

准备

Head

```
#include <bits/stdc++.h>
#define SZ(x) (int)(x).size()
#define ALL(x) (x).begin(),(x).end()
#define PB push_back
#define EB emplace_back
#define MP make_pair
#define FI first
#define SE second
using namespace std;
typedef double DB;
typedef long double LD;
typedef long long LL;
typedef unsigned long long ULL;
typedef pair<int,int> PII;
typedef vector<int> VI;
typedef vector<LL> VLL;
typedef vector<PII> VPII;
// head
```

快速读入

```
template<typename T>
bool read(T &t){
    static char ch;
    int f=1;
    while(ch!=EOF&&!isdigit(ch)) {
        if(ch=='-') f=-1;
        ch=getchar();
    }
    if(ch==EOF) return false;
    for(t=0;isdigit(ch);ch=getchar()) t=t*10+ch-'0';
    t*=f;
    return true;
}

template<typename T>
void print(T t) {
    static int stk[70],top;
    if(t==0) {putchar('0');return;}
    if(t<0) {t=-t;putchar('-');}
    while(t) stk[++top]=t%10,t/=10;
    while(top) putchar(stk[top--]+'0');
}
```

对拍

Windows

```
@echo off
:loop
    rand.exe>data.in
    std.exe<data.in>std.out
    my.exe<data.in>my.out
    fc my.out std.out
    if not errorlevel 1 goto loop
pause
```

Linux

```
cs=1
while true
do
    cs=$((cs+1))
    echo "Case $cs:"
    ./data>data.in
    ./my<data.in>my.out
    ./std<data.in>std.out
    if diff my.out std.out; then
        echo "AC\n"
    else
        echo "WA\n"
        exit 0
    fi
done
```

字符串

最小表示法

找出与一个字符串循环同构的串中字典序最小的串

考虑对于一对字符串 A, B ，它们在原字符串 S 中的起始位置分别为 i, j ，且它们的前 k 个字符均相同，即：

$$A[i \cdots i + k - 1] = B[j \cdots j + k - 1]$$

不妨先考虑 $A[i + k] > B[j + k]$ 的情况

我们发现起始位置下标 l 满足 $i \leq l \leq i + k$ 的字符串均不能成为答案。因为对于任意一个字符串 A_{i+p} （表示以 $i + p$ 为起始位置的字符串）一定存在字符串 B_{j+p} 比它更优

所以我们比较时可以跳过后下标 $l \in [i, i + k]$ ，直接比较 A_{i+k+1}

时间复杂度： $O(n)$

```

int get_min(const string &s) {
    int k=0,i=0,j=1,len=SZ(s);
    while(k<len&&i<len&&j<len) {
        if(s[(i+k)%len]==s[(j+k)%len]) ++k;
        else {
            s[(i+k)%len]>s[(j+k)%len]?i=i+k+1:j=j+k+1;
            if(i==j) ++i;
            k=0;
        }
    }
    return min(i,j);
}

```

KMP

- 字符串下标从 0 开始，前 i 位的最小循环节长度为 $i - next_i$ ，若 $i \mid (i - next_i)$ ，则前 i 位的循环周期为 $i / (i - next_i)$ 。

```

void get_next(const string &t) { // next 数组表示 0 ~ i - 1 的最长 border
    int i=0,j=-1;
    nxt[0]=-1;
    while(i<SZ(t)) {
        if(j==-1||t[i]==t[j]) nxt[++i]=++j;
        else j=nxt[j];
    }
}

```

- 字符串匹配

```

// 优化后的 next 不再是最长 border
void get_next(const string &t) {
    int i=0,j=-1;
    nxt[0]=-1;
    while(i<SZ(t)) {
        if(j==-1||t[i]==t[j]) {
            ++i,++j;
            if(t[i]!=t[j]) nxt[i]=j;
            else nxt[i]=nxt[j];
        }
        else j=nxt[j];
    }
}

void kmp(const string &s,const string &t) {
    get_next(t);
    int i=0,j=0;
    while(i<SZ(s)) {
        if(j==-1||s[i]==t[j]) {
            ++i,++j;
            if(j==SZ(t)) cout<<i-j+1<<'\n',j=nxt[j]; // 匹配成功
        }
        else j=nxt[j];
    }
}

```

扩展 KMP

- 求 S 的每一个后缀与 T 的最长公共前缀

```
int nxt[N]; // 以 T[i] 开始的子串与 T 的最长相同前缀长度
int extend[N]; // 以 S[i] 开始的子串与 T 的最长相同前缀长度
void get_next(const string &t) {
    int l=0,r=0; // l 为当前最长匹配起点, r 为当前最长匹配终点
    int len=t.length();
    nxt[0]=len;
    for(int i=1;i<len;i++) {
        if(i>r||i+nxt[i-1]>=r) {
            if(i>r) r=i;
            while(r<len&&t[r]==t[r-i]) r++; // 重新暴力匹配更新 r
            nxt[i]=r-i;
            l=i;
        }
        else nxt[i]=nxt[i-1];
    }
}
void get_extend(const string &s,const string &t) {
    get_next(t);
    int l=0,r=0;
    int len_s=s.length(),len_t=t.length();
    for(int i=0;i<len_s;i++) {
        if(i>r||i+nxt[i-1]>=r) {
            if(i>r) r=i;
            while(r<len_s&&r-i<len_t&&s[r]==t[r-i]) r++;
            extend[i]=r-i;
            l=i;
        }
        else extend[i]=nxt[i-1];
    }
}
```

Manacher

```
char t[N<<1];
int len[N<<1];
int manacher(const string &s) {
    int k=0,mid=0,r=0,res=0;
    t[0]='$';
    for(int i=0;s[i];i++) t[++k]='#',t[++k]=s[i];
    t[++k]='#';
    for(int i=1;i<=k;i++) {
        len[i]=i<r?min(r-i,len[2*mid-i]):1;
        while(i-len[i]>=1&&i+len[i]<=k&&t[i-len[i]]==t[i+len[i]]) ++len[i];
        if(len[i]+i>r) {
            r=len[i]+i;
            mid=i;
            res=max(res,len[i]); // 更新id回文串中点和mx回文串最右端
        }
    }
    return res-1; // 返回最大回文串长度
}
```

Trie 树

```

int tr[N][26], tot[N], sz;
void insert(const string &s) {
    int u=0;
    for(auto c:s) {
        int v=c-'a';
        if(!tr[u][v]) tr[u][v]=++sz;
        u=tr[u][v];
    }
    tot[u]++;
}
int find(const string &s) {
    int u=0;
    for(auto c:s) {
        int v=c-'a';
        if(!tr[u][v]) return 0;
        u=tr[u][v];
    }
    return tot[u]; // 返回该字符串的个数
}

```

AC 自动机

- 多组数据字符集大小不一样时，初始化按最大字符集大小处理

```

struct AC {
    int tr[N][26], fail[N], sz; // fail 指针指向与当前前缀匹配的最长后缀的位置
    void init() {
        for(int i=0; i<=sz; i++) {
            for(int j=0; j<26; j++) tr[i][j]=0;
            fail[i]=0;
        }
        sz=0;
    }
    void insert(const string &s) {
        int u=0;
        for(auto c:s) {
            int v=c-'a';
            if(!tr[u][v]) tr[u][v]=++sz;
            u=tr[u][v];
        }
    }
    void build() {
        queue<int> q;
        for(int v=0; v<26; v++) if(tr[0][v]) q.push(tr[0][v]);
        while(!q.empty()) {
            int u=q.front();
            q.pop();
            for(int v=0; v<26; v++) {
                if(tr[u][v]) fail[tr[u][v]]=tr[fail[u]][v], q.push(tr[u][v]);
                else tr[u][v]=tr[fail[u]][v];
            }
        }
    }
} ac;

```

Hash 字符串

```
// 双 Hash
const LL HASH_MOD[2]={1000000007,1000000009};
const LL HASH_BASE=13331;
LL ha[2][N],power[2][N];
void hash_init(const string &s) {
    power[0][0]=power[1][0]=1;
    for(int i=1;i<=SZ(s);i++) { // ha[0/1][1] 存 s[0] 的 Hash 值
        ha[0][i]=(ha[0][i-1]*HASH_BASE%HASH_MOD[0]+s[i-1])%HASH_MOD[0];
        ha[1][i]=(ha[1][i-1]*HASH_BASE%HASH_MOD[1]+s[i-1])%HASH_MOD[1];
        power[0][i]=power[0][i-1]*HASH_BASE%HASH_MOD[0];
        power[1][i]=power[1][i-1]*HASH_BASE%HASH_MOD[1];
    }
}
pair<LL,LL> get_hash(int l,int r) {
    return MP(
        (ha[0][r]+HASH_MOD[0]-ha[0][l-1]*power[0][r-
l+1])%HASH_MOD[0])%HASH_MOD[0],
        (ha[1][r]+HASH_MOD[1]-ha[1][l-1]*power[1][r-
l+1])%HASH_MOD[1])%HASH_MOD[1]
    );
}
```

后缀数组

- 两子串最长公共前缀: $lcp(sa_i, sa_j) = \min\{height_{i+1\dots j}\}$
- 两个子串的大小关系:
假设需要比较的是 $A = S_{a\dots b}$ 和 $B = S_{c\dots d}$ 的大小关系
若 $lcp(a, c) \geq \min(|A|, |B|)$, $A < b \iff |A| < |B|$
否则, $A < b \iff rk_a < rk_b$
- 不同子串的数目: $\frac{n(n+1)}{2} - \sum_{i=2}^n height_i$
- 合并多个字符串中间用没有出现过的字符隔开
- 多组数据时, 注意初始化 $s_{n+1} = 0$

```
int n,s[N],sa[N],id[N],rk[N],oldrk[N<<1],cnt[N],height[N];
void DA() {
    int m=200; // 初始最大排名
    for(int i=1;i<=m;i++) cnt[i]=0;
    for(int i=1;i<=n;i++) ++cnt[rk[i]=s[i]];
    for(int i=1;i<=m;i++) cnt[i]+=cnt[i-1];
    for(int i=n;i>=1;i--) sa[cnt[rk[i]]--]=i;
    for(int w=1,p;w<=n;w<=1,m=p) {
        p=0;
        for(int i=n;i>=n-w;i--) id[++p]=i;
        for(int i=1;i<=n;i++) if(sa[i]>w) id[++p]=sa[i]-w;
        for(int i=1;i<=m;i++) cnt[i]=0;
        for(int i=1;i<=n;i++) ++cnt[rk[id[i]]];
        for(int i=1;i<=m;i++) cnt[i]+=cnt[i-1];
        for(int i=n;i>=1;i--) sa[cnt[rk[id[i]]]--]=id[i];
        for(int i=1;i<=n;i++) oldrk[i]=rk[i];
        p=0;
        for(int i=1;i<=n;i++)
```

```

        rk[sa[i]]=(oldrk[sa[i]]==oldrk[sa[i-1]]&&oldrk[sa[i]+w]==oldrk[sa[i-1]+w])?p:++p;
    }
}
void get_height() {
    for(int i=1,t=0;i<=n;i++) {
        if(t) t--;
        while(s[i+t]==s[sa[rk[i]-1]+t]) t++;
        height[rk[i]]=t;
    }
}
bool check(int k) { // 是否有某长度为 k 的子串至少不重叠地出现了 2 次
    int mn=0,mx=0;
    for(int i=1;i<=n;i++) {
        if(height[i]<k) mn=sa[i],mx=sa[i];
        else {
            mn=min(mn,sa[i]);
            mx=max(mx,sa[i]);
            if(mx-mn>=k) return true;
        }
    }
    return false;
}
}

```

后缀自动机

- 字符集过大，使用 map
- 不同子串个数： $\sum_{i=1}^{sz} (len_i - len_{link_i})$
可以在线计算，每插入一个字符的贡献是： $len_{last} - len_{link_{last}}$
- 不同子串的总长度： $\sum_{i=1}^{sz} (\frac{len_i(len_i+1)}{2} - \frac{len_{link_i}(len_{link_i}+1)}{2})$
可以在线计算，每插入一个字符的贡献是： $\frac{len_{last}(len_{last}+1)}{2} - \frac{len_{link_{last}}(len_{link_{last}}+1)}{2}$
- 出现次数：将不是复制创建的非初始状态的 cnt 初始化为 1，再让 $cnt_{link_v} + = cnt_v$
- 字典序第 k 小子串：根据 nxt 处理子树大小，若求的是去重后的第 k 小，让子树大小初始值为 1，否则为该状态出现次数，然后 dfs 求出第 k 小

```

int n,sz,last,len[N<<1],link[N<<1],nxt[N<<1][26],tot[N<<1]; // tot 是每个状态出现的次数
void sam_init() {
    link[0]=-1;
}
void sam_insert(int x) {
    int cur=++sz,p=last;
    len[cur]=len[p]+1;
    tot[cur]=1;
    while(p!=-1&&!nxt[p][x]) nxt[p][x]=cur,p=link[p];
    if(p==-1) link[cur]=0;
    else {
        int q=nxt[p][x];
        if(len[p]+1==len[q]) link[cur]=q;
        else {
            int t=++sz;
            len[t]=len[p]+1;
            for(int i=0;i<26;i++) nxt[t][i]=nxt[q][i];

```

```

        link[t]=link[q];
        while(p!=-1&&nxt[p][x]==q) nxt[p][x]=t,p=link[p];
        link[q]=link[cur]=t;
    }
}
last=cur;
}
int c[N],a[N<<1]; // a 是拓扑次序
void topo(int t) {
    for(int i=1;i<=n;i++) c[i]=0;
    for(int i=1;i<=sz;i++) ++c[len[i]];
    for(int i=1;i<=n;i++) c[i]+=c[i-1];
    for(int i=sz;i>=1;i--) a[c[len[i]]--]=i;
    for(int i=sz;i>=1;i--) tot[link[a[i]]]+=tot[a[i]];
}
string lcs(const string &S,const string &T) { // 两个字符串的最长公共子串
    n=SZ(S);
    sam_init();
    for(int i=0;i<n;i++) sam_insert(S[i]);
    int v=0,l=0,mx=0,mxpos=0;
    for(int i=0;i<SZ(T);i++) {
        while(v&&!nxt[v].count(T[i])) v=link[v],l=len[v];
        if(nxt[v].count(T[i])) v=nxt[v][T[i]],++l;
        if(l>mx) mx=l,mxpos=i;
    }
    return T.substr(mxpos-mx+1,mx);
}

```

广义后缀自动机

- 多个字符串的最长公共子串长度：记录每个状态是否在某个字符串中存在，通过 link 转移

```

// N 为单个字符串最大长度
// M 为字符串个数
int n,sz,len[N*M*2],nxt[N*M*2][26],link[N*M*2];
void trie_insert(const string &s) {
    n=max(n,SZ(s));
    int rt=0;
    for(auto c:s) {
        int now=c-'a';
        if(!nxt[rt][now]) nxt[rt][now]=++sz;
        rt=nxt[rt][now];
    }
}
int gsa_insert(int last,int x) {
    int cur=nxt[last][x],p=link[last];
    len[cur]=len[last]+1;
    while(p!=-1&&!nxt[p][x]) nxt[p][x]=cur,p=link[p];
    if(p==-1) {link[cur]=0;return cur;}
    int q=nxt[p][x];
    if(len[p]+1==len[q]) {link[cur]=q;return cur;}
    int t=++sz;
    len[t]=len[p]+1;
    for(int i=0;i<26;i++) nxt[t][i]=len[nxt[q][i]]!=0?nxt[q][i]:0;
    link[t]=link[q];
    while(p!=-1&&nxt[p][x]==q) nxt[p][x]=t,p=link[p];
    link[q]=link[cur]=t;
}

```



```

        return cur;
    }
    void gsa_build() {
        link[0]=-1;
        queue<PII> q;
        for(int i=0;i<26;i++) if(nxt[0][i]) q.emplace(0,i);
        while(SZ(q)) {
            auto u=q.front();
            q.pop();
            auto last=gsa_insert(u.FI,u.SE);
            for(int i=0;i<26;i++) if(nxt[last][i]) q.emplace(last,i);
        }
    }
    int c[N],a[N*M*2];
    void gsa_topo() {
        for(int i=1;i<=n;i++) c[i]=0;
        for(int i=1;i<=sz;i++) ++c[len[i]];
        for(int i=1;i<=n;i++) c[i]+=c[i-1];
        for(int i=sz;i>=1;i--) a[c[len[i]]--]=i;
    }
}

```

动态规划

背包问题

01 背包

```

int n,m,dp[W],w[N],v[N];
for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        dp[j]=max(dp[j],dp[j-w[i]]+v[i]);

// 求01背包的方案数
dp[0]=1;
for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        dp[j]+=dp[j-w[i]];

// 取出第i个物品后的方案数
for(int i=1;i<=n;i++)
    for(int j=w[i];j<=m;j++)
        dp[j]-=dp[j-w[i]];

```

完全背包

```

int n,m,dp[W],w[N],v[N];
for(int i=1;i<=n;i++)
    for(int j=w[i];j<=m;j++)
        dp[j]=max(dp[j],dp[j-w[i]]+v[i]);

// 求完全背包的方案数
dp[0]=1;
for(int i=1;i<=n;i++)
    for(int j=w[i];j<=m;j++)
        dp[j]+=dp[j-w[i]];

```

```
// 取出第i个物品后的方案数
for(int i=1;i<n+1;i++)
    for(int j=m;j>=w[i];j--)
        dp[j]-=dp[j-w[i]];
```

多重背包

```
// 单调队列优化
int n,m,v,w,s,dp[M],pre[M],q[M];
for(int i=1;i<=n;i++) {
    if(m/v<s) s=m/v;
    memcpy(pre,dp,sizeof dp);
    for(int j=0;j<v;j++) {
        int l=1,r=0;
        for(int k=0;k*v+j<=m;k++) {
            int t=dp[k*v+j]-k*w;
            while(l<=r&&k-q[l]>s) l++;
            while(l<=r&&pre[q[r]*v+j]-q[r]*w<=t) r--;
            q[++r]=k;
            dp[k*v+j]=pre[q[l]*v+j]-q[l]*w+k*w;
        }
    }
}
```

分组背包

```
// 每组只能取一个物品
int n,m,c[N],v[N][N],w[N][N],dp[N];
for(int i=1;i<=n;i++)
    for(int j=m;j>=0;j--)
        for(int k=1;k<=c[i];k++)
            if(j-v[i][k]>=0)
                dp[j]=max(dp[j],dp[j-v[i][k]]+w[i][k]);
```

区间 DP

有 n 堆石子排成一排每堆石子有一定的质量，现在要将这 n 堆石子合并成为一堆，每次只能合并相邻的两堆，求最小代价

```
int dp[N][N],sum[N];
memset(dp,0x3f,sizeof dp);
for(int i=1;i<=n;i++) dp[i][i]=0;
for(int l=2;l<=n;l++)
    for(int i=1;j=i+l-1;j<=n;i++,j++)
        for(int k=i;k<=j;k++)
            dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+sum[j]-sum[i-1]);
```

状压 DP

- 预处理二进制状态 1 的个数

```
// i&(i-1)=i-lowbit(i)
for(int i=1;i<=n;i++) cnt[i]=cnt[i&(i-1)]+1;
```

- 枚举二进制状态的子集

```
for(int i=0;i<=n;i++)
    for(int j=i;j=(j-1)&i)
```

数位 DP

- 注意前导 0

斜率优化

- 将状态转移方程化成 $b = -kx + y$ 形式
- 假如经过判断，我们需要维护的是一个下凸包，那么我们需要找的最优决策点为 $k_l \leq k_x < k_r$ 的最小的 x
- x 非严格递增时，求斜率时若 x 相等，加上 `return y[a]<=y[b]?LDBL_MAX:LDBL_MIN`
- 比较斜率时，出队条件最好加上等于，防止分母出锅
- 根据题意判断队列初始化时是否要加入决策点 0
- x 单调， k 单调：用单调队列维护凸包
- x 单调， k 非单调：用单调栈维护凸包，每次二分找最佳决策点

数据结构

并查集

```
void init() {
    for(int i=1;i<=n;i++) fa[i]=i;
}
int find(int x) {
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
void merge(int x,int y) {
    int tx=find(x),ty=find(y);
    if(tx==ty) return;
    fa[tx]=ty;
}
```

- 边带权

```
int find(int x) {
    if(x==fa[x]) return x;
    int root=find(fa[x]);
    d[x]= ; // 更新x的边权（与fa[x]的边权的关系）
    return fa[x]=root;
}
void merge(int x,int y) {
    int tx=find(x),ty=find(y);
    fa[tx]=ty;
    d[tx]= ; // 更新x的边权
}
```

- 扩展域

扩展域：将一个节点拆成为多个节点

如：判断 x, y 奇偶性

可将节点分为奇数域和偶数域： `odd=x, even=x+n;`

若 x 和 y 奇偶性相同： `get(odd)=get(odd), get(even)=get(even);`

若 x 和 y 奇偶性不同： `get(odd)=get(even), get(even)=get(odd);`

ST 表

- 一维

```
int a[N], mx[25][N];
int hightbit(int x) {return 31-__builtin_clz(x);}
void init(int n) {
    for(int i=1; i<=n; i++) mx[0][i]=a[i];
    int t=hightbit(n);
    for(int i=1; i<=t; i++)
        for(int j=1; j+(1<<i)-1<=n; j++)
            mx[i][j]=max(mx[i-1][j], mx[i-1][j+(1<<(i-1))]);
}
int query(int l, int r) {
    int k=hightbit(r-l+1);
    return max(mx[k][l], mx[k][r-(1<<k)+1]);
}
```

- 二维

```
int a[N][N], mx[N][N][10][10];
int hightbit(int x) {return 31-__builtin_clz(x);}
void init(int n, int m) {
    int t1=hightbit(n);
    int t2=hightbit(m);
    for(int i=0; i<=t1; i++)
        for(int j=0; j<=t2; j++)
            for(int k=1; k<=n-(1<<i)+1; k++)
                for(int l=1; l<=m-(1<<j)+1; l++) {
                    if(!i&&!j) mx[k][l][i][j]=a[k][l];
                    else if(!i) mx[k][l][i][j]=max(mx[k][l][i][j-1], mx[k][l+(1<<(j-1))][i][j-1]);
                    else mx[k][l][i][j]=max(mx[k][l][i-1][j], mx[k+(1<<(i-1))][l][i-1][j]);
                }
}
int query(int x1, int y1, int x2, int y2) {
    int t1=hightbit(x2-x1+1);
    int t2=hightbit(y2-y1+1);
    return max({mx[x1][y1][t1][t2], mx[x2-(1<<t1)+1][y1][t1][t2],
                mx[x1][y2-(1<<t2)+1][t1][t2], mx[x2-(1<<t1)+1][y2-(1<<t2)+1][t1][t2]});
}
```

树状数组

- 单点修改 & 单点查询 & 区间查询

```
int n,b[N];
void update(int x,int v) {
    for(int i=x;i<=n;i+=i&-i) b[i]+=v;
}
int query(int x) {
    int ret=0;
    for(int i=x;i; i-=i&-i) ret+=b[i];
    return ret;
}
int query(int l,int r) {
    int ret=0;
    for(int i=r;i; i-=i&-i) ret+=b[i];
    for(int i=l-1;i; i-=i&-i) ret-=b[i];
    return ret;
}
```

- 区间修改 & 单点查询 & 区间查询

```
int n,b[N],c[N];
void update(int l,int r,int v) {
    for(int i=l;i<=n;i+=i&-i) b[i]+=v,c[i]+=l*v;
    for(int i=r+1;i<=n;i+=i&-i) b[i]-=v,c[i]-=(r+1)*v;
}
int query(int x) {
    int ret=0;
    for(int i=x;i; i-=i&-i) ret+=(x+1)*b[i]-c[i];
    return ret;
}
int query(int l,int r) {
    int ret=0;
    for(int i=r;i; i-=i&-i) ret+=(r+1)*b[i]-c[i];
    for(int i=l-1;i; i-=i&-i) ret-=l*b[i]-c[i];
    return ret;
}
```

线段树

```
int n,a[N],ls[N<<2],rs[N<<2],w[N<<2],add[N<<2];
void build(int p,int l,int r) {
    ls[p]=l,rs[p]=r;
    if(l==r) {w[p]=a[l];return;}
    int mid=l+r>>1;
    build(p<<1,l,mid);
    build(p<<1|1,mid+1,r);
    w[p]=w[p<<1]+w[p<<1|1];
}
void push_down(int p) {
    if(add[p]) {
        w[p<<1]+=(rs[p<<1]-ls[p<<1]+1)*add[p];
        w[p<<1|1]+=(rs[p<<1|1]-ls[p<<1|1]+1)*add[p];
        add[p<<1]+=add[p],add[p<<1|1]+=add[p];
        add[p]=0;
    }
}
```

```

}
void update(int p,int l,int r,int v) {
    if(ls[p]>=l&&rs[p]<=r) {
        w[p]+=rs[p]-ls[p]+1)*v;
        add[p]+=v;
        return;
    }
    push_down(p);
    int mid=ls[p]+rs[p]>>1;
    if(l<=mid) update(p<<1,l,r,v);
    if(r>mid) update(p<<1|1,l,r,v);
    w[p]=w[p<<1]+w[p<<1|1];
}
int query(int p,int l,int r) {
    if(ls[p]>=l&&rs[p]<=r) return w[p];
    push_down(p);
    int mid=ls[p]+rs[p]>>1;
    if(r<=mid) return query(p<<1,l,r);
    if(l>mid) return query(p<<1|1,l,r);
    return query(p<<1,l,r)+query(p<<1|1,l,r);
}

```

- 标记永久化

```

int n,a[N],ls[N<<2],rs[N<<2],w[N<<2],add[N<<2];
void build(int p,int l,int r) {
    ls[p]=l,rs[p]=r;
    if(l==r) {w[p]=a[l];return;}
    int mid=l+r>>1;
    build(p<<1,l,mid);
    build(p<<1|1,mid+1,r);
    w[p]=w[p<<1]+w[p<<1|1];
}
void update(int p,int l,int r,int v) {
    w[p]+=(min(rs[p],r)-max(ls[p],l)+1)*v;
    if(ls[p]>=l&&rs[p]<=r) {
        add[p]+=v;
        return;
    }
    int mid=ls[p]+rs[p]>>1;
    if(l<=mid) update(p<<1,l,r,v);
    if(r>mid) update(p<<1|1,l,r,v);
}
int query(int p,int l,int r) {
    if(ls[p]>=l&&rs[p]<=r) return w[p];
    int mid=ls[p]+rs[p]>>1;
    int ret=(r-l+1)*add[p];
    if(r<=mid) return ret+query(p<<1,l,r);
    if(l>mid) return ret+query(p<<1|1,l,r);
    return ret+query(p<<1,l,mid)+query(p<<1|1,mid+1,r);
}

```

可持久化线段树

- 区间修改，有两种方法，lazy 标记下传和标记永久化，前者空间需开大。

```

// 区间第 k 小
#include <cstdio>
using namespace std;
const int N=1e5+5;
int n,m,a[N];
int ls[N<<5],rs[N<<5],sum[N<<5],rt[N],cnt;
void update(int &p,int q,int L,int R,int x) {
    p=++cnt;
    ls[p]=ls[q],rs[p]=rs[q],sum[p]=sum[q]+1;
    if(L==R) return;
    int mid=L+R>>1;
    if(x<=mid) update(ls[p],ls[q],L,mid,x);
    else update(rs[p],rs[q],mid+1,R,x);
}
int query(int p,int q,int L,int R,int k) {
    if(L==R) return L;
    int mid=L+R>>1;
    int lsum=sum[ls[q]]-sum[ls[p]];
    if(k<=lsum) return query(ls[p],ls[q],L,mid,k);
    else return query(rs[p],rs[q],mid+1,R,k-lsum);
}
int main() {
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++) update(rt[i],rt[i-1],-1e9,1e9,a[i]);
    for(int i=1;i<=m;i++) {
        int l,r,k;
        scanf("%d%d%d",&l,&r,&k);
        printf("%d\n",query(rt[l-1],rt[r],-1e9,1e9,k));
    }
    return 0;
}

```

平衡树

Splay 树

```

struct Splay {
    int rt,cnt,ch[N][2],sz[N],tot[N],val[N],fa[N];
    bool get(int x) {return x==ch[fa[x]][1];}
    void clear(int x) {ch[x][0]=ch[x][1]=fa[x]=val[x]=sz[x]=tot[x]=0;}
    void push_up(int x) {sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+tot[x];}
    void rotate(int x) {
        int y=fa[x],z=fa[y],chk=get(x);
        ch[y][chk]=ch[x][chk^1];
        if(ch[x][chk^1]) fa[ch[x][chk^1]]=y;
        ch[x][chk^1]=y;
        fa[y]=x,fa[x]=z;
        if(z) ch[z][y==ch[z][1]]=x;
        push_up(y);
    }
    void splay(int x) {
        for(int f=fa[x];f=fa[x],f;rotate(x)) if(fa[f])
            rotate(get(x)==get(f)?f:x);
        push_up(x);
        rt=x;
    }
}

```

```

}
void insert(int x) {
    if(!rt) {
        val[++cnt]=x;
        ++tot[cnt];
        rt=cnt;
        push_up(rt);
        return;
    }
    int cur=rt,f=0;
    for(;;) {
        if(val[cur]==x) {
            ++tot[cur];
            push_up(cur),push_up(f);
            splay(cur);
            break;
        }
        f=cur;
        cur=ch[cur][val[cur]<x];
        if(!cur) {
            val[++cnt]=x;
            ++tot[cnt];
            fa[cnt]=f;
            ch[f][val[f]<x]=cnt;
            push_up(cnt),push_up(f);
            splay(cnt);
            break;
        }
    }
}

int rk(int x) { // 树中存在 x
    int res=0,cur=rt;
    for(;;) {
        if(x<val[cur]) cur=ch[cur][0];
        else {
            res+=sz[ch[cur][0]];
            if(x==val[cur]) break;
            res+=tot[cur];
            cur=ch[cur][1];
        }
    }
    splay(cur);
    return res+1;
}

int kth(int k) {
    int cur=rt;
    for(;;) {
        if(ch[cur][0]&& k<=sz[ch[cur][0]]) cur=ch[cur][0];
        else {
            k-=tot[cur]+sz[ch[cur][0]];
            if(k<=0) break;
            cur=ch[cur][1];
        }
    }
    splay(cur);
    return val[cur];
}

int pre() {

```



```

    int cur=ch[rt][0];
    while(ch[cur][1]) cur=ch[cur][1];
    splay(cur);
    return cur;
}
int nxt() {
    int cur=ch[rt][1];
    while(ch[cur][0]) cur=ch[cur][0];
    splay(cur);
    return cur;
}
void del(int x) {
    rk(x);
    if(!ch[rt][0]||!ch[rt][1]) {
        int cur=rt;
        fa[rt=ch[rt][0]+ch[rt][1]]=0;
        clear(cur);
    }
    else {
        // 合并左右子树，将左子树最大值 splay 到根，然后把它的右子树设置为右子树并更新节点的信息
        int cur=rt;
        x=pre();
        fa[ch[cur][1]]=x;
        ch[x][1]=ch[cur][1];
        clear(cur);
        push_up(rt);
    }
}
int get_pre(int x) {
    insert(x); // 插入 x，此时 x 位于树根，前驱即为 x 的左子树最右边的节点
    int ret=pre();
    del(x); // 删除 x
    return ret; // 返回节点 id
}
int get_nxt(int x) {
    insert(x); // 插入 x，此时 x 位于树根，后驱即为 x 的右子树最左边的节点
    int ret=nxt();
    del(x); // 删除 x
    return ret; // 返回节点 id
}
}splay;

```

- 平衡树维护区间翻转
- LibreOJ 104. 文艺平衡树

对于区间 $[l, r]$ ，将 $l - 1$ 上旋至树根， $r + 1$ 上旋至树根的右子树，那么树根的右子树的左子树就是区间 $[l, r]$ ，将其标记，注意下放标记并且交换左右子树。

```

#include <bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int n,m;
struct Splay {
    int rt,cnt,ch[N][2],sz[N],fa[N],mark[N];
    bool get(int x) {return x==ch[fa[x]][1];}
    void push_up(int x) {sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+1;}
}

```

```

void push_down(int x) {
    if(mark[x]) {
        mark[ch[x][0]]^=1;
        mark[ch[x][1]]^=1;
        mark[x]=0;
        swap(ch[x][0],ch[x][1]);
    }
}

void build(int l,int r,int f) {
    if(l>r) return;
    int mid=l+r>>1;
    ch[mid][0]=ch[mid][1]=0;
    if(f==-1) rt=mid,fa[rt]=0;
    else ch[f][r>f]=mid,fa[mid]=f;
    build(l,mid-1,mid);
    build(mid+1,r,mid);
    push_up(mid);
}

void rotate(int x) {
    int y=fa[x],z=fa[y];
    push_down(y);
    push_down(x);
    int chk=get(x);
    ch[y][chk]=ch[x][chk^1];
    if(ch[x][chk^1]) fa[ch[x][chk^1]]=y;
    ch[x][chk^1]=y;
    fa[y]=x,fa[x]=z;
    if(z) ch[z][y==ch[z][1]]=x;
    push_up(y);
}

void splay(int x,int t) {
    push_down(x);
    for(int f=fa[x];f=fa[x],f!=t;rotate(x)) if(fa[f]!=t)
        rotate(get(x)==get(f)?f:x);
    push_up(x);
    if(!t) rt=x;
}

int kth(int k) {
    int cur=rt;
    for(;;) {
        push_down(cur);
        if(ch[cur][0]&& k<=sz[ch[cur][0]]) cur=ch[cur][0];
        else {
            k-=1+sz[ch[cur][0]];
            if(k<=0) break;
            cur=ch[cur][1];
        }
    }
    splay(cur,0);
    return cur;
}

void reverse(int l,int r) {
    l=kth(l);
    r=kth(r+2);
    splay(l,0);
    splay(r,l);
    mark[ch[ch[rt][1]][0]]^=1;
}

```

```

void print(int u) {
    if(u) {
        push_down(u);
        print(ch[u][0]);
        if(u>1&&u<n+2) cout<<u-1<<' ';
        print(ch[u][1]);
    }
}
}splay;
int main() {
    cin>>n>>m;
    splay.build(1,n+2,-1);
    for(int i=1;i<=m;i++) {
        int l,r;
        cin>>l>>r;
        splay.reverse(l,r);
    }
    splay.print(splay.rt);
    cout<<'\n';
    return 0;
}

```

树套树

树状数组套可持久化线段树

```

// 动态区间第 k 小
// luogu p2617
#include <bits/stdc++.h>
#define SZ(x) (int)(x).size()
#define ALL(x) (x).begin(),(x).end()
#define PB push_back
using namespace std;
typedef vector<int> VI;
const int N=1e5+5;
int
n,m,a[N],cnt,rt[N],ls[N*2*18*18],rs[N*2*18*18],sum[N*2*18*18],tot1,tot2,t1[18],t2[18];
VI b;
struct Q {
    char o;
    int l,r,k;
    int x,v;
}q[N];
int find(int x) {
    return lower_bound(ALL(b),x)-b.begin()+1;
}
void update(int &p,int l,int r,int x,int v) {
    if(!p) p=++cnt;
    sum[p]+=v;
    if(l==r) return;
    int mid=l+r>>1;
    if(x<=mid) update(ls[p],l,mid,x,v);
    else update(rs[p],mid+1,r,x,v);
}
int query(int l,int r,int k) {

```

```

if(l==r) return 1;
int mid=l+r>>1,sz=0;
for(int i=1;i<=tot1;i++) sz-=sum[ls[t1[i]]];
for(int i=1;i<=tot2;i++) sz+=sum[ls[t2[i]]];
if(sz>=k) {
    for(int i=1;i<=tot1;i++) t1[i]=ls[t1[i]];
    for(int i=1;i<=tot2;i++) t2[i]=ls[t2[i]];
    return query(1,mid,k);
}
else {
    for(int i=1;i<=tot1;i++) t1[i]=rs[t1[i]];
    for(int i=1;i<=tot2;i++) t2[i]=rs[t2[i]];
    return query(mid+1,r,k-sz);
}
}
int lowbit(int x) {
    return x&-x;
}
void upd(int p,int x,int v) {
    for(;p<=n;p+=lowbit(p)) update(rt[p],1,SZ(b),x,v);
}
int qry(int l,int r,int k) {
    tot1=tot2=0;
    for(;l;l-=lowbit(l)) t1[++tot1]=rt[l];
    for(;r;r-=lowbit(r)) t2[++tot2]=rt[r];
    return query(1,SZ(b),k);
}
int main() {
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) {
        scanf("%d",&a[i]);
        b.PB(a[i]);
    }
    for(int i=1;i<=m;i++) {
        scanf(" %c",&q[i].o);
        if(q[i].o=='C') scanf("%d%d",&q[i].x,&q[i].v),b.PB(q[i].v);
        else scanf("%d%d%d",&q[i].l,&q[i].r,&q[i].k);
    }
    sort(ALL(b));
    b.resize(unique(ALL(b))-b.begin());
    for(int i=1;i<=n;i++) upd(i,find(a[i]),1);
    for(int i=1;i<=m;i++) {
        if(q[i].o=='C') {
            upd(q[i].x,find(q[i].v),1);
            upd(q[i].x,find(a[q[i].x]),-1);
            a[q[i].x]=q[i].v;
        }
        else printf("%d\n",b[qry(q[i].l-1,q[i].r,q[i].k)-1]);
    }
    return 0;
}

```

莫队

普通莫队

对于长度为 n 的序列上的 m 次区间询问问题，如果从 $[l, r]$ 的答案能够 $O(1)$ 扩展到 $[l-1, r], [l+1, r], [l, r-1], [l, r+1]$ 的答案，可以在 $O(n\sqrt{m})$ 的复杂度内求出所有询问的答案。

实现：离线后排序，顺序处理每个询问，暴力从上一个区间的答案转移到下一个区间答案（一步一步移动即可）。

排序方法：设定块的长度为 S ，取 $S = \lceil \frac{n}{\sqrt{m}} \rceil$ ，按照 $(\lfloor \frac{l}{S} \rfloor, r)$ 的二元组从小到大排序。

奇偶优化：设块的编号从 1 开始，对于属于奇数块的询问， r 按从小到大排序，对于属于偶数块的排序， r 从大到小排序。

```
// SPOJ DQUERY 区间不同数的个数的查询
// 数组均从 0 开始
int n,m,unit,ans,a[30005],cnt[1000005],res[200005];
struct Q {
    int l,r,id;
    Q() {}
    Q(int l,int r,int id):l(l),r(r),id(id) {}
    bool operator < (const Q &T) const {
        if(l/unit!=T.l/unit) return l<T.l;
        if((l/unit)&1) return r>T.r;
        return r<T.r;
    }
} q[200005];
void move(int x,int v) {
    if(v== -1&&--cnt[a[x]]==0) --ans;
    if(v== 1&&++cnt[a[x]]==1) ++ans;
}
void mo() {
    unit=int(ceil(n/pow(n,0.5)));
    sort(q,q+m);
    int l=0,r=-1;
    for(int i=0;i<m;i++) {
        while(l>q[i].l) move(--l,1);
        while(r<q[i].r) move(++r,1);
        while(l<q[i].l) move(l++, -1);
        while(r>q[i].r) move(r--, -1);
        res[q[i].id]=ans;
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>n;
    for(int i=0;i<n;i++) cin>>a[i];
    cin>>m;
    for(int i=0;i<m;i++) {
        cin>>q[i].l>>q[i].r;
        --q[i].l,--q[i].r;
        q[i].id=i;
    }
    mo();
    for(int i=0;i<m;i++) cout<<res[i]<<'\n';
    return 0;
}
```

```
}
```

带修莫队

因为多了修改操作，所以在普通莫队的基础上多加一个时间轴 t ，可以在 $O(\sqrt[3]{n^4t})$ 的复杂度内求出所有询问的答案。

排序方法：设定块的长度为 S ，取 $S = \lceil \sqrt[3]{nt} \rceil$ ，按照 $(\lfloor \frac{l}{S} \rfloor, \lfloor \frac{r}{S} \rfloor, t)$ 的三元组从小到大排序。

```
// BZOJ 2120 带修改的区间不同数的个数的查询
int n,m,unit,t=1,ans,a[10005],cnt[1000005],pre[10005],res[10005];
struct R {
    int x,v,pre;
    R() {}
    R(int x,int v,int pre):x(x),v(v),pre(pre) {}
} b[1005];
struct Q {
    int l,r,t,id;
    Q() {}
    Q(int l,int r,int t,int id):l(l),r(r),t(t),id(id) {}
    bool operator < (const Q &T) const {
        if(l/unit!=T.l/unit) return l<T.l;
        if(r/unit!=T.r/unit) return r<T.r;
        return t<T.t;
    }
};
vector<Q> q;
void update_t(int t,int l,int r,int v) {
    if(v==1) {
        if(b[t].x>=l&&b[t].x<=r) {
            if(--cnt[b[t].pre]==0) --ans;
            if(++cnt[b[t].v]==1) ++ans;
        }
        a[b[t].x]=b[t].v;
    }
    else {
        if(b[t].x>=l&&b[t].x<=r) {
            if(--cnt[b[t].v]==0) --ans;
            if(++cnt[b[t].pre]==1) ++ans;
        }
        a[b[t].x]=b[t].pre;
    }
}
void update(int x,int v) {
    if(v==-1&&--cnt[a[x]]==0) --ans;
    if(v==1&&++cnt[a[x]]==1) ++ans;
}
void mo() {
    unit=int(ceil(pow(n*t,1.0/3)));
    sort(ALL(q));
    int t=1,l=0,r=-1;
    for(auto x:q) {
        while(t<x.t) update_t(++t,l,r,1);
        while(t>x.t) update_t(t--,l,r,-1);
        while(l>x.l) update(--l,1);
        while(r<x.r) update(++r,1);
        while(l<x.l) update(l++, -1);
    }
}
```

```

        while(r>x.r) update(r--,-1);
        res[x.id]=ans;
    }
    for(int i=0;i<SZ(q);i++) cout<<res[i]<<'\n';
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>n>>m;
    for(int i=0;i<n;i++) {
        cin>>a[i];
        pre[i]=a[i];
    }
    int cnt_q=0;
    for(int i=0;i<m;i++) {
        char o;cin>>o;
        if(o=='R') {
            int x,v;cin>>x>>v;
            --x;
            b[++t]=R(x,v,pre[x]);
            pre[x]=v;
        }
        else {
            int l,r;cin>>l>>r;
            --l,--r;
            q.EB(l,r,t,cnt_q++);
        }
    }
    mo();
    return 0;
}

```

图论

最短路

Dijkstra 算法

- 无法处理负权

```

int dist[N];
bool st[N];
VPII g[N];
void dijkstra() {
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;
    priority_queue<PII,VPII,greater<PII>> q;
    q.emplace(0,1);
    while(SZ(q)) {
        int u=q.top().SE;
        q.pop();
        if(st[u]) continue;
        st[u]=true;
        for(auto x:g[u]) {
            int v=x.FI,w=x.SE;
            if(!st[v]&&dist[u]+w<dist[v]) {

```

```

        dist[v]=dist[u]+w;
        q.emplace(dist[v],v);
    }
}
}
}

```

SPFA 算法

- 一条最短路最多有 $n - 1$ 条边组成，若经过 $n - 1$ 次更新后还能更新，则存在负环
- LLL 优化：将普通队列换成双端队列，每次将入队结点距离和队内距离平均值比较，如果更大则插入至队尾，否则插入队首
- SLF 优化：将普通队列换成双端队列，每次将入队结点距离和队首比较，如果更大则插入至队尾，否则插入队首

```

int dist[N];
bool st[N];
VPII g[N];
void SPFA() {
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;
    queue<int> q;
    q.push(1);
    st[1]=true;
    while(SZ(q)) {
        int u=q.front();
        q.pop();
        st[u]=false;
        for(auto x:g[u]) {
            int v=x.FI,w=x.SE;
            if(dist[u]+w<dist[v]) {
                dist[v]=dist[u]+w;
                if(!st[v]) q.push(v),st[v]=true;
            }
        }
    }
}

```

Floyd 算法

- 全源最短路

```

int n,dist[N][N];
void floyd() {
    memset(dist,0x3f,sizeof dist);
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
}

```

同余最短路

- 解决形如 $k = \sum_{i=1}^n a_i x_i$ 的问题

假设数组 a 最小的是 a_1 , 记 $y = \sum_{i=2}^n a_i x_i$

设 $f[j]$ 为使 $y \bmod a_1 = j$ 成立的最小的 y

可以得到转移式: $f[(j + a_i) \% a_1] = \min(f[(j + a_i) \% a_1], f[j] + a_i) \ (2 \leq i \leq n, 0 \leq j < a_1)$

因此可以在点 j 和点 $(j + a_i) \% a_1$ 间建立一条权值为 a_i 的边, 跑最短路

差分约束

一个系统 n 个变量和 m 个约束条件组成, 每个约束条件形如 $x_j - x_i \leq b_k$

可以发现每个约束条件都形如最短路中的三角不等式 $d_u - d_v \leq w_{u,v}$, 因此连一条边 (i, j, b_k) 建图

若要使得所有量两两的值最接近 (最小解), 源点到各点的距离初始成 0, 跑最长路

若要使得某一变量与其他变量的差尽可能大 (最大解), 则源点到各点距离初始化成 ∞ , 跑最短路

- 若约束条件中的变量为整数, $d_u - d_v < w_{u,v}$ 可改为 $d_u - d_v \leq w_{u,v} - 1$
- 若约束条件中的变量为实数, $d_u - d_v < w_{u,v}$ 可近似为 $d_u - d_v \leq w_{u,v}$

根据约束条件, 判断跑最短路还是最长路

跑最短路, 有负环则无法满足所有约束条件

跑最长路, 有正环则无法满足所有约束条件

最小生成树

Kruskal 算法

```
int n, fa[N];
struct E {
    int u, v, w;
    E() {}
    E(int u, int v, int w): u(u), v(v), w(w) {}
    bool operator < (const E &a) const {
        return w < a.w;
    }
};
vector<E> e;
int find(int x) {
    return x == fa[x] ? x : fa[x] = find(fa[x]);
}
int kruskal() {
    for(int i=1; i<=n; i++) fa[i]=i;
    sort(ALL(e));
    int ret=0;
    for(auto x:e) {
        int u=x.u, v=x.v, w=x.w;
        int tx=find(u), ty=find(v);
        if(tx==ty) continue;
        fa[tx]=ty;
        res+=w;
    }
    return ret;
}
```

Prim 算法

- Prim 算法可以堆优化，但用优先队列实现堆不如直接用 Kruskal 算法。
- 稠密图和完全图用暴力 Prim 算法比 Kruskal 算法更优，但不一定跑得快，时间复杂度 $O(n^2 + m)$ 。

```
int w[N][N], f[N];
bool vis[N];
int prim(int n) {
    memset(f, 0x3f, sizeof f);
    memset(vis, false, sizeof vis);
    f[1] = 0;
    int ret = 0;
    for (int i = 1; i <= n; i++) {
        int x = 0;
        for (int j = 1; j <= n; j++) if (!vis[j] && (x == 0 || f[j] < f[x])) {
            x = j;
        }
        vis[x] = true;
        ret += f[x];
        for (int j = 1; j <= n; j++) if (!vis[j] && w[x][j] < f[j]) {
            f[j] = w[x][j];
        }
    }
    return ret;
}
```

最小生成树的唯一性

考虑最小生成树的唯一性。如果一条边不在最小生成树的边集中，并且可以替换与其权值相同、并且在最小生成树边集的另一条边。那么这个最小生成树就是不唯一的

对于 Kruskal 算法，只要计算为当前权值的边可以放几条，实际放了几条，如果这两个值不一样，那么就说明这几条边与之前的边产生了一个环（这个环中至少有两条当前权值的边，否则根据并查集，这条边是不能放的），即最小生成树不唯一

寻找权值与当前边相同的边，我们只需要记录头尾指针，用单调队列即可在基本与原算法时间相同的时间复杂度里优秀解决这个问题

```
//POJ 1679
#include <cstdio>
#include <algorithm>
using namespace std;
const int N=105;
struct E {
    int u,v,w;
    E() {}
    E(int u,int v,int w):u(u),v(v),w(w) {}
    bool operator < (const E &a) const {
        return w<a.w;
    }
}e[N*N];
int n,m,fa[N],cnt;
int find(int x) {
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
```

```

void kruskal() {
    for(int i=1;i<=n;i++) fa[i]=i;
    sort(e+1,e+1+cnt);
    e[++cnt]=E(0,0,0);
    bool f=true;
    int res=0,tail=0,sum1=0,sum2=0;
    for(int i=1;i<=cnt;i++) {
        if(i>tail) {
            if(sum1!=sum2) {f=false;break;}
            sum1=0;
            for(int j=i;j<=cnt;j++) {
                if(e[j].w!=e[i].w) {tail=j-1;break;}
                if(find(e[j].u)!=find(e[j].v)) sum1++;
            }
            sum2=0;
        }
        if(i>cnt) break;
        int tx=find(e[i].u),ty=find(e[i].v);
        if(tx==ty) continue;
        fa[tx]=ty;
        res+=e[i].w;
        sum2++;
    }
    if(f) printf("%d\n",res);
    else puts("Not Unique!");
}

int main() {
    int T;
    scanf("%d",&T);
    while(T--) {
        scanf("%d%d",&n,&m);
        cnt=0;
        for(int i=1;i<=m;i++) {
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            e[++cnt]=E(u,v,w);
        }
        kruskal();
    }
    return 0;
}

```

次小生成树

非严格次小生成树

求出无向图的最小生成树 T ，设其权值和为 M

遍历每条未被选中的边 $e = (u, v, w)$ ，找到 T 中 u 到 v 路径上边权最大的一条边 $e' = (s, t, w')$ ，则在 T 中以 e 替换 e' ，可得一棵权值和为 $M' = M + w - w'$ 的生成树 T'

对所有替换得到的答案 M' 取最小值即可

使用倍增预处理出每个节点的 2^i 级祖先及到达其 2^i 级祖先路径上最大的边权，这样在倍增求 LCA 的过程中可以直接求得 u, v 路径上的边权最大值

严格次小生成树

维护到 2^i 级祖先路径上的最大边权的同时维护严格次大边权，当用于替换的边的权值与原生成树中路径最大边权相等时，我们用严格次大值来替换即可

时间复杂度 $O(m \log m)$

```
// 洛谷 P4180
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int N=1e5+5,M=3e5+5;
int n,m,t,fa[N],d[N],f[N][20],g[N][20][2];
bool st[M];
vector<pair<int,int>> G[N];
struct E {
    int u,v,w;
    E() {}
    E(int u,int v,int w):u(u),v(v),w(w) {}
    bool operator < (const E &a) const {
        return w<a.w;
    }
}e[M];
int find(int x) {
    return fa[x]==x?x:fa[x]=find(fa[x]);
}
int kruskal() {
    for(int i=1;i<=n;i++) fa[i]=i;
    sort(e+1,e+1+m);
    int res=0;
    for(int i=1;i<=m;i++) {
        int tx=find(e[i].u),ty=find(e[i].v);
        if(tx==ty) continue;
        st[i]=true;
        fa[tx]=ty;
        res+=e[i].w;
        G[e[i].u].push_back(make_pair(e[i].v,e[i].w));
        G[e[i].v].push_back(make_pair(e[i].u,e[i].w));
    }
    return res;
}
void bfs() {
    queue<int> q;
    q.push(1);
    d[1]=0;
    while(!q.empty()) {
        int u=q.front();
        q.pop();
        for(auto x:G[u]) {
            int v=x.first;
            if(v==f[u][0]) continue;
            d[v]=d[u]+1;
            f[v][0]=u;
            for(int i=1;i<=t;i++) f[v][i]=f[f[v][i-1]][i-1];
            g[v][0][0]=x.second;
            g[v][0][1]=0;
            for(int j=1;j<=t;j++) {
```

```

        g[v][j][0]=max(g[v][j-1][0],g[f[v][j-1]][j-1][0]);
        if(g[v][j-1][0]==g[f[v][j-1]][j-1][0])
            g[v][j][1]=max(g[v][j-1][1],g[f[v][j-1]][j-1][1]);
        else if(g[v][j-1][0]<g[f[v][j-1]][j-1][0])
            g[v][j][1]=max(g[v][j-1][0],g[f[v][j-1]][j-1][1]);
        else if(g[v][j-1][0]>g[f[v][j-1]][j-1][0])
            g[v][j][1]=max(g[v][j-1][1],g[f[v][j-1]][j-1][0]);
    }
    q.push(v);
}
}
}
int LCA(int a,int b) {
    if(d[a]>d[b]) swap(a,b);
    for(int i=t;i>=0;i--) if(d[f[b][i]]>=d[a]) b=f[b][i];
    if(a==b) return a;
    for(int i=t;i>=0;i--) if(f[a][i]!=f[b][i]) a=f[a][i],b=f[b][i];
    return f[a][0];
}
pair<int,int> calc(int a,int b) {
    pair<int,int> res(0,0);
    for(int i=t;i>=0;i--)
        if(d[f[a][i]]>=d[b]) {
            if(g[a][i][0]>=res.first) {
                res.second=max(res.second,max(res.first,g[a][i][1]));
                res.first=g[a][i][0];
            }
            else res.second=max(res.second,g[a][i][0]);
            a=f[a][i];
        }
    return res;
}
int solve() {
    int mst=kruskal();
    bfs();
    int res=0x7fffffff;
    for(int i=1;i<=m;i++) {
        if(st[i]) continue;
        int mx,lca=LCA(e[i].u,e[i].v);
        pair<int,int> t1=calc(e[i].u,lca),t2=calc(e[i].v,lca);
        if(t1.first!=t2.first) {
            if(max(t1.first,t2.first)==e[i].w)
                mx=max(max(t1.second,t2.second),min(t1.first,t2.first));
            else mx=max(t1.first,t2.first);
        }
        else {
            if(t1.first==e[i].w) mx=max(t1.second,t2.second);
            else mx=t1.first;
        }
        res=min(res,mst-mx+e[i].w);
    }
    return res;
}
signed main() {
    scanf("%lld%lld",&n,&m);
    t=(int)(log(n)/log(2))+1;
    for(int i=1;i<=m;i++) {
        int u,v,w;

```

```

scanf("%lld%lld%lld",&u,&v,&w);
e[i]=E(u,v,w);
}
printf("%lld\n",solve());
return 0;
}

```

最小 k 度限制生成树

去除 v_0 , 找到所有连通分量, 每个连通分量跑最小生成树

添加 v_0 与每个连通分量的最小的边 (若有 t 个连通分量, 则此时为 t 度最小生成树)

若 $t < k$, 则还可添加 $k - t$ 条边, 对于每条不属于最小生成树的 (v_0, v) , 边权为 z , 找出最小生成树中 v 到 v_0 路径上的最大边, 边权为 w , 求出使 $w - z$ 最大的点 v , 若 $w - z > 0$, 则加入 (v_0, v)

- 求 v 到 v_0 路径上最大边可以用 DP

设 $dp[i]$ 为路径 v_0 到 v 无关联且权值最大的边, $fa[v]$ 为 v 的父节点

转移方程: $dp[v] = \max(dp[fa[v]], dist[fa[v]][v])$

边界条件: $dp[v_0] = -\infty, dp[v'] = -\infty ((v_0, v') \in E(T))$

```

//POJ 1639
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
using namespace std;
const int N=505;
struct E {
    int u,v,w;
    E() {}
    E(int u,int v,int w):u(u),v(v),w(w) {}
    bool operator < (const E &a) const {
        return w<a.w;
    }
}mn[N],mx[N];
int n,m,k,dist[N][N],fa[N],id[N],vis[N];
bool st[N][N];
vector<E> e;
map<string,int> mp;
int find(int x) {
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
int kruskal() {
    for(int i=1;i<=n;i++) fa[i]=i;
    sort(e.begin(),e.end());
    int res=0;
    for(int i=0;i<e.size();i++)
        if(id[e[i].u]==id[e[i].v]) {
            int tx=find(e[i].u),ty=find(e[i].v);
            if(tx==ty) continue;
            fa[tx]=ty;
            res+=e[i].w;
            st[e[i].u][e[i].v]=st[e[i].v][e[i].u]=true;
        }
}

```

```

        return res;
    }
    void dfs1(int u,int t) {
        id[u]=t;
        for(int i=2;i<=n;i++) if(dist[u][i]&&!id[i]) dfs1(i,t);
    }
    void dp(int u,int fa) {
        for(int i=2;i<=n;i++) {
            if(i==fa||!st[u][i]) continue;
            if(u!=1) {
                if(mx[u].w>dist[u][i]) mx[i]=mx[u];
                else mx[i]=E(u,i,dist[u][i]);
            }
            dp(i,u);
        }
    }
    int solve() {
        int t=0;
        for(int i=2;i<=n;i++) if(!id[i]) dfs1(i,++t);
        int res=kruskal();
        for(int i=1;i<=t;i++) mn[i].w=0x3f3f3f3f;
        for(int i=2;i<=n;i++)
            if(dist[1][i]&&dist[1][i]<mn[id[i]].w)
                mn[id[i]]=E(1,i,dist[1][i]);
        for(int i=2;i<=n;i++) {
            if(vis[id[i]]) continue;
            st[1][mn[id[i]].v]=st[mn[id[i]].v][1]=true;
            res+=mn[id[i]].w;
            vis[id[i]]=true;
            k--;
        }
        while(k) {
            mx[1].w=0xc0c0c0c0;
            for(int i=2;i<=n;i++) {
                if(st[1][i]) mx[i].w=0xc0c0c0c0;
                else mx[i].w=0;
            }
            dp(1,0);
            int temp=1;
            for(int i=2;i<=n;i++) {
                if(st[1][i]||!dist[1][i]) continue;
                if(dist[1][i]-mx[i].w<dist[1][temp]-mx[temp].w) temp=i;
            }
            st[1][temp]=st[temp][1]=true;
            st[mx[temp].u][mx[temp].v]=st[mx[temp].v][mx[temp].u]=false;
            if(dist[1][temp]-mx[temp].w>0) break;
            res+=dist[1][temp]-mx[temp].w;
            k--;
        }
        return res;
    }
    int main() {
        ios::sync_with_stdio(false);
        cin.tie(0);
        mp["Park"]+=n;
        cin>>m;
        for(int i=1;i<=m;i++) {
            string a,b;

```

```

int w;
cin>>a>>b>>w;
if(!mp.count(a)) mp[a]=++n;
if(!mp.count(b)) mp[b]=++n;
dist[mp[a]][mp[b]]=dist[mp[b]][mp[a]]=w;
e.push_back(E(mp[a],mp[b],w));
}
cin>>k;
cout<<"Total miles driven: "<<solve()<<'\n';
return 0;
}

```

瓶颈生成树

定义：无向图 G 的瓶颈生成树是这样的一个生成树，它的最大的边权值在 G 的所有生成树中最小

性质：最小生成树是瓶颈生成树的充分不必要条件

最小瓶颈路

定义：无向图 G 中 x 到 y 的最小瓶颈路是这样的一类简单路径，满足这条路径上的最大的边权在所有 x 到 y 的简单路径中是最小的

性质：根据最小生成树定义， x 到 y 的最小瓶颈路上的最大边权等于最小生成树上 x 到 y 路径上的最大边权。虽然最小生成树不唯一，但是每种最小生成树 x 到 y 路径的最大边权相同且为最小值。也就是说，每种最小生成树上的 x 到 y 的路径均为最小瓶颈路

- 并不是所有最小瓶颈路都存在一棵最小生成树满足其为树上 x 到 y 的简单路径

最小树形图

- 有向图最小生成树

朱刘算法

- $O(nm)$

```

int in[N],st[N],id[N],pre[N];
struct E {
    int u,v,w;
    E() {}
    E(int u,int v,int w):u(u),v(v),w(w) {}
};
vector<E> e;
int edmonds(int rt,int n) {
    int ret=0,v,nn=n;
    for(;;) {
        for(int i=1;i<=n;i++) in[i]=0x3f3f3f3f;
        for(auto x:e) if(x.u!=x.v&&x.w<in[x.v]) pre[x.v]=x.u,in[x.v]=x.w;
        for(int i=1;i<=n;i++) if(i!=rt&&in[i]==0x3f3f3f3f) return -1;
        int tn=0;
        for(int i=1;i<=nn;i++) id[i]=st[i]=-1;
        in[rt]=0;
        for(int i=1;i<=n;i++) {
            ret+=in[v=i];
            while(st[v]!=i&&id[v]==-1&&v!=rt) st[v]=i,v=pre[v];
            if(v!=rt&&id[v]==-1) {

```



```

        ++tn;
        for(int u=pre[v];u!=v;u=pre[u]) id[u]=tn;
        id[v]=tn;
    }
}
if(!tn) break;
for(int i=1;i<=n;i++) if(id[i]==-1) id[i]=++tn;
for(int i=0;i<SZ(e);) {
    auto &x=e[i];
    v=x.v;
    x.u=id[x.u],x.v=id[x.v];
    if(x.u!=x.v) x.w-=in[v],++i;
    else swap(x,e.back()),e.pop_back();
}
n=tn,rt=id[rt];
}
return ret;
}

```

无定根最小树形图

```

// 输出编号最小的根
int in[N],st[N],id[N],pre[N],inn[N],sum,t;
struct E {
    int u,v,w,num;
    E() {}
    E(int u,int v,int w,int num):u(u),v(v),w(w),num(num) {}
};
vector<E> e;
int edmonds(int rt,int n) {
    int ret=0;
    int v,nn=n;
    while(1) {
        for(int i=1;i<=n;i++) in[i]=0x3f3f3f3f;
        for(auto x:e) {
            if(x.w<in[x.v]) {
                pre[x.v]=x.u,in[x.v]=x.w;
                inn[x.v]=x.u;
            }
        }
        t=0x3f3f3f3f;
        for(int i=1;i<=n;i++) if(inn[i]==rt) {
            for(auto x:e) if(x.u==rt&&x.v==i&&in[x.v]==x.w) t=min(t,x.num);
        }
        int tn=0;
        for(int i=1;i<=nn;i++) id[i]=st[i]=-1;
        in[rt]=0;
        for(int i=1;i<=n;i++) {
            ret+=in[v=i];
            while(st[v]!=i&&id[v]==-1&&v!=rt) st[v]=i,v=pre[v];
            if(v!=rt&&id[v]==-1) {
                ++tn;
                for(int u=pre[v];u!=v;u=pre[u]) id[u]=tn;
                id[v]=tn;
            }
        }
        if(!tn) break;
    }
}

```

```

        for(int i=1;i<=n;i++) if(id[i]==-1) id[i]=++tn;
        for(int i=0;i<SZ(e);) {
            auto &x=e[i];
            v=x.v;
            x.u=id[x.u],x.v=id[x.v];
            if(x.u!=x.v) x.w-=in[v],++i;
            else swap(x,e.back()),e.pop_back();
        }
        n=tn,rt=id[rt];
    }
    return ret;
}

int main() {
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=m;i++) {
        int u,v,w;
        cin>>u>>v>>w;
        e.EB(u,v,w,0);
        sum+=w;
    }
    ++sum;
    for(int i=1;i<=n;i++) e.EB(n+1,i,sum,i);
    int res=edmonds(n+1,n+1);
    if(res==-1||res>=2*sum) cout<<"impossible"<<"\n";
    else cout<<res-sum<<" "<<t<<"\n";
    return 0;
}

```

欧拉图

通过图中所有边恰好一次且行遍所有顶点的通路称为欧拉通路

通过图中所有边恰好一次且行遍所有顶点的回路称为欧拉回路

具有欧拉回路的无向图称为欧拉图

具有欧拉通路但不具有欧拉回路的无向图称为半欧拉图

性质

欧拉图中所有顶点的度数都是偶数

若 G 是欧拉图，则它为若干个边不重的圈的并

若 G 是半欧拉图，则它为若干个边不重的圈和一条简单路径的并

判别法

对于无向图 G ， G 是欧拉图当且仅当 G 是连通的且没有奇度顶点

对于无向图 G ， G 是半欧拉图当且仅当 G 是连通的且 G 中恰有 0 个或 2 个奇度顶点

对于有向图 G ， G 是欧拉图当且仅当 G 的所有顶点属于同一个强连通分量且每个顶点的入度和出度相同

对于有向图 G ， G 是半欧拉图当且仅当

- 如果将 G 中的所有有向边退化为无向边时，那么 G 的所有顶点属于同一个连通分量。
- 有一个顶点的出度与入度差为 1，有一个顶点的入度与出度差为 1，其他顶点的入度和出度相同或者所有顶点入度和出度相同。

```

int cnt,nxt[M<<1],to[M<<1],head[N];
int top,stk[M<<1],t,ans[M<<1];
bool st[M<<1];
void init() {
    cnt=1;
    for(int i=1;i<=n;i++) head[i]=0;
    top=t=0;
}
void add_edge(int u,int v) {
    nxt[++cnt]=head[u];
    to[cnt]=v;
    head[u]=cnt;
}
void hierholzer(int s) {
    stk[++top]=s;
    while(top) {
        int u=stk[top],i=head[u];
        while(i&&st[i]) i=nxt[i];
        if(i) {
            stk[++top]=to[i];
            st[i]=st[i^1]=true;
            head[u]=nxt[i];
        }
        else top--,ans[++t]=u; // 倒序输出
    }
}

```

对于混合图 G ，先将无向边随意规定一个方向，计算出各个顶点的入度和出度。

欧拉图：

如果将 G 中的所有有向边退化为无向边时， G 的所有顶点不属于同一个连通分量或者入度出度差是奇数的顶点个数不是 0 个（对于无向边，无论方向如何，入度出度差奇偶性不变），则图 G 不可能是欧拉图。

接下来可以用网络流确定无向边方向判断图 G 是否是欧拉图。

建立起点 s 和终点 t ，将 s 与出度大于入度的顶点与相连，出度小于入度的顶点与 t 相连，容量为出度入度差的绝对值的 $\frac{1}{2}$ ，将之前定向的无向边之间建立一条容量为 1 的边，当网络流中与起点相连的边都满流时存在欧拉图。

我们的目的是调整无向边的方向使所有点的出度等于入度，假设网络流中经过 (a, b) 这条边，意味着 (a, b) 需要改变方向，使 a 的出度 -1 ，入度 $+1$ ， b 的入度 -1 ，出度 $+1$ ，所以网络流中与起点相连的边都满流时存在欧拉图。

半欧拉图：

如果将 G 中的所有有向边退化为无向边时， G 的所有顶点不属于同一个连通分量或者入度出度差是奇数的顶点个数不是 0 或 2 个，则图 G 不可能是半欧拉图。

将入度出度差是奇数的顶点之间加一条容量为 1 的边，这样则所有顶点入度出度差都为偶数，这样只要判断是否是欧拉图即可。

拓扑排序

```

int n,in[N];
VI g[N];
VI res; // 排序后的结果

```

```

void add_edge(int u,int v) {
    g[u].PB(v);
    in[v]++;
}
bool topo() {
    queue<int> q;
    for(int i=1;i<=n;i++) if(!in[i]) q.push(i);
    while(SZ(q)) {
        int u=q.front();
        q.pop();
        res.PB(u);
        for(auto v:g[u])
            if(--in[v]==0)
                q.push(v);
    }
    if(SZ(res)==n) return true;
    return false;
    // 如果res长度不等于结点个数说明存在环,无拓扑排序
}

```

连通性

边双连通分量

- 树形图至少添加 $(\text{叶子的数量 (若树根只有一个儿子, 树根也算叶子)} + 1) / 2$ 条边使其变为边双连通图

```

VI G[N];
void init() {
    for(int i=1;i<=n;i++) G[i].clear();
}
// 割边
int cnt,dfn[N],low[N],fa[N];
bool isb[N];
VI b;
void tarjan(int u) {
    dfn[u]=low[u]=++cnt;
    for(auto v:G[u]) {
        if(!dfn[v]) {
            fa[v]=u;
            tarjan(v);
            low[u]=min(low[u],low[v]);
            if(dfn[u]<low[v]) isb[v]=true,b.PB(v);
        }
        else if(v!=fa[u]) low[u]=min(low[u],dfn[v]);
    }
}
void getBridge() {
    cnt=0;
    b.clear();
    for(int i=1;i<=n;i++) dfn[i]=low[i]=fa[i]=0,isb[i]=false;
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
}
// 边双连通分量
int dcc,c[N];
void dfs(int u) {

```

```

c[u]=dcc;
for(auto v:G[u]) {
    if(c[v]||u==fa[v]&&isb[v]||v==fa[u]&&isb[u]) continue;
    dfs(v);
}
}
void shrink() {
    dcc=0;
    for(int i=1;i<=n;i++) c[i]=0;
    for(int i=1;i<=n;i++) if(!c[i]) ++dcc,dfs(i);
}
// 缩点建图
vector<int> g[N];
void build() { // 将边双连通分量缩点与桥形成一棵树,原图不连通则形成森林
    for(int i=1;i<=dcc;i++) g[i].clear();
    for(int i=0;i<b.size();i++) {
        int u=c[b[i]],v=c[fa[b[i]]];
        g[u].PB(v);
        g[v].PB(u);
    }
}
}

```

点双连通分量

```

VI G[N];
void init() {
    for(int i=1;i<=n;i++) G[i].clear();
}
// 点双连通分量
int root,cnt,cntDcc,top,dfn[N],low[N],stk[N];
bool cut[N];
VI dcc[N];
void tarjan(int u) {
    dfn[u]=low[u]=++cnt;
    stk[++top]=u;
    if(u==root&&!G[u].size()) { // 孤立点
        dcc[++cntDcc].PB(u);
        return;
    }
    int flag=0;
    for(auto v:G[u]) {
        if(!dfn[v]) {
            tarjan(v);
            low[u]=min(low[u],low[v]);
            if(dfn[u]<=low[v]) {
                flag++;
                if(u!=1||flag>1) cut[u]=true;
                ++cntDcc;
                int x;
                do {
                    x=stk[top--];
                    dcc[cntDcc].PB(x);
                } while(x!=v);
                dcc[cntDcc].PB(u);
            }
        }
        else low[u]=min(low[u],dfn[v]);
    }
}

```

```

    }
}
void shrink() {
    for(int i=1;i<=cntDcc;i++) dcc[i].clear();
    cnt=cntDcc=top=0;
    for(int i=1;i<=n;i++) dfn[i]=low[i]=0,cut[i]=false;
    for(int i=1;i<=n;i++) if(!dfn[i]) root=i,tarjan(i);
}
// 缩点建图
int num,id[N];
VI g[N];
void build() {
    num=cntDcc;
    for(int i=1;i<=n;i++) id[i]=0;
    for(int i=1;i<=n;i++) if(cut[i]) id[i]=++num;
    for(int i=1;i<=num;i++) g[i].clear();
    for(int i=1;i<=cntDcc;i++)
        for(auto x:dcc[i])
            if(id[x]) {
                g[id[x]].PB(i);
                g[i].PB(id[x]);
            }
}
}

```

强连通分量

```

VI G[N];
void init() {
    for(int i=1;i<=n;i++) G[i].clear();
}
// 强连通分量
int cnt,dfn[N],low[N],top,stk[N],cntScc,c[N];
bool ins[N];
VI scc[N];
void tarjan(int u) {
    dfn[u]=low[u]=++cnt;
    stk[++top]=u;
    ins[u]=true;
    for(auto v:G[u]) {
        if(!dfn[v]) {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(ins[v]) low[u]=min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u]) {
        ++cntScc;
        int x;
        do {
            x=stk[top--];
            ins[x]=false;
            c[x]=cntScc;
            scc[cntScc].PB(x);
        } while(u!=x);
    }
}
void shrink() {
}

```

```

    for(int i=1;i<=cntScc;i++) scc[i].clear();
    cnt=cntScc=top=0;
    for(int i=1;i<=n;i++) dfn[i]=low[i]=c[i]=0,ins[i]=false;
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
}
// 缩点建图
vector<int> g[N];
void build() {
    for(int i=1;i<=cntScc;i++) g[i].clear();
    for(int i=1;i<=n;i++)
        for(int j=0;j<SZ(G[i]);j++) {
            int u=c[i],v=c[G[i][j]];
            if(u==v) continue;
            g[u].PB(v);
        }
}
}

```

2 - SAT

给出 n 个集合，每个集合有两个元素，已知若干个 $\langle a, b \rangle$ ，表示 a 与 b 矛盾（其中 a 与 b 属于不同的集合）

然后从每个集合选择一个元素，判断能否一共选 n 个两两不矛盾的元素

显然可能有多种选择方案，一般题中只要求出一种即可

假设有两个集合 $\{a, \neg a\}, \{b, \neg b\}$

若 a 与 b 矛盾，则选了 a 就必须选 $\neg b$ ，选了 b 就必须选 $\neg a$ ，按此关系建立有向边（原命题和逆否命题是成对出现的）

若选 a 本身就是不合法的，那么在 a 和 $\neg a$ 之间建立一条有向边，让它直接产生矛盾

- SCC 缩点

若一个集合中的两个元素在同一个 SCC 中，无解

输出方案时可以通过变量在图中的反拓扑序确定该变量的取值

如果变量 $\neg x$ 的反拓扑序在 x 之后，那么取 x 值为真

```

int n,cnt,cnt_scc,top,dfn[N],low[N],c[N],stk[N];
int in[N],val[N],opp[N];
bool ins[N];
VI G[N],g[N];
void init() {
    cnt=cnt_scc=top=0;
    for(int i=1;i<=n*2;i++) {
        G[i].clear();
        g[i].clear();
        dfn[i]=low[i]=c[i]=in[i]=val[i]=0;
        ins[i]=false;
    }
}
void tarjan(int u) {
    dfn[u]=low[u]=++cnt;
    stk[++top]=u;
    ins[u]=true;
    for(auto v:G[u]) {

```

```

        if(!dfn[v]) {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(ins[v]) low[u]=min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u]) {
        cnt_scc++;
        int x;
        do {
            x=stk[top--];
            ins[x]=false;
            c[x]=cnt_scc;
        } while(u!=x);
    }
}

void rebuild() {
    for(int i=1;i<=n*2;i++) {
        for(auto x:G[i]) {
            int u=c[i],v=c[x];
            if(u==v) continue;
            in[u]++;
            g[v].PB(u);
        }
    }
}

void topo() {
    queue<int> q;
    for(int i=1;i<=cnt_scc;i++) if(!in[i]) q.push(i);
    while(SZ(q)) {
        int u=q.front();
        q.pop();
        if(!val[u]) val[u]=1,val[opp[u]]=2;
        for(auto v:g[u]) {
            in[v]--;
            if(!in[v]) q.push(v);
        }
    }
}

void solve() {
    for(int i=1;i<=n*2;i++) if(!dfn[i]) tarjan(i);
    for(int i=1;i<=n;i++) {
        if(c[i]==c[i+n]) {puts("NO");return;}
        opp[c[i]]=c[i+n],opp[c[i+n]]=c[i];
    }
    rebuild();
    topo();
    for(int i=1;i<=n;i++) printf("%d\n",val[i]<val[i+n]?0:1);
}

```

因为 Tarjan 算法求强连通分量时使用了栈，所以 Tarjan 求得的 SCC 编号相当于反拓扑序，所在 SCC 编号在 $\neg x$ 之前时，取 x 为真

```
for(int i=1;i<=n;i++) printf("%d\n",c[i]<c[i+n]?0:1);
```

- 爆搜


```

// HDU 1814
// 按字典序输出方案
#include <bits/stdc++.h>
using namespace std;
const int N=16005;
int n,m,top,stk[N];
bool st[N];
vector<int> g[N];
bool dfs(int u) {
    st[u]=true;
    stk[++top]=u;
    for(auto v:g[u]) {
        if(st[v^1]) return false;
        if(st[v]) continue;
        if(!dfs(v)) return false;
    }
    return true;
}
int main() {
    while(~scanf("%d%d",&n,&m)) {
        for(int i=0;i<n*2;i++) g[i].clear(),st[i]=false;
        for(int i=1;i<=m;i++) {
            int u,v;
            scanf("%d%d",&u,&v);
            u--,v--;
            g[u].push_back(v^1);
            g[v].push_back(u^1);
        }
        bool f=false;
        for(int i=0;i<n*2;i+=2) {
            if(st[i]||st[i^1]) continue;
            top=0;
            if(!dfs(i)) {
                while(top) st[stk[top--]]=false;
                if(!dfs(i^1)) {f=true;break;}
            }
        }
        if(f) puts("NIE");
        else for(int i=0;i<n*2;i++) if(st[i]) printf("%d\n",i+1);
    }
    return 0;
}

```

重链剖分

- 所有重链长度和等于节点数。
- 一个点到另一个点经过的轻链最多有 $\log_2 n$ 条。
- 一个点到另一个点经过的重链最多有 $\log_2 n$ 条。
- 每条链的 DFS 序是连续的。

```

int cnt,sz[N],dep[N],fa[N],top[N],son[N],dfn[N];
void dfs1(int u) {
    dep[u]=dep[fa[u]]+1;
    sz[u]=1;
    for(auto v:g[u]) if(v!=fa[u]) {
        fa[v]=u;

```

```

        dfs1(v);
        sz[u]+=sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v;
    }
}
void dfs2(int u,int t) {
    top[u]=t;
    dfn[u]++;cnt;
    if(son[u]) dfs2(son[u],t);
    for(auto v:g[u]) if(!top[v]) {
        dfs2(v,v);
    }
}
}

```

树分治

点分治

[CF 161D](#): 节点对 (u, v) 和节点对 (v, u) 被认为是相同的节点对, 求树上距离恰好为 k 的不同节点对的数量。

```

int n,k,rt,cnt,sz[N],mx[N],dist[N],a[N],tot[N];
LL res;
bool st[N];
VI g[N];
void get_rt(int u,int fa,int sum) {
    sz[u]=1;
    mx[u]=0;
    for(auto v:g[u]) if(v!=fa&&!st[v]) {
        get_rt(v,u,sum);
        sz[u]+=sz[v];
        mx[u]=max(mx[u],sz[v]);
    }
    mx[u]=max(mx[u],sum-sz[u]);
    if(mx[u]<mx[rt]) rt=u;
}
void get_dist(int u,int fa) {
    if(dist[u]>k) return;
    a[++cnt]=dist[u];
    for(auto v:g[u]) if(v!=fa&&!st[v]) {
        dist[v]=dist[u]+1;
        get_dist(v,u);
    }
}
LL calc(int u,int t) {
    dist[u]=t;
    cnt=0;
    get_dist(u,0);
    for(int i=0;i<=k;i++) tot[i]=0;
    for(int i=1;i<=cnt;i++) ++tot[a[i]];
    LL ret=0;
    for(int i=1;i<=cnt;i++) {
        --tot[a[i]];
        ret+=tot[k-a[i]];
    }
    return ret;
}

```

```

}
void divide(int u) {
    mx[rt=0]=INT_MAX;
    get_rt(u,0,sz[u]);
    st[u=rt]=true;
    res+=calc(u,0);
    for(auto v:g[u]) if(!st[v]) {
        res-=calc(v,1);
        divide(v);
    }
}
int main() {
    cin>>n>>k;
    for(int i=1;i<n;i++) {
        int u,v;cin>>u>>v;
        g[u].PB(v);
        g[v].PB(u);
    }
    sz[1]=n;divide(1);
    cout<<res<<'\n';
    return 0;
}

```

树上启发式合并

[CF 600E](#): 一棵树有 n 个结点，每个结点都是一种颜色，每个颜色有一个编号，求树中每个子树的最多的颜色编号的和。

用 cnt_i 表示颜色 i 的出现次数。

遍历一个节点 u ，我们按以下的步骤进行遍历：

- 先遍历 u 的轻（非重）儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；
- 遍历它的重儿子，保留它对 cnt 数组的影响；
- 再次遍历 u 的轻儿子的子树结点，加入这些结点的贡献，以得到 u 的答案。

时间复杂度： $O(n \log(n))$

```

VI g[N];
int w[N],sz[N],son[N],cnt[N],mx,sn;
LL sum,res[N];
void dfs(int u,int fa) {
    sz[u]=1;
    for(auto v:g[u]) if(v!=fa) {
        dfs(v,u);
        sz[u]+=sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v;
    }
}
void update(int u,int fa,int val) {
    cnt[w[u]]+=val;
    if(cnt[w[u]]>mx) mx=cnt[w[u]],sum=w[u];
    else if(cnt[w[u]]==mx) sum+=w[u];
    for(auto v:g[u]) if(v!=fa&&v!=sn) {
        update(v,u,val);
    }
}
void dsu(int u,int fa,bool op) {

```

```

    for(auto v:g[u]) if(v!=fa&&v!=son[u]) {
        dsu(v,u,true);
    }
    if(son[u]) dsu(son[u],u,false),sn=son[u];
    update(u,fa,1);
    res[u]=sum;
    sn=0;
    if(op) update(u,fa,-1),mx=0,sum=0;
}
int main() {
    int n;cin>>n;
    for(int i=1;i<=n;i++) cin>>w[i];
    for(int i=1;i<n;i++) {
        int u,v;
        cin>>u>>v;
        g[u].PB(v);
        g[v].PB(u);
    }
    dfs(1,0);
    dsu(1,0,true);
    for(int i=1;i<=n;i++) cout<<res[i]<<" \n"[i==n];
    return 0;
}

```

LCA

重链剖分算法

```

int fa[N],dep[N],fa[N],son[N],top[N];
VI g[N];
void dfs1(int u) {
    dep[u]=dep[fa[u]]+1;
    sz[u]=1;
    for(auto v:g[u]) {
        if(v==fa[u]) continue;
        fa[v]=u;
        dfs1(v);
        sz[u]+=sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v;
    }
}
void dfs2(int u,int t) {
    top[u]=t;
    if(son[u]) dfs2(son[u],t);
    for(auto v:g[u]) {
        if(v==son[u]||v==fa[u]) continue;
        dfs2(v,v);
    }
}
int LCA(int a,int b) {
    while(top[a]!=top[b]) {
        if(dep[top[a]]>dep[top[b]]) a=fa[top[a]];
        else b=fa[top[b]];
    }
    return dep[a]<dep[b]?a:b;
}

```

树上倍增算法

```
int dep[N], f[N][20];
VI g[N];
void dfs(int u) {
    dep[u] = dep[f[u][0]] + 1;
    for(auto v: g[u]) if(v != f[u][0]) {
        f[v][0] = u;
        for(int i = 1; i <= LIM; i++) f[v][i] = f[f[v][i-1]][i-1];
        dfs(v);
    }
}
int LCA(int a, int b) {
    if(dep[a] > dep[b]) swap(a, b);
    for(int i = LIM; ~i; i--) if(dep[f[b][i]] >= dep[a]) b = f[b][i];
    if(a == b) return b;
    for(int i = LIM; ~i; i--) if(f[a][i] != f[b][i]) a = f[a][i], b = f[b][i];
    return f[a][0];
}
```

树上 Tarjan 算法

```
void init() {
    for(int i = 1; i <= n; i++) fa[i] = i;
}
void add_query(int a, int b, int id) {
    query[a].EB(b, id);
    query[b].EB(a, id);
}
int get(int x) {
    if(fa[x] == x) return x;
    return fa[x] = get(fa[x]);
}
void tarjan(int u) {
    st[u] = 1;
    for(auto v: g[u]) {
        if(st[v]) continue;
        tarjan(v);
        fa[v] = u;
    }
    for(int i = 0; i < SZ(query[u]); i++) {
        int v = query[u][i].FI, id = query[u][i].SE;
        if(st[v] == 2) ans[id] = get(v);
    }
    st[u] = 2;
}
```

网络流

最大流

- 最大流最小割定理：网络流图中，最大流的值等于最小割的容量

```
const int INF=0x3f3f3f3f;
struct MAXIMUM_FLOW {
    int n,s,t,d[N],cur[N];
    VI g[N];
    VPII e;
    void init() {
        for(int i=1;i<=n;i++) g[i].clear();
        e.clear();
    }
    void add_edge(int u,int v,int c) {
        e.EB(v,c);
        e.EB(u,0);
        g[u].PB(SZ(e)-2);
        g[v].PB(SZ(e)-1);
    }
    bool bfs() {
        for(int i=1;i<=n;i++) d[i]=0;
        queue<int> q;
        q.push(s);
        d[s]=1;
        while(SZ(q)) {
            int u=q.front();
            q.pop();
            for(auto x:g[u]) {
                int v=e[x].FI,c=e[x].SE;
                if(d[v]==0 || c<=0) continue;
                d[v]=d[u]+1;
                q.push(v);
            }
        }
        return d[t];
    }
    int dfs(int u,int a) {
        if(u==t) return a;
        int f,flow=0;
        for(int &i=cur[u];i<SZ(g[u]);i++) {
            int v=e[g[u][i]].FI,&c=e[g[u][i]].SE;
            if(d[v]!=d[u]+1 || c<=0 || (f=dfs(v,min(a,c)))<=0) continue;
            c-=f;
            e[g[u][i]^1].SE+=f;
            a-=f;
            flow+=f;
            if(a==0) break;
        }
        return flow;
    }
    int dinic() {
        int flow=0;
        while(bfs()) {
            for(int i=1;i<=n;i++) cur[i]=0;
            flow+=dfs(s,INF);
        }
        return flow;
    }
}
```

```

    }
} mf;

```

最小费用最大流

```

const int INF=0x3f3f3f3f;
int dist[N],h[N],preu[N],pree[M];
VI g[N];
struct E {
    int v,c,w;
    E(){}
    E(int v,int c,int w):v(v),c(c),w(w){}
};
vector<E> e;
void init(int n) {
    for(int i=1;i<=n;i++) {
        h[i]=0;
        g[i].clear();
    }
    e.clear();
}
void add_edge(int u,int v,int c,int w) {
    e.EB(v,c,w);
    e.EB(u,0,-w);
    g[u].PB(SZ(e)-2);
    g[v].PB(SZ(e)-1);
}
bool dijkstra(int n,int s,int t) {
    for(int i=1;i<=n;i++) dist[i]=INF;
    priority_queue<PII,VPII,greater<PII>> q;
    dist[s]=0;
    q.emplace(0,s);
    while(SZ(q)) {
        int d=q.top().FI,u=q.top().SE;
        q.pop();
        if(dist[u]!=d) continue;
        for(auto x:g[u]) {
            int v=e[x].v,c=e[x].c,w=e[x].w;
            if(c>0&&dist[v]>dist[u]-h[v]+w+h[u]) {
                dist[v]=dist[u]-h[v]+w+h[u];
                preu[v]=u;
                pree[v]=x;
                q.emplace(dist[v],v);
            }
        }
    }
    return dist[t]!=INF;
}
PII mcmf(int n,int s,int t) {
    int flow=0,cost=0;
    while(dijkstra(n,s,t)) {
        int c=INF;
        for(int i=1;i<=n;i++) h[i]=min(INF,h[i]+dist[i]);
        for(int u=t;u!=s;u=preu[u]) c=min(c,e[pree[u]].c);
        flow+=c;
        cost+=c*h[t];
        for(int u=t;u!=s;u=preu[u]) {

```

```

        e[pree[u]].c-=c;
        e[pree[u]^1].c+=c;
    }
}
return MP(flow, cost);
}

```

二分图最大匹配

- 定理：一张无向图是二分图当且仅当图中不存在奇环。

匈牙利算法

时间复杂度： $O(n^2)$

```

int n, match[N];
bool m[N][N], st[N];
bool dfs(int u) {
    for(int v=1; v<=n; v++) {
        if(!m[u][v] || st[v]) continue;
        st[v]=true;
        if(match[v]==-1 || dfs(match[v])) {
            match[v]=u;
            return true;
        }
    }
    return false;
}
int hungary() {
    for(int i=1; i<=n; i++) match[i]=-1;
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) st[j]=false;
        if(dfs(i)) res++;
    }
    return res;
}

```

HK 算法

- 时间复杂度： $O(\sqrt{nm})$

```

int n, m, match[N<<1], dep[N<<1];
VI g[N];
bool bfs() {
    queue<int> q;
    for(int i=1; i<=n+m; i++) dep[i]=0;
    for(int i=1; i<=n; i++) if(match[i]==-1) dep[i]=1, q.push(i);
    bool f=false;
    while(sz(q)) {
        int u=q.front();
        q.pop();
        for(auto v:g[u]) {
            if(dep[v]) continue;
            dep[v]=dep[u]+1;
            if(match[v]==-1) f=true;
            else dep[match[v]]=dep[v]+1, q.push(match[v]);
        }
    }
    return f;
}

```



```

    }
}
return f;
}
bool dfs(int u) {
    for(auto v:g[u]) {
        if(dep[v]!=dep[u]+1) continue;
        dep[v]=0;
        if(match[v]==-1||dfs(match[v])) {
            match[v]=u;
            match[u]=v;
            return true;
        }
    }
    return false;
}
int HK() {
    for(int i=1;i<=n+m;i++) match[i]=-1;
    int res=0;
    while(bfs()) for(int i=1;i<=n;i++) if(match[i]==-1&&dfs(i)) res++;
    return res;
}

```

二分图最小点覆盖

图的最小点覆盖就是求出一个最小点集 S ，使得图中任意一条边都有至少一个端点属于 S

- Konig 定理：二分图最小点覆盖包含的点数等于二分图最大匹配包含的边数

二分图最大独立集

图的最大独立集就是“任意两点之间都没有边相连”的最大点集

- 定理：二分图中，最大独立集 = n - 最小点覆盖

对应地，“任意两点之间都有一条边相连”的子图被称为无向图的“团”，点数最多的团被称为图的最大团

- 定理：无向图 G 的最大团等于其补图 G' 的最大独立集

有向无环图的最小不相交路径覆盖

- 有向无环图 G 的最小不相交路径覆盖包含的路径条数 = 点数 - 拆点二分图 G_2 的最大匹配数

有向无环图的最小可相交路径覆盖

- 有向无环图 G 的最小可相交路径覆盖等价于先对有向图传递闭包，得到有向无环图 G' ，再在 G' 上求最小不相交路径覆盖

无向图的最小路径覆盖

- 无向图 G 的最小路径覆盖包含的路径条数 = 点数 - 拆点二分图 G_2 的最大匹配数 / 2

二分图最大权匹配

KM 算法

KM 算法只能在满足带权最大匹配一定是完美匹配的图中求解。

两边点数不同，需增加虚点让两边点数相同，虚点与另一边的所有点相连构成虚边。

若题目允许不完美匹配，虚边边权设为 0，否则为 $-\infty$ 。

注意根据题意考虑邻接矩阵初始值是 0 还是 $-\infty$ 。

```
int w[N][N], la[N], lb[N], ma[N], mb[N], vb[N], slk[N], pre[N];
int km(int n) {
    for(int i=1; i<=n; i++) {
        la[i]=0xc0c0c0c0;
        lb[i]=ma[i]=mb[i]=0;
        for(int j=1; j<=n; j++) la[i]=max(la[i], w[i][j]);
    }
    for(int i=1; i<=n; i++) {
        for(int j=0; j<=n; j++) vb[j]=pre[j]=0, slk[j]=0x3f3f3f3f;
        int b=0, p=-1;
        for(mb[b]=i; mb[b]; b=p) {
            int d=0x3f3f3f3f, a=mb[b];
            vb[b]=1;
            for(int j=1; j<=n; j++)
                if(!vb[j]) {
                    int t=la[a]+lb[j]-w[a][j];
                    if(t<slk[j]) slk[j]=t, pre[j]=b;
                    if(slk[j]<d) d=slk[j], p=j;
                }
            for(int j=0; j<=n; j++) {
                if(vb[j]) la[mb[j]]-=d, lb[j]+=d;
                else slk[j]-=d;
            }
        }
        for(; b=pre[b];) mb[b]=mb[pre[b]], ma[mb[b]]=b;
    }
    int res=0;
    for(int i=1; i<=n; i++) res+=w[i][ma[i]];
    return res;
    // nn 为二分图左边集合大小，不存在完美匹配返回 -1
    // int res=0, cnt=0;
    // for(int i=1; i<=nn; i++) {
    //     if(ma[i]&&w[i][ma[i]]!=0xc0c0c0c0) {
    //         ++cnt;
    //         res+=w[i][ma[i]];
    //     }
    // }
    // return cnt==nn?res:-1;
}
```

- HDU 2853

二分图两边集合大小为 n, m ，给定匹配边，改变最少的匹配边使二分图权值匹配最大。

将权值扩大 k ($k > n$) 倍，再将原匹配边权值 $+1$ ，跑 KM 算法，新二分图最大权值匹配 $/k$ 为原二分图最大权值匹配，最少改变的匹配边为 $n - \text{新二分图最大权值匹配} \% k$ 。

一般图最大匹配

```

int n,m,cnt,fa[N],vis[N],pre[N],dfn[N],match[N];
VI g[N];
queue<int> q;
void init() {
    for(int i=1;i<=n;i++) g[i].clear(),match[i]=dfn[i]=0;
    cnt=0;
}
void add_edge(int u,int v) {
    g[u].PB(v);
    g[v].PB(u);
}
int find(int x) {
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
int LCA(int u,int v) {
    ++cnt;
    u=find(u),v=find(v);
    while(dfn[u]!=cnt) {
        dfn[u]=cnt;
        u=find(pre[match[u]]);
        if(v) swap(u,v);
    }
    return u;
}
void blossom(int u,int v,int lca) {
    while(find(u)!=lca) {
        pre[u]=v;
        v=match[u];
        if(vis[v]==2) vis[v]=1,q.push(v);
        if(find(u)==u) fa[u]=lca;
        if(find(v)==v) fa[v]=lca;
        u=pre[v];
    }
}
bool aug(int s) {
    for(int i=1;i<=n;i++) fa[i]=i,vis[i]=pre[i]=0;
    while(SZ(q)) q.pop();
    q.push(s);
    vis[s]=1;
    while(SZ(q)) {
        int u=q.front();
        q.pop();
        for(auto v:g[u]) {
            if(find(u)==find(v)||vis[v]==2) continue;
            if(!vis[v]) {
                vis[v]=2,pre[v]=u;
                if(!match[v]) {
                    for(int x=v,lst;x=x==lst) {
                        lst=match[pre[x]];
                        match[x]=pre[x];
                        match[pre[x]]=x;
                    }
                    return true;
                }
                vis[match[v]]=1,q.push(match[v]);
            }
        }
    }
    else {

```

```

        int lca=LCA(u,v);
        blossom(u,v,lca);
        blossom(v,u,lca);
    }
}
}
return false;
}
int edmonds() {
    int res=0;
    for(int i=1;i<=n;i++) if(!match[i]) res+=aug(i);
    return res;
}

```

数学

龟速乘

```

LL qmul(LL a,LL b,LL m) { // b >= 0
    LL ret=0;
    while(b) {
        if(b&1) ret=(ret+a)%m;
        a=(a+a)%m;
        b>>=1;
    }
    return ret;
}

```

快速幂

```

LL qpow(LL a,LL b,LL m) {
    LL ret=1;
    while(b) {
        if(b&1) ret=ret*a%m;
        a=a*a%m;
        b>>=1;
    }
    return ret;
}

```

- 防爆 LL

```

// 前置模板：龟速乘
LL qpow(LL a,LL b,LL m) {
    LL ret=1;
    while(b) {
        if(b&1) ret=qmul(ret,a,m);
        a=qmul(a,a,m);
        b>>=1;
    }
    return ret;
}

```

Miller-Rabin 素性测试

- 二次探测定理：如果 p 是奇素数，则 $x^2 \equiv 1 \pmod{p}$ 的解为 $x \equiv 1 \pmod{p}$ 或者 $x \equiv p-1 \pmod{p}$
- int 范围内只需检查 2, 7, 61
- long long 范围内 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- $3e15$ 内 2, 2570940, 880937, 610386380, 4130785767
- $4e13$ 内 2, 2570940, 211991001, 3749873356
- <http://miller-rabin.appspot.com>

```
// 前置模板：龟速乘、快速幂
bool check(LL a, LL n) {
    if(n==2) return true;
    if(n==1 || !(n&1)) return false;
    LL d=n-1;
    while(!(d&1)) d>>=1;
    LL t=qpow(a,d,n);
    while(d!=n-1&&t!=1&&t!=n-1) {
        t=qmul(t,t,n);
        d<<=1;
    }
    return t==n-1 || d&1;
}

bool miller_rabin(LL n) {
    static vector<LL> P={2,325,9375,28178,450775,9780504,1795265022};
    if(n<=1) return false;
    for(LL x:P) {
        if(x>n) break;
        if(!check(x,n)) return false;
    }
    return true;
}
```

Pollard-Rho 因式分解

```
// 前置模板：Miller-Rabin 素性测试
mt19937 mt(time(0));
LL pollard_rho(LL n, LL c) {
    LL x=uniform_int_distribution<LL>(1,n-1)(mt), y=x;
    auto f=[&](LL v) {
        LL t=qmul(v,v,n)+c;
        return t<n?t:n;
    };
    while(1) {
        x=f(x), y=f(f(y));
        if(x==y) return n;
        LL d=__gcd(abs(x-y),n);
        if(d!=1) return d;
    }
}

VI fac; // 无序，有重复质因数
void get_fac(LL n, LL cc=19260817) {
    if(n==4) {fac.pb(2), fac.pb(2); return;}
    if(miller_rabin(n)) {fac.pb(n); return;}
    LL p=n;
    while(1) {
        LL d=pollard_rho(p, cc);
        fac.pb(d);
        p/=d;
    }
}
```

```

while(p==n) p=pollard_rho(n,--cc);
get_fac(p),get_fac(n/p);
}
void go_fac(LL n) {
    fac.clear();
    if(n<=1) return;
    get_fac(n);
}

```

积性函数

定义

若函数 $f(n)$ 满足 $f(1) = 1$ 且 $\forall x, y \in \mathbb{N}^+, \gcd(x, y) = 1$ 都有 $f(xy) = f(x)f(y)$, 则 $f(n)$ 为积性函数。

若函数 $f(n)$ 满足 $f(1) = 1$ 且 $\forall x, y \in \mathbb{N}^+$ 都有 $f(xy) = f(x)f(y)$, 则 $f(n)$ 为完全积性函数。

性质

若 $f(x)$ 和 $g(x)$ 均为积性函数, 则以下函数也为积性函数:

- $h(x) = f(x^p)$
- $h(x) = f^p(x)$
- $h(x) = f(x)g(x)$
- $h(x) = \sum_{d|x} f(x)g(\frac{x}{d})$

设 $x = \prod p_i^{k_i}$,

- 若 $F(x)$ 为积性函数, 则有 $F(x) = \prod F(p_i^{k_i})$ 。
- 若 $F(x)$ 为完全积性函数, 则有 $F(x) = \prod F(p_i)^{k_i}$ 。

定理

定理 1.1:

如果 f 是积性函数, 那么 $F(n) = \sum_{d|n} f(d)$ 也是积性函数。

证明:

当 $\gcd(n, m) = 1$ 时, nm 的因子必能写成 n 的因子 d_1 与 m 的因子 d_2 之积,

所以 $F(nm) = \sum_{i|n} \sum_{j|m} f(ij) = \sum_{i|n} f(i) \sum_{j|m} f(j) = F(n)F(m)$ 。

例子

- 单位函数: $\varepsilon(n) = [n = 1]$ (完全积性)
- 恒等函数: $\text{id}_k(n) = n^k$, $\text{id}_1(n)$ 通常简记作 $\text{id}(n)$ (完全积性)
- 常数函数: $1(n) = 1$ (完全积性)
- 除数函数: $\sigma_k(n) = \sum_{d|n} d^k$, $\sigma_0(n)$ 通常简记为 $d(n)$ 或 $\tau(n)$, $\sigma_1(n)$ 通常简记为 $\sigma(n)$
- 欧拉函数: $\varphi(n) = \sum_{i=1}^n [\gcd(i, n) = 1]$
- 莫比乌斯函数: $\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 | n \\ (-1)^k & k \text{ 为 } n \text{ 的不同质因子个数} \end{cases}$

Dirichlet 卷积

定义

对于两个数论函数 $f(x)$ 和 $g(x)$, 则它们的 Dirichlet 卷积得到的结果 $h(x)$ 定义为:

$$h(x) = \sum_{d|x} f(d)g(x/d) = \sum_{ab=x} f(a)g(b)$$

简记为: $h = f * g$.

性质

- 交换律: $f * g = g * f$.
- 结合律: $(f * g) * h = f * (g * h)$.
- 分配律: $(f + g) * h = f * h + g * h$.
- 等式的性质: $f = g$ 的充要条件是 $f * g = g * h$, 其中数论函数 $h(x)$ 要满足 $h(1) \neq 0$.
- 单位元: 单位函数 ε 是 Dirichlet 卷积运算中的单位元, 即对于任何数论函数 f , 都有 $f * \varepsilon = f$.

定理

定理 2.1:

两个积性函数的 Dirichlet 卷积也是积性函数。

证明:

设两个积性函数为 $f(x)$ 和 $g(x)$, 再记 $h = f * g$.

设 $\gcd(a, b) = 1$, 则 $h(a) = \sum_{d_1|a} f(d_1)g(a/d_1)$, $h(b) = \sum_{d_2|b} f(d_2)g(b/d_2)$,

所以 $h(a)h(b) = \sum_{d_1|a} f(d_1)g(a/d_1) \sum_{d_2|b} f(d_2)g(b/d_2) = \sum_{d|ab} f(d)g(ab/d) = h(ab)$.

例子

- $\varepsilon = \mu * 1 \iff \varepsilon(n) = \sum_{d|n} \mu(d)$
- $d = 1 * 1 \iff d(n) = \sum_{d|n} 1$
- $\sigma = \text{id} * 1 \iff \sigma(n) = \sum_{d|n} d$
- $\text{id} = \varphi * 1 \iff n = \sum_{d|n} \varphi(d)$
- $\varphi = \text{id} * \mu \iff \varphi(n) = \sum_{d|n} \mu(d)(n/d)$

欧拉函数

定理

定理 3.1:

$$n = \sum_{d|n} \varphi(d).$$

证明:

$f(n) = \sum_{d|n} \varphi(d)$ 为积性函数。

证明: 当 $\gcd(n, m) = 1$ 时,

$$\begin{aligned}
& f(n)f(m) \\
&= \sum_{i|n} \varphi(i) \sum_{j|m} \varphi(j) \\
&= \sum_{i|n} \sum_{j|m} (\varphi(i)\varphi(j)) \\
&= \sum_{i|n} \sum_{j|m} \varphi(ij) \\
&= \sum_{d|nm} \varphi(d) \\
&= f(nm)
\end{aligned}$$

证毕。

那么根据算术基本定理 $n = p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k}$,

由 f 是积性函数可知 $f(n) = f(p_1^{c_1})f(p_2^{c_2}) \cdots f(p_k^{c_k})$,

又因为

$$f(p^c) = \varphi(1) + \varphi(p) + \varphi(p^2) + \cdots + \varphi(p^c) = 1 + (p-1) + (p^2-p) + \cdots + (p^c - p^{c-1}) = p^c$$

所以 $f(n) = f(p_1^{c_1})f(p_2^{c_2}) \cdots f(p_k^{c_k}) = p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k} = n$ 。

定理 3.2:

$$\varphi(n) = \sum_{d|n} (\mu(d)(n/d)).$$

证明:

根据定理 3.1 可得 $\varphi * 1 = \text{id} \iff \varphi * \varepsilon = \text{id} * \mu \iff \varphi(n) = \sum_{d|n} (\mu(d)(n/d))$ 。

莫比乌斯函数

定义

μ 为莫比乌斯函数，定义为：

$$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 | n \\ (-1)^k & k \text{ 为 } n \text{ 的不同质因子个数} \end{cases}$$

定理

定理 4.1:

$\mu(n)$ 是积性函数。

证明:

假设 $\gcd(n, m) = 1$

- 当 $n = 1$ 或 $m = 1$ 时，若 $n = 1$, $\mu(nm) = \mu(n)\mu(m) = \mu(m)$, $m = 1$ 时同理；
- 当 n 和 m 中至少有一个数含有平方质因子时， nm 也有平方质因子， $\mu(nm) = \mu(n)\mu(m) = 0$ ；
- 当 n, m 都没有平方质因子且都不为 1 时，假设 n 有 p 个不同质因子个数， m 有 q 个不同质因子个数， $\mu(nm) = (-1)^{p+q} = (-1)^p + (-1)^q = \mu(n)\mu(m)$ 。

定理 4.2:

$$F(n) = \sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n > 1 \end{cases} = \varepsilon(n) = [n = 1]$$

证明:

- 当 $n = 1$ 时, $F(1) = \sum_{d|1} \mu(d) = \mu(1) = 1$;
- 当 $n > 1$ 时, 因为 $\mu(n)$ 是积性函数, 所以 $\mu(n)$ 的因数和函数 $F(n) = \sum_{d|n} \mu(d)$ 也是积性函数,

现在假设 p 是质数, $F(n) = F(p_1^{k_1})F(p_2^{k_2}) \cdots F(p_t^{k_t})$,

而

$$F(p^k) = \sum_{d|p^k} \mu(d) = \mu(1) + \mu(p) + \mu(p^2) + \cdots + \mu(p^k) = 1 + (-1) + 0 + \cdots + 0 = 0$$

,

所以 $F(n) = 0$ 。

定理 4.3:

若 f 是数论函数, $F(n) = \sum_{d|n} f(d)$, 则 $f(n) = \sum_{d|n} \mu(d)F(n/d)$ 。

证明:

$$\sum_{d|n} \mu(d)F(n/d) = \sum_{d|n} \mu(d) \sum_{e|(n/d)} f(e) = \sum_{e|n} f(e) \sum_{d|(n/e)} \mu(d),$$

根据定理 4.2, 当 $e = n$ 时, $\sum_{d|(n/e)} \mu(d) = 1$, 否则为 0,

所以 $\sum_{e|n} (f(e) \sum_{d|(n/e)} \mu(d)) = f(n)$ 。

筛

线性筛

- $1 \sim n$ 的素数个数近似为 $\frac{n}{\ln n}$

```
int mn[N];
VI p;
void prime_seive(int n) {
    for(int i=2; i<=n; i++) {
        if(!mn[i]) mn[i]=i, p.pb(i);
        for(auto x:p) {
            if(x>mn[i] || x*i>n) break;
            mn[i*x]=x;
        }
    }
}
```

线性筛 & 欧拉函数

$$\varphi(N) = N * \prod_{p|N} (1 - \frac{1}{p}) \quad (p \text{ 为质数})$$

```

int mn[N], phi[N];
VI p;
void get_phi(int n) {
    phi[1]=1;
    for(int i=2; i<=n; i++) {
        if(!mn[i]) mn[i]=i, p.pb(i), phi[i]=i-1;
        for(auto x:p) {
            if(x>mn[i] || x*i>n) break;
            mn[i*x]=x;
            phi[i*x]=i%x?phi[i]*(x-1):phi[i]*x;
        }
    }
}

```

线性筛 & 莫比乌斯函数

```

int mn[N], mu[N];
VI p;
void get_mu(int n) {
    mu[1]=1;
    for(int i=2; i<=n; i++) {
        if(!mn[i]) mn[i]=i, mu[i]=-1, p.pb(i);
        for(auto x:p) {
            if(x*i>n) break;
            mn[x*i]=x;
            if(i%x==0) {mu[x*i]=0; break;}
            mu[x*i]=-mu[i];
        }
    }
}

```

杜教筛

求 $S(n) = \sum_{i=1}^n f(i)$, 其中 f 是一个数论函数。

构造一个积性函数 g , 求 $g * f$ 的前缀和:

$$\begin{aligned}
 \sum_{i=1}^n (g * f)(i) &= \sum_{i=1}^n \sum_{d|i} g(d) f\left(\frac{i}{d}\right) \\
 &= \sum_{d=1}^n g(d) \sum_{d|i} f\left(\frac{i}{d}\right) \\
 &= \sum_{d=1}^n g(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i) \\
 &= \sum_{d=1}^n g(d) S\left(\left\lfloor \frac{n}{d} \right\rfloor\right)
 \end{aligned}$$

容斥一下, 有

$$g(1)S(n) = \sum_{i=1}^n g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right) - \sum_{i=2}^n g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right) = \sum_{i=1}^n (g * f)(i) - \sum_{i=2}^n g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right).$$

前半部分是 Dirichlet 卷积的前缀和, 后半部分可以数论分块求。

我们要做的是构造一个 g 使 $g * f$ 的前缀和好算。

例子:

$$1. \sum_{i=1}^n \mu(i)$$

由 $\sum_{d|n} \mu(d) = [n=1]$, 选择 $g = 1(i)$, 那么 $\sum_{i=1}^n (1 * \mu)(i) = 1$, 所以 $S(n) = \sum_{i=1}^n \mu(i) = 1 - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ 。

$$2. \sum_{i=1}^n \varphi(i)$$

由 $\sum_{d|n} \varphi(d) = n$, 选择 $g = 1(i)$, 那么 $\sum_{i=1}^n (1 * \varphi)(i) = \frac{n(n+1)}{2}$, 所以 $S(n) = \sum_{i=1}^n \varphi(i) = \frac{n(n+1)}{2} - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ 。

要能够由此计算 $S(n)$, 对 f, g 有一些要求:

- $f * g$ 能够快速求前缀和。
- g 能够快速求分段和 (前缀和)。

线性筛预处理 S 的前 $n^{\frac{2}{3}}$ 项, 剩余部分的时间复杂度为 $O(n^{\frac{2}{3}})$, 对于较大的值, 用 map 存其对应的值。

```
LL mn[N], phi[N], mu[N];
VI p;
unordered_map<LL, LL> _phi, _mu;
void init(int n) {
    phi[1] = mu[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!mn[i]) mn[i] = i, mu[i] = -1, phi[i] = i - 1, p.pb(i);
        for(auto x : p) {
            if(x * i > n) break;
            mn[x * i] = x;
            if(i % x == 0) {mu[x * i] = 0, phi[x * i] = phi[i] * x; break;}
            mu[x * i] = -mu[i];
            phi[x * i] = i % x ? phi[i] * (x - 1) : phi[i] * x;
        }
    }
    for(int i = 1; i <= n; i++) phi[i] += phi[i - 1], mu[i] += mu[i - 1];
}
LL calc_phi(LL n) {
    if(n <= LIM) return phi[n];
    if(_phi.count(n)) return _phi[n];
    LL ret = 0;
    for(LL l = 2, r; l <= n; l = r + 1) {
        r = n / (n / l);
        ret += (r - l + 1) * calc_phi(n / l);
    }
    return _phi[n] = n * (n + 1) / 2 - ret;
}
LL calc_mu(LL n) {
    if(n <= LIM) return mu[n];
    if(_mu.count(n)) return _mu[n];
    LL ret = 0;
    for(LL l = 2, r; l <= n; l = r + 1) {
        r = n / (n / l);
        ret += (r - l + 1) * calc_mu(n / l);
    }
    return _mu[n] = 1 - ret;
}
```

欧拉定理

- 欧拉定理: $a^{\varphi(m)} \equiv 1 \pmod{m} \ (a, m \in \mathbb{Z}^+, a \perp m)$
- 费马小定理: $a^{p-1} \equiv 1 \pmod{p} \ (p \text{ 为质数}, a \perp p)$
- 扩展欧拉定理:

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)} & a \perp m \\ a^b & a \nmid m, b < \varphi(m) \\ a^{b \bmod \varphi(m) + \varphi(m)} & a \nmid m, b \geq \varphi(m) \end{cases} \pmod{m}$$

也可表示为:

$$a^b \equiv \begin{cases} a^b & b < \varphi(m) \\ a^{b \bmod \varphi(m) + \varphi(m)} & b \geq \varphi(m) \end{cases} \pmod{m}$$

- 若 $a \perp m$, 则满足 $a^x \equiv 1 \pmod{m}$ 的最小正整数 x_0 是 $\varphi(m)$ 的约数

扩展欧几里得算法

裴蜀定理: 对于任意整数 a, b , 存在一对整数, 满足 $ax + by = \gcd(a, b)$

即 $ax + by = c$, 当 $\gcd(a, b) \mid c$, 等式成立

- 若 x', y' 为方程 $ax + by = \gcd(a, b)$ 的一组解, 则该方程的任意解表示为:
 $x = x' + kb / \gcd(a, b), y = y' - ka / \gcd(a, b)$, 且对任意整数 k 都成立
- 设 $t = b / \gcd(a, b)$, 那么 x 的最小非负整数解 $x = (x' \% t + t) \% t$

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(!b) {x=1, y=0; return a;}
    LL ret=exgcd(b, a%b, y, x);
    y-=a/b*x;
    return ret;
}
```

线性同余方程

形如 $ax \equiv b \pmod{m}$ 的方程被称为线性同余方程

方程 $ax \equiv b \pmod{m}$ 等价于方程 $ax + my = b$, 有整数解的充要条件为 $\gcd(a, m) \mid b$

乘法逆元

如果一个线性同余方程 $ax \equiv 1 \pmod{m}, a \perp m$, 则称 x 为 $a \bmod m$ 的乘法逆元, 记作 a^{-1}

- 当模数 m 为质数时, a^{m-2} 为 a 的乘法逆元
- 当模数 m 为合数时, 通过求解同余方程 $ax \equiv 1 \pmod{m}$ 可得到乘法逆元

```
// 前置模板: 扩展欧几里得
LL get_inv(LL a, LL m) {
    LL x, y;
    if(exgcd(a, m, x, y) != 1) return -1;
    return (x%m+m)%m;
}
```

- 预处理 $1 \sim n$ 的乘法逆元

```
LL inv[N];
void init_inv(LL n) {
    inv[1]=1;
    for(LL i=2;i<=n;i++) inv[i]=(MOD-MOD/i)*inv[MOD%i]%MOD;
}
```

- 预处理 $1 \sim n$ 的阶乘及其乘法逆元

```
// 前置模板：快速幂
LL fac[N],invfac[N];
void init_invfac(LL n) {
    n=min(n,MOD-1);
    fac[0]=1;
    for(LL i=1;i<=n;i++) fac[i]=fac[i-1]*i%MOD;
    invfac[n]=qpow(fac[n],MOD-2);
    for(LL i=n-1;~i;i--) invfac[i]=invfac[i+1]*(i+1)%MOD;
}
```

中国剩余定理

中国剩余定理可求解如下形式的一元线性同余方程组（其中 m_1, m_2, \dots, m_k 两两互质）：

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

算法流程

1. 计算所有模数的积 M
2. 对于第 i 个方程：
 - a. 计算 $b_i = \frac{m}{m_i}$
 - b. 计算 b_i 在模 m_i 意义下的逆元 b_i^{-1}
 - c. 计算 $c_i = b_i b_i^{-1}$ （不要对 m_i 取模）
3. 方程组的特解为： $Y = \sum_{i=1}^k a_i c_i \pmod{M}$ ，通解为 $kM + Y (k \in \mathbb{Z})$

```
// 前置模板：龟速乘，扩展欧几里得
LL crt(LL *m, LL *a, int n) {
    LL M=1, res=0;
    for(int i=1; i<=n; i++) M*=m[i];
    for(int i=1; i<=n; i++) {
        LL b=M/m[i], x, y;
        LL d=exgcd(b, m[i], x, y);
        res=(res+qmul(qmul(b, a[i], M), (x%m[i]+m[i])%m[i], M))%M;
    }
    return res;
}
```

扩展：模数不互质的情况

设两个方程分别是 $x \equiv a_1 \pmod{m_1}$ 、 $x \equiv a_2 \pmod{m_2}$

将它们转为不定方程: $x = m_1 + a_1 = m_2q + a_2$, 其中 p, q 是整数, 则有 $m_1p - m_2q = a_2 - a_1$

由裴蜀定理, 当 $a_2 - a_1$ 不能被 $\gcd(m_1, m_2)$ 整除时, 无解

其他情况下, 可以通过扩展欧几里得算法解出来一组可行解 (p, q)

则原来的两方程组成的模方程组的解为 $x \equiv b \pmod{M}$, 其中 $b = m_1p + a_1$, $M = \text{lcm}(m_1, m_2)$

两两合并多个方程, 能得出特解 x , 通解为 $k * \text{lcm}(m_1, m_2, \dots, m_n) + x (k \in \mathbb{Z})$

```
// 前置模板: 龟速乘, 扩展欧几里得
LL excrt(LL *m, LL *a, int n) {
    if(!n) return 0;
    LL M=m[1], A=a[1], x, y;
    for(int i=2; i<=n; i++) {
        LL d=exgcd(M, m[i], x, y);
        LL c=(a[i]-A%m[i]+m[i])%m[i];
        if(c%d) return -1;
        x=qmul(x, c/d, m[i]/d); // 防爆 LL
        A+=M*x;
        M*=m[i]/d;
        A%=M;
    }
    return (A%M+M)%M;
}
```

离散对数

BSGS

在 $O(\sqrt{m})$ 的时间内求解 $a^x \equiv b \pmod{m}, a \perp m$

方程的解 x 满足 $0 \leq x < m$

令 $x = A\lceil\sqrt{m}\rceil - B (0 \leq A, B \leq \lceil\sqrt{m}\rceil)$, 则有 $a^{A\lceil\sqrt{m}\rceil - B} \equiv b \pmod{m}$, 即 $a^{A\lceil\sqrt{m}\rceil} \equiv ba^B \pmod{m}$

枚举 B , 算出 ba^B 的所有取值, 然后枚举 A , 逐一计算 $a^{A\lceil\sqrt{m}\rceil}$

```
LL BSGS(LL a, LL b, LL m) {
    a%=m, b%=m;
    if(!a&&!b) return 1;
    if(!a) return -1;
    static unordered_map<LL, LL> mp;
    mp.clear();
    LL r=sqrt(m+1.5), v=1;
    for(int i=1; i<=r; i++) {
        v=v*a%m;
        mp[b*v%m]=i;
    }
    LL vv=v;
    for(int i=1; i<=r; i++) {
        auto it=mp.find(vv);
        if(it!=mp.end()) return i*r-it->second;
        vv=vv*v%m;
    }
}
```

```

    return -1;
}

```

数论分块

- $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$
- $O(\sqrt{n})$

```

LL solve(int n) {
    LL res=0;
    for(int l=1,r;l<=n;l=r+1) {
        r=n/(n/l);
        res+=1ll*(r-l+1)*(n/l);
    }
    return res;
}

```

- 卡常 $O(\sqrt{n})$

```

LL solve(int n) {
    LL res=0;
    int t=sqrt(n);
    for(int i=1;i<=t;i++) res+=n/i;
    res=res*2-1ll*t*t;
    return res;
}

```

- $\sum_{i=1}^n \sum_{j=1}^m \lfloor \frac{n}{i} \rfloor \lfloor \frac{m}{i} \rfloor$

```

LL solve(int n,int m) {
    LL res=0;
    int lim=min(n,m);
    for(int l=1,r;l<=lim;l=r+1) {
        r=min(n/(n/l),m/(m/l));
        res+=1ll*(r-l+1)*(n/l)*(m/l);
    }
    return res;
}

```

二次剩余

一个数 a ，如果不是 p 的倍数且模 p 同余于某个数的平方，则称 a 为模 p 的二次剩余。

而一个不是 p 的倍数的数 b ，不同余于任何数的平方，则称 b 为模 p 的非二次剩余。

$x^2 \equiv n \pmod{p}$ ($n > 0$, p 为质数)，对常数 n 求 x 。

当解得 $x > 0$ 时， $p - x$ 也是一个解。

Cipolla 算法

- 时间复杂度： $O(\log p)$

```

LL w;
struct C {

```

```

    LL r,i;
    C() {}
    C(LL r,LL i):r(r),i(i) {}
};

C mul(C a,C b,LL p) {
    C ret=C(0,0);
    ret.r=((a.r*b.r%p+a.i*b.i%p*w%p)%p+p)%p;
    ret.i=((a.r*b.i%p+a.i*b.r%p)%p+p)%p;
    return ret;
}

LL qpow_r(LL a,LL b,LL p) {
    LL ret=1;
    while(b) {
        if(b&1) ret=ret*a%p;
        a=a*a%p;
        b>>=1;
    }
    return ret;
}

LL qpow_i(C a,LL b,LL p) {
    C ret=C(1,0);
    while(b) {
        if(b&1) ret=mul(ret,a,p);
        a=mul(a,a,p);
        b>>=1;
    }
    return ret.r;
}

LL cioplla(LL n,LL p) {
    n%=p;
    if(!n) return 0;
    if(p==2) return n;
    if(qpow_r(n,(p-1)/2,p)==p-1) return -1;
    LL a;
    for(;;) {
        a=rand()%p;
        w=((a*a%p-n)%p+p)%p;
        if(qpow_r(w,(p-1)/2,p)==p-1) break;
    }
    C x=C(a,1);
    return qpow_i(x,(p+1)/2,p);
}

```

矩阵运算

```

struct M {
    LL a[N][N];
    void clear() {memset(a,0,sizeof a);}
    M() {clear();}
    void init() {
        clear();
        for(int i=0;i<N;i++) a[i][i]=1;
    }
    M operator + (const M &T) const {
        M ret;
        for(int i=0;i<N;i++)
            for(int j=0;j<N;j++)

```



```

        ret.a[i][j]=(a[i][j]+T.a[i][j])%MOD;
    return ret;
}
M operator - (const M &T) const {
    M ret;
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            ret.a[i][j]=(a[i][j]-T.a[i][j]+MOD)%MOD;
    return ret;
}
M operator * (const LL &v) const {
    M ret;
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++) if(a[i][j])
            ret.a[i][j]=a[i][j]*v%MOD;
    return ret;
}
M operator * (const M &T) const {
    M ret;
    for(int i=0;i<N;i++)
        for(int k=0;k<N;k++) if(a[i][k])
            for(int j=0;j<N;j++) if(T.a[k][j])
                ret.a[i][j]=(ret.a[i][j]+a[i][k]*T.a[k][j]%MOD)%MOD;
    return ret;
}
M operator ^ (LL b) const {
    M ret,bas;
    for(int i=0;i<N;i++) ret.a[i][i]=1;
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            bas.a[i][j]=a[i][j];
    while(b) {
        if(b&1) ret=ret*bas;
        bas=bas*bas;
        b>>=1;
    }
    return ret;
}
void print() {
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            cout<<a[i][j]<<" \n"[j==N-1];
}
};

```

高斯消元

高斯消元解线性方程组

- 浮点数版本

```

const DB EPS=1e-9;
DB a[N][N],x[N]; // a 是增广矩阵, x 是解
bool free_x[N]; // 是否为自由变量
int sgn(DB x) {return fabs(x)<EPS?0:(x>0?1:-1);}
int gauss(int n,int m) { // n 个方程, m 个变量

```

```

memset(x,0,sizeof x);
memset(&free_x,true,sizeof free_x);
int r=0,c=0;
while(r<n&& c<m) {
    int R=r;
    for(int i=r+1;i<n;i++) if(fabs(a[i][c])>fabs(a[R][c])) R=i;
    if(R!=r) for(int j=c;j<=m;j++) swap(a[r][j],a[R][j]);
    if(!sgn(a[r][c])) {a[r][c]=0;++c;continue;}
    for(int i=r+1;i<n;i++) if(a[i][c]) {
        DB t=a[i][c]/a[r][c];
        for(int j=c;j<=m;j++) a[i][j]-=a[r][j]*t;
    }
    ++r,++c;
}
for(int i=r;i<n;i++) if(sgn(a[i][m])) return -1;
if(r<m) {
    for(int i=r-1;i>=0;i--) {
        int cnt=0,k=-1;
        for(int j=0;j<m;j++) if(sgn(a[i][j])&&free_x[j]) ++cnt,k=j;
        if(cnt>1) continue;
        DB s=a[i][m];
        for(int j=0;j<m;j++) if(sgn(a[i][j])&&j!=k) s-=a[i][j]*x[j];
        x[k]=s/a[i][k];
        free_x[k]=false;
    }
    return m-r; // 自由变量个数
}
for(int i=m-1;i>=0;i--) {
    DB s=a[i][m];
    for(int j=i+1;j<m;j++) s-=a[i][j]*x[j];
    // 整数版本中 if(s%a[i][i]!=0) return -2; // 说明有浮点数解，但无整数解
    x[i]=s/a[i][i];
}
return 0;
}

```

高斯消元解 XOR 方程组

```

int a[N][N],x[N]; // a 是增广矩阵，x 是解
VI free_x;
int gauss(int n,int m) { // n 个方程，m 个变量
    memset(x,0,sizeof x);
    free_x.clear();
    int r=0,c=0;
    while(r<n&& c<m) {
        int R=r;
        for(int i=r+1;i<n;i++) if(a[i][c]>a[R][c]) R=i;
        if(R!=r) for(int j=c;j<=m;j++) swap(a[r][j],a[R][j]);
        if(!a[r][c]) {free_x.pb(c),++c;continue;}
        for(int i=r+1;i<n;i++) if(a[i][c]) {
            int t=a[i][c]^a[r][c];
            for(int j=c;j<=m;j++) a[i][j]^=a[r][j]^t;
        }
        ++r,++c;
    }
    for(int i=r;i<n;i++) if(a[i][m]) return INT_MAX; // 无解
    int mn=INT_MAX;

```

```

// 自由变量个数为 m - r 个
for(int i=0;i<1<<(m-r);i++) { //枚举变元
    int cnt=0,idx=i;
    for(int j=0;j<m-r;j++) {
        x[free_x[j]]=idx&1;
        if(x[free_x[j]]) ++cnt;
        idx>>=1;
    }
    for(int j=r-1;j>=0;j--) {
        int k=j;
        while(!a[j][k]) ++k;
        x[k]=a[j][m];
        for(int l=k+1;l<m;l++) x[k]^=a[j][l]&x[l];
        if(x[k]) ++cnt;
    }
    mn=min(mn,cnt);
}
return mn; // 最小操作数
}

```

高斯消元解同余方程组

```

const int MOD=7;
int a[N][N],x[N]; // a 是增广矩阵, x 是解
int LCM(int a,int b) {
    return a/__gcd(a,b)*b;
}
LL get_inv(LL a,LL m) {
    if(a==1) return 1;
    return get_inv(m%a,m)*(m-m/a)%m;
}
int gauss(int n,int m) { // n 个方程, m 个变量
    memset(x,0,sizeof x);
    int r=0,c=0;
    while(r<n&& c<m) {
        int R=r;
        for(int i=r+1;i<n;i++) if(abs(a[i][c])>abs(a[R][c])) R=i;
        if(R!=r) for(int j=c;j<=m;j++) swap(a[r][j],a[R][j]);
        if(!a[r][c]) {++c;continue;}
        for(int i=r+1;i<n;i++) if(a[i][c]) {
            int lcm=LCM(a[i][c],a[r][c]);
            int ta=lcm/a[i][c],tb=lcm/a[r][c];
            for(int j=c;j<=m;j++) a[i][j]=(a[i][j]*ta%MOD-a[r][j]*tb%MOD+MOD)%MOD;
        }
        ++r,++c;
    }
    for(int i=r;i<n;i++) if(a[i][m]) return -1; // 无解
    if(r<m) return m-r; // 自由变量个数
    for(int i=m-1;i>=0;i--) { // 必有唯一解
        int s=a[i][m];
        for(int j=i+1;j<m;j++) s=(s-a[i][j]*x[j]%MOD+MOD)%MOD;
        x[i]=s*get_inv(a[i][i],MOD)%MOD;
    }
    return 0;
}

```

线性基

线性基是向量空间的一组基，通常可以解决有关异或的一些题目

通俗一点的讲法就是由一个集合构造出来的另一个集合，它有以下几个性质：

- 线性基的元素能相互异或得到原集合的元素的所有相互异或得到的值
- 线性基是满足性质 1 的最小的集合
- 线性基没有异或和为 0 的子集
- 线性基中每个元素的异或方案唯一，也就是说，线性基中不同的异或组合异或出的数都是不一样的
- 线性基中每个元素的二进制最高位互不相同

```
// 查询原集合内任意几个元素异或后的值
const int N=100;
int n,tot;
LL d[N];
bool add(LL x) { // 插入
    for(int i=62;~i;i--) if((x>>i)&1) {
        if(!d[i]) {d[i]=x;return true;}
        x^=d[i];
    }
    return false;
}
void calc() { // 统计线性基元素数量
    tot=0;
    for(int i=0;i<63;i++) if(d[i]) ++tot;
}
LL get_max() { // 最大值
    LL ret=0;
    for(int i=62;~i;i--) if(ret^d[i]>ret) ret^=d[i];
    return ret;
}
LL get_min() { // 最小值
    if(tot<n) return 0;
    for(int i=0;i<63;i++) if(d[i]) return d[i];
    return -1;
}
void update() {
    for(int i=0;i<63;i++)
        for(int j=0;j<i;j++)
            if((d[i]>>j)&1)
                d[i]^=d[j];
}
LL kth(int k) { // k 小值
    if(tot<n) --k; // 去掉 0
    LL ret=0;
    for(int i=0;i<63;i++) if(d[i]) {
        if(k&1) ret^=d[i];
        k>>=1;
    }
    if(k) return -1;
    return ret;
}
void init() {
    memset(d,0,sizeof d);
    for(int i=1;i<=n;i++) {
        LL x;cin>>x;
```

```
    add(x);  
}  
calc();  
update();  
}
```

线性递推数列

一阶线性递推数列

对于一阶线性递推数列，形如 $a_{n+1} = pa_n + q$ ，可采用参数法求通项。

对于 $a_{n+1} = pa_n + q$ ，设 $a_{n+1} - t = p(a_n - t)$ ，

得： $q = t - pt$ ，解出 t 代入上式，用等比数列通项得到： $a_n = p^{n-1}(a_1 - t) + t$ 。

二阶线性递推数列

对于二阶线性递推数列，形如 $a_{n+2} = pa_{n+1} + qa_n$ ($q \neq 0$)，可采用特征方程法求通项。

对于 $a_{n+2} = pa_{n+1} + qa_n$ ，设有 s 和 r ，使 $a_{n+2} - sa_{n+1} = r(a_{n+1} - sa_n)$ 。

得： $s + r = p$ ， $rs = -q$ ，易发现这是韦达定理。

那么 s, r 为一元二次方程 $x^2 - px - q = 0$ 的解。

称一元二次方程 $x^2 - px - q = 0$ 为递推数列 $a_{n+2} = pa_{n+1} + qa_n$ ($q \neq 0$) 的特征方程。

在等式 $a_{n+2} - sa_{n+1} = r(a_{n+1} - sa_n)$ 中，易发现 s, r 是可以交换的。

那么可以得到方程组：

$$\begin{cases} a_{n+2} - sa_{n+1} = r^{n-1}(a_2 - sa_1) \\ a_{n+2} - ra_{n+1} = s^{n-1}(a_2 - ra_1) \end{cases}$$

- 若 $s \neq r$ ， $a_n = \frac{r^{n-1}(a_2 - sa_1) - s^{n-1}(a_2 - ra_1)}{r - s}$ ；
- 若 $s = r$ ，则 $a = b \neq 0$ ， $a_n = (2 - n)s^{n-1}a_1 + (n - 1)s^{n-2}a_2$ 。

概率论

几何分布

在 n 次伯努利试验中，试验 k 次才得到第一次成功的机率。

记在伯努利试验中，成功的概率为 p ，试验次数为 x 。

$$P(x) = (1 - p)^{x-1}p$$

$$E(x) = \frac{1}{p}$$

超几何分布

多项式

FFT

- n 必须超过 a, b 最高指数之和
- 空间大于 a, b 最高指数之和的两倍
- 所求系数是整数时, 卷积后注意四舍五入

```
const LD PI=acos(-1);
namespace FFT {
    struct C {
        LD r,i;
        C(LD r=0,LD i=0):r(r),i(i) {}
        C operator + (const C &T) const {
            return C(r+T.r,i+T.i);
        }
        C operator - (const C &T) const {
            return C(r-T.r,i-T.i);
        }
        C operator * (const C &T) const {
            return C(r*T.r-i*T.i,r*T.i+i*T.r);
        }
    };
    void FFT(C x[],int n,int on) {
        VI rev(n);
        for(int i=0;i<n;i++) {
            rev[i]=rev[i>>1]>>1|(i&1?n>>1:0);
            if(i<rev[i]) swap(x[i],x[rev[i]]);
        }
        for(int mid=1;mid<n;mid<=<1) {
            C wn(cos(PI/mid),sin(on*PI/mid));
            for(int i=0;i<n;i+=(mid<<1)) {
                C w(1,0);
                for(int j=0;j<mid;j++,w=w*wn) {
                    C t1=x[i+j],t2=w*x[i+j+mid];
                    x[i+j]=t1+t2,x[i+j+mid]=t1-t2;
                }
            }
        }
        if(on==<1) for(int i=0;i<n;i++) x[i].r/=n;
    }
    void conv(C a[],C b[],int n) {
        FFT(a,n,1);
        FFT(b,n,1);
        for(int i=0;i<n;i++) a[i]=a[i]*b[i];
        FFT(a,n,<1);
    }
}
```

NTT

- 998244353, 1004535809, 469762049 原根都为 3

```
namespace NTT {
    const LL G=3,MOD=998244353;
    LL qpow(LL a,LL b) {
        LL ret=1;
        while(b) {
```

```

        if(b&1) ret=ret*a%MOD;
        a=a*a%MOD;
        b>>=1;
    }
    return ret;
}

void NTT(LL x[],int n,int on) {
    VI rev(n);
    for(int i=0;i<n;i++) {
        rev[i]=rev[i>>1]>>1|(i&1?n>>1:0);
        if(i<rev[i]) swap(x[i],x[rev[i]]);
    }
    for(int mid=1;mid<n;mid<=<1) {
        LL wn=qpow(on==1?G:qpow(G,MOD-2),(MOD-1)/(mid<<1));
        for(int i=0;i<n;i+=(mid<<1)) {
            LL w=1;
            for(int j=0;j<mid;j++,w=w*wn%MOD) {
                LL t1=x[i+j],t2=w*x[i+j+mid]%MOD;
                x[i+j]=(t1+t2)%MOD,x[i+j+mid]=(t1-t2+MOD)%MOD;
            }
        }
    }
    if(on==<1) {
        LL inv=qpow(n,MOD-2);
        for(int i=0;i<n;i++) x[i]=x[i]*inv%MOD;
    }
}

void conv(LL a[],LL b[],int n) {
    NTT(a,n,1);
    NTT(b,n,1);
    for(int i=0;i<n;i++) a[i]=a[i]*b[i]%MOD;
    NTT(a,n,<1);
}
}

```

博弈论

巴什博弈

一堆 n 个物品，两个人轮流从中取出 $1 \sim m$ 个，最后取光者胜。

先手必败条件： $n \% (m + 1) = 0$ 。

威佐夫博弈

有两堆各若干个物品，两个人轮流从任一堆取至少一个或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

先手必败条件： $|n - m| * \frac{1+\sqrt{5}}{2} = \min(n, m)$ 。

斐波那契博弈

一堆石子有 n 个，两人轮流取，先取者第一次可以取任意多个，但是不能取完，以后每次取的石子数不能超过上次取子数的 2 倍。取完者胜。

先手必败条件： n 是斐波那契数。

Nim 游戏

有 n 堆石子，两人轮流取，每次取某堆中不少于 1 个，最后取完者胜。

先手必败条件：各堆石子数目全部异或后结果等于 0。

阶梯 Nim 游戏

有 n 个位置 $1 \dots n$ ，每个位置上有 a_i 个石子。两人轮流操作。每次挑选 $1 \dots n$ 中任一个存在石子的位置 i ，将至少 1 个石子移动至 $i - 1$ 位置（也就是最后所有石子都堆在 0 这个位置），谁不能操作谁输。

先手必败条件：奇数位置石子数目全部异或后结果等于 0。

反 Nim 游戏

有 n 堆石子，两人轮流取，每次取某堆中不少于 1 个，最后取完者败。

先手必胜条件：

1. 各堆石子数目异或结果等于 0，且所有石子堆数目全部为 1。
2. 各堆石子数目异或结果不等于 0，且存在有石子数目大于 1 的石子堆。

公式

高数公式

- $\lim_{x \rightarrow 0} \frac{\sin x}{x} = \lim_{x \rightarrow 0} \frac{x}{\sin x} = 1$
- $\lim_{x \rightarrow \infty} (1 + \frac{1}{x})^x = \lim_{x \rightarrow 0} (1 + x)^{\frac{1}{x}} = e$
- $\int_0^1 x^{a-1} (1-x)^{b-1} = \frac{1}{b \binom{a+b-1}{a-1}} (a, b \in \mathbb{Z}^+)$

莫比乌斯反演

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

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

数论公式

$$1. \sum_{i=1}^n \sum_{j=1}^m \gcd(i, j)$$

由 $\varphi * 1 = \text{id}$ 得：

$$\sum_{i=1}^n \sum_{j=1}^m \gcd(i, j) = \sum_{i=1}^n \sum_{j=1}^m \sum_{d|\gcd(i, j)} \varphi(d) = \sum_{d=1} \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$$

$$2. \sum_{i=1}^n i[\gcd(i, n) = 1]$$

- 当 $n = 1$ 时, $\sum_{i=1}^n i[\gcd(i, n) = 1] = 1$ 。
- 当 $n > 1$ 时, $\gcd(i, n) = \gcd(n - i, n)$, 与 n 互质的数是成对出现的。

$$\text{因此 } \sum_{i=1}^n i[\gcd(i, n) = 1] = \frac{n\varphi(n)}{2}。$$

$$\text{所以 } \sum_{i=1}^n i[\gcd(i, n) = 1] = \frac{n\varphi(n) + [n=1]}{2}。$$

$$3. \mu^2(n) = \sum_{d^2|n} \mu(d)$$

证明:

- n 无平方因子, $\mu^2(n) = 1$ 。
- n 有平方因子, 假设 d' 是最大的数满足 $d'^2 \mid n$ 且 d' 无平方因子, 那么 n 的其它因子当且仅当其质因子构成的可重集是 d' 的质因子集合的子集时, 才对 $\sum_{d^2|n} \mu(d)$ 产生贡献。

假设 d' 的质因子集合大小为 m , 那么考虑枚举 d' 质因子集合的子集:

$$\sum_{i=0}^m (-1)^i \binom{m}{i} = (1-1)^m = 0$$

所以 $\mu^2(n) = \sum_{d^2|n} \mu(d)$ 成立。

$$4. \sum_{i=1}^n \mu^2(i)$$

$$\sum_{i=1}^n \mu^2(i) = \sum_{i=1}^n \sum_{d^2|i} \mu(d) = \sum_{d=1}^{\sqrt{n}} \lfloor \frac{n}{d^2} \rfloor \mu(d)$$

$$5. \sum_{i=1}^n \text{lcm}(i, n)$$

$$\begin{aligned} \sum_{i=1}^n \text{lcm}(i, n) &= \sum_{i=1}^n \frac{in}{\gcd(i, n)} \\ &= \sum_{d|n} (n \sum_{i=1}^n \frac{i}{d} [\gcd(i, n) = d]) \\ &= \sum_{d|n} n \sum_{i=1}^{\frac{n}{d}} i [\gcd(i, \frac{n}{d}) = 1] \\ &= n \sum_{d|n} \frac{(\frac{n}{d})\varphi(\frac{n}{d}) + [\frac{n}{d} = 1]}{2} \\ &= n \sum_{d|n} \frac{d\varphi(d) + [d = 1]}{2} \end{aligned}$$

$$6. \sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j)$$

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j) &= \sum_{i=1}^n \sum_{j=1}^m \frac{ij}{\gcd(i, j)} \\ &= \sum_{d=1}^n \sum_{i=1}^n \sum_{j=1}^m \frac{ij [\gcd(i, j) = d]}{d} \\ &= \sum_{d=1}^n \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} ij [\gcd(i, j) = 1] \\ &= \sum_{d=1}^n d \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} ij [\gcd(i, j) = 1] \end{aligned}$$

$$\text{设 } f(n, m) = \sum_{i=1}^n \sum_{j=1}^m ij [\gcd(i, j) = 1],$$

$$\begin{aligned}
f(n, m) &= \sum_{i=1}^n \sum_{j=1}^m \sum_{d|\gcd(i, j)} ij\mu(d) \\
&= \sum_{d=1}^n \sum_{d|i} \sum_{d|j}^m ij\mu(d) \\
&= \sum_{d=1} d^2 \mu(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} ij
\end{aligned}$$

$$\text{设 } g(n, m) = \sum_{i=1}^n \sum_{j=1}^m ij,$$

$$\text{根据等差数列求和公式得 } g(n, m) = (n + \frac{n(n-1)}{2})(m + \frac{m(m-1)}{2}) = \frac{nm(n+1)(m+1)}{4},$$

$$f(n, m) = \sum_{d=1} d^2 \mu(d) g(\lfloor \frac{n}{d} \rfloor, \lfloor \frac{m}{d} \rfloor), \text{ 数论分块加速求和.}$$

$$\text{因此 } \sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j) = \sum_{d=1} df(\lfloor \frac{n}{d} \rfloor, \lfloor \frac{m}{d} \rfloor), \text{ 数论分块加速求和.}$$

$$7. \sum_{i=1}^n \sum_{j=1}^m d(ij)$$

$$\begin{aligned}
d(ij) &= \sum_{x|i} \sum_{y|j} [\gcd(x, y) = 1] \\
&= \sum_{x|i} \sum_{y|j} \sum_{d|\gcd(x, y)} \mu(d) \\
&= \sum_{d=1} \mu(d) \sum_{x|i} \sum_{y|j} [d | \gcd(x, y)] \\
&= \sum_{d|i, d|j} \mu(d) d(\frac{i}{d}) d(\frac{j}{d})
\end{aligned}$$

代入原式：

$$\begin{aligned}
\sum_{i=1}^n \sum_{j=1}^m d(ij) &= \sum_{i=1}^n \sum_{j=1}^m \sum_{d|i, d|j} \mu(d) d(\frac{i}{d}) d(\frac{j}{d}) \\
&= \sum_{d=1} \mu(d) \sum_{d|i} \sum_{d|j} d(\frac{i}{d}) d(\frac{j}{d}) \\
&= \sum_{d=1} \mu(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} d(i) \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} d(j)
\end{aligned}$$

$O(n)$ 预处理 μ, d 的前缀和, $O(\sqrt{n})$ 数论分块加速求和。

组合数学公式

- $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$
- $(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$
- n 个无差别的球放进 k 个不同的盒子, 不允许空盒子, 方案数为 $\binom{n-1}{k-1}$
- n 个无差别的球放进 k 个不同的盒子, 允许空盒子, 方案数为 $\binom{n+k-1}{k-1}$
- Catalan 数:

$$\text{递推式: } H(n) = \frac{H(n-1) * (4n-2)}{n+1}$$

$$\text{该递推关系的解为: } H(n) = \binom{2n}{n} - \binom{2n}{n-1} = \frac{\binom{2n}{n}}{n+1}$$

$$H_n = \begin{cases} \sum_{i=1}^n H_{i-1} H_{n-i} & n \geq 2, n \in \mathbb{N}^+ \\ 1 & n = 0, 1 \end{cases}$$

以下问题属于 Catalan 数列：

1. 有 $2n$ 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外 n 人只有 10 元钞票，剧院无其它钞票，问有多少中方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？
2. 一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 $2n$ 个街区去上班。如果他从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？
3. 在圆上选择 $2n$ 个点，将这些点成对连接起来使得所得到的 n 条线段不相交的方法数？
4. 对角线不相交的情况下，将一个凸多边形区域分成三角形区域的方法数？
5. n 个结点可构造多少个不同的二叉树？
6. n 个不同的数依次进栈，求不同的出栈结果的种数？
7. n 个 $+1$ 和 n 个 -1 构成 $2n$ 项 a_1, a_2, \dots, a_{2n} ，其部分和满足 $a_1 + a_2 + \dots + a_k \geq 0 (k = 1, 2, 3, \dots, 2n)$ 的方法数？

低阶等幂求和

- $\sum_{i=1}^n i^1 = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$
- $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$
- $\sum_{i=1}^n i^3 = \left[\frac{n(n+1)}{2} \right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$
- $\sum_{i=1}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$
- $\sum_{i=1}^n i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

约瑟夫环

n 个人围成一圈，从第一个开始报数，第 m 个将被杀掉，最后剩下一个，其余人都将被杀掉

$f[i]$ 表示 i 个人玩游戏报 m 退出最后胜利者的编号

$$f_n = \begin{cases} 0 & n = 1 \\ (f_{n-1} + m) \% n & n > 1 \end{cases}$$

勾股数

所谓勾股数，一般是指能够构成直角三角形三条边的三个正整数

即 $a^2 + b^2 = c^2, a, b, c \in N$

当 a 为大于 1 的奇数 $2n + 1$ 时, $b = 2n^2 + 2n, c = 2n^2 + 2n + 1$

当 a 为大于 4 的偶数 $2n$ 时, $b = n^2 - 1, c = n^2 + 1$

计算几何

二维

点

```
const DB EPS=1e-8;
const DB PI=acos(-1);
int sgn(DB x) {return fabs(x)<EPS?0:(x>0?1:-1);}
struct P {
    DB x,y;
    P() {}
    P(DB x,DB y):x(x),y(y) {}
    bool operator == (const P &a) const {return !sgn(x-a.x)&&!sgn(y-a.y);}
    bool operator < (const P &a) const {return sgn(x-a.x)<0||sgn(x-a.x)==0&&sgn(y-a.y)<0;}
}
```

```

P operator + (const P &a) const {return P(x+a.x,y+a.y);}
P operator - (const P &a) const {return P(x-a.x,y-a.y);}
P operator * (const DB &k) const {return P(x*k,y*k);}
P operator / (const DB &k) const {return P(x/k,y/k);}
// 点积
DB operator * (const P &a) const {return x*a.x+y*a.y;}
// 叉积
DB operator ^ (const P &a) const {return x*a.y-y*a.x;}
// 这个点与 a, b 所成的夹角
DB rad(const P &a,const P &b) {return fabs(atan2(fabs((a-*this)^(b-*this)),
(a-*this)*(b-*this))));}
// 点到原点的距离
DB len() {return hypot(x,y);}
// 点到原点的距离的平方
DB len2() {return x*x+y*y;}
// 两点距离
DB dist(const P &p) {return hypot(x-p.x,y-p.y);}
// 绕原点逆时针旋转 90 度
P rotright() {return P(-y,x);}
// 绕原点顺时针旋转 90 度
P rotright() {return P(y,-x);}
// 绕 P 点逆时针旋转 angle 度
P rotate(const P &p,const DB &angle) {
    P v=*this-p;
    DB c=cos(angle),s=sin(angle);
    return P(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
}
};

```

线

```

struct L {
    P s,t;
    L() {}
    L(P s,P t):s(s),t(t) {}
    bool operator == (const L &a) const {return s==a.s&&t==a.t;}
    // -1在左侧, 1在右侧, 0在直线上
    int relation(const P &p) {return sgn((p-s)^(t-s));}
    // -1 在 T 的逆时针方向, 1 在 T 的顺时针方向, 0 共线
    int relation(const L &T) const {
        return sgn((t-s)^(T.t-T.s));
    }
    P cross_point(const L &a) { // 直线交,需保证不平行不重合
        DB s1=(t-s)^(a.s-s);
        DB s2=(t-s)^(a.t-s);
        return (a.s*s2-a.t*s1)/(s2-s1);
    }
    bool parallel(const L &a) {return sgn((t-s)^(a.t-a.s))==0;} // 平行
    bool P_on_seg(const P &p) {
        return !sgn((p-s)^(t-s))&&sgn((p-s)*(p-t))<=0;
    }
    // 求 a, b 直线交是否在线段 b 上无需判断 sgn((p-s)^(t-s))
    // 因为已经保证了该点在线段 b 或线段 b 的延长线上, 加上会有精度问题
};

```



```
struct C {
    P p;
    DB r;
    C() {}
    C(P p,DB r):p(p),r(r) {}
    DB area() {return PI*r*r;}
    DB circumference() {return 2*PI*r;}
    // 点和圆的关系
    // 2 圆外
    // 1 圆上
    // 0 圆内
    int p_relation(P a) {
        DB d=p.dist(a);
        if(sgn(d-r)<0) return 0;
        if(sgn(d-r)==0) return 1;
        return 2;
    }
    // 圆与圆的关系
    // 5 相离
    // 4 外切
    // 3 相交
    // 2 内切
    // 1 内含
    int c_relation(C c) {
        DB d=p.dist(c.p);
        if(sgn(d-r-c.r)>0) return 5;
        if(sgn(d-r-c.r)==0) return 4;
        DB l=fabs(r-c.r);
        if(sgn(d-r-c.r)<0&&sgn(d-l)>0) return 3;
        if(sgn(d-l)==0) return 2;
        if(sgn(d-l)<0) return 1;
    }
    DB c_area(C c) {
        int rel=c_relation(c);
        if(rel>=4) return 0;
        if(rel<=2) return min(area(),c.area());
        DB d=p.dist(c.p);
        DB hf=(r+c.r+d)/2;
        DB ss=2*sqrt(hf*(hf-r)*(hf-c.r)*(hf-d));
        DB a1=acos((r*r+d*d-c.r*c.r)/(2*r*d))*r*r;
        DB a2=acos((c.r*c.r+d*d-r*r)/(2*c.r*d))*c.r*c.r;
        return a1+a2-ss;
    }
};

P compute_circle_center(P a,P b) {return(a+b)/2;} // 两点定圆
P compute_circle_center(P a,P b,P c) { // 三点定圆
    b=(a+b)/2;
    c=(a+c)/2;
    return L(b,b+(a-b).rotright()).cross_point({c,c+(a-c).rotright()});
}
```

凸包

```
void andrew() {
    sort(p+1,p+1+n);
    // 有重复点需要去重，至少三个不相同的点才能构成凸包，根据题意判断点或线段的特殊情况
    for(int i=1;i<=2;i++) stk[++top]=p[i];
    for(int i=3;i<=n;i++) {
        while(top>=2&&sgn((stk[top]-stk[top-1])^(p[i]-stk[top]))<=0) --top;
        stk[++top]=p[i];
    }
    int temp=top;
    stk[++top]=p[n-1];
    for(int i=n-2;i>=1;i--) {
        while(top>=temp&&sgn((stk[top]-stk[top-1])^(p[i]-stk[top]))<=0) --top;
        stk[++top]=p[i];
    }
    // 1 和 top 为同一个点，凸包上共有 top - 1 个点
}
```

最小圆覆盖

```
C min_circle_cover(const vector<P> &T) {
    vector<P> a(ALL(T));
    random_shuffle(ALL(a));
    P c=a[0];DB r=0;
    for(int i=1;i<SZ(a);i++) if(!p_in_circle(a[i],{c,r})) {
        c=a[i];r=0;
        for(int j=0;j<i;j++) if(!p_in_circle(a[j],{c,r})) {
            c=compute_circle_center(a[i],a[j]);
            r=a[j].dist(c);
            for(int k=0;k<j;k++) if(!p_in_circle(a[k],{c,r})) {
                c=compute_circle_center(a[i],a[j],a[k]);
                r=a[k].dist(c);
            }
        }
    }
    return {c,r};
}
```

三维

点

```
struct P {
    DB x,y,z;
    P() {}
    P(DB x,DB y,DB z):x(x),y(y),z(z) {}
    DB dist(const P &p) {return sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y)+(z-p.z)*(z-p.z));}
};
```

球

```
struct S {
    P p;
    DB r;
    S() {}
    S(P p,DB r):p(p),r(r) {}
    DB vol() {return 4*PI*r*r*r/3;} // 体积
    DB s_s_area(S &T) { // 两球体积交
        DB d=p.dist(T.p);
        if(sgn(d-r-T.r)>=0) return 0;
        if(sgn(d-fabs(r-T.r))<=0) return r<T.r?vol():T.vol();
        DB h1=r-(r*r-T.r*T.r+d*d)/(2*d);
        DB h2=T.r-(T.r*T.r-r*r+d*d)/(2*d);
        return PI/3*(h1*h1*(3*r-h1)+h2*h2*(3*T.r-h2));
    }
};
```

其它

- 圆台体积公式: $V = \frac{1}{3}\pi h(R^2 + r^2 + Rr)$ (r 为上底半径、 R 为下底半径、 h 为高)

杂项

排序

归并排序

```
int a[N],temp[N];
void merge(int l,int mid,int r) {
    int i=l,j=mid+1;
    int t=l;
    while(i<=mid&&j<=r) {
        if(a[i]<=a[j]) temp[t++]=a[i++];
        else temp[t++]=a[j++];
    }
    while(i<=mid) temp[t++]=a[i++];
    while(j<=r) temp[t++]=a[j++];
    for(int i=l;i<=r;i++) a[i]=temp[i];
}
void merge_sort(int l,int r) {
    if(l<r) {
        int mid=l+r>>1;
        merge_sort(l,mid);
        merge_sort(mid+1,r);
        merge(l,mid,r);
    }
}
```

快速排序

```
void quick_sort(int l,int r) {
    if(l<r) {
        int i=l,j=r;
        int key=a[l];
        while(i<j) {
            while(i<j&& a[j]>=key) j--;
            if(i<j) a[i++]=a[j];
            while(i<j&& a[i]<=key) i++;
            if(i<j) a[j--]=a[i];
        }
        a[i]=key;
        quick_ort(l,i-1);
        quick_ort(i+1,r);
    }
}
```

离散化

```
for(int i=1;i<=n;i++) b.pb(a[i]);
sort(ALL(b));
b.resize(unique(ALL(b))-b.begin());
for(int i=1;i<=n;i++) a[i]=lower_bound(ALL(b),a[i])-b.begin()+1; // 最小值为1
```

二分

- 整数最小值

```
int find(int l,int r) {
    while(l<r) {
        int mid=l+r>>1;
        if(check(mid)) r=mid;
        else l=mid+1;
    }
    return l;
}
```

- 整数最大值

```
int find(int l, int r) { // 最大值
    while(l<r) {
        int mid=l+r+1>>1;
        if (check(mid)) l=mid;
        else r=mid-1;
    }
    return l;
}
```

- 实数


```

DB find(DB l,DB r) { // 实数二分
    for(int i=0;i<100;i++) {
        DB mid=(l+r)/2;
        if(check(mid)) l=mid;
        else r=mid;
    }
    return l;
}

```

三分

- 三分用来寻找凸函数或凹函数的极值
- 以寻找凸函数的极大值为例
- 实数三分

```

for(int i=0;i<100;i++) {
    DB mid=(r-l)/3.0;
    if(calc(l+mid)>calc(r-mid)) r=r-mid;
    else l=l+mid;
}
mx=calc(l);

```

- 整数三分

```

while(l<r) {
    int lmid=l+(r-l)/3,rmid=r-(r-l)/3;
    if(calc(lmid)>calc(rmid)) r=rmid-1;
    else l=lmid+1;
}
mx=calc(l);

```

高精度

```

struct bign {
    int len;
    int num[1005];
    bign() {
        len=1;
        memset(num,0,sizeof num);
    }
    bool operator < (const bign &a) const {
        if(len!=a.len) return len<a.len;
        for(int i=len;i>=1;i--) if(num[i]!=a.num[i]) return num[i]<a.num[i];
        return false;
    }
    bool operator > (const bign &a) const {return a<*this;}
    bool operator <= (const bign &a) const {return !(a<*this);}
    bool operator >= (const bign &a) const {return !(*this<a);}
    bool operator != (const bign &a) const {return a<*this||*this<a;}
    bool operator == (const bign &a) const {return !(a<*this)&&!(*this<a);}
};
bign get(const string &s) {
    bign res;
    res.len=SZ(s);
}

```

```

        for(int i=1;i<=res.len;i++) res.num[i]=s[res.len-i]-'0';
        return res;
    }
    bign get(int n) {
        bign res;
        while(n) {
            res.num[res.len++]=n%10;
            n/=10;
        }
        if(res.len>1) res.len--;
        return res;
    }
    void print(bign a) {
        for(int i=a.len;i;i--) cout<<a.num[i];
        cout<<'\n';
    }
    bool non_zero(bign T) {
        return T.len>1||T.len==1&&T.num[1];
    }
    bign operator + (bign a,bign b) {
        bign res;
        res.len=max(a.len,b.len);
        int x=0;
        for(int i=1;i<=res.len;i++) {
            res.num[i]=x+a.num[i]+b.num[i];
            x=res.num[i]/10;
            res.num[i]%=10;
        }
        if(x) res.num[++res.len]=x;
        return res;
    }
    bign operator - (bign a,bign b) { // a > b
        bign res;
        res.len=max(a.len,b.len);
        for(int i=1;i<=res.len;i++) {
            if(a.num[i]<b.num[i]) --a.num[i+1],a.num[i]+=10;
            res.num[i]=a.num[i]-b.num[i];
        }
        while(res.len>1&&res.num[res.len]==0) res.len--;
        return res;
    }
    bign operator * (bign a,int b) {
        bign res;
        int x=0;
        for(int i=1;i<=a.len;i++) {
            x+=a.num[i]*b;
            res.num[res.len++]=x%10;
            x/=10;
        }
        while(x) {
            res.num[res.len++]=x%10;
            x/=10;
        }
        while(res.len>1&&res.num[res.len]==0) res.len--;
        return res;
    }
    bign operator * (bign a,bign b) {
        bign res;

```

```

    res.len=a.len+b.len;
    for(int i=1;i<=res.len;i++) res.num[i]=0;
    for(int i=1;i<=a.len;i++) {
        int x=0;
        for(int j=1;j<=b.len;j++) {
            res.num[i+j-1]+=a.num[i]*b.num[j]+x;
            x=res.num[i+j-1]/10;
            res.num[i+j-1]%=10;
        }
        res.num[i+b.len]+=x;
    }
    while(res.len>1&&res.num[res.len]==0) res.len--;
    return res;
}

bign operator / (bign a,int b) {
    bign res;
    res.len=a.len;
    for(int i=a.len,t=0;i>=1;i--) {
        t=t*10+a.num[i];
        if(t>=b) res.num[i]=t/b,t%=b;
    }
    while(res.len>1&&res.num[res.len]==0) res.len--;
    return res;
}

bign operator / (bign a,bign b) {
    bign res,x;
    res.len=a.len;
    for(int i=1;i<=res.len;i++) res.num[i]=0;
    for(int i=a.len;i>=1;i--) {
        x=x*10; // bign * int
        x.num[1]=a.num[i];
        while(x>b||x==b) {
            x=x-b; // bign - bign
            res.num[i]++;
        }
    }
    while(res.len>1&&res.num[res.len]==0) res.len--;
    return res;
}

bign operator % (bign a,bign b) {
    bign res;
    for(int i=a.len;i>=1;i--) {
        res=res*10; // bign * int
        res.num[1]=a.num[i];
        while(res>b||res==b) res=res-b; // bign - bign
    }
    while(res.len>1&&res.num[res.len]==0) res.len--;
    return res;
}

bign gcd(bign a,bign b) {
    return non_zero(b)?gcd(b,a%b):a;
}

bign qpow(bign a,bign b,bign m) {
    bign ret;
    ret.num[1]=1;
    while(non_zero(b)) {
        if(b.num[1]&1) ret=ret*a%m;
        a=a*a%m;
    }
}

```

```

        b=b/2;
    }
    return ret;
}

```

浮点数的精度问题

```
int sgn(DB x) {return fabs(x)<EPS?0:(x>0?1:-1);}
```

传统意义	修正写法1	修正写法2
$a == b$	$\text{sgn}(a - b) == 0$	$\text{fabs}(a - b) < \text{eps}$
$a != b$	$\text{sgn}(a - b) != 0$	$\text{fabs}(a - b) > \text{eps}$
$a < b$	$\text{sgn}(a - b) < 0$	$a - b < -\text{eps}$
$a \leq b$	$\text{sgn}(a - b) \leq 0$	$a - b < \text{eps}$
$a > b$	$\text{sgn}(a - b) > 0$	$a - b > \text{eps}$
$a \geq b$	$\text{sgn}(a - b) \geq 0$	$a - b > -\text{eps}$

Lowbit 前缀和

```

int solve(int n) {
    int res=0;
    n++;
    for(int i=1;n>1;n--=(n>>1),i<=1) res+=i*(n>>1);
    return res;
}

```

随机数

```

mt19937 mt(time(0));
uniform_int_distribution<int> rd1(0,10000); // 整数
uniform_real_distribution<double> rd2(0,1); // 浮点数
cout<<rd1(mt)<<' '<<rd2(mt)<<'\n';

```

builtin 系列函数（位运算）

```

// 返回最低位的 1 是从后向前的第几位，比如 2(10)，返回 2
int __builtin_ffs(unsigned int x)
// 返回前导 0 的个数，比如 2(10)，返回 30，x = 0 时结果未定义
int __builtin_clz(unsigned int x)
// 返回末尾0的个数，比如 2(10)，返回 1，x = 0 时结果未定义
int __builtin_ctz(unsigned int x)
// 返回 1 的个数
int __builtin_popcount(unsigned int x)
// 返回 1 的个数模 2 的结果
int __builtin_parity(unsigned int x)
// unsigned long long 版本在函数名后加 ll

```

