# ACM 模板

黄佳瑞，钱智煊，林乐道

2025 年 8 月 29 日

# 目录

# 1 做题指导

## 1.1 上机前你应该注意什么

1. 预估你需要多少机时，以及写出来以后预计要调试多久，并在纸上记下。
2. 先想好再上机，不要一边写一边想。
3. 如果有好写的题，务必先写好写的。
4. 把题目交给擅长的人来写，而不是空闲的人。

## 1.2 机上你应该注意什么

1. 如果你遇到了问题（做法假了、需要分讨等），先下机并通知队友。不要占着机时想。
2. 建议使用整块时间写题，尽量不要断断续续的写。

## 1.3 交题前你应该注意什么

1. `long long` 开了没有。
2. 数组开够没有。
3. 多测清了没有。
4. 边界数据（`corner case`）考虑了没有
5. 调试输出删了没有

编译命令：

```
g++ X.cpp -Wall -O2 -fsanitize=undefined -fsanitize=address X
# -fsanitize=undefined: 检测未定义行为
# -fsanitize=address: 检测内存溢出
```

## 1.4 如果你的代码挂了

按照优先级列出：

1. 先 P 再下机。（P 还没有送来就分屏调。）
2. 看 `long long` 开了没有，数组开够没有，多测清了没有。
3. 检查 typo ，你有没有打错一些难蚌的地方。
4. 看看你题读错没有。
5. 检查你的代码逻辑，即你的代码实现是否与做法一致。同时让另一个人重新读题。
6. 怀疑做法假了。拉一个人一起看代码。

## 1.5 另外一些注意事项

- 千万注意节奏，不要让任何一个人在一道题上卡得太久。必要的时候可以换题。
- 如果你想要写题，请想好再上机。诸如分类讨论一类的题目，更要想好再上。
- 后期的时候可以讨论，没必要一个人挂题。

# 2 图论

## 2.1 Tarjan 点双、边双

$low$ 表示一个点**不经过父亲**，能到达的最小的 $dfn$ 。

割点：

- 若 $u$ 是根节点，当至少存在 2 条边满足 $low(v) \geq dfn(u)$ 则 $u$ 是割点。
- 若 $u$ 不是根节点，当至少存在 1 条边满足 $low(v) \geq dfn(u)$ 则 $u$ 是割点。

割边：

- 当存在一条边条边满足 $low(v) > dfn(u)$ 则边 $i$ 是割边

注意：记录上一个访问的边时要记录边的编号，不能记录上一个过来的节点（因为会有重边）!!! 或者在加边的时候特判一下，不过注意编号问题。(用输入顺序来对应数组中位置的时候，重边跳过，但是需要 'tot+=2')

## 2.2 Tarjan 圆方树

将点双连通分量定义为不含割点的联通图。

将原图中的点称作圆点，而每个点双对应一个方点。将每个圆点向其所在的所有点双对应方点连边，就得到了圆方树。

特别的，仅含一条边的图也是点双（因其不含割点），这保证了圆方树的连通性。

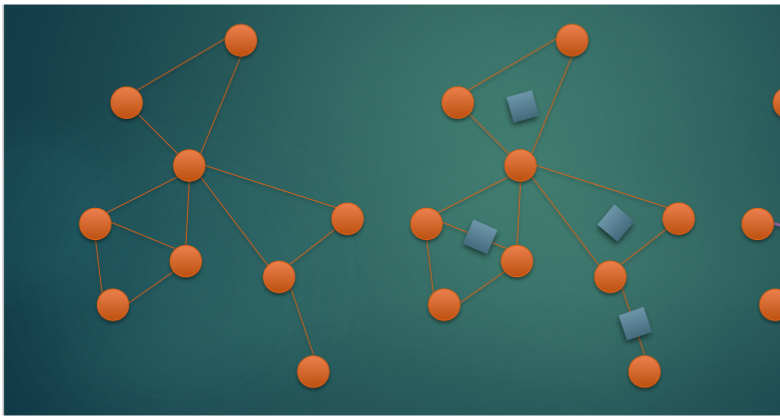


图 1: 圆方树示意图

圆方树会建立很多新的点，所以不要忘记**给数组开两倍**！

```
void tarjan(int u)
{
    dfn[u]=low[u]=++Time,sta[++tp]=u;
    for(int v:G[u])
    {
        if(!dfn[v])
        {
            tarjan(v),low[u]=min(low[u],low[v]);
            if(low[v]==dfn[u])
            {
                int hav=0; ++All;
                for(int x=0;x!=v;tp--) x=sta[tp],T[x].pb(All),T[All].pb(x),hav++;
                T[u].pb(All),T[All].pb(u),;
                siz[All]=++hav;
            }
        }
        else low[u]=min(low[u],dfn[v]);
    }
}
```

## 2.3 O(1) LCA

按照顺序遍历子树，并查集更新。

```
int qhea[Maxq],qver[Maxq],qnex[Maxq],qid[Maxq];
inline void addq(int x,int y,int d)
    { qver[++qtot]=y,qnex[qtot]=qhea[x],qhea[x]=qtot,qid[qtot]=d; }
void dfs(int x,int Last_node)
{
    vis[x]=true;
    for(int i=hea[x];i;i=nex[i])
        if(ver[i]!=Last_node)
            dfs(ver[i],x),fa[ver[i]]=x;
    for(int i=qhea[x];i;i=qnex[i])
        if(vis[qver[i]])
            ans[qid[i]]=Find(qver[i]);
}

for(int i=1;i<=n;i++) fa[i]=i;
for(int i=1,x,y;i<n;i++) x=rd(),y=rd(),add(x,y),add(y,x);
for(int i=1,x,y;i<=m;i++) x=rd(),y=rd(),addq(x,y,i),addq(y,x,i);
dfs(rt,rt);
for(int i=1;i<=m;i++) printf("%d\n",ans[i]);
```

### 2.3.1 2-SAT

2-sat 问题定义为：给定 $m$ 个布尔表达式，每个包含两个布尔变量，形如 $a \vee b$ 。求是否存在一种赋值方式，使得所有表达式都为真。

对于表达式 $a \lor b$ ，显然若 $a$ 假则 $b$ 必真，反之亦然。因此，考虑将每个变量拆为两个点，分别代表此变量取值为真或假。对于每个表达式 $a \lor b$ ，连边 $\neg a \to b$ 与 $\neg b \to a$ 。对于边 $u \to v$ ，意义为若 $u$ 为真则 $v$ 必为真。那么对于最后得到的图，如果同一变量拆出的点处在同一强连通分量中，则无解，因为在可行解中它们的取值必然是不同的，但同一强连通分量中的点取值必然相同；否则，考虑缩点以后得到的 DAG ，取其拓扑序，对于每一个变量，令其取拆出的拓扑序较大的点对应的值即可。特别的，以上的判定是必要的，但我还不知道怎么证明它是充分的。

## 2.4 欧拉路径

欧拉图的判定：考虑起点、终点与中间点的必要条件即可。可以证明必要条件也是充分的。以下给出已确定起点的无向图欧拉路径（回路）的构造算法。

```
int s[N], top, head[N], now[N], used[M << 1], cnt;
struct edge {
    int a, b, next;
} e[M << 1];
void add(int a, int b) {
    e[++cnt] = {a, b, head[a]};
    head[a] = cnt;
} // cnt 的初值为 1 ，head[] 的初值为 0 。即从 2 开始编号，以保证异
  ↪ 或 1 得到反向边。
void dfs(int x) {
    while (now[x]) { // now[] 的初值为 head[] 。
        int i = now[x];
        now[x] = e[now[x]].next;
        if (!used[i]) {
            used[i] = used[i ^1] = 1;
            dfs(e[i].b);
        }
    }
    s[++top] = x;
} // s[] 存储了反向的欧拉路。
```

## 2.5 网络流

### 2.5.1 最大流（dinic）

```
int n, m, flow, S, T, now[N], head[N], cnt = 1, dep[N];
queue<int> q;
struct edge {
    int a, b, next, f;
} e[M << 1];
void add_edge(inta, intb, intf){
    e[++cnt] = {a, b, head[a], f};
    head[a] = cnt;
} // cnt 的初值为 1 ，head[] 的初值为 0 。即从 2 开始编号，以保证异
  ↪ 或 1 得到反向边。
void add(int a, int b, int f) { add_edge(a, b, f), add_edge(b, a, 0); }

bool bfs() {
    for (int i = 1; i <= n; i++) dep[i] = 0;
    while (!q.empty()) q.pop();
    dep[S] = 1, q.push(S), now[S] = head[S];
    while (!q.empty()) {
        int x = q.front(); q.pop();
        if (x == T) return 1;
        for (int i = head[x]; i; i = e[i].next) {
            if (e[i].f  !dep[e[i].b]) {
                dep[e[i].b] = dep[x] + 1;
                now[e[i].b] = head[e[i].b];
                q.push(e[i].b);
            }
        }
    }
    return 0;
}
int dfs(int x, int mn) {
    if (x == T) return mn;
    int res = 0;
    for (int i = now[x]; i  mn; i = e[i].next) {
        now[x] = i;
        if (e[i].f  dep[e[i].b] == dep[x] + 1) {
            int k = dfs(e[i].b, min(mn, e[i].f));
            if (!k) dep[e[i].b] = 0;
            res += k, mn -= k;
            e[i].f -= k, e[i ^1].f+ = k;
        }
    }
    return res;
}
void dinic() {
    while (bfs()) flow += dfs(S, inf);
}
```

### 2.5.2 最小费用最大流（dinic）

```
struct Dinic_cost{
#define Maxn ?
#define Maxm ?
    int tot = 1;
    int tmphea[Maxn], hea[Maxn], nex[Maxm << 1], ver[Maxm << 1];
    ll sumflow, sumcost, edg[Maxm << 1], Cost[Maxm << 1];
    bool inq[Maxn];
    ll dis[Maxn];
    inline void init() { tot = 1, memset(hea, 0, sizeof(hea)); }
    inline void add_edge(intx, inty, lld, llc){
        ver[++tot] = y, nex[tot] = hea[x], hea[x] = tot, edg[tot]
          ↪ = d, Cost[tot] = c;
        ver[++tot] = x, nex[tot] = hea[y], hea[y] = tot, edg[tot]
          ↪ = 0, Cost[tot] = -c;
    }
    inline bool spfa(int s, int t) {
        memset(dis, 0x3f, sizeof(dis)), dis[s] = 0;
        memcpy(tmphea, hea, sizeof(hea));
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int cur = q.front();
            q.pop(), inq[cur] = false;
            for (int i = hea[cur]; i; i = nex[i])
                if (edg[i]  dis[ver[i]] > dis[cur] + Cost[i]) {
                    dis[ver[i]] = dis[cur] + Cost[i];
                    if (!inq[ver[i]])
                        q.push(ver[i]), inq[ver[i]] = true;
                }
        }
        return dis[t] != infll;
    }
    ll dfs(int x, ll flow, int t) {
        if (x == t || !flow) return flow;
        ll rest = flow, tmp;
        inq[x] = true;
        for (int i = tmphea[x]; i  rest; i = nex[i]) {
            tmphea[x] = i;
            if (!inq[ver[i]]  edg[i]  dis[ver[i]] == dis[x] +
              ↪ Cost[i]) {
                if (!(tmp = dfs(ver[i], min(edg[i], rest), t)))
                  ↪ dis[ver[i]] = infll;
                sumcost += Cost[i] * tmp, edg[i] -= tmp, edg[i
                    ^1]+ = tmp, rest− = tmp;
            }
        }
        inq[x] = false;
        return flow - rest;
    }
    inline pll solve(int s, int t) {
        sumflow = sumcost = 0;
        while (spfa(s, t))
            sumflow += dfs(s, infll, t);
        return pll(sumflow, sumcost);
    }
#undef Maxn
#undef Maxm
} G;
```

### 2.5.3 最小费用最大流（edmonds-karp）

```
struct edge {
    int a, b, next, f, v;
} e[M << 1];
int n, m, s, t, cnt = 1, flow, cost, head[N], dis[N], inq[N],
  ↪ pre[N], f[N];
queue<int> q;

void add_edge(inta, intb, intf, intv){
    e[++cnt] = {a, b, head[a], f, v};
    head[a] = cnt;
} // cnt 的初值为 1 ，head[] 的初值为 0 。即从 2 开始编号，以保证异
  ↪ 或 1 得到反向边。
void add(int a, int b, int f, int v) {
    add_edge(a, b, f, v), add_edge(b, a, 0, −v);
}

bool spfa() {
    for (int i = 1; i <= n; i++) f[i] = dis[i] = inf, inq[i] = 0,
      ↪ pre[i] = -1;
    q.push(s), inq[s] = 1, dis[s] = 0, pre[s] = -1;
    while (!q.empty()) {
        int x = q.front(); q.pop();
        inq[x] = 0;
```

```
        for (int i = head[x]; i; i = e[i].next) {
            if (e[i].f > 0  dis[e[i].b] > dis[x] + e[i].v) {
                dis[e[i].b] = dis[x] + e[i].v;
                pre[e[i].b] = i;
                f[e[i].b] = min(e[i].f, f[x]);
                if (!inq[e[i].b]) inq[e[i].b] = 1,
                  ↪ q.push(e[i].b);
            }
        }
    }
    return pre[t] != -1;
}
void mcmf() {
    while (spfa()) {
        flow += f[t], cost += f[t] * dis[t];
        int x = t;
        while (x != s) {
            e[pre[x]].f -= f[t], e[pre[x] ^1].f+ = f[t];
            x = e[pre[x]].a;
        }
    }
}
```

### 2.5.4 无源汇有上下界可行流

```
#include <bits/stdc++.h>
using namespace std;
#define infll 0x3f3f3f3f3f3f3f3f
#define inf 0x3f3f3f3f
#define pb push_back
#define eb emplace_back
#define pa pair<int, int>
#define fi first
#define se second
typedef long long ll;
inline int rd() {
    int x = 0;
    char ch, t = 0;
    while (!isdigit(ch = getchar())) t |= ch == '-';
    while (isdigit(ch)) x = x * 10 + (ch ^48), ch = getchar();
    return t ? -x : x;
}
struct Dini {
#define Maxn 505
#define Maxm 20005
    int tot = 1;
    int hea[Maxn], tmphea[Maxn], dep[Maxn];
    int nex[Maxm << 1], ver[Maxm << 1], num[Maxm << 1];
    ll edg[Maxm << 1];
    inline void addedge(int x, int y, int d, int _n){
        ver[++tot] = y, nex[tot] = hea[x], hea[x] = tot, edg[tot]
          ↪ = d, num[tot] = -1;
        ver[++tot] = x, nex[tot] = hea[y], hea[y] = tot, edg[tot]
            = 0, num[tot] = _n;
    }
    inline bool bfs(int s, int t) {
        memcpy(tmphea, hea, sizeof(hea));
        memset(dep, 0, sizeof(dep)), dep[s] = 1;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int cur = q.front();
            q.pop();
            if (cur == t) return true;
            for (int i = hea[cur]; i; i = nex[i])
                if (edg[i] > 0  !dep[ver[i]])
                    dep[ver[i]] = dep[cur] + 1, q.push(ver[i]);
        }
        return false;
    }
    ll dfs(int x, int t, ll flow) {
        if (!flow || x == t) return flow;
        ll rest = flow, tmp;
        for (int i = tmphea[x]; i  rest; i = nex[i]) {
            tmphea[x] = i;
            if (dep[ver[i]] == dep[x] + 1  edg[i] > 0) {
                if (!(tmp = dfs(ver[i], t, min(rest, edg[i]))))
                  ↪ dep[ver[i]] = 0;
                edg[i] -= tmp, edg[i ^1]+ = tmp, rest- = tmp;
            }
        }
        return flow - rest;
    }
    inline ll solve(int s, int t) {
        ll sum = 0;
        while (bfs(s, t))
            sum += dfs(s, t, infll);
        return sum;
    }
#undef Maxn
#undef Maxm
} G;
#define Maxn 205
#define Maxm 10205
int n, m, ss, tt;
int ans[Maxm];
ll needin, needout;
ll Ind[Maxn], Outd[Maxn];
int main() {
    n = rd(), m = rd(), ss = n + 1, tt = n + 2;
    for (int i = 1, x, y, Inf, Sup; i <= m; i++) {
        x = rd(), y = rd(), Inf = rd(), Sup = rd();
        G.addedge(x, y, Sup - Inf, i);
        Outd[x] += Inf;
        Ind[y] += Inf;
        ans[i] = Inf;
    }
    for (int i = 1; i <= n; i++) {
        if (Ind[i] > Outd[i])
            G.addedge(ss, i, Ind[i] - Outd[i], -1), needin +=
              ↪ Ind[i] - Outd[i];
        if (Ind[i] < Outd[i])
            G.addedge(i, tt, Outd[i] - Ind[i], -1), needout +=
              ↪ Outd[i] - Ind[i];
    }
    ll tmp = G.solve(ss, tt);
    if (needin != needout || needin != tmp) printf("NO\n");
    else {
        for (int i = 2; i <= G.tot; i++)
            if (G.num[i] != -1) ans[G.num[i]] += G.edg[i];
        printf("YES\n");
        for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    }
    return 0;
}
```

### 2.5.5 有源汇有上下界最大流

```
#include <bits/stdc++.h>
using namespace std;
#define infll 0x3f3f3f3f3f3f3f3f
#define inf 0x3f3f3f3f
#define pb push_back
#define eb emplace_back
#define pa pair<int, int>
#define fi first
#define se second
typedef long long ll;
inline int rd() {
    int x = 0;
    char ch, t = 0;
    while (!isdigit(ch = getchar())) t |= ch == '-';
    while (isdigit(ch)) x = x * 10 + (ch ^48), ch = getchar();
    return t ? -x : x;
}
struct Dinic {
#define Maxn 505
#define Maxm 20005
    int tot = 1;
    int hea[Maxn], tmphea[Maxn], dep[Maxn];
    int nex[Maxm << 1], ver[Maxm << 1], num[Maxm << 1];
    ll edg[Maxm << 1];
    inline int addedge(int x, int y, ll d, int _n){
        ver[++tot] = y, nex[tot] = hea[x], hea[x] = tot, edg[tot]
          ↪ = d, num[tot] = -1;
        ver[++tot] = x, nex[tot] = hea[y], hea[y] = tot, edg[tot]
            = 0, num[tot] = _n;
        return tot;
    }
    inline bool bfs(int s, int t) {
        memcpy(tmphea, hea, sizeof(hea));
        memset(dep, 0, sizeof(dep)), dep[s] = 1;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int cur = q.front();
            q.pop();
            if (cur == t) return true;
            for (int i = hea[cur]; i; i = nex[i])
                if (edg[i] > 0  !dep[ver[i]])
                    dep[ver[i]] = dep[cur] + 1, q.push(ver[i]);
        }
        return false;
    }
    ll dfs(int x, int t, ll flow) {
        if (!flow || x == t) return flow;
        ll rest = flow, tmp;
        for (int i = tmphea[x]; i  rest; i = nex[i]) {
            tmphea[x] = i;
```

```
            if (dep[ver[i]] == dep[x] + 1  edg[i] > 0) {
                if (!(tmp = dfs(ver[i], t, min(rest, edg[i]))))
                    dep[ver[i]] = 0;
                edg[i] -= tmp, edg[i ^1]+ = tmp, rest− = tmp;
            }
        }
        return flow - rest;
    }
    inline ll solve(int s, int t) {
        ll sum = 0;
        while (bfs(s, t))
            sum += dfs(s, t, infll);
        return sum;
    }
#undef Maxn
#undef Maxm
} G;
#define Maxn 505
#define Maxm 20005
int n, m, ss, tt, s, t;
int ans[Maxm];
ll needin, needout;
ll Ind[Maxn], Outd[Maxn];
int main() {
    n = rd(), m = rd(), s = rd(), t = rd(), ss = n + 1, tt = n +
      ↪ 2;
    for (int i = 1, x, y, Inf, Sup; i <= m; i++) {
        x = rd(), y = rd(), Inf = rd(), Sup = rd();
        G.addedge(x, y, Sup - Inf, i);
        Outd[x] += Inf;
        Ind[y] += Inf;
        ans[i] = Inf;
    }
    for (int i = 1; i <= n; i++) {
        if (Ind[i] > Outd[i])
            G.addedge(ss, i, Ind[i] - Outd[i], -1), needin +=
              ↪ Ind[i] - Outd[i];
        if (Ind[i] < Outd[i])
            G.addedge(i, tt, Outd[i] - Ind[i], -1), needout +=
              ↪ Outd[i] - Ind[i];
    }
    G.addedge(t, s, infll, -1);
    ll tmp = G.solve(ss, tt);
    if (needin != needout || needin != tmp) printf("please go
      ↪ home to sleep\n");
    else {
        tmp = G.edg[G.tot];
        G.edg[G.tot] = G.edg[G.tot - 1] = 0;
        tmp += G.solve(s, t);
        printf("%lld\n", tmp);
    }
    return 0;
}
```

# 3 数据结构

## 3.1 平衡树

### 3.1.1 无旋 Treap

structure/fhq.cpp

### 3.1.2 平衡树合并

注意这里的合并是指合并两个值域相交的集合，并且这个操作应该只有 treap 支持。代码如下：

structure/treap-merge.cpp 可以证明，以上操作的总时间复杂度为 $O(n\log n\log V)$，其中 $n$ 为元素总个数，$V$ 为值域。而在特定的场景下，时间复杂度会更优。（例如若只支持合并与分裂操作，则时间复杂度为 $O(n\log n)$ 。）

### 3.1.3 Splay

structure/splay.cpp

## 3.2 动态树

structure/LCT.cpp

## 3.3 珂朵莉树

structure/ODT.cpp

## 3.4 李超线段树

### 3.4.1 修改与询问

structure/lichao.cpp

### 3.4.2 合并

structure/lichao-merge.cpp

## 3.5 二维树状数组

structure/2D-BIT.cpp

## 3.6 虚树

structure/virtual-tree.cpp

## 3.7 左偏树

structure/heap.cpp 特别的，若要求给定节点所在左偏树的根，须使用并查集。对于每个节点维护 rt[] 值，查找根时使用函数：

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

在合并节点时，加入：

```
rt[x] = rt[y] = merge(x, y);
```

在弹出最小值时加入：

```
rt[ls(x)] = rt[rs(x)] = rt[x] = merge(ls(x), rs(x));
```

另外，删除过的点是不能复用的，因为这些点可能作为并查集的中转节点。

## 3.8 吉司机线段树

- 区间取 min 操作：通过维护区间次小值实现，即将区间取 min 转化为对区间最大值的加法，当要取 min 的值 v 大于次小值时停止递归。时间复杂度通过标记回收证明，即将区间最值视作标记，这样每次多余的递归等价于标记回收，总时间复杂度为 $O(m\log n)$。
- 区间历史最大值：通过维护加法标记的历史最大值实现。应该可以通过 max+ 矩乘维护。
- 区间历史版本和：使用矩阵乘法维护。由于矩乘具备结合律，历史版本和可以简单实现。

总而言之，主要的手段是标记回收和矩乘合并标记。

structure/seg-beats.cpp

## 3.9 树分治

熟知序列分治的过程是选取恰当的分治点并考虑所有跨过分治点的区间。而树分治的过程也是类似的，以点分治为例，每一次选择当前联通块的重心作为分治点，然后考虑所有跨越分治点的路径，并对分割出的联通块递归。

若要处理树上邻域问题，可以考虑建出点分树。处理点 x 的询问时，只需考虑 x 在点分树上到根的路径，每一次加上除开 x 所在子树的答案即可。

structure/treer-divide.cpp

# 4 字符串

## 4.1 Hash 类

```cpp
const ll P = (ll)1e18+9;
static constexpr u128 inv = [](){
    u128 ret = P;
    for(int i=0;i<6;i++)ret*=2-ret*P;
    return ret;
}();
constexpr u128 chk = u128(-1) / P;
bool check(i128 a,i128 b){
    if(a<b)swap(a,b);
    return (a-b)*inv<=chk;
}
```

## 4.2 后缀数组（与后缀树）

```cpp
const int N = 1e6 + 5;

char s[N];
int sa[N], rk[N], n, h[N];
// 后缀数组。h[i] = lcp(sa[i], sa[i - 1])
int rt, ls[N], rs[N], fa[N], val[N];
// 后缀树。实际上就是 height 数组的笛卡尔树。
// val[x] : x 与 fa[x] 对应的子串等价类的大小之差，也就是 x 贡献的本质不同子串数。

struct suffix {
    int k1[N], k2[N << 1], cnt[N], mx, stk[N], top;
    void radix_sort(){
        for (int i = 1; i <= mx; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) cnt[k1[i]]++;
        for (int i = 1; i <= mx; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i >= 1; i--) sa[cnt[k1[k2[i]]]--] = k2[i];
    } // 基数排序
    void sort() {
        mx = 'z';
        for (int i = 1; i <= n; i++) k1[i] = s[i], k2[i] = i;
        radix_sort();
        for (int j = 1; j <= n; j <<= 1) {
            int num = 0;
            for (int i = n - j + 1; i <= n; i++) k2[++num] = i;
            for (int i = 1; i <= n; i++) if (sa[i] > j) k2[++num] = sa[i] - j;
            radix_sort();
            for (int i = 1; i <= n; i++) k2[i] = k1[i];
            k1[sa[1]] = mx = 1;
            for (int i = 2; i <= n; i++) k1[sa[i]] = k2[sa[i]] == k2[sa[i - 1]]  k2[sa[i] + j] == k2[sa[i - 1] + j] ? mx : ++mx;
        }
    } // 后缀排序
    void height() {
        for (int i = 1; i <= n; i++) rk[sa[i]] = i;
        int k = 0;
        for (int i = 1; i <= n; i++) {
            if (k) k--;
            if (rk[i] == 1) continue;
            int j = sa[rk[i] - 1];
            while (i + k <= n  j + k <= n  s[i + k] == s[j + k]) k++;
            h[rk[i]] = k;
        }
    } // 计算 height 数组
    void build() {
        if (n == 1) return rt = 1, void();
        ls[2] = n + 1, rs[2] = n + 2, fa[ls[2]] = fa[rs[2]] = rt = stk[++top] = 2;
        for (int i = 3; i <= n; i++) {
            while (top  h[stk[top]] > h[i]) top--;
            int p = stk[top];
            if (top) ls[i] = rs[p], fa[rs[p]] = i, rs[p] = i, fa[i] = p;
            else ls[i] = rt, fa[rt] = i, rt = i;
            rs[i] = n + i, fa[rs[i]] = i, stk[++top] = i;
        }
        for (int i = 2; i <= n + n; i++) val[i] = (i > n ? n - sa[i - n] + 1 : h[i]) - h[fa[i]];
    } // 构建后缀树
} SA;
```

## 4.3 AC 自动机

```cpp
struct ACAM {
    int ch[N][26], cnt, fail[N], vis[N];
    queue <int> q;
    void insert(char *s, int n, int id) {
        int x = 0;
        for (int i = 1; i <= n; i++) {
            int nw = s[i] - 'a';
            if (!ch[x][nw]) ch[x][nw] = ++cnt;
            x = ch[x][nw];
        }
        vis[x]++;
    }
    void build() {
        for (int i = 0; i < 26; i++) {
            if (ch[0][i]) q.push(ch[0][i]);
        }
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int i = 0; i < 26; i++) {
                if (ch[x][i]) {
                    fail[ch[x][i]] = ch[fail[x]][i];
                    q.push(ch[x][i]);
                } else ch[x][i] = ch[fail[x]][i];
            }
        }
    }
    void match(char *s, int n) {
        int x = 0;
        for (int i = 1; i <= n; i++) {
            int nw = s[i] - 'a';
            x = ch[x][nw];
        }
    }
} AC;
```

## 4.4 回文自动机

```cpp
struct PAM {
   int fail[N], ch[N][26], len[N], s[N], tot, cnt, lst;
    // fail : 当前节点的最长回文后缀。
    // ch : 在当前节点的前后添加字符，得到的回文串。
   PAM() {
      len[0] = 0, len[1] = -1, fail[0] = 1;
      tot = lst = 0, cnt = 1, s[0] = -1;
   }
   int get_fail(intx){
      while (s[tot - 1 - len[x]] != s[tot]) x = fail[x];
      return x;
   }
   void insert(char c) {
      s[++tot] = c - 'a';
      int p = get_fail(lst);
      if(!ch[p][s[tot]]) {
         len[++cnt] = len[p] + 2;
         int t = get_fail(fail[p]);
         fail[cnt] = ch[t][s[tot]];
         ch[p][s[tot]] = cnt;
      }
      lst=ch[p][s[tot]];
   }
} pam;
```

## 4.5 Manacher 算法

```cpp
char s[N], t[N];
// s[] : 原串，t[] : 加入分割字符的串，这样就只需考虑奇回文串了。
int mxp, cen, r[N], n;
// mxp : 最右回文串的右端点的右侧，cen : 最右回文串的中心，r[i] : 以位置 i 为中心的回文串半径，即回文串的长度一半向上取整。

void manacher() {
    t[0] = '~', t[1] = '#';
    // 在 t[0] 填入特殊字符，防止越界。
    int m = 1;
    for (int i = 1; i <= n; i++) {
        t[++m] = s[i], t[++m] = '#';
    }
    for (int i = 1; i <= m; i++) {
        r[i] = mxp > i ? min(r[2 * cen - i], mxp - i) : 1;
        // 若 i (cen, mxp)，则由对称性 r[i] 至少取 min(r[2 * cen - i], mxp - i)。否则直接暴力扩展。
        while (t[i + r[i]] == t[i - r[i]]) r[i]++;
        if (i + r[i] > mxp) mxp = i + r[i], cen = i;
    }
}
```

## 4.6 KMP 算法与 border 理论

```
char s[N], t[N];
int nex[N], n, m;

void kmp() {
    int j = 0;
    for (int i = 2; i <= n; i++) {
        while (j  s[j + 1] != s[i]) j = nex[j];
        if (s[j + 1] == s[i]) j++;
        nex[i] = j;
    }
}

void match() {
    int j = 0;
    for (int i = 1; i <= m; i++) {
        while (j  s[j + 1] != t[i]) j = nex[j];
        if (s[j + 1] == t[i]) j++;
        if (j == n) {
            //match.
            j = nex[j];
        }
    }
}
```

字符串的 border 理论：以下记字符串 $S$ 的长度为 $n$ 。

- 若串 $S$ 具备长度为 $m$ 的 border ，则其必然具备长度为 $n-m$ 的周期，反之亦然。
- 弱周期性引理：若串 $S$ 存在周期 $p$ 、$q$ ，且 $p+q\le n$ ，则 $S$ 必然存在周期 $\gcd(p,q)$ 。
- 引理 1：若串 $S$ 存在长度为 $m$ 的 border $T$，且 $T$ 具备周期 $p$ ，满足 $2m-n\ge p$ ，则 $S$ 同样具备周期 $p$ 。
- 周期性引理：若串 $S$ 存在周期 $p$ 、$q$ ，满足 $p+q-\gcd(p,q)\le n$ ，则串 $S$ 必然存在周期 $\gcd(p,q)$ 。
- 引理 2：串 $S$ 的所有 border 的长度构成了 $O(\log n)$ 个不交的等差数列。更具体的，记串 $S$ 的最小周期为 $p$ ，则其所有长度包含于区间 $[n \bmod p+p, n)$ 的 border 构成了一个等差数列。
- 引理 3：若存在串 $S$ 、$T$ ，使得 $2|T|\ge n$ ，则 $T$ 在 $S$ 中的所有匹配位置构成了一个等差数列。
- 引理 4：PAM 的失配链可以被划分为 $O(\log n)$ 个等差数列。

## 4.7 Z 函数

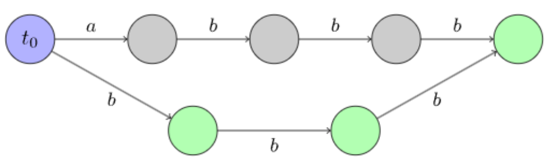
Z 函数用于求解字符串的每一个后缀与其本身的 lcp 。其思路和 manacher 算法基本一致，都是维护一个扩展过的最右端点和对应的起点，而当前点要么暴力扩展使最右端点右移，要么处在记录的起点和终点间，从而可以利用已有的信息快速转移。

```
char s[N];
int z[N];

void zfunc() {
    z[1] = n;
    int j = 0;
    for (int i = 2; i <= n; i++) {
        if (j  j + z[j] - 1 >= i) z[i] = min(z[i - j + 1], j + z[j] - i);
        while (i + z[i] <= n  s[i + z[i]] == s[1 + z[i]]) z[i]++;
        if (i + z[i] > j + z[j]) j = i;
    }
}
```

## 4.8 后缀自动机

字符串 $S$ 的 SAM 是一个接受 $S$ 的所有后缀的最小 DFA 。例如 $S=\text{abbb}$ 的 SAM 如下：




SAM 的后缀链接构成了原串的后缀树。

对于代码实现，首先，我们实现一种存储一个转移的全部信息的数据结构。如果需要的话，你可以在这里加入一个终止标记，也可以是一些其它信息。我们将用一个 map 存储转移的列表，允许我们在总计 $O(n)$ 的空间复杂度和 $O(n\log|\Sigma|)$ 的时间复杂度内处理整个字符串。

```
struct state {
    int len, link;
    std::map<char, int> next;
};
```

SAM 本身将会存储在一个 `state` 结构体数组中。我们记录当前自动机的大小 `sz` 和变量 `last` ，当前整个字符串对应的状态。

```
const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;
```

我们定义一个函数来初始化 SAM（创建一个只有初始状态的 SAM）。

```
void sam_init(){
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}
```

最终我们给出主函数的实现：给当前行末增加一个字符，对应的在之前的基础上建造自动机。

```
void sam_extend(char c){
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1  !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1  st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

# 5 线性代数

## 5.1 高斯消元

```
scanf("%d", n);
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n + 1; j++)
        scanf("%lf", a[i][j]);
for (int i = 1, Max = 1; i <= n; Max = ++i) {
    for (int s = i + 1; s <= n; s++)
        if (fabs(a[s][i]) > fabs(a[Max][i]))
            Max = s;  // 找出绝对值最大的
    for (int j = 1; j <= n + 1; j++)
        swap(a[i][j], a[Max][j]);
    if (a[i][i] < 10e-8  a[i][i] > -10e-8) {
        p = false;
        break;
    }  // 记得 double 的精度问题
    for (int s = 1; s <= n; s++)
        if (s != i)  // 这样省去了第二步处理的麻烦
        {
            double tmp = 0 - (a[s][i] / a[i][i]);
            a[s][i] = 0;
            for (int j = i + 1; j <= n + 1; j++)
```

```
                a[s][j] += tmp * a[i][j];
        }
}
if (p)
    for (int i = 1; i <= n; i++)
        printf("%.2lf\n", a[i][n + 1] / a[i][i]);
else
    printf("No Solution\n");
```

## 5.2 线性基

```
void Insert(ll x) // 加入一个数
{
    for (int i = 62; i >= 0; i--)
        if ((x >> i)  1) {
            if (!p[i]) {
                p[i] = x;
                break;
            }
            x =p[i];// [0,i−1]
        }
}
bool exist(ll x) // 查询一个元素是否可以被异或出来
{
    for (int i = 62; i >= 0; i--)
        if ((x >> i)  1) x =p[i];
    return x == 0;
}
ll query_max()//
{
    ll ret = 0;
    for (int i = 62; i >= 0; i--)
        if ((ans p[i]) > ans)ans=p[i];
    return ans;
}
ll query_min()//
{
    for (int i = 0; i <= 62; i++)
        if (p[i]) return p[i];
    return 0;
}
ll kth(ll k) // 查询异或第 k 小
{
    // 重建 d 数组，求出哪些位可以被异或为 1
    // d[i] 中任意两个数在任意一个二进制位上不可能同时为 1
    for (int i = 62; i >= 1; i--) // 从高到低防止后效性
        for (int j = i - 1; j >= 0; j--)
            if ((p[i] >> j)  1) p[i] =p[j];
    for (int i = 0; i <= 62; i++)
        if (p[i]) d[cnt++] = p[i];

    if (!k) return 0ll;                    // 特判 0
    if (k >= (1ll << (ll)cnt)) return -1ll; // k 大于可以表示出的
    ↪ 数的个数
    ll ret = 0;
    for (int i = 62; i >= 0; i--)
        if ((k >> i)  1) ret =d[i];
    return ret;
}
ll Rank(ll k) {
    ll Now = 1, ret = 0;
    for (int i = 0; i <= 62; i++)
        if (p[i]) // 记得保证 k 可以被异或出来
        {
            if ((k >> i)  1) ret += Now;
            Now <<= 1;
        }
    return ret;
}
```

## 5.3 行列式

容易证明转置后行列式相等，积的行列式等于行列式的积。但是这样的性质并不对和成立，而每次只能拆一行或一列。
以下是任意模数求行列式的算法：

```
// 解法 1: 类似辗转相除
n = rd(), mod = rd();
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        a[i][j] = rd() % mod;
for (int i = 1; i <= n; i++) {
    for (int j = i + 1; j <= n; j++) {
        while (a[j][i]) {
            ll tmp = a[i][i] / a[j][i];
            for (int k = i; k <= n; k++)
                a[i][k] = (a[i][k] - tmp * a[j][k] % mod + mod) %
                ↪ mod;
            swap(a[i], a[j]), w = -w; // ?
        }
    }
}
for (int i = 1; i <= n; i++)
    ans = ans * a[i][i] % mod;
printf("%lld\n", (mod + w * ans) % mod);
```

## 5.4 矩阵树定理

以下叙述允许重边，不允许自环。
对于无向图 $G$ ，定义度数矩阵 $D$ 为：

$$D_{ij} = \deg(i)[i = j]$$

设 $\#e(i, j)$ 为连接点 $i$ 和 $j$ 的边数，定义邻接矩阵 $A$ 为：

$$A_{ij} = \#e(i, j)$$

显然 $A_{ii} = 0$ 。定义 Laplace 矩阵 $L$ 为 $D - A$ ，记 $G$ 的生成树个数为 $t(G)$ ，则其恰为 $L$ 的任意一个 $n - 1$ 阶主子式的值。
对于有向图 $G$ ，分别定义出度矩阵 $D^{out}$ 和入度矩阵 $D^{in}$ 为：

$$D^{out}_{ij} = \deg^{out}(i)[i = j]$$
$$D^{in}_{ij} = \deg^{in}(i)[i = j]$$

设 $\#e(i, j)$ 为从点 $i$ 到 $j$ 的边数，定义邻接矩阵 $A$ 为：

$$A_{ij} = \#e(i, j)$$

显然 $A_{ii} = 0$ 。再分别定义出度 Laplace 矩阵 $L^{out}$ 和入度 Laplace 矩阵 $L^{in}$ 为：

$$L^{out} = D^{out} - A$$
$$L^{in} = D^{in} - A$$

分别记 $G$ 的以 $k$ 为根的根向树形图个数为 $t^{root}(k)$ ，以及以 $k$ 为根的叶向树形图个数为 $t^{leaf}(k)$ 。则 $t^{root}(k)$ 恰为 $L^{out}$ 的删去 $k$ 行 $k$ 列的 $n - 1$ 阶主子式的值；$t^{leaf}(k)$ 恰为 $L^{in}$ 的删去 $k$ 行 $k$ 列的 $n - 1$ 阶主子式的值。

## 5.5 单纯形法

线性规划的标准型：

$$\begin{aligned} \text{maximize :} \quad & c^{\mathrm{T}} x \\ \text{constraints :} \quad & Ax \le b \\ & x \ge 0 \end{aligned}$$

在标准型的基础上得到松弛型：

$$\begin{aligned} \text{maximize :} \quad & c^{\mathrm{T}} x \\ \text{constraints :} \quad & \alpha = b - Ax \\ & \alpha, x \ge 0 \end{aligned}$$

单纯性法以松弛型为基础。具体的，松弛型隐含了一个基本解，即 $x = 0$ ，$\alpha = b$（这里要求 $b \ge 0$ ）。我们称 $\alpha$ 中的变量为基变量，其余为非基变量。单纯性的主过程被称作 pivot 操作。一次 pivot 操作的本质就是进行基变量与非基变量之间的变换以使得带入基本解的目标函数更大。具体的，我们每一次选定一个在目标函数中系数为正的变量为换入变量，再选择对这个换入变量约束最紧的的线性约束所对应的基变量，称其为换出变量。然后，我们将换入变量和换出变量分别换为基变量和非基变量，并对其余的式子做出对应的代换以使得定义满足即可。
另外单纯型法的时间复杂度虽然是指数级别的，但是跑起来效果还是很好的，期望迭代次数貌似可以大致看作约束个数的平方级别。

```
typedef double db;
const db eps = 1e-8, inf = 1e9;
int n, m, ans;
db b[N], a[N][M], c[M]; // n: 约束个数。m: 变量个数。
```

```
bool dcmp(db x) { return fabs(x) > eps; }

db pivot(int out, int in) {
    b[out] /= a[out][in];
    for (int i = 1; i <= m; i++) if (i != in) a[out][i] /= a[out][in];
    a[out][in] = 1; // 理论上是 1 / a[out][in] ，但这个系数可以任取，但也不要随便取。
    for (int i = 1; i <= n; i++) {
        if (i != out  dcmp(a[i][in])) {
            b[i] -= a[i][in] * b[out];
            for (int j = 1; j <= m; j++)
                if (j != in) a[i][j] -= a[i][in] * a[out][j];
            a[i][in] *= -a[out][in];
        }
    }
    db res = c[in] * b[out];
    for (int i = 1; i <= m; i++) if (i != in) c[i] -= a[out][i] * c[in];
    c[in] *= -a[out][in];
    return res;
}
void simplex() {
    db res = 0;
    while (1) {
        int in = 0;
        for (int i = 1; i <= m; i++) {
            if (c[i] > eps) {
                in = i;
                break;
            }
        }
        if (!in) break; // simplex 完成，找到最优解。
        int out = 0;
        db mn = inf;
        for (int i = 1; i <= n; i++)
            if (a[i][in] > eps  b[i] / a[i][in] < mn) mn = b[i] / a[i][in], out = i;
        if (!out) {
            res = inf;
            break;
        } // 解为无穷大。
        res += pivot(out, in);
    }
    ans = round(res);
}
```

## 5.6 全幺模矩阵

当一个矩阵的任意一个子方阵的行列式都为 $\pm 1, 0$ 时，我们称这个矩阵是全幺模的。

如果单纯形矩阵是全幺模的，那么单纯形就具有整数解。

## 5.7 对偶原理

线性规划的对偶原理：原线性规划与对偶线性规划的最优解相等。即：

$$
\begin{array}{rcccrc}
\text{minimize}: & c^{\mathrm{T}}x & & & \text{maximize}: & b^{\mathrm{T}}y \\
\text{constraints}: & Ax \geq b & & \text{dual} & \text{constraints}: & A^{\mathrm{T}}y \leq c \\
& x \geq 0 & & & & y \geq 0
\end{array}
$$

直观上看，对于一个最小化的线性规划，我们尝试构造一个最大化的线性规划，使得它们目标函数的最优解相同。具体的，为每个约束设置一个非负的新变量，代表其系数。对于每个原变量，其对应了一个新约束，要求原约束的线性组合的对应系数不大于原目标函数的系数，从而得到原目标函数的下界。而新目标函数则要使得原约束的组合最大化，从而得到最紧的下界。而线性规划对偶性则指出，原线性规划的最优解必然与对偶线性规划的最优解相等。

对偶线性规划具备互补松弛性。即，设 $x$ 和 $y$ 分别为原问题与对偶问题的可行解，则 $x$ 和 $y$ 均为最优解，当且仅当以下两个命题同时成立：

$$\forall j \in [1, m], x_j = 0 \vee \sum_{i=1}^{n} a_{ij} y_i = c_j$$

$$\forall i \in [1, n], y_i = 0 \vee \sum_{j=1}^{m} a_{ij} x_j \leq b_i$$

对偶松弛性的意义是，其指出若最优解中的变量不取 0 ，则对应约束在最优解中一定取等。

# 6 多项式

## 6.1 FFT

```
struct comp {
    double x, y;
    comp(double X = 0, double Y = 0) { x = X, y = Y; }
    comp operator + (const comp rhs) const { return comp(x + rhs.x, y + rhs.y); }
    comp operator - (const comp rhs) const { return comp(x - rhs.x, y - rhs.y); }
    comp operator * (const comp rhs) const { return comp(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    comp operator / (const comp rhs) const {
        double t = rhs.x * rhs.x + rhs.y * rhs.y;
        return {(x * rhs.x + y * rhs.y) / t, (y * rhs.x - x * rhs.y) / t};
    }
};
const double pi = acos(-1);
int lim, tr[N];
void adjust(int n) { // n: 多项式的次数。
    lim = 1;
    while (lim <= n) lim <<= 1;
    for (int i = 0; i < lim; i++) tr[i] = (tr[i >> 1] >> 1) | ((i & 1) ? lim >> 1 : 0);
} // 准备蝶形变换。
void fft(comp *f, int op) { // op: 1 为 dft , -1 为 idft 。
    for (int i = 0; i < lim; i++) if (tr[i] < i) swap(f[tr[i]], f[i]);
    for (int l = 1; l < lim; l <<= 1) {
        comp w1(cos(2 * pi / (l << 1)), sin(2 * pi / (l << 1)) * op);
        for (int i = 0; i < lim; i += l << 1) {
            comp w(1, 0);
            for (int j = i; j < i + l; j++, w = w * w1) {
                comp x = f[j], y = f[j + l] * w;
                f[j] = x + y, f[j + l] = x - y;
            }
        }
    }
    if (op == -1) {
        for (int i = 0; i < lim; i++) f[i].x /= lim;
    }
}
```

## 6.2 NTT

```
const int mod = 998244353, G = 3, iG = 332748118;
int tr[N], lim;
void adjust(int n) { // n: 多项式的次数。
    lim = 1;
    while (lim <= n) lim <<= 1;
    for (int i = 0; i < lim; i++) tr[i] = (tr[i >> 1] >> 1) | ((i & 1) ? lim >> 1 : 0);
} // 准备蝶形变换。
void ntt(int *f, int op) { // op: 1 为 dft , -1 为 idft 。
    for (int i = 0; i < lim; i++) if (tr[i] < i) swap(f[tr[i]], f[i]);
    for (int l = 1; l < lim; l <<= 1) {
        int w1 = qpow(op == 1 ? G : iG, (mod - 1) / (l << 1)); // qpow: 快速幂。
        for (int i = 0; i < lim; i += l << 1) {
            for (int j = i, w = 1; j < i + l; j++, (w *= w1) %= mod) {
                int x = f[j], y = f[j + l] * w % mod;
                f[j] = (x + y) % mod, f[j + l] = (x - y) % mod;
            }
        }
    }
    if (op == -1) {
        int iv = qpow(lim); // 算逆元。
        for (int i = 0; i < lim; i++) (f[i] *= iv) %= mod;
    }
}
```

## 6.3 集合幂级数

### 6.3.1 并卷积、交卷积与子集卷积

集合并等价于二进制按位或，因此并卷积的计算实际上就是做高维前缀和以及差分，也被称作莫比乌斯变换。

```
void fmt(int *f, int op) { // op: 1 为 dft , -1 为 idft 。
    for (int l = 1; l < lim; l <<= 1) {
        for (int i = 0; i < lim; i += l << 1) {
            for (int j = i; j < i + l; j++) {
                f[j + l] += f[j] * op;
            }
        }
    }
}
```

而集合交卷积则对应后缀和。

```
void fmt(int *f, int op) { // op: 1 为 dft , -1 为 idft 。
    for (int l = 1; l < lim; l <<= 1) {
        for (int i = 0; i < lim; i += l << 1) {
            for (int j = i; j < i + l; j++) {
                f[j] += f[j + l] * op;
            }
        }
    }
}
```

子集卷积则较为特殊，为了使得产生贡献的集合没有交集，考虑引入代表集合大小的占位符。这样只需做 $n$ 次 FMT，再枚举长度做 $n^2$ 次卷积。因为 FMT 具备线性性，所以最后只需做 $n$ 次 iFMT 即可。

```
void multi(int *f, int *g, int *h) { // 对 f 和 g 做子集卷积，答案为 h 。
    static int a[n][lim], b[n][lim], c[n][lim]; // lim = 1 << n
    memset(a, 0, sizeof a);
    memset(b, 0, sizeof b);
    memset(c, 0, sizeof c);
    for (int i = 0; i < lim; i++) a[pcnt[i]][i] = f[i]; // pcnt[i] = popcount(i)
    for (int i = 0; i < lim; i++) b[pcnt[i]][i] = g[i];
    for (int i = 0; i <= n; i++) fmt(a[i], 1), fmt(b[i], 1);
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= i; j++)
            for (int k = 0; k < lim; k++)
                c[i][k] += a[j][k] * b[i - j][k];
    for (int i = 0; i <= n; i++) fmt(c[i], -1);
    for (int i = 0; i < lim; i++) h[i] = c[pcnt[i]][i];
}
```

特别的，子集卷积等价于 $n$ 元保留到一次项的线性卷积。

### 6.3.2 对称差卷积

集合对称差等价于按位异或，而异或卷积则等价于 $n$ 元模 2 的循环卷积，因此，FWT 实质上和 $n$ 元 FFT 没有什么区别。

```
void fwt(int *f, int op) { // op: 1 为 dft , -1 为 idft 。
    for (int l = 1; l < lim; l <<= 1) {
        for (int i = 0; i < lim; i += l << 1) {
            for (int j = i; j < i + l; j++) {
                int x = f[j], y = f[j + l];
                f[j] = x + y, f[j + l] = x - y;
                if (op == -1) f[j] /= 2, f[j + l] /= 2;
                // 模意义下改成乘逆元。
            }
        }
    }
}
```

## 6.4 多项式全家桶

```
const int N = 5000005;
const long long mod = 998244353;
#define int long long
namespace poly {
    const long long G = 3, iG = 332748118;
    int tr[N], lim;
    int qpow(int a, int b = mod - 2) {
        int res = 1;
        while (b) {
            if (b  1) (res *= a) %= mod;
            (a *= a) %= mod, b >>= 1;
        }
        return res;
    }
    void adjust(int n) {
        lim = 1;
        while (lim <= n) lim <<= 1;
        for (int i = 0; i < lim; i++) tr[i] = (tr[i >> 1] >> 1) | ((i  1) ? lim >> 1 : 0);
    } // 准备蝶形变换
    void copy(int *f, int *g, int n) { for (int i = 0; i <= n; i++) f[i] = g[i]; }
    void clear(int *f, int n) { for (int i = 0; i <= n; i++) f[i] = 0; }
    void erase(int *f, int l, int r) { for (int i = l; i <= r; i++) f[i] = 0; }
    void reverse(int *f, int n) { for (int i = 0; i <= n / 2; i++) swap(f[i], f[n - i]); }
    void integral(int *f, int n) {
        for (int i = n + 1; i >= 1; i--) f[i] = f[i - 1] * qpow(i) % mod;
        f[0] = 0;
    } // 对 n 次多项式 f 求积分，默认常数项为 0 。
    void dif(int *f, int n) {
        for (int i = 0; i < n; i++) f[i] = f[i + 1] * (i + 1) % mod;
        f[n] = 0;
    } // 对 n 次多项式 f 求导。
    void ntt(int *f, int op) { // op: 1 为 dft , -1 为 idft 。
        for (int i = 0; i < lim; i++) if (tr[i] < i) swap(f[tr[i]], f[i]);
        for (int l = 1; l < lim; l <<= 1) {
            int w1 = qpow(op == 1 ? G : iG, (mod - 1) / (l << 1)); // qpow: 快速幂。
            for (int i = 0; i < lim; i += l << 1) {
                for (int j = i, w = 1; j < i + l; j++, (w *= w1) %= mod) {
                    int x = f[j], y = f[j + l] * w % mod;
                    f[j] = (x + y) % mod, f[j + l] = (x - y) % mod;
                }
            }
        }
        if (op == -1) {
            int iv = qpow(lim); // 算逆元。
            for (int i = 0; i < lim; i++) (f[i] *= iv) %= mod;
        }
    }
    void multiply(int *h, int *f, int n, int *g, int m) {
        static int a[N], b[N];
        copy(a, f, n), copy(b, g, m);
        adjust(n + m);
        ntt(a, 1), ntt(b, 1);
        for (int i = 0; i < lim; i++) h[i] = a[i] * b[i] % mod;
        ntt(h, -1);
        clear(a, lim - 1), clear(b, lim - 1);
    } // 计算 f 与 g 的积，存放在 h 中，f 与 g 不变。
    void inverse(int *f, int *g, int n) {
        static int t[N];
        if (!n) return f[0] = qpow(g[0]), void();
        inverse(f, g, n >> 1);
        adjust(2 * n), copy(t, g, n);
        ntt(t, 1), ntt(f, 1);
        for (int i = 0; i < lim; i++) f[i] = f[i] * (2 - f[i] * t[i] % mod) % mod;
        ntt(f, -1), erase(f, n + 1, lim - 1), clear(t, lim - 1);
    } // 计算 g 的 n 次逆，存放在 f 中，g 不变。不要让 f 和 g 为同一个数组。
    void ln(int *f, int *g, int n) {
        static int t[N];
        copy(t, g, n), inverse(f, g, n), dif(t, n);
        adjust(n * 2), ntt(t, 1), ntt(f, 1);
        for (int i = 0; i < lim; i++) (f[i] *= t[i]) %= mod;
        ntt(f, -1), integral(f, n);
        erase(f, n + 1, lim - 1), clear(t, lim - 1);
    } // 要求 g[0] = 1 。
    void exp(int *f, int *g, int n) {
        static int t[N];
        if (!n) return f[0] = 1, void();
        exp(f, g, n >> 1), ln(t, f, n);
        for (int i = 0; i <= n; i++) t[i] = (g[i] - t[i]) % mod;
        t[0]++;
        adjust(n * 2), ntt(f, 1), ntt(t, 1);
        for (int i = 0; i < lim; i++) (f[i] *= t[i]) %= mod;
        ntt(f, -1), clear(t, lim - 1), erase(f, n + 1, lim - 1);
    } // 要求 g[0] = 0 。
    void pow(int *f, int *g, int n, int k) {
        static int t[N];
        ln(t, g, n);
        for (int i = 0; i <= n; i++) (t[i] *= k) % mod;
        exp(f, t, n);
    } // 要求 g[0] = 1 。
    void divide(int *q, int *r, int *f, int n, int *g, int m) {
        static int a[N], b[N], c[N];
        copy(a, f, n), copy(b, g, m);
        reverse(a, n), reverse(b, m);
        inverse(c, b, n - m);
```

```
        multiply(q, a, n - m, c, n - m);
        reverse(q, n - m);
        multiply(a, g, m, q, n - m);
        for (int i = 0; i < m; i++) r[i] = (f[i] - a[i] + mod) %
          ↪ mod;
    } // 多项式带余除法，其中 q 为商，r 为余数。
}
using namespace poly;
```

# 7 数论

## 7.1 中国剩余定理

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

求解 $x$ ，其中 $m_1, m_2, \dots m_n$ 互素。

$$x \equiv \sum_{i=1}^{n} a_i \prod_{j \neq i}^{n} m_j \times \left( \left( \prod_{j \neq i}^{n} m_j \right)^{-1} \pmod{m_i} \right) \pmod{\prod_{i=1}^{n} m_i}$$

## 7.2 扩展中国剩余定理

用于求解同余方程组的模数并不互素的情况。我们考虑如何合并两个同余式：

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

显然其等价于：

$$\begin{cases} x = a_1 + k_1 \times m_1 \\ x = a_2 + k_2 \times m_2 \end{cases}$$

联立可得：

$$k_1 m_1 - k_2 m_2 = a_2 - a_1$$

我们解这个方程即可得出当前的解 $x_0$ 。且注意到我们若给 $x_0$ 加上若干个 $\operatorname{lcm}(m_1, m_2)$ ，上式仍然成立，即当前的解是在 $\bmod \operatorname{lcm}(m_1, m_2)$ 意义下的。这样我们得出新的同余式：

$$x \equiv x_0 \pmod{\operatorname{lcm}(m_1, m_2)}$$

与其它式子继续合并即可。注意在数据范围比较大的时候需要龟速加。

## 7.3 BSGS

在 $\sqrt{p}$ 的时间内求解 $a^x \equiv b \pmod{p}$，要求 $a$ 与 $p$ 互质。

theory/bsgs.cpp

## 7.4 扩展 BSGS

不要求 $a, p$ 互质。

theory/exbsgs.cpp

## 7.5 Lucas 定理

theory/lucas.cpp

## 7.6 扩展 Lucas 定理

theory/exlucas.cpp

## 7.7 杜教筛

实际上是利用迪利克雷卷积来构造递推式，从而对一些积性函数快速求和的方法。

我们现在考虑求取积性函数 $f$ 的前缀和 $F$ 。设存在函数 $g$ ，使得 $f * g$ 的前缀和可以被快速计算，那么：

$$\begin{aligned} \sum_{k=1}^{n} (f * g)(k) &= \sum_{k=1}^{n} \sum_{d|k} f\left(\frac{k}{d}\right) \times g(d) \\ &= \sum_{d=1}^{n} \sum_{k=1}^{\lfloor n/d \rfloor} f(k) \times g(d) \\ &= \sum_{d=1}^{n} g(d) \times F\left(\left\lfloor \frac{n}{d} \right\rfloor\right) \\ &= \sum_{d=2}^{n} g(d) \times F\left(\left\lfloor \frac{n}{d} \right\rfloor\right) + g(1) \times F(n) \end{aligned}$$

则：

$$F(n) = \left( \sum_{k=1}^{n} (f * g)(k) - \sum_{d=2}^{n} g(d) \times F\left(\left\lfloor \frac{n}{d} \right\rfloor\right) \right) \Big/ g(1)$$

若 $f * g$ 的前缀和可以被快速计算，我们就可以使用整除分块，从而把 $F(n)$ 划分为若干个子问题。使用时使用线性筛来预处理 $F$ 的前 $n^{\frac{2}{3}}$ 项，这样杜教筛的时间复杂度为 $O(n^{\frac{2}{3}})$ 。

## 7.8 Min-25 筛

Min-25 筛本质上是对埃氏筛进行了扩展，用于求解积性函数的前缀和，要求其在质数与质数的次幂处的取值可以被快速计算。

下面对 Min-25 筛的运行过程做一个简要的推导：

记 $\mathbb{P}$ 中的数为 $p$ ，$p_k$ 为 $\mathbb{P}$ 中第 $k$ 小的数，$\operatorname{lpf}(n)$（lowest prime factor ）为 $n$ 的最小素因子，$F(n) = \sum_{p \le n} f(p)$ ，$F_k(n) = \sum_{i=2}^{n} [p_k \le \operatorname{lpf}(i)] f(i)$ ，不难发现答案即为 $F_1(n) + 1$ 。

考虑在 $F_k$ 与 $F$ 之间建立递推关系，从素因子的角度出发，并应用积性函数的性质，我们有：

$$\begin{aligned} F_k(n) &= \sum_{i=1}^{n} [\operatorname{lpf}(i) \ge k] f(i) \\ &= \sum_{i \ge k, p_i^2 \le n} \sum_{c \ge 1, p_i^c \le n} f(p_i^c) \times ([c > 1] + F_{k+1}(n/p_i^c)) + F(n) - F(p_{k-1}) \end{aligned}$$

现在的问题在于如何快速求取 $F$ 。首先可以注意到只有 $O(\sqrt{n})$ 处的 $F_k$ 和 $F$ 的取值对我们来说是有用的，这一点保障了我们的计算复杂度。现在我们只关注 $f$ 在质数处的取值，在具体的问题中，这一部分往往可以被表示为一个低次多项式，因此我们可以考虑分别计算每一项的贡献，最后再把它们加起来。也就是说现在我们只需要求 $g(n) = n^s$ 在只考虑素数处的取值的情况下的前缀和。注意到 $g$ 具有非常优美的性质，其是一个完全积性函数，因此我们考虑构造 $G_k(n) = \sum_{i=2}^{n} [\operatorname{lpf}(i) > p_k \vee i \in \mathbb{P}] g(i)$ ，即埃氏筛第 $k$ 轮以后剩下的数处的 $g$ 的取值之和。从埃氏筛的过程入手，我们考虑如何递推求解 $G_k(n)$ ，即：

1. 对于 $p_k^2 > n$ ，显然，所有满足的条件的数都会是素数，因此 $G_k(n) = G_{k-1}(n)$ 。2. 否则我们希望去除掉所有 lpf 为 $p_k$ 的合数处的取值，即减去 $g(p_k) G_{k-1}(n/p_k)$ 。3. 在第 2 步中会多减一部分，这部分均仅含一个小于 $p_k$ 的素因子，因此我们加上 $g(p_k) G_{k-1}(p_{k-1})$ 。

概括一下，我们有：

$$G_k(n) = G_{k-1}(n) - [p_k^2 \le n] g(p_k) (G_{k-1}(n/p_k) - G_{k-1}(p_{k-1}))$$

以下是洛谷 P5325 求积性函数 $f(p^k) = p^k(p^k - 1), p \in \mathbb{P}$ 的前缀和的代码：

theory/min25.cpp

# 8 计算几何

## 8.1 声明与宏

```
#define cp const P
#define cl const L
#define cc const C
#define vp vector<P>
#define cvp const vector<P>
#define D double
#define LD long double
std::mt19937 rnd(time(0));
const LD eps = 1e-12;
const LD pi = std::numbers::pi;
const LD INF = 1e9;
int sgn(LD x) {
    return x > eps ? 1 : (x < -eps ? -1 : 0);
}
```

## 8.2 点与向量

```
struct P {
    LD x, y;

    P(const LD x = 0, const LD y = 0) : x(x), y(y) {}
    P(cp a) : x(a.x), y(a.y) {}

    P operator=(cp a) {
        x = a.x, y = a.y;
        return *this;
    }
    P rot(LD t) const {
        LD c = cos(t), s = sin(t);
        return P(x * c - y * s, x * s + y * c);
    } // counterclockwise
    P rot90() const { return P(-y, x); } // counterclockwise
    P _rot90()const{returnP(y, −x); }//clockwise
    LD len() const { return sqrtl(x * x + y * y); }
    LD len2() const { return x * x + y * y; }
    P unit() const {
        LD d = len();
        return P(x / d, y / d);
    }
    void read() { scanf("%Lf%Lf", x, y); }
    void print() const { printf("(%.9Lf,%.9Lf)", x, y); }
};

bool operator<(cp a, cp b) { return a.x == b.x ? a.y < b.y : a.x
  ↪ < b.x; }
bool operator>(cp a, cp b) { return a.x == b.x ? a.y > b.y : a.x
  ↪ > b.x; }
bool operator==(cp a, cp b) { return !sgn(a.x - b.x)  !sgn(a.y -
  ↪ b.y); }
P operator+(cp a, cp b) { return P(a.x + b.x, a.y + b.y); }
P operator-(cp a, cp b) { return P(a.x - b.x, a.y - b.y); }
P operator*(const LD a, cp b) { return P(a * b.x, a * b.y); }
P operator*(cp b, const LD a) { return P(a * b.x, a * b.y); }
P operator/(cp b, const LD a) { return P(b.x / a, b.y / a); }
LD operator*(cp a, cp b) { return a.x * b.x + a.y * b.y; }
  ↪ // 点积
LD operator^(cpa, cpb){returna.x ∗ b.y − a.y ∗ b.x; }//
LD rad(cp a, cp b) { return acosl((a.x == 0  a.y == 0 || b.x == 0
  ↪ b.y == 0) ? 0 : (a * b / a.len() / b.len())); } // 夹角
bool left(cp a, cp b) { return sgn(a ^b) > 0; }//
```

## 8.3 线

```
struct L {
    P s, t;

    L(cp a, cp b) : s(a), t(b) {}
    L(cl l) : s(l.s), t(l.t) {}
    void read() { s.read(), t.read(); }
};

bool point_on_line(cpa, cll){returnsgn((a − l.s)^(l.t − l.s)) == 0; }
bool point_on_segment(cpa, cll){returnsgn((a − l.s)^(l.t − l.s)) ==
     0sgn((a − l.s) ∗ (a − l.t)) <= 0; }
bool two_side(cpa, cpb, cll){returnsgn((a − l.s)^(a − l.t)) ∗ sgn((b −
     l.s)^(b − l.t)) < 0; }
bool
    intersection_judge_strict(cla, clb){returntwo_side(a.s, a.t, b)two_side(b.s, b.t, a)}
bool
    intersection_judge(cla, clb){returnpoint_on_segment(a.s, b)||point_on_segment(a.t, b)||point_on_segment(b.s, a)||point_on_segment(b.t, a)||intersection_judge
P ll_intersection(cla, clb){
    LD s1 = (a.t - a.s) ^(b.s − a.s), s2 = (a.t − a.s)^(b.t − a.s);
    return (s2 * b.s - s1 * b.t) / (s2 - s1);
}
bool point_on_ray(cpa, clb){returnsgn((a − b.s)^(b.t − b.s)) ==
     0sgn((a − b.s) ∗ (b.t − b.s)) >= 0; }
bool ray_intersection_judge(cla, clb){
    P p(ll_intersection(a, b));
    return sgn((p - a.s) * (a.t - a.s)) >= 0  sgn((p - b.s) *
      ↪ (b.t - b.s)) >= 0;
} // 似乎有更快的做法，但是看不懂
LD point_to_line(cpa, clb){returnfabs((b.t − b.s)^(a − b.s))/(b.t −
     b.s).len(); }//
P project_to_line(cpa, clb){returnb.s + (a − b.s) ∗ (b.t − b.s)/(b.t −
     b.s).len2() ∗ (b.t − b.s); }//
LD point_to_segment(cpa, clb){
    if (b.s == b.t) return (a - b.s).len();
    if (sgn((a - b.s) * (b.t - b.s)) * sgn((a - b.t) * (b.t -
      ↪ b.s)) <= 0)
        return fabs((b.t - b.s) ^(a − b.s))/(b.t − b.s).len();
    return min((a - b.s).len(), (a - b.t).len());
}
```

## 8.4 圆

```
struct C {
    P c;
    LD r;

    C(cp c, const LD r = 0) : c(c), r(r) {}
    C(cc a) : c(a.c), r(a.r) {};
    C(cp a, cp b) {
        c = (a + b) / 2;
        r = (a - c).len();
    }
    C(cp x, cp y, cp z) {
        P p(y - x), q(z - x);
        P s(p * p / 2, q * q / 2);
        LD d = p ^q;
        p = P(s ^P(p.y, q.y), P(p.x, q.x)^s)/d;
        c = x + p, r = p.len();
    }
};

bool in_circle(cpa, ccb){returnsgn((b.c − a).len() − b.r) <= 0; }
C min_circle(vpp)//O(n)
{
    shuffle(p.begin(), p.end(), rnd);
    int n = p.size(), i, j, k;
    C ret(p[0], 0);
    for (i = 1; i < n; ++i)
        if (!in_circle(p[i], ret))
            for (ret = C(p[i], p[0]), j = 1; j < i; ++j)
                if (!in_circle(p[j], ret))
                    for (ret = C(p[i], p[j]), k = 0; k < j; ++k)
                        if (!in_circle(p[k], ret))
                            ret = C(p[i], p[j], p[k]);
    return ret;
}

vp lc_intersection(cll, ccc){
    LD x = point_to_line(c.c, l);
    if (sgn(x - c.r) > 0) return vp();
    x = sqrtl(c.r * c.r - x * x);
    P p(project_to_line(c.c, l));
    if (sgn(x) == 0) return vp({p});
    return vp({p + x * (l.t - l.s).unit(), p - x * (l.t -
      ↪ l.s).unit()});
}

LD cc_intersection_area(cca, ccb){
    LD d = (a.c - b.c).len();
    if (sgn(d - (a.r + b.r)) >= 0) return 0;
    if (sgn(d - fabs(a.r - b.r)) <= 0) {
        LD r = min(a.r, b.r);
        return pi * r * r;
    }
    LD x = (d * d + a.r * a.r - b.r * b.r) / (2 * d), t1 =
      ↪ acosl(min((LD)1, max((LD)-1, x / a.r))), t2 =
      ↪ acosl(min((LD)1, max((LD)-1, (d - x) / b.r)));
    return a.r * a.r * t1 + b.r * b.r * t2 - d * a.r * sinl(t1);
}

vp cc_intersection(cca, ccb){
    LD d = (a.c - b.c).len();
    if (a.c == b.c || sgn(d - a.r - b.r) > 0 || sgn(d - fabs(a.r
      ↪ - b.r)) < 0) return vp();
    P r = (b.c - a.c).unit();
    LD x = (d * d + a.r * a.r - b.r * b.r) / (d * 2);
```

```
    LD h = sqrtl(a.r * a.r - x * x);
    if (sgn(h) == 0) return vp({a.c + r * x});
    return vp({a.c + r * x + r.rot90() * h, a.c + r * x -
      ↪ r.rot90() * h});
}

vp tangent(cp a, cc b) { return cc_intersection(b, C(a, b.c)); }//

vector<L> tangent(cc a, cc b, int f) // f = 1 extangent;f = -1
  ↪ intangent
{
    D d = (b.c - a.c).len();
    int sg = sgn(fabs((b.r - f * a.r) / d) - 1);
    if (sg == 1)
        return {};
    else if (sg == 0) {
        P p = a.c - sgn(b.r - f * a.r) * f * a.r * (b.c -
          ↪ a.c).unit();
        return {{p, p + (b.c - a.c).rot90()}};
    } // 内切/外切
    D theta = asin(min((D)1, max((D)-1, f * (b.r - f * a.r) /
      ↪ d)));
    P du = (b.c - a.c).unit(), du1 = du.rot(theta + pi / 2), du2
      ↪ = du.rot(-theta - pi / 2);
    return {{a.c + a.r * du1, b.c + f * b.r * du1}, {a.c + a.r *
      ↪ du2, b.c + f * b.r * du2}};
}
```

## 8.5 凸包

```
vp convex(vp a) {
    if (a.size() < 2) return a;
    sort(a.begin(), a.end());
    int n = a.size(), cnt = 0;
    vp con({p[0]});
    for (int i = 0; i < n; ++i) {
        while (cnt > 0  sgn(((p[i] - con[cnt - 1])
            ^(con[cnt] - con[cnt - 1]))) > 0)// >=
            --cnt, con.pop_back();
        ++cnt, con.push_back(p[i]);
    }
    int fixed = cnt;
    for (int i = n - 2; ~i; --i) {
        while (cnt > fixed  sgn(((p[i] - con[cnt - 1])
            ^(con[cnt] - con[cnt - 1]))) > 0)// >=
            --cnt, con.pop_back();
        ++cnt, con.push_back(p[i]);
    }
    con.pop_back();
    return con;
}

vp minkowski(vp p1, vp p2) {
    if (p1.empty() || p2.empty()) return vp();
    int n1 = p1.size(), n2 = p2.size();
    vp ret;
    if (n1 == 1) {
        for (int i = 0; i < n2; ++i)
            ret.push_back(p1[0] + p2[i]);
        return ret;
    }
    if (n2 == 1) {
        for (int i = 0; i < n1; ++i)
            ret.push_back(p1[i] + p2[0]);
        return ret;
    }
        p1.push_back(p1[0]), p1.push_back(p1[1]), p2.push_back(p2[0]), p2.push_back(p2[1]);
    ret.push_back(p1[0] + p2[0]);
    int i1 = 0, i2 = 0;
    while (i1 < n1 || i2 < n2) {
        if (((p1[i1 + 1] - p1[i1]) ^(p2[i2 + 1] - p2[i2]))) > 0)
            ret.push_back(p1[++i1] + p2[i2]);
        else
            ret.push_back(p2[++i2] + p1[i1]);
    } // p1.pop_back(), p1.pop_back(), p2.pop_back(), p2.pop_back();
    ret.pop_back();
    return ret;
}

struct Con {
    int n;
    vp a, upper, lower;

    Con(cvp _a) : a(_a){
        n = a.size();
        int k = 0;
        for (int i = 1; i < n; ++i)
            if (a[i] > a[k]) k = i;
        for (int i = 0; i <= k; ++i)
            lower.push_back(a[i]);
        for (int i = k; i < n; ++i)
            upper.push_back(a[i]);
        upper.push_back(a[0]);
    }

}

// Below is from Nemesis

// Convex
int n;
vector<point> a; // 可以封装成一个 struct
bool inside(cp u) { // 点在凸包内
    int l = 1, r = n - 2;
    while (l < r) {
        int mid = (l + r + 1) / 2;
        if (turn(a[0], a[mid], u) >= 0)
            l = mid;
        else
            r = mid - 1;
    }
    return turn(a[0], a[l], u) >= 0  turn(a[l], a[l + 1], u) >= 0
      ↪ turn(a[l + 1], a[0], u) >= 0;
}
int search(auto f) { // 凸包求极值, 需要 C++17
    int l = 0, r = n - 1;
    int d = f(a[r], a[l]) ? (swap(l, r), -1) : 1;
    while (d * (r - l) > 1) {
        int mid = (l + r) / 2;
        if (f(a[mid], a[l])  f(a[mid], a[mid - d]))
            l = mid;
        else
            r = mid;
    }
    return l;
}
pair<int, int> get_tan(cp u){//
    return // 严格在凸包外; 需要边界上时, 特判 a[n-1] -> a[0]
        {search([](cp x, cp y) { return turn(u, y, x) > 0; }),
         search([](cp x, cp y) { return turn(u, x, y) > 0; })};
}
point at(int i) { return a[i % n]; }
int inter(cp u, cp v, int l, int r) {
    int sl = turn(u, v, at(l));
    while (l + 1 < r) {
        int m = (l + r) / 2;
        if (sl == turn(u, v, at(m)))
            l = m;
        else
            r = m;
    }
    return l % n;
}
bool get_inter(cp u, cp v, int i, int j){//
    int p0 = search([](cp x, cp y) { return det(v - u, x - u) <
      ↪ det(v - u, y - u); }),
        p1 = search([](cp x, cp y) { return det(v - u, x - u) >
          ↪ det(v - u, y - u); });
    if (turn(u, v, a[p0]) * turn(u, v, a[p1]) < 0) {
        if (p0 > p1) swap(p0, p1);
        i = inter(u, v, p0, p1);
        j = inter(u, v, p1, p0 + n);
        return true;
    } else
        return false;
}
LD near(cp u, int l, int r) {
    if (l > r) r += n;
    int sl = sgn(dot(u - at(l), at(l + 1) - at(l)));
    LD ret = p2s(u, {at(l), at(l + 1)});
    while (l + 1 < r) {
        int m = (l + r) / 2;
        if (sl == sgn(dot(u - at(m), at(m + 1) - at(m))))
            l = m;
        else
            r = m;
    }
    return min(ret, p2s(u, {at(l), at(l + 1)}));
}
LD get_near(cp u){//
    if (inside(u)) return 0;
    auto [x, y] = get_tan(u);
    return min(near(u, x, y), near(u, y, x));
}

// Dynamic convex hull
struct hull { // upper hull, left to right
    set<point> a;
    LL tot;
```

```
    hull() { tot = 0; }
    LL calc(auto it) {
        auto u = it == a.begin() ? a.end() : prev(it);
        auto v = next(it);
        LL ret = 0;
        if (u != a.end()) ret += det(*u, *it);
        if (v != a.end()) ret += det(*it, *v);
        if (u != a.end()  v != a.end()) ret -= det(*u, *v);
        return ret;
    }
    void insert(point p) {
        if (!a.size()) {
            a.insert(p);
            return;
        }
        auto it = a.lower_bound(p);
        bool out;
        if (it == a.begin())
            out = (p < *it); // special case
        else if (it == a.end())
            out = true;
        else
            out = turn(*prev(it), *it, p) > 0;
        if (!out) return;
        while (it != a.begin()) {
            auto o = prev(it);
            if (o == a.begin() || turn(*prev(o), *o, p) < 0)
                break;
            else
                erase(o);
        }
        while (it != a.end()) {
            auto o = next(it);
            if (o == a.end() || turn(p, *it, *o) < 0)
                break;
            else
                erase(it), it = o;
        }
        tot += calc(a.insert(p).first);
    }
    void erase(auto it) {
        tot -= calc(it);
        a.erase(it);
    }
};
```

## 8.6 三角形

```
// From Nemesis
point incenter(cp a, cp b, cp c) { // 内心
    double p = dis(a, b) + dis(b, c) + dis(c, a);
    return (a * dis(b, c) + b * dis(c, a) + c * dis(a, b)) / p;
}
point circumcenter(cp a, cp b, cp c) { // 外心
    point p = b - a, q = c - a, s(dot(p, p) / 2, dot(q, q) / 2);
    double d = det(p, q);
    return a + point(det(s, {p.y, q.y}), det({p.x, q.x}, s)) / d;
}
point orthocenter(cp a, cp b, cp c) { // 重心
    return a + b + c - circumcenter(a, b, c) * 2.0;
}
point fermat_point(cp a, cp b, cp c){//
    if (a == b) return a;
    if (b == c) return b;
    if (c == a) return c;
    double ab = dis(a, b), bc = dis(b, c), ca = dis(c, a);
    double cosa = dot(b - a, c - a) / ab / ca;
    double cosb = dot(a - b, c - b) / ab / bc;
    double cosc = dot(b - c, a - c) / ca / bc;
    double sq3 = PI / 3.0;
    point mid;
    if (sgn(cosa + 0.5) < 0)
        mid = a;
    else if (sgn(cosb + 0.5) < 0)
        mid = b;
    else if (sgn(cosc + 0.5) < 0)
        mid = c;
    else if (sgn(det(b - a, c - a)) < 0)
        mid = line_inter({a, b + (c - b).rot(sq3)}, {b, c + (a - c).rot(sq3)});
    else
        mid = line_inter({a, c + (b - c).rot(sq3)}, {c, b + (a - b).rot(sq3)});
    return mid;
} // minimize(|A-x|+|B-x|+|C-x|)
```

## 8.7 多边形

```
vp Poly_cut(vp p, cl l)//              O(n)
{
    if (p.empty()) return vp();
    vp ret;
    int n = p.size();
    p = push_back(p[0]);
    for (int i = 0; i < n; ++i) {
        if (((p[i] - l.s) ^ (l.t - l.s)) <= 0)
            ret.push_back(p[i]);
        if (two_size(p[i], p[i + 1], l))
            ret.push_back(ll_intersection(l, L(p[i], p[i + 1])));
    } // p.pop_back();
    return ret;
}

LD Poly_area(vp p){
    LD ar = 0;
    int n = p.size();
    for (int i = 0, j = n - 1; i < n; j = i++)
        ar += p[i] ^ p[j];
    return fabs(ar) / 2;
}

// Below is from Nemesis

// 多边形与圆交
LD angle(cp u, cp v) {
    return 2 * asin(dis(u.unit(), v.unit()) / 2);
}
LD area(cp s, cp t, LD r) { // 2 * area
    LD theta = angle(s, t);
    LD dis = p2s({0, 0}, {s, t});
    if (sgn(dis - r) >= 0) return theta * r * r;
    auto [u, v] = line_circle_inter({s, t}, {{0, 0}, r});
    point lo = sgn(det(s, u)) >= 0 ? u : s;
    point hi = sgn(det(v, t)) >= 0 ? v : t;
    return det(lo, hi) + (theta - angle(lo, hi)) * r * r;
}
LD solve(vector<point> p, cc c) {
    LD ret = 0;
    for (int i = 0; i < (int)p.size(); ++i) {
        auto u = p[i] - c.c;
        auto v = p[(i + 1) % p.size()] - c.c;
        int s = sgn(det(u, v));
        if (s > 0)
            ret += area(u, v, c.r);
        else if (s < 0)
            ret -= area(v, u, c.r);
    }
    return abs(ret) / 2;
} // ret 在 p 逆时针时为正
```

## 8.8 半平面交

```
// From Nemesis
int half(cp a) { return a.y > 0 || (a.y == 0  a.x > 0) ? 1 : 0; }
bool turn_left(cl a, cl b, cl c){
    return turn(a.s, a.t, line_inter(b, c)) > 0;
}
bool is_para(cl a, cl b){return !sgn(det(a.t - a.s, b.t - b.s));}
bool cmp(cl a, cl b) {
    int sign = half(a.t - a.s) - half(b.t - b.s);
    int dir = sgn(det(a.t - a.s, b.t - b.s));
    if (!dir  !sign)
        return turn(a.s, a.t, b.t) < 0;
    else
        return sign ? sign > 0 : dir > 0;
}
vector<point> hpi(vector<line> h) { // 半平面交
    sort(h.begin(), h.end(), cmp);
    vector<line> q(h.size());
    int l = 0, r = -1;
    for (auto i : h) {
        while (l < r  !turn_left(i, q[r - 1], q[r]))
            --r;
        while (l < r  !turn_left(i, q[l], q[l + 1]))
            ++l;
        if (l <= r  is_para(i, q[r]))continue;
        q[++r] = i;
    }
    while (r - l > 1  !turn_left(q[l], q[r - 1], q[r]))
        --r;
    while (r - l > 1  !turn_left(q[r], q[l], q[l + 1]))
        ++l;
    if (r - l < 2) return {};
    vector<point> ret(r - l + 1);
    for (int i = l; i <= r; i++)
        ret[i - l] = line_inter(q[i], q[i == r?l : i + 1]);
    return ret;
```

```
}
// 空集会在队列里留下一个开区域；开区域会被判定为空集。
// 为了保证正确性，一定要加足够大的框，尽可能避免零面积区域。
// 实在需要零面积区域边缘，需要仔细考虑 turn_left
```

# 9 杂项

## 9.1 生成树计数

【根据度数求方案】对于给定每个点度数为 $d_i$ 的无根树，方案数为：

$$\frac{(n-2)!}{\prod_{i=1}^{n}(d_i-1)!}$$

【根据连通块数量与大小求方案】一个 $n$ 个点 $m$ 条边的带标号无向图有 $k$ 个连通块，每个连通块大小为 $s_i$，需要增加 $k-1$ 条边使得整个图联通，方案数为：（但是当 $k=1$ 时需要特判）

$$n^{k-2}\cdot\prod_{i=1}^{k}s_i$$

证明只需考虑 prufer 序列即可。

## 9.2 类欧几里得

$ax+by=n$ 的几何意义可以想象为一条直线，那么 $[0,n]$ 中可以被表示出来的整数就是 $(0,0),\left(\frac{n}{a},0\right),\left(0,\frac{n}{b}\right)$ 为顶点的三角形在第一象限内含有的整点个数。显然的结论就是，在 $[0,n]$ 可以表示出的整数数量为：

$$\sum_{x=0}^{\lfloor\frac{n}{a}\rfloor}\left\lfloor\frac{n-ax}{b}\right\rfloor$$

类欧几里得可以在 $\mathcal{O}(\log\max(\dashv,\lfloor))$ 的时间内解决此类问题。

求 $\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$：

```
ll solve(ll a, ll b, ll c, ll n) {
    if (a == 0) return (n + 1) * (b / c) % mod;
    if (a >= c || b >= c)
        return (n * (n + 1) % mod * inv2 % mod * (a / c) % mod + (n + 1) * (b / c) % mod + solve(a % c, b % c, c, n)) % mod;
    ll m = (a * n + b) / c;
    return (n * (m % mod) % mod - solve(c, c - b - 1, a, m - 1) + mod) % mod;
}
```

求 $\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor^2$ 和 $\sum_{i=0}^{n}i\lfloor\frac{ai+b}{c}\rfloor$，分别对应以下的 g 和 h：

```
struct Euclid {
    ll f, g, h;
};
Euclid solve(ll a, ll b, ll c, ll n) {
    Euclid ans, tmp;
    if (a == 0) {
        ans.f = (n + 1) * (b / c) % mod;
        ans.g = n * (n + 1) % mod * (b / c) % mod * inv2 % mod;
        ans.h = (n + 1) * (b / c) % mod * (b / c) % mod;
        return ans;
    }
    if (a >= c || b >= c) {
        tmp = solve(a % c, b % c, c, n);
        ans.f = (n * (n + 1) % mod * inv2 % mod * (a / c) % mod + (n + 1) * (b / c) % mod + tmp.f) % mod;
        ans.g = (n * (n + 1) % mod * (2 * n + 1) % mod * (a / c) % mod * inv6 % mod + n * (n + 1) % mod * (b / c) % mod * inv2 % mod + tmp.g) % mod;
        ans.h = ((a / c) * (a / c) % mod * n % mod * (n + 1) % mod * (n * 2 + 1) % mod * inv6 % mod +
                (b / c) * (b / c) % mod * (n + 1) % mod + (a / c) * (b / c) % mod * n % mod * (n + 1) % mod +
                2 * (b / c) % mod * tmp.f % mod + 2 * (a / c) % mod * tmp.g % mod + tmp.h) %
                mod;
        return ans;
    }
    ll m = (a * n + b) / c;
    tmp = solve(c, c - b - 1, a, m - 1);
    ans.f = (n * (m % mod) % mod + mod - tmp.f) % mod;
    ans.g = (n * (m % mod) % mod * (n + 1) % mod - tmp.h - tmp.f) % mod * inv2 % mod;
    ans.h = (n * (m % mod) % mod * (m + 1) % mod - 2 * tmp.g - 2 * tmp.f - ans.f) % mod;
    return ans;
}

ans.f = (ans.f % mod + mod) % mod;
ans.g = (ans.g % mod + mod) % mod;
ans.h = (ans.h % mod + mod) % mod;
```

# 10 其它工具

## 10.1 编译命令

```
g++ X.cpp -Wall -O2 -fsanitize=undefined -fsanitize=address X
# -fsanitize=undefined: 检测未定义行为
# -fsanitize=address: 检测内存溢出
```

## 10.2 快读

```
char *p1, *p2, buf[100000];
#define gc() (p1 == p2  (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++)
int read() {
    int x = 0, f = 1;
    char ch = gc();
    while (!isdigit(ch)) { if (ch == '-') f = -1; ch = gc(); }
    while (isdigit(ch)) x = x * 10 + ch - '0', ch = gc();
    return x * f;
}
```

## 10.3 Python Hints

```
from itertools import * # itertools 库
# 笛卡尔积
product('ABCD', 'xy') # Ax Ay Bx By Cx Cy Dx Dy
product(range(2), repeat=3) # 000 001 010 011 100 101 110 111
# 排列
permutations('ABCD', 2) # AB AC AD BA BC BD CA CB CD DA DB DC
# 组合
combinations('ABCD', 2) # AB AC AD BC BD CD
# 有重复的组合
combinations_with_replacement('ABC',2)#AAABACBBBCCC


from random import * # random 库
randint(l, r) # 在 [l, r] 内的随机整数
choice([1, 2, 3, 5, 8]) # 随机选择序列中一个元素
sample([1, 2, 3, 4, 5], k=2) # 随机抽样两个元素
shuffle(x) # 原地打乱序列 x
l,r = sorted(choices(range(1, N+1), k=2)) # 生成随机区间 [l,r]
binomialvariate(n, p) # 返回服从 B(n,p) 的一个变量
normalvariate(mu, sigma) # 返回服从 N(mu,sigma) 的一个变量

# 列表操作
l = sample(range(100000), 10)
l.sort() # 原地排序
l.sort(key=lambda x:x%10) # 按末尾排序
from functools import cmp_to_key
l.sort(key=cmp_to_key(lambdax,y:y-x))#
sorted(l) # 非原地排序
l.reverse() reversed(l)

# 复数
a = 1+2j
print(a.real, a.imag, abs(a), a.conjugate())

# 高精度小数
from decimal import Decimal, getcontext, FloatOperation,
    ROUND_HALF_EVEN
getcontext().prec = 100 # 设置有效位数
getcontext().rounding = getattr(ROUND_HALF_EVEN)#
getcontext().traps[FloatOperation] = True # 禁止 float 混合运算
a = Decimal("114514.1919810")
print(a, f"{a:.2f}")
a.ln() a.log10() a.sqrt() a**2
```

## 10.4 对拍器

```python
import os

while True:
    os.system("python3 data.py > in")
    os.system("/usr/bin/time -f 'test Time=%es' ./test < in >
     ↪ out")
    os.system("/usr/bin/time -f 'std Time=%es' ./std < in > ans")
    if os.system("diff out ans >/dev/null"):
        print("WA")
        exit(1)
    print("AC")
```

| $n \le$ | $10^1$ | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| $\max\{\omega(n)\}$ | 2 | 3 | 4 | 5 | 26 | 7 |
| $\max\{d(n)\}$ | 4 | 12 | 32 | 64 | 128 | 240 |
| $\pi(n)$ | 4 | 25 | 168 | 1229 | 9592 | 78498 |
| $n \le$ | $10^7$ | $10^8$ | $10^9$ | $10^{10}$ | $10^{11}$ | $10^{12}$ |
| $\max\{\omega(n)\}$ | 8 | 8 | 9 | 10 | 10 | 11 |
| $\max\{d(n)\}$ | 448 | 768 | 1344 | 2304 | 4032 | 6720 |
| $\pi(n)$ | 664579 | 5761455 | 5.08e7 | 4.55e8 | 4.12e9 | 3.7e10 |
| $n \le$ | $10^{13}$ | $10^{14}$ | $10^{15}$ | $10^{16}$ | $10^{17}$ | $10^{18}$ |
| $\max\{\omega(n)\}$ | 12 | 12 | 13 | 13 | 14 | 15 |
| $\max\{d(n)\}$ | 10752 | 17280 | 26880 | 41472 | 64512 | 103680 |
| $\pi(n)$ | $\pi(x) \sim x/\ln(x)$ | | | | | |

## 10.5 常数表

| $n$ | $\log_{10} n$ | $n!$ | $C(n, n/2)$ | $\mathrm{lcm}(1 \ldots n)$ |
|---|---|---|---|---|
| 2 | 0.301030 | 2 | 2 | 2 |
| 3 | 0.477121 | 6 | 3 | 6 |
| 4 | 0.602060 | 24 | 6 | 12 |
| 5 | 0.698970 | 120 | 10 | 60 |
| 6 | 0.778151 | 720 | 20 | 60 |
| 7 | 0.845098 | 5040 | 35 | 420 |
| 8 | 0.903090 | 40320 | 70 | 840 |
| 9 | 0.954243 | 362880 | 126 | 2520 |
| 10 | 1.000000 | 3628800 | 252 | 2520 |
| 11 | 1.041393 | 39916800 | 462 | 27720 |
| 12 | 1.079181 | 479001600 | 924 | 27720 |
| 15 | 1.176091 | 1.31e12 | 6435 | 360360 |
| 20 | 1.301030 | 2.43e18 | 184756 | 232792560 |
| 25 | 1.397940 | 1.55e25 | 5200300 | 2.68e10 |
| 30 | 1.477121 | 2.65e32 | 155117520 | 2.33e12 |

## 10.6 试机赛

- 测试编译器版本。
  - `#include<bits/stdc++.h>`
  - `pb_ds`
  - C++20: `cin>>(s+1);`
  - C++17: `auto [x,y]=pair1,"abc";`
  - C++11: `auto x=1;`
- 测试 __int128, __float128, long double
- 测试 pragma 是否 CE 。
- 测试 -fsanitize=address,undefined
- 测试本地性能

## 10.7 阴间错误集锦

- 多测不清空。
- 该开 `long long` 的地方不开 `long long` 。一般情况下建议直接 `#define int long long` 。
- 注意变量名打错。例如 u 打成 v 或 a 打成 t 。建议读代码的时候专门检查此类错误。