版)
int n;
char s[N];
struct AC{
    struct dat{
        int tp[26],fail,ed;
        void init(){memset(tp,0,sizeof tp);fail=ed=0;}
    }tr[N];int tot;
    void init(){for(int i=0;i<=tot;++i)tr[i].init();tot=0;}

```

```
    void insert(char*s){//C 风格字符串 下标从 1 开始
        int p=0;
        for(int i=1;s[i]!=0;++i){//C 风格字符串 下标从 1 开始
            int&tp=tr[p].tp[s[i]-'a'];
            if(tp==0)tp=(++tot);
            p=tp;
        }
        ++tr[p].ed;
    }

    void build(){//先 insert 所有模式串 然后 build
        queue<int>q;
        for(int i=0;i<26;++i)if(tr[0].tp[i])q.push(tr[0].tp[i]);
        while(q.size()){
            dat&now=tr[q.front()];q.pop();
            for(int i=0;i<26;++i){
                int&tp=now.tp[i];
                if(tp)tr[tp].fail=tr[now.fail].tp[i],q.push(tp);
                else tp=tr[now.fail].tp[i];//路径压缩
            }
        }
    }

    int ask(char*s){//C 风格字符串 下标从 1 开始
        int p=0,res=0;
        for(int i=1;s[i]!=0;++i){//C 风格字符串 下标从 1 开始
            p=tr[p].tp[s[i]-'a'];
            for(int j=p;tr[j].ed!=-1;j=tr[j].fail)res+=tr[j].ed,tr[j].ed=-1;
        }
        return res;
    }
}ac;

signed main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%s",s+1);
        ac.insert(s);
    }
    ac.build();
    scanf("%s",s+1);
    printf("%d\n",ac.ask(s));
    return 0;
}
```

# 杂项

## 整数哈希

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//https://codeforces.com/blog/entry/62393

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
```

```
};

unordered_map<long long, int, custom_hash> safe_map;
gp_hash_table<long long, int, custom_hash> safe_hash_table;
```

### GCC 位运算

需要 gcc 编译器

- `int __builtin_ffs (unsigned int x)` 返回 x 的最后一位 1 的是从后向前第几位，比如 7368（1110011001000）返回 4。
- `int __builtin_clz (unsigned int x)` 返回前导的 0 的个数。
- `int __builtin_ctz (unsigned int x)` 返回后面的 0 个个数，和 `__builtin_clz` 相对。
- `int __builtin_popcount (unsigned int x)` 返回二进制表示中 1 的个数。
- `int __builtin_parity (unsigned int x)` 返回 x 的奇偶校验位，也就是 x 的 1 的个数模 2 的结果。

此外，这些函数都有相应的 `usigned long` 和 `usigned long long` 版本，只需要在函数名后面加上 l 或 ll 就可以了，比如 `int __builtin_clzll`。

来源 https://blog.csdn.net/yuer158462008/article/details/46383635

### 对拍

for windows

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

//输入文件   input.in
//正确的程序名    ac.cpp   ac.exe   ac.out
//待验证的程序名   wa.cpp   wa.exe   wa.out

void gen(){//生成 INPUT 内的数据
    FILE *op=fopen("input.in","w");
    //mt19937 RAND(time(nullptr));
    //uniform_int_distribution<int>(1,100)(RAND);//范围内均匀随机整数
    //fprintf(op,"%d\n",233);
    fclose(op);
}

bool check(){//比较 ac.out 和 wa.out 的内容   一致则返回 0   否则返回非 0
    return system("fc wa.out ac.out");
}

int main(){
    system("g++ wa.cpp -o wa.exe");
    system("g++ ac.cpp -o ac.exe");
    while(1){
        gen(); int st,ed;

        st=clock();
        system("ac.exe < input.in > ac.out");
        ed=clock();
        printf("ac cost %lld ms\n",(ed-st)*1000ll/CLOCKS_PER_SEC);

        st=clock();
        system("wa.exe < input.in > wa.out");
        ed=clock();
        printf("wa cost %lld ms\n",(ed-st)*1000ll/CLOCKS_PER_SEC);

        if(check()){
            puts("Wrong Answer");
            system("pause");
            return 0;
        }
    }
```

```
    return 0;
}
```

## fread/fwrite 实现的读写类

没用

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

struct MY_IO {
    #define MAXSIZE (1 << 20) //缓冲区大小 不小于 1 << 14
    inline bool isdigit(const char &x) {return x >= '0' && x <= '9';} //字符集 看情况改
    inline bool blank(const char &c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }

    char buf[MAXSIZE + 1], *p1, *p2, pbuf[MAXSIZE + 1], *pp;
    MY_IO() : p1(buf), p2(buf), pp(pbuf) {}
    ~MY_IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }

    char gc() {
        if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
        return p1 == p2 ? EOF : *p1++;
    }

    void pc(const char &c) {
        if (pp - pbuf == MAXSIZE) fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
    }

    template <typename T>
    bool read(T &x) {
        x = 0; char c = gc(); int f = 1;
        while (!isdigit(c) && (c != '-') && (c != EOF)) c = gc();
        if (c == EOF) return 0;
        if (c == '-') f = -1, c = gc();
        while (isdigit(c)) { x = x * 10 + (c & 15); c = gc();}
        x *= f; return 1;
    }

    template <typename T, typename... Args>
    bool read(T &x, Args &...args) {
        bool res = 1;
        res &= read(x);
        res &= read(args...);
        return res;
    }

    int gets(char *s) {
        char c = gc();
        while (blank(c) && c != EOF) c = gc();
        if (c == EOF) return 0;
        int len = 0;
        while (!blank(c) && c != EOF) *s++ = c, c = gc(), ++len;
        *s = 0; return len;
    }

    void getc(char &c) { for (c = gc(); blank(c) && c != EOF; c = gc());}

    template <typename T>
    void write(T x) {
        if (x < 0) x = -x, putchar('-');
        static char sta[55];
        int top = 0;
        do sta[top++] = x % 10 + '0', x /= 10; while (x);
        while (top) putchar(sta[--top]);
    }

    template <typename T>
    void write(T x, const char &Lastchar) {write(x); pc(Lastchar);}

    void puts(char *s) {while ((*s) != 0)pc(*s++);}
```

```
65
66      int getline(char *s){
67          char c = gc(); int len = 0;
68          while (c != '\n' && c != EOF)*s++ = c, c = gc(), ++len;
69          *s = 0; return len;
70      }
71
72      void putline(char *s){ while ((*s) != 0) pc(*s++); pc('\n');}
73  } IO;
74
75  #define read  IO.read    //读一个/多个整数
76  #define write IO.write   //写一个整数
77  #define gc    IO.gc      //getchar()
78  #define pc    IO.pc      //putchar()
79  #define gets  IO.gets    //读一个指定字符集的 C 风格字符串 到 s[0..len-1] 返回 len
80  #define getc  IO.getc    //读一个指定字符集的字符
81  #define puts  IO.puts    //写一个 C 风格字符串
82  #define getl  IO.getline //读一行
83  #define putl  IO.putline //写一个 C 风格字符串并换行
```