

ACM 模板

黄佳瑞，钱智煊，林乐道

2025 年 12 月 13 日

目录

| | | | |
|--|-----------|---------------------------|-----------|
| 1 做题指导 | 3 | 5 线性代数 | 19 |
| 1.1 上机前你应该注意什么 | 3 | 5.1 高斯消元 | 19 |
| 1.2 机上你应该注意什么 | 3 | 5.2 线性基 | 19 |
| 1.3 交题前你应该注意什么 | 3 | 5.3 行列式 | 19 |
| 1.4 如果你的代码挂了 | 3 | 5.4 矩阵树定理 | 19 |
| 1.5 另外一些注意事项 | 3 | 5.5 单纯形法 | 20 |
| 1.6 阴间错误集锦 | 3 | 5.6 全幺模矩阵 | 20 |
| | | 5.7 对偶原理 | 20 |
| 2 图论 | 4 | 6 多项式 | 21 |
| 2.1 Tarjan 边双、点双 (圆方树) | 4 | 6.1 $O(nV)$ 多重背包 (单调队列优化) | 21 |
| 2.2 离线 $O(1)$ LCA (Tarjan) | 4 | 6.2 二项式反演 | 21 |
| 2.3 在线 $O(1)$ LCA (DFS 序) | 4 | 6.3 FFT | 21 |
| 2.4 某些路径问题单 \log 做法 | 5 | 6.4 NTT | 21 |
| 2.5 重链剖分 | 5 | 6.5 拉格朗日插值 | 21 |
| 2.6 笛卡尔树 | 5 | 6.6 集合幂级数 | 21 |
| 2.7 2-SAT | 5 | 6.6.1 并卷积、交卷积与子集卷积 | 21 |
| 2.8 Dinic 网络流、费用流 | 5 | 6.6.2 对称差卷积 | 22 |
| 2.8.1 无源汇有上下界可行流 | 6 | 6.7 多项式全家桶 | 22 |
| 2.8.2 有源汇有上下界最大流 | 7 | | |
| 2.8.3 网络流总结 | 8 | | |
| 2.9 差分约束 | 9 | 7 数论 | 24 |
| 2.10 欧拉路径 | 9 | 7.1 exgcd | 24 |
| 2.10.1 有向图 | 9 | 7.2 类欧几里得 | 24 |
| 2.10.2 无向图 | 9 | 7.3 中国剩余定理 | 24 |
| 2.11 同余最短路 | 9 | 7.4 扩展中国剩余定理 | 24 |
| 2.12 树的计数 | 10 | 7.5 BSGS | 24 |
| 2.12.1 树的计数 Prufer 序列 | 10 | 7.6 扩展 BSGS | 25 |
| 2.12.2 有根树计数 1,1,2,4,9,20,48,115,286,719,1842,4766 | 10 | 7.7 Lucas 定理 | 25 |
| 2.12.3 无根树计数 | 10 | 7.8 扩展 Lucas 定理 | 25 |
| 2.12.4 生成树计数 Kirchhoff's Matrix-Tree Theorem | 10 | 7.9 杜教筛 | 26 |
| 2.12.5 有向图欧拉回路计数 BEST Theorem | 10 | 7.10 Min-25 筛 | 26 |
| | | 7.11 没有精度问题的整除 | 27 |
| 3 数据结构 | 11 | 8 计算几何 | 28 |
| 3.1 平衡树 | 11 | 8.1 声明与宏 | 28 |
| 3.1.1 FHQ Treap | 11 | 8.2 点与向量 | 28 |
| 3.1.2 平衡树合并 | 12 | 8.3 线 | 28 |
| 3.1.3 Splay | 12 | 8.4 圆 | 28 |
| 3.2 LCT 动态树 | 12 | 8.5 凸包 | 29 |
| 3.3 ODT 珂朵莉树 | 13 | 8.6 三角形 | 30 |
| 3.4 李超线段树 | 13 | 8.7 多边形 | 31 |
| 3.5 二维树状数组 | 13 | 8.8 半平面交 | 31 |
| 3.6 虚树 | 14 | | |
| 3.7 左偏树 | 14 | 9 其它工具 | 32 |
| 3.8 吉司机线段树 | 14 | 9.1 编译命令 | 32 |
| 3.9 树分治 (点分治) | 15 | 9.2 快读 | 32 |
| | | 9.3 Python Hints | 32 |
| 4 字符串 | 17 | 9.4 对拍器 | 32 |
| 4.1 Hash 类 | 17 | 9.5 常数表 | 33 |
| 4.2 后缀数组 | 17 | 9.6 试机赛 | 33 |
| 4.3 AC 自动机 | 17 | 9.7 Bashrc | 33 |
| 4.4 回文自动机 | 17 | | |
| 4.5 Manacher 算法 | 17 | | |
| 4.6 KMP 算法与 border 理论 | 17 | | |
| 4.7 Z 函数 | 18 | | |
| 4.8 后缀自动机 | 18 | | |
| 4.9 后缀自动机 (map 版) | 18 | | |
| 4.10 最小表示法 | 18 | | |

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：

```
1 g++ X.cpp -Wall -Wshadow -Wextra -Wconversion -O2  
  ↳ -fsanitize=undefined -fsanitize=address X  
2 # -fsanitize=undefined: 检测未定义行为  
3 # -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。

1.6 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。
- 树剖，倍增 LCA 之类的 `while` 打成 `if`。

2 图论

2.1 Tarjan 边双、点双 (圆方树)

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)!!!

(如果选择在加边的时候特判, 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 $\text{tot} += 2$.)

圆方树示意图:

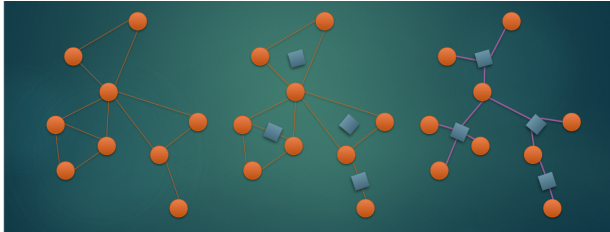


图 1: 圆方树示意图

```
1  /*** 缩点 ***/
2  // 找出强联通分量后, 对每一条边查看是否在同一个 scc 中, 如
   ↳ 果不在就加边
3  void tarjan(int x)
4  {
5      dfn[x]=low[x]=++Time, sta[++tp]=x, ins[x]=true;
6      for(int i=hea[x]; i; i=nex[i])
7      {
8          if(!dfn[ver[i]])
9              ↳ tarjan(ver[i]), low[x]=min(low[x], low[ver[i]]);
10         else if(ins[ver[i]])
11             ↳ low[x]=min(low[x], dfn[ver[i]]);
12     }
13     if(dfn[x]==low[x])
14     {
15         scc++;
16         do { x=sta[tp], tp--, ins[x]=false, bel[x]=scc; }
17         while (dfn[x]!=low[x]);
18     }
19 }
20 /*** 割点 ***/
21 void tarjan(int x, int Last)
22 // Last 是边的编号, tot 初始值为 1, i 与 i^1 互为反边
23 {
24     dfn[x]=low[x]=++Time;
25     for(int i=hea[x]; i; i=nex[i])
26     {
27         if(!dfn[ver[i]])
28             ↳ tarjan(ver[i], i), low[x]=min(low[x], low[ver[i]]);
29         if(low[ver[i]]>dfn[x]) edg[i]=edg[i^1]=true;
30     }
31     else if(dfn[ver[i]]<dfn[x] && i!=(Last^1))
32         ↳ low[x]=min(low[x], dfn[ver[i]]);
33 }
34 /*** 圆方树、点双 ***/
35 // G 表示原图, T 是新建的圆方树
36 // 一条边也是点双
37 // 圆方树只存在圆 - 方边!!!
38 int Time, dfn[N], low[N], sta[N], siz[N];
39 vector<int> G[N], T[N<<1]; // 不要忘记给数组开两倍
40 void tarjan(int u)
41 {
42     dfn[u]=low[u]=++Time, sta[++tp]=u;
43     for(int v:G[u])
44     {
45         if(!dfn[v])
```

```
46         tarjan(v), low[u]=min(low[u], low[v]);
47         // 这里对 low[v]>=dfn[u] 进行计数, 根节点需要
           ↳ 2 个, 其他节点需要 1 个, 那么就是割点。
48         if(low[v]==dfn[u])
49         {
50             int hav=0; ++All;
51             for(int x=0; x!=v; tp--) x=sta[tp],
52                 ↳ T[x].pb(All), T[All].pb(x), hav++;
53             T[u].pb(All), T[All].pb(u);
54             siz[All]++; hav;
55         }
56         else low[u]=min(low[u], dfn[v]);
57     }
58 }
```

2.2 离线 O(1) LCA (Tarjan)

按照顺序遍历子树, 并查集更新。

```
1  int qhea[Maxq], qver[Maxq], qnex[Maxq], qid[Maxq];
2  inline void addq(int x, int y, int d)
3  { qver[++qtot]=y, qnex[qtot]=qhea[x], qhea[x]=qtot,
   ↳ qid[qtot]=d; }
4  void dfs(int x, int Last_node)
5  {
6      vis[x]=true;
7      for(int i=hea[x]; i; i=nex[i])
8          if(ver[i]!=Last_node)
9              ↳ dfs(ver[i], x), fa[ver[i]]=x;
10     for(int i=qhea[x]; i; i=qnex[i])
11         if(vis[qver[i]])
12             ↳ ans[qid[i]]=Find(qver[i]);
13 }
14
15 for(int i=1; i<=n; i++) fa[i]=i;
16 for(int i=1, x, y; i<=m; i++)
17     ↳ x=rd(), y=rd(), add(x, y), add(y, x);
18 for(int i=1, x, y; i<=m; i++)
19     ↳ x=rd(), y=rd(), addq(x, y, i), addq(y, x, i);
20 dfs(rt, rt);
21 for(int i=1; i<=m; i++) printf("%d\n", ans[i]);
```

2.3 在线 O(1) LCA (DFS 序)

对于求出 u, v 的 LCA 来说 (不妨设 $dfn_u \leq dfn_v$), $u = v$ 显然, 否则在 DFS 序中 $[dfn_u + 1, dfn_v]$ 区间最浅的节点的父亲就是 LCA。

```
1  int dep[N], dfn[N], tot, st[N][20];
2  vector<int> G[N];
3
4  void dfs(int u, int fa){
5      st[++tot][0]=fa; // 直接记录父亲
6      dfn[u]=tot;
7      dep[u]=dep[fa]+1;
8      for(int v:G[u]){
9          if(v==fa) continue;
10         ↳ dfs(v, u);
11     }
12 }
13
14 int n;
15 int get(int u, int v){ // 返回深度较浅的节点
16     return dep[u]<dep[v]?u:v;
17 }
18 void init(){ // 初始化 ST 表
19     for(int j=1; j<20; j++){
20         for(int i=1; (i+(1<<j))-1<=n; i++){
21             st[i][j] =
22                 ↳ get(st[i][j-1], st[i+(1<<j)-1][j-1]);
23         }
24     }
```

```

23     }
24 }
25 int lca(int u,int v){
26     if(u==v)return u;
27     // 记得给左端点 +1
28     int l=min(dfn[u],dfn[v])+1,r=max(dfn[u],dfn[v]);
29     int o = __lg(r-l+1);
30     return get(st[l][o],st[r-(1<<o)+1][o]);
31 }

```

2.4 某些路径问题单 log 做法

路径加/单点求和 & 子树求和: 设 $a[u]$ 为子树内 $d[u]$ 之和, 则

1. 路径加: $d[u] + w, d[v] + w, d[lca(u, v)] - wd[fa[lca(u, v)]] - w$
2. 单点求和: $a[u] = sumd(u)$
3. 子树求和: $suma = \sum_v d[v] * (dep[v] - dep[u] + 1)$, 树状数组维护 2 系数即可

单点加 & 子树加/路径求和: 设 $g[u]$ 表示 1 到 u 权值之和, 则

1. 单点加: 子树 d 加
2. 子树加: u 每个子树内 v 有 $g[v] += w * (dep[v] - dep[u] + 1)$ 则维护系数加即可
3. 路径和: $sum(u, v) = g[u] + g[v] - g[lca(u, v)] - g[fa[lca(u, v)]]$

2.5 重链剖分

```

1 vector<int> G[N];
2 int dep[N],fa[N],hson[N],sz[N],top[N],dfn[N],d;
3 void dfs1(int u,int f){
4     dep[u]=dep[f]+1;
5     fa[u]=f;sz[u]=1;
6     for(int v:G[u]){
7         if(v==f)continue;
8         dfs1(v,u);sz[u]+=sz[v];
9         if(sz[v]>sz[hson[u]])hson[u]=v;
10    }
11 }
12 void dfs2(int u,int t){
13     dfn[u]++;top[u]=t;
14     if(hson[u])dfs2(hson[u],t);
15     for(int v:G[u]){
16         if(v==fa[u]||v==hson[u])continue;
17         dfs2(v,v);
18    }
19 }
20 void operate_path(int u,int v){ // 维护路径
21     while(top[u]!=top[v]){
22         if(dep[top[u]] < dep[top[v]]) swap(u,v);
23         OPERATE(dfn[u],dfn[u]); // 对这一区间执行任
24         // 意操作
25         u = fa[top[u]];
26     }
27     if(dep[u]>dep[v])swap(u,v);
28     OPERATE(dfn[u],dfn[v]); // 跳到同一条重链上了
29 }
30 void operate_tree(int u){ // 维护子树
31     OPERATE(dfn[u],dfn[u]+sz[u]-1);
32 }
33 int lca(int u,int v){ // 求最近公共祖先
34     while(top[u]!=top[v]){
35         if(dep[top[u]] < dep[top[v]]) swap(u,v);
36         u = fa[top[u]];
37     }
38     if(dep[u]>dep[v])swap(u,v);
39     return u;
40 }

```

2.6 笛卡尔树

```

1 for(int i=1;i<=n;i++){
2     int k = tp;
3     // 大根堆
4     while(k && a[stk[k]]<a[i])ls[i]=stk[k],--k;
5     if(k)rs[stk[k]]=i;
6     stk[++k]=i;
7     tp=k;
8 }

```

2.7 2-SAT

2-sat 至多支持对两个变量之间关系进行限制。

n 个点中恰好只有一个点为 'true' 怎么办? 这是 $K-sat$!! 至多有 1 个是可做的 (前缀优化建图), 但 "恰好有 1 个" 是不行的。

构造方案: Tarjan 对强联通分量的拓扑排序时逆序的, 序号小的不可能到达序号大的。如果 $bel_i < bel_{i+n}$, 则 i 所在的强联通分量不会推出 $i+n$, 因此不产生矛盾, 选择 i ; 反之, 则选择 $i+n$ 。

DFS 求解 2-sat 有些题目要求某些条件优先满足, 比如第 i 位优先选择 0/1 之类, 可以采用 DFS 的方式求解。相当于二分图染色, 如果染过了就不染, 如果自己的反面染过了就返回失败。这样最坏复杂度是 $O(nm)$ 的, 但是能够解决优先的问题。

2.8 Dinic 网络流、费用流

```

1 /*** Dinic 最大流 ***/
2 struct Dinic
3 {
4     #define Maxn 点数
5     #define Maxm 边数
6     int tot=1;
7     int hea[Maxn],nex[Maxm<<1],ver[Maxm<<1];
8     int tmphea[Maxn],dep[Maxn];
9     ll edg[Maxm<<1],sum;
10    inline void init()
11    { tot=1,memset(hea,0,sizeof(hea)); }
12    inline void add_edge(int x,int y,ll d)
13    {
14        ver[++tot]=y,nex[tot]=hea[x],hea[x]=tot,edg[tot]=d;
15        ver[++tot]=x,nex[tot]=hea[y],hea[y]=tot,edg[tot]=0;
16    }
17    inline bool bfs(int s,int t)
18    {
19        memset(dep,0,sizeof(dep)),dep[s]=1;
20        memcpy(tmphea,hea,sizeof(hea));
21        queue<int> q; q.push(s);
22        while(!q.empty())
23        {
24            int cur=q.front(); q.pop();
25            if(cur==t) return true;
26            for(int i=hea[cur];i;i=nex[i]) if(edg[i]>0 &&
27                !dep[ver[i]])
28                dep[ver[i]]=dep[cur]+1,q.push(ver[i]);
29        }
30        return false;
31    }
32    ll dfs(int x,ll flow,int t)
33    {
34        if(x==t || !flow) return flow;
35        ll rest=flow,tmp;
36        for(int i=tmphea[x];i && rest;i=nex[i])
37        {
38            tmphea[x]=i;
39            if(dep[ver[i]]==dep[x]+1 && edg[i]>0)
40            {
41                if(!(tmp = dfs(ver[i],min(edg[i],rest),t)))
42                    dep[ver[i]]=0;
43                edg[i]-=tmp,edg[i^1]+=tmp,rest-=tmp;
44            }
45        }
46        return rest;
47    }
48 }

```

```

42     }
43 }
44 return flow-rest;
45 }
46 inline ll solve(int s,int t)
47 {
48     sum=0;
49     while(bfs(s,t)) sum+=dfs(s,inf,t);
50     return sum;
51 }
52 #undef Maxn
53 #undef Maxm
54 }G;

```

```

54     sumflow=sumcost=0;
55     while(spfa(s,t)) sumflow+=dfs(s,infll,t);
56     return pll(sumflow,sumcost);
57 }
58 #undef Maxn
59 #undef Maxm
60 }G;

```

2.8.1 无源汇有上下界可行流

给定无源汇流量网络 G , 询问是否存在一种标定每条边流量的方式, 使得每条边流量满足上下界同时每一个点流量平衡。

```

1  /** Dinic 最小费最大流 **/
2  struct Dinic_cost
3  {
4      #define Maxn 点数
5      #define Maxm 边数
6      int tot=1;
7      int
8      ↪ tmphea[Maxn],hea[Maxn],nex[Maxm<<1],ver[Maxm<<1];
9      ll sumflow,sumcost,edg[Maxm<<1],Cost[Maxm<<1];
10     bool inq[Maxn];
11     ll dis[Maxn];
12     inline void init() {
13         ↪ tot=1,memset(hea,0,sizeof(hea)); }
14     inline void add_edge(int x,int y,ll d,ll c)
15     {
16         ver[++tot]=y, nex[tot]=hea[x], hea[x]=tot,
17         ↪ edg[tot]=d, Cost[tot]=c;
18         ver[++tot]=x, nex[tot]=hea[y], hea[y]=tot,
19         ↪ edg[tot]=0, Cost[tot]=-c;
20     }
21     inline bool spfa(int s,int t)
22     {
23         memset(dis,0x3f,sizeof(dis)),dis[s]=0;
24         memcpy(tmphea,hea,sizeof(hea));
25         queue<int> q; q.push(s);
26         while(!q.empty())
27         {
28             int cur=q.front(); q.pop(),inq[cur]=false;
29             for(int i=hea[cur];i;i=nex[i])
30                 if(edg[i] &&
31                    ↪ dis[ver[i]]>dis[cur]+Cost[i])
32                 {
33                     dis[ver[i]]=dis[cur]+Cost[i];
34                     if(!inq[ver[i]])
35                         ↪ q.push(ver[i]),inq[ver[i]]=true;
36                 }
37         }
38         return dis[t]!=infll;
39     }
40     ll dfs(int x,ll flow,int t)
41     {
42         if(x==t || !flow) return flow;
43         ll rest=flow,tmp;
44         inq[x]=true;
45         for(int i=tmphea[x];i && rest;i=nex[i])
46         {
47             tmphea[x]=i;
48             if(!inq[ver[i]] && edg[i] &&
49                ↪ dis[ver[i]]==dis[x]+Cost[i])
50             {
51                 if(!(tmp =
52                    ↪ dfs(ver[i],min(edg[i],rest),t)))
53                     dis[ver[i]]=infll;
54                 sumcost+=Cost[i]*tmp, edg[i]-=tmp,
55                 ↪ edg[i^1]+=tmp, rest-=tmp;
56             }
57         }
58         inq[x]=false;
59         return flow-rest;
60     }
61     inline pll solve(int s,int t)
62     {

```

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define infll 0x3f3f3f3f3f3f3f3f
4  #define inf 0x3f3f3f3f
5  #define pb push_back
6  #define eb emplace_back
7  #define pa pair<int, int>
8  #define fi first
9  #define se second
10 #typedef long long ll;
11 inline int rd() {
12     int x = 0;
13     char ch, t = 0;
14     while (!isdigit(ch = getchar())) t |= ch == '-';
15     while (isdigit(ch)) x = x * 10 + (ch ^ 48), ch =
16     ↪ getchar();
17     return t ? -x : x;
18 }
19 struct Dini {
20     #define Maxn 505
21     #define Maxm 20005
22     int tot = 1;
23     int hea[Maxn], tmphea[Maxn], dep[Maxn];
24     int nex[Maxm << 1], ver[Maxm << 1], num[Maxm << 1];
25     ll edg[Maxm << 1];
26     inline void addedge(int x, int y, int d, int _n) {
27         ver[++tot] = y, nex[tot] = hea[x], hea[x] = tot,
28         ↪ edg[tot] = d, num[tot] = -1;
29         ver[++tot] = x, nex[tot] = hea[y], hea[y] = tot,
30         ↪ edg[tot] = 0, num[tot] = _n;
31     }
32     inline bool bfs(int s, int t) {
33         memcpy(tmphea, hea, sizeof(hea));
34         memset(dep, 0, sizeof(dep)), dep[s] = 1;
35         queue<int> q;
36         q.push(s);
37         while (!q.empty()) {
38             int cur = q.front();
39             q.pop();
40             if (cur == t) return true;
41             for (int i = hea[cur]; i; i = nex[i])
42                 if (edg[i] > 0 && !dep[ver[i]])
43                     dep[ver[i]] = dep[cur] + 1,
44                     ↪ q.push(ver[i]);
45         }
46         return false;
47     }
48     ll dfs(int x, int t, ll flow) {
49         if (!flow || x == t) return flow;
50         ll rest = flow, tmp;
51         for (int i = tmphea[x]; i && rest; i = nex[i]) {
52             tmphea[x] = i;
53             if (dep[ver[i]] == dep[x] + 1 && edg[i] > 0)
54                 ↪ {
55                     if (!(tmp = dfs(ver[i], t, min(rest,
56                        ↪ edg[i])))) dep[ver[i]] = 0;
57                     edg[i] -= tmp, edg[i ^ 1] += tmp, rest
58                     ↪ -= tmp;
59                 }
60         }
61     }

```

```

54     return flow - rest;
55 }
56 inline ll solve(int s, int t) {
57     ll sum = 0;
58     while (bfs(s, t))
59         sum += dfs(s, t, infll);
60     return sum;
61 }
62 #undef Maxn
63 #undef Maxm
64 } G;
65 #define Maxn 205
66 #define Maxm 10205
67 int n, m, ss, tt;
68 int ans[Maxm];
69 ll needin, needout;
70 ll Ind[Maxn], Outd[Maxn];
71 int main() {
72     n = rd(), m = rd(), ss = n + 1, tt = n + 2;
73     for (int i = 1, x, y, Inf, Sup; i <= m; i++) {
74         x = rd(), y = rd(), Inf = rd(), Sup = rd();
75         G.addedge(x, y, Sup - Inf, i);
76         Outd[x] += Inf;
77         Ind[y] += Inf;
78         ans[i] = Inf;
79     }
80     for (int i = 1; i <= n; i++) {
81         if (Ind[i] > Outd[i])
82             G.addedge(ss, i, Ind[i] - Outd[i], -1),
            ↳ needin += Ind[i] - Outd[i];
83         if (Ind[i] < Outd[i])
84             G.addedge(i, tt, Outd[i] - Ind[i], -1),
            ↳ needout += Outd[i] - Ind[i];
85     }
86     ll tmp = G.solve(ss, tt);
87     if (needin != needout || needin != tmp)
88         ↳ printf("NO\n");
89     else {
90         for (int i = 2; i <= G.tot; i++)
91             if (G.num[i] != -1) ans[G.num[i]] +=
            ↳ G.edg[i];
92         printf("YES\n");
93         for (int i = 1; i <= m; i++) printf("%d\n",
            ↳ ans[i]);
94     }
95     return 0;
96 }

```

2.8.2 有源汇有上下界最大流

给定有源汇流量网络 G ，询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。

假设源点为 S ，汇点为 T ，则我们可以加入一条 T 到 S 的上界为 ∞ ，下界为 0 的边转化为无源汇上下界可行流问题。

若有解，则 S 到 T 的可行流流量等于 T 到 S 的附加边的流量。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define infll 0x3f3f3f3f3f3f3f3f
4 #define inf 0x3f3f3f3f
5 #define pb push_back
6 #define eb emplace_back
7 #define pa pair<int, int>
8 #define fi first
9 #define se second
10 typedef long long ll;
11 inline int rd() {
12     int x = 0;
13     char ch, t = 0;
14     while (!isdigit(ch = getchar())) t |= ch == '-';
15     while (isdigit(ch)) x = x * 10 + (ch ^ 48), ch =
        ↳ getchar();
16     return t ? -x : x;

```

```

17 }
18 struct Dinic {
19     #define Maxn 505
20     #define Maxm 20005
21     int tot = 1;
22     int hea[Maxn], tmphea[Maxn], dep[Maxn];
23     int nex[Maxm << 1], ver[Maxm << 1], num[Maxm << 1];
24     ll edg[Maxm << 1];
25     inline int addedge(int x, int y, ll d, int _n) {
26         ver[++tot] = y, nex[tot] = hea[x], hea[x] = tot,
            ↳ edg[tot] = d, num[tot] = -1;
27         ver[++tot] = x, nex[tot] = hea[y], hea[y] = tot,
            ↳ edg[tot] = 0, num[tot] = _n;
28         return tot;
29     }
30     inline bool bfs(int s, int t) {
31         memcpy(tmphea, hea, sizeof(hea));
32         memset(dep, 0, sizeof(dep)), dep[s] = 1;
33         queue<int> q;
34         q.push(s);
35         while (!q.empty()) {
36             int cur = q.front();
37             q.pop();
38             if (cur == t) return true;
39             for (int i = hea[cur]; i; i = nex[i])
40                 if (edg[i] > 0 && !dep[ver[i]])
41                     dep[ver[i]] = dep[cur] + 1,
                        ↳ q.push(ver[i]);
42         }
43         return false;
44     }
45     ll dfs(int x, int t, ll flow) {
46         if (!flow || x == t) return flow;
47         ll rest = flow, tmp;
48         for (int i = tmphea[x]; i && rest; i = nex[i]) {
49             tmphea[x] = i;
50             if (dep[ver[i]] == dep[x] + 1 && edg[i] > 0)
51                 ↳ {
52                     if (!(tmp = dfs(ver[i], t, min(rest,
                        ↳ edg[i]))))
53                         dep[ver[i]] = 0;
54                     edg[i] -= tmp, edg[i ^ 1] += tmp, rest
                        ↳ -= tmp;
55                 }
56         }
57         return flow - rest;
58     }
59     inline ll solve(int s, int t) {
60         ll sum = 0;
61         while (bfs(s, t))
62             sum += dfs(s, t, infll);
63         return sum;
64     }
65 #undef Maxn
66 #undef Maxm
67 } G;
68 #define Maxn 505
69 #define Maxm 20005
70 int n, m, ss, tt, s, t;
71 int ans[Maxm];
72 ll needin, needout;
73 ll Ind[Maxn], Outd[Maxn];
74 int main() {
75     n = rd(), m = rd(), s = rd(), t = rd(), ss = n + 1,
        ↳ tt = n + 2;
76     for (int i = 1, x, y, Inf, Sup; i <= m; i++) {
77         x = rd(), y = rd(), Inf = rd(), Sup = rd();
78         G.addedge(x, y, Sup - Inf, i);
79         Outd[x] += Inf;
80         Ind[y] += Inf;
81         ans[i] = Inf;
82     }
83     for (int i = 1; i <= n; i++) {
84         if (Ind[i] > Outd[i])
85             G.addedge(ss, i, Ind[i] - Outd[i], -1),
            ↳ needin += Ind[i] - Outd[i];

```



```

85         if (Ind[i] < Outd[i])
86             G.addedge(i, tt, Outd[i] - Ind[i], -1),
               ↳ needout += Outd[i] - Ind[i];
87     }
88     G.addedge(t, s, infll, -1);
89     ll tmp = G.solve(ss, tt);
90     if (needin != needout || needin != tmp)
        ↳ printf("please go home to sleep\n");
91     else {
92         tmp = G.edg[G.tot];
93         G.edg[G.tot] = G.edg[G.tot - 1] = 0;
94         tmp += G.solve(s, t);
95         printf("%lld\n", tmp);
96     }
97     return 0;
98 }

```

2.8.3 网络流总结

最小割集, 最小割必须边以及可行边

最小割条件 $S \setminus T$ 是最小割当前仅当残余网络没有 S 指向 T 的边。

最小割集 从 S 出发, 在残余网络中 BFS 所有权值非 0 的边 (包括反向边), 得到点集 $\{S\}$, 另一集为 $\{V\} - \{S\}$ 。

最小割集必须点 残余网络中 S 直接连向的点必在 S 的割集中, 直接连向 T 的点必在 T 的割集中; 若这些点的并集为全集, 则最小割方案唯一。

最小割可行边 在残余网络中求强联通分量, 将强联通分量缩点后, 剩余的满流边即为最小割可行边。

最小割必须边 在残余网络中求强联通分量, 若 $scc_S = scc_u$ 且 $scc_T = scc_v$, 则该边为必须边。

常见问题

最大权闭合子图 点权, 限制条件形如: 选择 A 则必须选择 B, 选择 B 则必须选择 C, D. 建图方式: A 向 B 连边, B 向 CD 连边. 求解: S 向正权点连边权为权值的边, 负权点向 T 连边权为权值的边, 其余边容量 ∞ , 求最小割, 答案的方案为 S 所在最小割集, 答案为 (正权点权值和-最小割)。

二次布尔型 (文理分科) n 个点分为两类, i 号点有 l_i 或 r_i 的代价, i, j 同属一侧分别获得 l_{ij} 或 r_{ij} 的代价, 问最小代价. $L \rightarrow i : (l_i + 1/2 \sum_j l_{ij})$, $i \rightarrow R : (r_i + 1/2 \sum_j r_{ij})$, $i \leftrightarrow j : 1/2(l_{ij} + r_{ij})$. 实现时边权乘 2 为整数, 求解后答案除 2 为整数. 图拆点可以看作二分图。

如果是二元限制是分类不同时有一个代价 d_{ij} , 建图可以简化为 $L \rightarrow i : l_i$, $i \rightarrow R : r_i$, $i \leftrightarrow j : d_{ij}$. 经典例子: xor 最小值, 按位拆开建图。

混合图欧拉回路 把无向边随便定向, 计算每个点的入度和出度, 如果有某个点出入度之差 $\deg_i = \text{in}_i - \text{out}_i$ 为奇数, 肯定不存在欧拉回路. 对于 $\deg_i > 0$ 的点, 连接边 $(i, T, \deg_i/2)$; 对于 $\deg_i < 0$ 的点, 连接边 $(S, i, -\deg_i/2)$. 最后检查是否满流即可。

二物流 水源 S_1 , 水汇 T_1 , 油源 S_2 , 油汇 T_2 , 每根管道流量共用. 求流量和最大. 建超级源 SS_1 汇 TT_1 , 连边 $SS_1 \rightarrow S_1, SS_1 \rightarrow S_2, T_1 \rightarrow TT_1, T_2 \rightarrow TT_1$, 设最大流为 x_1 . 建超级源 SS_2 汇 TT_2 , 连边 $SS_2 \rightarrow S_1, SS_2 \rightarrow T_2, T_1 \rightarrow TT_2, S_2 \rightarrow TT_2$, 设最大流为 x_2 . 则最大流中水流量 $\frac{x_1+x_2}{2}$, 油流量 $\frac{x_1-x_2}{2}$ 。

无源汇有上下界可行流 每条边 (u, v) 有一个上界容量 $C_{u,v}$ 和下界容量 $B_{u,v}$, 我们让下界变为 0, 上界变为 $C_{u,v} - B_{u,v}$, 但这样做流量不守恒. 建立超级源点 SS 和超级汇点 TT , 用 du_i 来记录每个节点的流量情况, $du_i = \sum B_{j,i} - \sum B_{i,j}$, 添加一些附加弧. 当 $du_i > 0$ 时, 连边 (SS, i, du_i) ; 当 $du_i < 0$ 时, 连边 $(i, TT, -du_i)$. 最后对 (SS, TT) 求一次最大流即可, 当所有附加边全部满流时 (即 $\text{maxflow} == du_i > 0$) 时有可行解。

有源汇有上下界最大可行流 建立超级源点 SS 和超级汇点 TT , 首先判断是否存在可行流, 用无源汇有上下界可行流的方法判断. 增设一条从 T 到 S 没有下界容量为无穷的边, 那么原图就变成了一个无源汇有上下界可行流问题. 同样地建图后, 对 (SS, TT) 进行一次最大流, 判断是否有可行解. 如果有可行解, 删除超级源点 SS 和超级汇点 TT , 并删去 T 到 S 的这条边, 再对 (S, T) 进行一次最大流, 此时得到的 maxflow 即为有源汇有上下界最大可行流。

有源汇有上下界最小可行流 建立超级源点 SS 和超级汇点 TT , 和无源汇有上下界可行流一样新增一些边, 然后从 SS 到 TT 跑最大流. 接着加上边 (T, S, ∞) , 再从 SS 到 TT 跑一遍最大流. 如果所有新增边都是满的, 则存在可行流, 此时 T 到 S 这条边的流量即为最小可行流。

有上下界费用流 如果求无源汇有上下界最小费用可行流或有源汇有上下界最小费用最大可行流, 用 1.6.3.1/1.6.3.2 的构图方法, 给边加上费用即可. 求有源汇有上下界最小费用最小可行流, 要先用 1.6.3.3 的方法建图, 先求出一个保证必要边满流情况下的最小费用. 如果费用全部非负, 那么这时的费用就是答案. 如果费用有负数, 那么流多了可能更好, 继续做从 S 到 T 的流量任意的最小费用流, 加上原来的费用就是答案。

费用流消负环 新建超级源 SS 汇 TT , 对于所有流量非空的负权边 e , 先流满 ($\text{ans} += e.f * e.c$, $e.\text{rev}.f += e.f$, $e.f = 0$), 再连边 $SS \rightarrow e.to$, $e.from \rightarrow TT$, 流量均为 $e.f (> 0)$, 费用均为 0. 再连边 $T \rightarrow S$ 流量 ∞ 费用 0. 此时没有负环了. 做一遍 SS 到 TT 的最小费用最大流, 将费用累加 ans , 拆掉 $T \rightarrow S$ 的那条边 (此边的流量为残量网络中 $S \rightarrow T$ 的流量). 此时负环已消, 再继续跑最小费用最大流。

二分图最小点覆盖和最大独立集

最小点覆盖: 求出一个最大匹配, 从左部开始每次寻找一个未匹配点, 从该点出发可以得到“未匹配-匹配-未匹配...”形式的交错树, 标记所有这些点. 则最小点覆盖方案为右部未标记点与左部标记点的并集. 显然最小点覆盖集合大小 = 最大匹配。

最大独立集 = 全集 - 最小点覆盖。

Hall 定理

二分图存在完备匹配 $\iff \forall S \subseteq X, |S| \leq |N(S)|$

最大匹配 = $|X| - \max_{S \subseteq X} \{|S| - |N(S)|\}$, 其中 X 为左部点, S 为左部点一个子集, $N(S)$ 为 S 中点连接右部的所有点。

整数线性规划转费用流

首先将约束关系转化为所有变量下界为 0, 上界没有要求, 并满足一些等式, 每个变量在均在等式左边且出现恰好两次, 系数为 +1 和 -1, 优化目标为 $\max \sum v_i x_i$ 的形式. 将等式看做点, 等式 i 右边的值 b_i 若为正, 则 S 向 i 连边 $(b_i, 0)$, 否则 i 向 T 连边 $(-b_i, 0)$. 将变量看做边, 记变量 x_i 的上界为 m_i (无上界则 $m_i = \text{inf}$), 将 x_i 系数为 +1 的那个等式 u 向系数为 -1 的等式 v 连边 (m_i, v_i) .

最小路径覆盖

给定一个 DAG, 求把 DAG 划分为若干条路径, 使得每个点都恰好某个路径上的方案中, 路径条数最小是多少. 那么我们就把每个点拆成两个, (u, u') , u 和 s 连接, u' 和 t 连接, 边 (u, v) 变成网络中 (u, v') , 最大流就是最小路径覆盖条数.

DAG 最长反链

做法 1: 考虑把一个点拆成两个 u, u' , 其中 u 的权值为 -1, u' 的权值为 1, 以及需要建立 (u, u') 之间的边. 然后对于每一条原图上的边 (u, v) , 在新图上建立 (u', v) . 该图的最大权闭合子图即为最大反链长度. 构造方案: u 在反链中 \iff u 在 T 里面且 u' 在 S 里面.
做法 2: 根据 Dilworth 定理, 最长反链长度等于最小链划分, 可以对原图先求一次传递闭包, 然后求最小路径覆盖.

2.9 差分约束

差分约束系统是一种特殊的 n 元一次不等式组. 差分约束系统中的每个约束条件 $x_i - x_j \leq c_k$ 都可以变形成 $x_i \leq x_j + c_k$ 与 $x_j \geq x_i - c_k$, 这与单源最短路中的三角形不等式非常相似. 因此, 我们可以把每个变量 x_i 看做图中的一个结点, 对于每个约束条件连边. 需要注意的是, 有些题目看会对解的上、下界进行约束, 因此我们需要对这些条件处理 (这里只考虑对于这 n 个元素 只约束了上界或 只约束了下界):

- 只约束下界: 有 0 号点向每一个点连一条长为 Lim_i 的边, 表示第 i 号元素的下界为 Lim_i , 如图所示建边:

| 题意 | 转化 | 连边 |
|--------------------|------------------------------|------------------------|
| $x_a - x_b \geq c$ | $x_a \geq x_b + c$ | add(b,a,c) |
| $x_a - x_b \leq c$ | $x_b \geq x_a - c$ | add(a,b,-c) |
| $x_a = x_b$ | $x_a \geq x_b, x_b \leq x_a$ | add(a,b,0), add(b,a,0) |

之后对整张图跑 最长路。

- 只约束上界: 有 0 号点向每一个点连一条长为 Lim_i 的边, 表示第 i 号元素的上界为 Lim_i , 如图所示建边:

| 题意 | 转化 | 连边 |
|--------------------|------------------------------|------------------------|
| $x_a - x_b \geq c$ | $x_b \leq x_a - c$ | add(a,b,-c) |
| $x_a - x_b \leq c$ | $x_a \leq x_b + c$ | add(b,a,c) |
| $x_a = x_b$ | $x_a \leq x_b, x_b \leq x_a$ | add(a,b,0), add(b,a,0) |

之后对整张图跑 最短路。

设 $dist[0] = 0$, 若存在负环 / 正环, 则不等式无解, 否则 $x_i = dist[i]$ 是该差分约束系统的一组解. 最坏情况下 (存在负环 / 正环) 复杂度为 $O(nm)$.
注意: 整个图不一定是联通的!

2.10 欧拉路径

2.10.1 有向图

```
1 vector<int> G[N];
2 int cur[N];
3 void dfs(int u){
4     for(int& i=cur[u];i<G[u].size();){
5         i++; // 提前加 i
6         dfs(G[u][i-1]);
7         // 在这里往栈中加入边 (u -> G[u][i-1])
8     }
9     // 在这里往栈中加入点 u
10    ans.push(u);
11 }
```

2.10.2 无向图

```
1 struct Edge{
2     int to,rev; // 终点, 反向边编号
3     bool exist;
4 };
5 vector<Edge> G[N];
6 int cur[N];
7 stack<int> ans;
8 void add_edge(int u,int v){
9     G[u].push_back(Edge{v,(int)G[v].size(),true});
10    // if(u!=v)G[v].push_back(Edge{u,(int)G[u].size()-1,true});
11 }
12 void dfs(int u){
13     for(int& i=cur[u];i<G[u].size();){
14         i++;
15         if(!G[u][i-1].exist)continue;
16         G[u][i-1].exist = 0;
17         G[G[u][i-1].to][G[u][i-1].rev].exist=0;
18         dfs(G[u][i-1].to);
19         // 在这里加入边 (u->G[u][i-1].to)
20     }
21     // 在这里加入节点 u
22 }
```

2.11 同余最短路

形如:

- 设问 1: 给定 n 个整数, 求这 n 个整数在 $h(h \leq 2^{63} - 1)$ 范围内 能拼凑出多少的其他整数 (整数可以重复取)。
- 设问 2: 给定 n 个整数, 求这 n 个整数 不能拼凑出的最小 (最大) 的整数。

设 x 为 n 个数中最小的一个, 令 $ds[i]$ 为只通过增加其他 $n - 1$ 种数能够达到的最低楼层 p , 并且满足 $p \equiv i \pmod{x}$. 对于 $n - 1$ 个数与 x 个 $ds[i]$, 可以如下连边:

```
1 for(int i=0;i<x;i++) for(int j=2;j<=n;j++){
2     // add(i,(i+a[j])*x,a[j]);
3 }
```

之后进行最短路, 对于:

- 问在 h 范围内能够到达的点的数量: 答案为 (加一因为 i 本身也要计算)

$$\sum_{i=0}^{x-1} [d[i] \leq h] \times \frac{h - d[i]}{x} + 1$$

- 问不能达到的最小的数: 答案为: (i 一定时最小表示的数为 $d[i] = s \times x + i$, 则 $(s - 1) \times x + i$ 一定不能被表示出来)

$$\min_{i=1}^{x-1} \{d[i] - x\}$$

注意: ds 与 h 范围相同, 一般也要开 long long !

2.12 树的计数

2.12.1 树的计数 Prufer 序列

prufer 编码长度为 $n - 2$, 且度数为 d_i 的点在 prufer 编码中出现 $d_i - 1$ 次.

由树得到序列: 总共需要 $n - 2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为 Prufer 序列的第 i 个元素 p_i , 并将此叶子节点从树中删除, 直到最后得到一个长度为 $n - 2$ 的 Prufer 序列和一个只有两个节点的树.

由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在 Prufer 序列中出现的次数, 得到每个点的度; 执行 $n - 2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v = p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.

相关结论: n 个点完全图, 每个点度数依次为 d_1, d_2, \dots, d_n , 这样生成树的棵树为: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$.

左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵树为 $n_1^{n_2-1} \times n_2^{n_1-1}$.

m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2} \prod c_i$

2.12.2 有根树计数 1,1,2,4,9,20,48,115,286,719,1842,4766

无标号 $a_{n+1} = 1/n \sum_{k=1}^n (\sum_{d|k} d \cdot a(d)) \cdot a(n-k+1)$

2.12.3 无根树计数

n 是奇数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i}$ 种不同的无根树.

n 是偶数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i} + \frac{1}{2} a_{n/2} (a_{n/2} + 1)$ 种不同的无根树.

2.12.4 生成树计数 Kirchhoff's Matrix-Tree Theorem

Kirchhoff Matrix $T = Deg - A$, Deg 是度数对角阵, A 是邻接矩阵. 无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.

邻接矩阵 $A[u][v]$ 表示 $u \rightarrow v$ 边个数, 重边按照边数计算, 自环不计入度数.

无向图生成树计数: $c = |K|$ 的任意 1 个 $n - 1$ 阶主子式 |

有向图外向树计数: $c = |$ 去掉根所在的那阶得到的主子式 |

2.12.5 有向图欧拉回路计数 BEST Theorem

$$ec(G) = t_w(G) \prod_{v \in V} (\deg(v) - 1)!$$

其中 \deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数. 相关计算参考生成树计数.

欧拉连通图中任意两点外向树个数相同: $t_v(G) = t_w(G)$.

3 数据结构

3.1 平衡树

3.1.1 FHQ Treap

```

1 struct FHQ_number
2 {
3     #define Maxn 点数
4     #define ls tree[p].pl
5     #define rs tree[p].pr
6     private:
7         int All=0,root=0;
8         struct NODE { int pl,pr,siz,cnt,rnd; ll val; };
9         NODE tree[Maxn];
10        inline int Dot() { return ++All; }
11        inline int New(ll Val)
12        {
13            int p=Dot();
14            tree[p].rnd=rand(),tree[p].val=Val;
15            tree[p].siz=tree[p].cnt=1;
16            tree[p].pl=tree[p].pr=0;
17            return p;
18        }
19        inline void pushdown(int p) { p--; }
20        inline void pushup(int p)
21        {
22            ↪ tree[p].siz=tree[ls].siz+tree[rs].siz+tree[p].cnt;
23            ↪ }
24        void split(int p,int k,int &x,int &y)
25        {
26            if(!p) { x=y=0; return; }
27            pushdown(p);
28            if(tree[p].val<=k) x=p,split(rs,k,rs,y);
29            else y=p,split(ls,k,x,ls);
30            pushup(p);
31        }
32        int merge(int x,int y)
33        {
34            if(!x || !y) return x+y;
35            if(tree[x].rnd<tree[y].rnd)
36            {
37                pushdown(x),tree[x].pr=merge(tree[x].pr,y),pushup(x);
38                return x;
39            }
40            else
41            {
42                pushdown(y),tree[y].pl=merge(x,tree[y].pl),pushup(y);
43                return y;
44            }
45        }
46        inline int kth(int p,int Rank)
47        {
48            while(p)
49            {
50                if(tree[ls].siz>=Rank) p=ls;
51                else if(tree[ls].siz+tree[p].cnt>=Rank)
52                    ↪ return p;
53                else Rank-=tree[ls].siz+tree[p].cnt,p=rs;
54            }
55            return p;
56        }
57        int x,y,z;
58        public:
59        inline void Insert(ll Val)
60        {
61            ↪ split(root,Val,x,y),root=merge(merge(x,New(Val)),y);
62            ↪ }
63        inline void Delete_one(int Val)
64        {
65            split(root,Val,x,z),split(x,Val-1,x,y);
66            y=merge(tree[y].pl,tree[y].pr);
67            root=merge(merge(x,y),z);
68        }
69        inline ll Rank_to_Value(int Rank)
70        {
71            { return tree[kth(root,Rank)].val; }
72            inline int Value_to_Rank(ll Value)
73            {
74                split(root,Value-1,x,y);
75                int ret=tree[x].siz+1;
76                root=merge(x,y);
77                return ret;
78            }
79            inline ll Findpre(ll Value)
80            {
81                split(root,Value-1,x,y);
82                ll ret=tree[kth(x,tree[x].siz)].val;
83                root=merge(x,y);
84                return ret;
85            }
86            inline ll Findnex(ll Value)
87            {
88                split(root,Value,x,y);
89                ll ret=tree[kth(y,1)].val;
90                root=merge(x,y);
91                return ret;
92            }
93        }T;
94        struct FHQ_sequence
95        {
96            #define Maxn 点数
97            #define ls tree[p].pl
98            #define rs tree[p].pr
99            int All=0,root=0;
100            struct NODE { int pl,pr,siz,cnt,rnd,val; bool tag; };
101            NODE tree[Maxn];
102            inline int Dot() { return ++All; }
103            inline int New(int Val)
104            {
105                int p=Dot();
106                tree[p].rnd=rand(),tree[p].val=Val;
107                tree[p].cnt=tree[p].siz=1;
108                tree[p].pl=tree[p].pr=0;
109                return p;
110            }
111            inline void pushdown(int p)
112            {
113                if(!tree[p].tag) return;
114                swap(tree[ls].pl,tree[ls].pr);
115                swap(tree[rs].pl,tree[rs].pr);
116                tree[ls].tag^=1,tree[rs].tag^=1;
117                tree[p].tag=false;
118            }
119            inline void pushup(int p)
120            {
121                ↪ tree[p].siz=tree[ls].siz+tree[p].cnt+tree[rs].siz;
122                ↪ }
123            void split(int p,int k,int &x,int &y)
124            {
125                if(!p) { x=y=0; return; }
126                pushdown(p);
127                if(tree[ls].siz<k)
128                    ↪ x=p,split(rs,k-tree[ls].siz-1,rs,y);
129                else y=p,split(ls,k,x,ls);
130                pushup(p);
131            }
132            int merge(int x,int y)
133            {
134                if(!x || !y) return x+y;
135                if(tree[x].rnd<tree[y].rnd)
136                {
137                    pushdown(x),tree[x].pr=merge(tree[x].pr,y),pushup(x);
138                    return x;
139                }
140                else
141                {
142                    pushdown(y),tree[y].pl=merge(x,tree[y].pl),pushup(y);
143                    return y;
144                }
145            }
146            int kth(int p,int Rank)
147            {

```

```

139 while(p)
140 {
141     if(tree[ls].siz>=Rank) p=ls;
142     else if(tree[ls].siz+tree[p].cnt>=Rank) return
        ↳ p;
143     else Rank-=tree[ls].siz+tree[p].cnt,p=ls;
144 }
145 return p;
146 }
147 void Insert(int Val) // 插到末尾
148 { root=merge(root,New(Val)); }
149 int x,y,z;
150 inline void Reverse(int l,int r)
151 {
152     split(root,r,x,z),split(x,l-1,x,y);
153     swap(tree[y].pl,tree[y].pr),tree[y].tag^=1;
154     root=merge(merge(x,y),z);
155 }
156 void print(int p)
157 {
158     pushdown(p);
159     if(ls) print(ls);
160     printf("%d ",tree[p].val);
161     if(rs) print(rs);
162 }
163 }T;

```

3.1.2 平衡树合并

如果需要合并两个有交集的 Treap 时该怎么做？我们可以每次将较小的数合并到较大的树中去，这样每个点最多只会合并 $\log n$ 次，每次合并复杂度 $O(n \log n)$ ，总时间复杂度 $O(n \log n \log V)$ 。

代码其实非常暴力，就是直接对更小的那棵树直接一个插入进去：

```

1 // ls,rs 为 tree[p].pl 和 tree[p].pr
2 void mergetree(int p,int &rt) // rt = (rt+p)
3 {
4     if(!p) return;
5     mergetree(ls,rt);
6     mergetree(rs,rt);
7     int x,y;
8     split(rt,tree[p].val,x,y);
9     ls=rs=0,rt=merge(merge(x,p),y);
10 }

```

可以证明，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

```

1 // 洛谷 P3391
2 // 理论上 splay 的每次操作以后都应该立刻做 splay 操作以保
    ↳ 证均摊时间复杂度，但是某些地方又不能立刻做，例如下面的
    ↳ kth。
3 int rt;
4 struct Splay {
5     struct node {
6         int val, siz, fa, ch[2], rv;
7         #define ls(x) nod[x].ch[0]
8         #define rs(x) nod[x].ch[1]
9         #define val(x) nod[x].val
10        #define siz(x) nod[x].siz
11        #define fa(x) nod[x].fa
12        #define rv(x) nod[x].rv
13    } nod[N];
14    int cnt;
15    bool chk(int x) { return x == rs(fa(x)); }
16    void reverse(int x) { rv(x) ^= 1, swap(ls(x),
        ↳ rs(x)); }
17    void pushup(int x) { siz(x) = siz(ls(x)) + 1 +
        ↳ siz(rs(x)); }
18    void pushdown(int x) { if (rv(x)) reverse(ls(x)),
        ↳ reverse(rs(x)), rv(x) = 0; }

```

```

19 void connect(int x, int fa, int son) { fa(x) = fa,
    ↳ nod[fa].ch[son] = x; }
20 void rotate(int x) {
21     int y = fa(x), z = fa(y), ys = chk(x), zs =
        ↳ chk(y), u = nod[x].ch[!ys];
22     connect(u, y, ys), connect(y, x, !ys),
        ↳ connect(x, z, zs), pushup(y), pushup(x);
23 }
24 void pushall(int x) { if (fa(x)) pushall(fa(x));
    ↳ pushdown(x); }
25 void splay(int x, int to) {
26     pushall(x);
27     while (fa(x) != to) {
28         int y = fa(x);
29         if (fa(y) != to) rotate(chk(x) == chk(y) ? y
            ↳ : x);
30         rotate(x);
31     }
32     if (!to) rt = x;
33 } // 将 x 伸展为 to 的儿子。
34 void append(int val) {
35     if (!rt) nod[rt = ++cnt] = {val, 1};
36     else {
37         int x = rt;
38         while (rs(x)) pushdown(x), x = rs(x);
39         splay(x, 0), nod[rs(x) = ++cnt] = {val, 1,
            ↳ x}, pushup(x);
40     }
41 }
42 int kth(int k) {
43     int x = rt;
44     while (x) {
45         pushdown(x);
46         if (siz(ls(x)) + 1 == k) return x;
47         else if (k <= siz(ls(x))) x = ls(x);
48         else k -= siz(ls(x)) + 1, x = rs(x);
49     }
50     return -1;
51 } // kth 做完以后不能立刻 splay，因为需要提取区间。
52 void reverse(int l, int r) {
53     splay(kth(r + 2), 0), splay(kth(l), rt);
54     reverse(rs(ls(rt))), pushup(ls(rt)), pushup(rt);
55 } // 这里添加了前后两个哨兵，以避免额外的分类讨论。
56 } spl;

```

3.2 LCT 动态树

```

1 // 洛谷 P3690
2 struct LCT {
3     struct node {
4         int rv, ch[2], fa, sm, val;
5         #define ls(x) nod[x].ch[0]
6         #define rs(x) nod[x].ch[1]
7         #define fa(x) nod[x].fa
8         #define sm(x) nod[x].sm
9         #define rv(x) nod[x].rv
10        #define val(x) nod[x].val
11    } nod[N];
12    // 根节点的父亲：链顶节点的树上父亲。
13    // 其余节点的父亲：splay 中的父亲。
14
15    bool chk(int x) { return rs(fa(x)) == x; }
16    bool isroot(int x) { return nod[fa(x)].ch[chk(x)] !=
        ↳ x; }
17    void pushup(int x) { sm(x) = sm(ls(x)) ^ val(x) ^
        ↳ sm(rs(x)); }
18    void reverse(int x) { rv(x) ^= 1, swap(ls(x),
        ↳ rs(x)); }
19    void pushdown(int x) {
20        if (rv(x)) reverse(ls(x)), reverse(rs(x)), rv(x)
            ↳ ^= 0;
21    }

```

```

22 void connect(int x, int fa, int son) { fa(x) = fa,
    ↳ nod[fa].ch[son] = x; }
23 void rotate(int x) {
24     int y = fa(x), z = fa(y), ys = chk(x), zs =
    ↳ chk(y), u = nod[x].ch[!ys];
25     if (isroot(y)) fa(x) = z;
26     else connect(x, z, zs);
27     connect(u, y, ys), connect(y, x, !ys),
    ↳ pushup(y), pushup(x);
28 }
29 void pushall(int x) { if (!isroot(x))
    ↳ pushall(fa(x)); pushdown(x); }
30 void splay(int x) {
31     pushall(x);
32     while (!isroot(x)) {
33         if (!isroot(fa(x))) rotate(chk(x) ==
    ↳ chk(fa(x)) ? fa(x) : x);
34         rotate(x);
35     }
36 }
37 void access(int x) { for (int y = 0; x; y = x, x =
    ↳ fa(x)) splay(x), rs(x) = y, pushup(x); }
38 void makeroot(int x) { access(x), splay(x),
    ↳ reverse(x); }
39 int findroot(int x) { access(x), splay(x); while
    ↳ (ls(x)) pushdown(x), x = ls(x); return splay(x),
    ↳ x; }
40 void link(int x, int y) { makeroot(y); if
    ↳ (findroot(x) != y) fa(y) = x; }
41 void split(int x, int y) { makeroot(y), access(x),
    ↳ splay(x); }
42 void cut(int x, int y) { split(x, y); if (ls(x) ==
    ↳ y) ls(x) = fa(y) = 0, pushup(x); }
43 void modify(int x, int val) { splay(x), val(x) =
    ↳ val, pushup(x); }
44 int sum(int x, int y) { split(x, y); return sm(x); }
45 // 任何操作过后都应该立即 splay 以保证均摊复杂度。
46 } lct;

```

3.3 ODT 珂朵莉树

```

1 // 珂朵莉树的本质是颜色段均摊。若保证数据随机，可以证明其期
    ↳ 望时间复杂度为  $O(n \log n)$ 。
2 struct ODT {
3     struct node {
4         int l, r;
5         mutable int val;
6         node(int L, int R, int V) { l = L, r = R, val =
    ↳ V; }
7         bool operator < (const node &rhs) const { return
    ↳ l < rhs.l; }
8     };
9     set<node> s;
10    auto split(int x) {
11        auto it = s.lower_bound(node(x, 0, 0));
12        if (it != s.end() && it->l == x) return it;
13        it--;
14        int l = it->l, r = it->r, val = it->val;
15        s.erase(it), s.insert(node(l, x - 1, val));
16        return s.insert(node(x, r, val)).first;
17    }
18    void assign(int l, int r, int val) {
19        // 此处须先 split(r + 1)。因为若先 split(l)，则
    ↳ 后来的 split(r + 1) 可能致使 itl 失效。
20        auto itr = split(r + 1), itl = split(l);
21        s.erase(itl, itr), s.insert(node(l, r, val));
22    }
23    void perform (int l, int r) {
24        auto itr = split(r + 1), itl = split(l);
25        while (itl != itr) {
26            // do something...
27            itl++;
28        }
29    }

```

```

30 } odt;

```

3.4 李超线段树

```

1 struct Line { double b,k; } a[Maxn];
2 inline double calc(int p,int x){ return
    ↳ a[p].b+a[p].k*1.0*x; }
3 inline int tomax(int p1,int p2,int x)
4     { return (calc(p1,x)>calc(p2,x)+eps)?p1:p2; }
5 struct Segment_Tree
6 {
7     int tree[Maxn<<2];
8     void add(int p,int nl,int nr,int l,int r,int x)
9     {
10        int mid=(nl+nr)>>1;
11        if(nl>=l && nr<=r)
12        {
13            if(calc(x,mid)>calc(tree[p],mid)+eps)
    ↳ swap(tree[p],x);
14            if(calc(x,nl)>calc(tree[p],nl)+eps)
    ↳ add(p<<1,nl,mid,l,r,x);
15            if(calc(x,nr)>calc(tree[p],nr)+eps)
    ↳ add(p<<1|1,mid+1,nr,l,r,x);
16            return;
17        } // 插入  $O(\log^2 n)$  : 定位到  $O(\log n)$  个区间, 每个
    ↳ 区间  $O(\log n)$  递归到叶子。
18        if(mid>=l) add(p<<1,nl,mid,l,r,x);
19        if(mid<r) add(p<<1|1,mid+1,nr,l,r,x);
20    }
21    int query(int p,int nl,int nr,int x)
22    {
23        if(nl==nr) return tree[p];
24        int mid=(nl+nr)>>1;
25        if(mid>=x) return
26            ↳ tomax(tree[p],query(p<<1,nl,mid,x),x);
27        else return
28            ↳ tomax(tree[p],query(p<<1|1,mid+1,nr,x),x);
29    }
30 }T;

```

李超线段树的合并本质上和线段树没什么区别，只是在 merge 完两棵子树后的 update 中改为将 y 的最优线段再放到 x 中进行一次 add 即可。

```

1 int merge(int x,int y,int nl,int nr)
2 {
3     if(!x || !y) return x+y;
4     int mid=(nl+nr)>>1;
5     tree[x].pl=merge(tree[x].pl,tree[y].pl,nl,mid);
6     tree[x].pr=merge(tree[x].pr,tree[y].pr,mid+1,nr);
7     add(x,nl,nr,tree[y].num);
8     return x;
9 }

```

3.5 二维树状数组

```

1 struct BIT {
2     int sm[N][N][4]; // sm 是 sum 的缩写。
3     int lowbit(int x) { return x & -x; }
4     void add(int x, int y, int k) {
5         int a = k, b = k * x, c = k * y, d = k * x * y;
6         for (int i = x; i <= n; i += lowbit(i)) {
7             for (int j = y; j <= m; j += lowbit(j)) {
8                 sm[i][j][0] += a;
9                 sm[i][j][1] += b;
10                sm[i][j][2] += c;
11                sm[i][j][3] += d;
12            }
13        }

```

```

14 }
15 int query(int x,int y) {
16     int ret = 0, a = x * y + x + y + 1, b = y + 1, c
        ↳ = x + 1, d = 1;
17     for (int i = x; i; i -= lowbit(i)) {
18         for(int j = y; j; j -= lowbit(j)) {
19             ret += sm[i][j][0] * a;
20             ret -= sm[i][j][1] * b;
21             ret -= sm[i][j][2] * c;
22             ret += sm[i][j][3] * d;
23         }
24     }
25     return ret;
26 }
27 } bit;
28
29 // [a, c] * [b, d] + x : add(a, b, x), add(a, d + 1,
        ↳ -x), add(c + 1, b, -x), add(c + 1, d + 1, x);
30 // sum of [a, c] * [b, d] : query(c, d) - query(a - 1,
        ↳ d) - query(c, b - 1) + query(a - 1, b - 1);

```

```

13     if (dis(ls(x)) < dis(rs(x))) swap(ls(x), rs(x));
14     dis(x) = dis(rs(x)) + 1;
15 } // 若根的 dis 为 d, 则左偏树至少包含  $O(2^d)$  个节点,
        ↳ 因此  $d=O(\log n)$ 。
16 } hp;

```

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:

```

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

```

在合并节点时, 加入:

```

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

```

在弹出最小值时加入:

```

1 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。

3.6 虚树

★attention: 由于整个图需要用到到的边与点很少, 所以在每次新建虚树的时候不能全局清空, 而是在把一个新的点加入栈中的时候清空这个点连过的边。

★attention: 一定要在将这个点加入栈的时候清空这个点的连边情况!!!! 不能早也不能晚。

```

1 inline void build(int s)
2 {
3     sort(dot+1,dot+m+1,cmp),tot=0;
4     sta[tp=1]=1,hea[1]=0;
5     for(int i=1,l;i<=m;i++)
6     {
7         if(dot[i]==1) continue; // 别忘了 1!
8         l=Lca(sta[tp],dot[i]);
9         if(l!=sta[tp])
10        {
11            while(dfn[l]<dfn[sta[tp-1]])
12                ↳ add(sta[tp-1],sta[tp],dist(sta[tp]-sta[tp-1])),
                    ↳ tp--;
13            if(sta[tp-1]==l)
14                add(l,sta[tp],dist(sta[tp]-l)),tp--;
15            else
16                hea[l]=0,
                    ↳ add(l,sta[tp],dist(sta[tp]-l)),
                    ↳ sta[tp]=l;
17        }
18        hea[dot[i]]=0,sta[++tp]=dot[i];
19    }
20    for(int i=1;i<tp;i++)
21        ↳ add(sta[i],sta[i+1],dist(sta[i+1]-sta[i]));

```

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。

```

1 void Max(auto &x, auto y) { x = max(x, y); }
2 void Min(auto &x, auto y) { x = min(x, y); }
3 struct SMT {
4     #define ls (k << 1)
5     #define rs (k << 1 | 1)
6     #define mid ((l + r) >> 1)
7     struct node {
8         int sm, mx, mx2, c, hmx, ad, ad2, had, had2;
9         // 区间和, 最大值, 次大值, 最大值个数, 历史最大值,
            ↳ 最大值加法标记, 其余值加法标记, 最大值的历史最
            ↳ 大加法标记, 其余值的历史最大加法标记。
10        node operator + (const node &x) const {
11            node t = *this;
12            t.sm += x.sm, Max(t.hmx, x.hmx);
13            if (t.mx > x.mx) Max(t.mx2, x.mx);
14            else if (t.mx < x.mx) t.mx2 = max(t.mx,
                ↳ x.mx2), t.mx = x.mx, t.c = x.c;
15            else Max(t.mx2, x.mx2), t.c += x.c;
16            t.ad = t.ad2 = t.had = t.had2 = 0;
17            return t;
18        }
19    } nod[N << 2];
20    #define sm(x) nod[x].sm
21    #define mx(x) nod[x].mx
22    #define mx2(x) nod[x].mx2
23    #define c(x) nod[x].c
24    #define hmx(x) nod[x].hmx
25    #define ad(x) nod[x].ad
26    #define ad2(x) nod[x].ad2
27    #define had(x) nod[x].had

```

3.7 左偏树

```

1 struct heap {
2     struct node {
3         int val, dis, ch[2];
4         #define val(x) nod[x].val
5         #define ls(x) nod[x].ch[0]
6         #define rs(x) nod[x].ch[1]
7         #define dis(x) nod[x].dis
8     } nod[N]; // dis : 节点到叶子的最短距离。
9     int merge(int x, int y) {
10        if (!x || !y) return x | y;
11        if (val(x) > val(y)) swap(x, y);
12        rs(x) = merge(rs(x), y);

```



```

28 #define had2(x) nod[x].had2
29 void pushup(int k) { nod[k] = nod[ls] + nod[rs]; }
30 void add(int k, int l, int r, int ad, int ad2, int
    ↪ had, int had2) {
31     Max(had(k), ad(k) + had, Max(had2(k), ad2(k) +
    ↪ had2), Max(hmx(k), mx(k) + had);
32     sm(k) += c(k) * ad + (r - l + 1 - c(k)) * ad2,
    ↪ mx(k) += ad, ad(k) += ad, ad2(k) += ad2,
    ↪ mx2(k) += ad2;
33 }
34 void pushdown(int k, int l, int r) {
35     int mx = max(mx[ls], mx[rs]);
36     if (mx[ls] == mx) add(ls, l, mid, ad(k), ad2(k),
    ↪ had(k), had2(k));
37     else add(ls, l, mid, ad2(k), ad2(k), had2(k),
    ↪ had2(k));
38     if (mx[rs] == mx) add(rs, mid + 1, r, ad(k),
    ↪ ad2(k), had(k), had2(k));
39     else add(rs, mid + 1, r, ad2(k), ad2(k),
    ↪ had2(k), had2(k));
40     ad(k) = ad2(k) = had(k) = had2(k) = 0;
41 }
42 void build(int k, int l, int r) {
43     if (l == r) return nod[k] = {a[l], a[l], -inf,
    ↪ 1, a[l]}, void();
44     build(ls, l, mid), build(rs, mid + 1, r);
45     pushup(k);
46 }
47 void add(int k, int l, int r, int x, int y, int v) {
48     if (x <= l && r <= y) return add(k, l, r, v, v,
    ↪ v, v);
49     pushdown(k, l, r);
50     if (x <= mid) add(ls, l, mid, x, y, v);
51     if (y > mid) add(rs, mid + 1, r, x, y, v);
52     pushup(k);
53 }
54 void Min(int k, int l, int r, int x, int y, int v) {
55     if (v >= mx(k)) return;
56     if (x <= l && r <= y && v > mx2(k)) return
    ↪ add(k, l, r, v - mx(k), 0, v - mx(k), 0),
    ↪ void();
57     pushdown(k, l, r);
58     if (x <= mid) Min(ls, l, mid, x, y, v);
59     if (y > mid) Min(rs, mid + 1, r, x, y, v);
60     pushup(k);
61 }
62 node query(int k, int l, int r, int x, int y) {
63     if (x <= l && r <= y) return nod[k];
64     pushdown(k, l, r);
65     if (y <= mid) return query(ls, l, mid, x, y);
66     else if (x > mid) return query(rs, mid + 1, r,
    ↪ x, y);
67     else return query(ls, l, mid, x, y) + query(rs,
    ↪ mid + 1, r, x, y);
68 }
69 } smt;

```

3.9 树分治 (点分治)

```

1 /*** (静态) 点分治 ***/
2 void Find1(int x, int fa)
3 {
4     siz[x] = 1, subsiz++;
5     for (int i = hea[x]; i; i = nex[i]) if (!used[ver[i]] &&
    ↪ ver[i] != fa)
6         Find1(ver[i], x), siz[x] += siz[ver[i]];
7 }
8 void Find2(int x, int fa)
9 {
10     bool isrt = true;
11     for (int i = hea[x]; i; i = nex[i]) if (!used[ver[i]] &&
    ↪ ver[i] != fa)
12     {
13         Find2(ver[i], x);

```

```

14         if ((siz[ver[i]] << 1) > subsiz) isrt = false;
15     }
16     if (((subsiz - siz[x]) << 1) > subsiz) isrt = false;
17     if (isrt) rt = x;
18 }
19 void solve(int x)
20 {
21     subsiz = 0, Find1(x, 0), Find2(x, 0);
22     // 处理和 rt 有关的答案
23     used[rt] = true;
24     for (int i = hea[rt]; i; i = nex[i]) if (!used[ver[i]])
    ↪ solve(ver[i]);
25 }

```

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。

```

1 int siz[N], rt, tot, dfa[N], mxp[N], vis[N], mxd;
2 // rt: 当前重心, tot: 联通块大小, dfa: 点分树父亲, mxp: 最
    ↪ 大子树大小, mxd: 最大深度。
3 void find(int x, int fa) {
4     siz[x] = 1, mxp[x] = 0;
5     for (int i = head[x]; i; i = e[i].next)
6     {
7         if (e[i].b != fa && !vis[e[i].b])
8         {
9             find(e[i].b, x);
10            siz[x] += siz[e[i].b];
11            mxp[x] = max(mxp[x], siz[e[i].b]);
12        }
13    }
14    mxp[x] = max(mxp[x], tot - siz[x]);
15    rt = mxp[rt] > mxp[x] ? x : rt;
16 }
17 void get_dis(int x, int fa, int dep) {
18     mxd = max(mxd, dep);
19     for (int i = head[x]; i; i = e[i].next)
20         if (e[i].b != fa && !vis[e[i].b])
21             get_dis(e[i].b, x, dep + 1);
22 }
23 // bit[x][0]: 存储了以 x 为重心时 x 的子树内的信息。
24 // bit[x][1]: 存储了以 dfa[x] 为重心时 x 的子树内的信息。
25 void divide(int x, int lim) {
26     vis[x] = 1, mxd = 0, get_dis(x, 0, 0);
27     bit[x][0].build(mxd);
28     if (dfa[x]) bit[x][1].build(lim);
29     for (int i = head[x]; i; i = e[i].next) {
30         if (!vis[e[i].b]) {
31             rt = 0, tot = siz[e[i].b] < siz[x] ?
    ↪ siz[e[i].b] : lim - siz[x];
32             find(e[i].b, x);
33             dfa[rt] = x, divide(rt, tot);
34         }
35     }
36 }
37 void modify(int ct, int v) {
38     int x = ct;
39     while (x) {
40         bit[x][0].add(dis(x, ct), v); // dis(x, y): 返回
    ↪ x 到 y 的距离。
41         if (dfa[x]) bit[x][1].add(dis(dfa[x], ct), v);
42         x = dfa[x];
43     }
44 } // 将点 ct 的值加上 v。
45 int query(int ct, int k) {
46     int x = ct, res = 0, lst = 0;
47     while (x) {
48         int d = dis(x, ct);

```



```
49         if (d <= k) {
50             res += bit[x][0].query(k - d);
51             if (lst) res -= bit[lst][1].query(k - d);
52         }
53         lst = x, x = dfa[x];
54     }
55     return res;
56 }
57 void init() {
58     // dis 的预处理。
59     mxp[rt = 0] = tot = n, find(1, 0), divide(rt, tot);
60 }
```

4 字符串

4.1 Hash 类

```
1 const ll P = (1ll)1e18+9;
2 static constexpr u128 inv = [](){
3     u128 ret = P;
4     for(int i=0;i<6;i++)ret*=2-ret*P;
5     return ret;
6 }();
7 constexpr u128 chk = u128(-1) / P;
8 bool check(i128 a,i128 b){
9     if(a<b)swap(a,b);
10    return (a-b)*inv<=chk;
11 }
```

4.2 后缀数组

```
1 // 0-下标, h[i] 表示后缀 sa[i] 和 sa[i+1] 的 LCP 长度
2 struct SA {
3     vector<int> sa, rk, h;
4     SA(const string &s) {
5         int n = s.length();
6         sa.resize(n); rk.resize(n); h.resize(n - 1);
7         iota(sa.begin(), sa.end(), 0);
8         sort(sa.begin(), sa.end(), [&](int a, int b){return
9             s[a]<s[b];});
10        rk[sa[0]] = 0;
11        for(int i=1;i<n;i++)
12            rk[sa[i]]=rk[sa[i-1]]+(s[sa[i]]!=s[sa[i-1]]);
13        int k = 1;
14        vector<int> t, c(n); t.reserve(n);
15        while(rk[sa[n - 1]] < n - 1) {
16            t.clear();
17            for(int i=0;i<k;i++) t.push_back(n-k+i);
18            for(auto i:sa)if(i>=k) t.push_back(i-k);
19            fill(c.begin(), c.end(), 0);
20            for(int i=0;i<n;i++) c[rk[i]]++;
21            for(int i=1;i<n;i++) c[i]+=c[i-1];
22            for(int i=n-1;i>=0;i--) sa[--c[rk[t[i]]]]=t[i];
23            swap(rk, t);
24            rk[sa[0]] = 0;
25            for (int i=1;i<n;i++){
26                rk[sa[i]]=rk[sa[i-1]];
27                if(t[sa[i-1]]<t[sa[i]] || sa[i-1]+k==n ||
28                    t[sa[i-1]+k]<t[sa[i]+k])
29                    rk[sa[i]]++;
30            }
31            k *= 2;
32        }
33        for(int i=0,j=0;i<n;i++){
34            if(rk[i]==0) j=0;
35            else{
36                if(j) j--;
37                while(i+j<n && sa[rk[i]-1]+j<n &&
38                    s[sa[rk[i]-1]+j]==s[sa[rk[i]-1]+j])j++;
39                h[rk[i]-1] = j;
40            }
41        }
42    }
43 }
```

4.3 AC 自动机

```
1 // 记得修改字符集大小和字符种类
2 const int SIGMA = 26;
3 const char BASE = 'a';
4 int ch[N][SIGMA], fail[N], cnt[N], tot=1;
5 void ins(const string& s){
6     int u=1;
7     for(int i=0;i<s.size();i++){
8         int c=s[i]-BASE;
```

```
9         if(!ch[u][c])ch[u][c]=++tot;
10        u=ch[u][c];
11    }
12    cnt[u]++;
13 }
14 void get_fail(){
15     for(int i=0;i<SIGMA;i++)ch[0][i]=1;
16     fail[1]=0;
17     queue<int> q;q.push(1);
18     while(q.size()){
19         int u=q.front();q.pop();
20         for(int c=0;c<SIGMA;c++){
21             int v=ch[u][c];
22             if(!v)ch[u][c]=ch[fail[u]][c];
23             else{
24                 fail[v]=ch[fail[u]][c];
25                 q.push(v);
26             }
27         }
28     }
29 }
```

4.4 回文自动机

```
1 struct PAM {
2     // tot: 字符串长度, cnt: 节点个数
3     int fail[N], ch[N][26], len[N], s[N], tot, cnt, lst;
4     // fail : 当前节点的最长回文后缀。
5     // ch : 在当前节点的前后添加字符, 得到的回文串。
6     PAM() {
7         len[0] = 0, len[1] = -1, fail[0] = 1;
8         tot = lst = 0, cnt = 1, s[0] = -1;
9     }
10    int get_fail(int x) {
11        while (s[tot-1-len[x]] != s[tot]) x = fail[x];
12        return x;
13    }
14    void insert(char c) {
15        s[++tot] = c - 'a';
16        int p = get_fail(lst);
17        if(!ch[p][s[tot]]) {
18            len[++cnt] = len[p] + 2;
19            int t = get_fail(fail[p]);
20            fail[cnt] = ch[t][s[tot]];
21            ch[p][s[tot]] = cnt;
22        }
23        lst=ch[p][s[tot]];
24    }
25 }
```

4.5 Manacher 算法

```
1 string s;
2
3 string t="#~#";
4 for(int i=0;i<n;i++)t+=s[i], t+='#';
5 vector<int> d(t.size());
6 // r 是最远的右边加 1
7 // 每个位置在 [i-d[i]+1, i+d[i]-1] 是回文的
8 for(int i=1,r=0,mid=0;i<t.size();i++){
9     d[i] = r>i ? min(d[2*mid-i], r-i):1;
10    while(t[i+d[i]]==t[i-d[i]])d[i]++;
11    if(i+d[i]>r)r=i+d[i],mid=i;
12 }
```

4.6 KMP 算法与 border 理论

```
1 for(int i=2,j=0;i<=m;i++){
2     while(j && t[j+1]!=t[i])j=nxt[j];
3     if(t[j+1]==t[i])j++;
```

```

4     nxt[i]=j;
5 }
6
7 for(int i=1,j=0;i<=n;i++){
8     while(j && t[j+1]!=s[i])j=nxt[j];
9     if(t[j+1] == s[i])j++;
10    if(j==m)cout<<i-m+1<<'\n';
11 }

```

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 $p+q \leq n$ ，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 $2m-n \geq p$ ，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 $p+q-\gcd(p, q) \leq n$ ，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 $2|T| \geq n$ ，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。

```

1 // 下标从 0 开始
2 // z[i] 表示 s[i..len-1] 与 s 最长公共前缀长度
3 vector<int> z_function(const string& s) {
4     int n = s.length();
5     vector<int> z(n);
6     for(int i=1,l=0,r=0;i<n;i++) {
7         if(i<=r && z[i-l]<r-i+1)z[i]=z[i-l];
8         else{
9             z[i] = max(0,r-i+1);
10            while(i+z[i]<n && s[z[i]]==s[i+z[i]])++z[i];
11        }
12        if(i+z[i]-1>r)l=i,r=i+z[i]-1;
13    }
14    return z;
15 }

```

4.8 后缀自动机

```

1 struct SAM{ // 注意修改字符集！字符集是小写字母吗？
2     int last = 1, tot = 1;
3     int ch[N<<1][26], len[N<<1], f[N<<1];
4     void ins(char c){
5         c -= 'a';
6         int p = last, cur = last = ++tot;
7         len[cur]=len[p]+1;
8         for(;p&&!ch[p][c];p=f[p])ch[p][c]=cur;
9         if(!p){f[cur]=1;return;}
10        int q=ch[p][c];
11        if(len[q]==len[p]+1){f[cur]=q;return;}
12        int clone=++tot;
13        for(int i=0;i<26;i++)ch[clone][i]=ch[q][i];
14        f[clone]=f[q],len[clone]=len[p]+1;
15        f[q]=f[cur]=clone;

```

```

16        for(;p&&ch[p][c]==q;p=f[p])ch[p][c]=clone;
17    }
18 }sam; // 注意任何跟 SAM 有关的数组都要开两倍

```

4.9 后缀自动机 (map 版)

```

1 struct SAM{ // 注意修改字符集！字符集是小写字母吗？
2     int last = 1, tot = 1;
3     map<int,int> ch[N<<1];
4     int len[N<<1], f[N<<1];
5     void ins(char c){
6         c -= 'a';
7         int p = last, cur = last = ++tot;
8         len[cur]=len[p]+1;
9         for(;p&&!ch[p].count(c);p=f[p])ch[p][c]=cur;
10        if(!p){f[cur]=1;return;}
11        int q=ch[p][c];
12        if(len[q]==len[p]+1){f[cur]=q;return;}
13        int clone=++tot;
14        ch[clone]=ch[q];
15        f[clone]=f[q],len[clone]=len[p]+1;
16        f[q]=f[cur]=clone;
17        for(;p&&ch[p].count(c)&&ch[p][c]==q;p=f[p])
18            ch[p][c]=clone;
19    }
20 }sam; // 注意任何跟 SAM 有关的数组都要开两倍

```

4.10 最小表示法

```

1 // 0-下标的！
2 int minpos(const string& s){
3     int n = s.size();
4     int k = 0, i = 0, j = 1;
5     while (k < n && i < n && j < n) {
6         if (s[(i+k)%n]==s[(j+k)%n])++k;
7         else{
8             if(s[(i+k)%n]>s[(j+k)%n]) i+=k+1;
9             else j+=k+1;
10            k = 0;
11            if(i==j)i++;
12        }
13    }
14    return min(i,j);
15 }

```

5 线性代数

5.1 高斯消元

```

1 scanf("%d", &n);
2 for (int i = 1; i <= n; i++)
3     for (int j = 1; j <= n + 1; j++)
4         scanf("%lf", &a[i][j]);
5 for (int i = 1, Max = 1; i <= n; Max = ++i) {
6     for (int s = i + 1; s <= n; s++)
7         if (fabs(a[s][i]) > fabs(a[Max][i]))
8             Max = s; // 找出绝对值最大的
9     for (int j = 1; j <= n + 1; j++)
10        swap(a[i][j], a[Max][j]);
11    if (a[i][i] < 10e-8 && a[i][i] > -10e-8) {
12        p = false;
13        break;
14    } // 记得 double 的精度问题
15    for (int s = 1; s <= n; s++)
16        if (s != i) // 这样省去了第二步处理的麻烦
17        {
18            double tmp = 0 - (a[s][i] / a[i][i]);
19            a[s][i] = 0;
20            for (int j = i + 1; j <= n + 1; j++)
21                a[s][j] += tmp * a[i][j];
22        }
23 }
24 if (p)
25     for (int i = 1; i <= n; i++)
26         printf("%.21f\n", a[i][n + 1] / a[i][i]);
27 else
28     printf("No Solution\n");

```

5.2 线性基

```

1 void Insert(ll x) // 加入一个数
2 {
3     for (int i = 62; i >= 0; i--)
4         if ((x >> i) & 1) {
5             if (!p[i]) {
6                 p[i] = x;
7                 break;
8             }
9             x ^= p[i]; // 更新 [0,i-1] 位的更优答案
10        }
11 }
12 bool exist(ll x) // 查询一个元素是否可以被异或出来
13 {
14     for (int i = 62; i >= 0; i--)
15         if ((x >> i) & 1) x ^= p[i];
16     return x == 0;
17 }
18 ll query_max() // 查询异或最大值
19 {
20     ll ret = 0;
21     for (int i = 62; i >= 0; i--)
22         if ((ans ^ p[i]) > ans) ans ^= p[i];
23     return ans;
24 }
25 ll query_min() // 查询异或最小值
26 {
27     for (int i = 0; i <= 62; i++)
28         if (p[i]) return p[i];
29     return 0;
30 }
31 ll kth(ll k) // 查询异或第 k 小
32 {
33     // 重建 d 数组, 求出哪些位可以被异或为 1
34     // 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
35     for (int i = 62; i >= 1; i--) // 从高到低防止后效性
36         for (int j = i - 1; j >= 0; j--)
37             if ((p[i] >> j) & 1) p[i] ^= p[j];
38     for (int i = 0; i <= 62; i++)

```

```

39     if (p[i]) d[cnt++] = p[i];
40
41     if (!k) return 0ll; // 特判 0
42     if (k >= (1ll << (1ll)cnt)) return -1ll; // k 大于可
43     // 以表示出的数的个数
44     ll ret = 0;
45     for (int i = 62; i >= 0; i--)
46         if ((k >> i) & 1) ret ^= d[i];
47     return ret;
48 }
49 ll Rank(ll k) {
50     ll Now = 1, ret = 0;
51     for (int i = 0; i <= 62; i++)
52         if (p[i]) // 记得保证 k 可以被异或出来
53         {
54             if ((k >> i) & 1) ret += Now;
55             Now <<= 1;
56         }
57     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:

```

1 // 解法 1: 类似辗转相除
2 n = rd(), mod = rd();
3 for (int i = 1; i <= n; i++)
4     for (int j = 1; j <= n; j++)
5         a[i][j] = rd() % mod;
6 for (int i = 1; i <= n; i++) {
7     for (int j = i + 1; j <= n; j++) {
8         while (a[j][i]) {
9             ll tmp = a[i][i] / a[j][i];
10            for (int k = i; k <= n; k++)
11                a[i][k] = (a[i][k] - tmp * a[j][k] % mod
12                // + mod) % mod;
13            swap(a[i], a[j]), w = -w; // ?
14        }
15    }
16    for (int i = 1; i <= n; i++)
17        ans = ans * a[i][i] % mod;
18    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:

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

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

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:

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

在标准型的基础上得到松弛型:

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

单纯形法以松弛型为基础。具体的, 松弛型隐含了一个基本解, 即 $x = 0$, $\alpha = b$ (这里要求 $b \geq 0$)。我们称 α 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。

```

1 typedef double db;
2 const db eps = 1e-8, inf = 1e9;
3 int n, m, ans;
4 db b[N], a[N][M], c[M]; // n: 约束个数. m: 变量个数。
5 bool dcmp(db x) { return fabs(x) > eps; }
6
7 db pivot(int out, int in) {
8     b[out] /= a[out][in];
9     for (int i = 1; i <= m; i++) if (i != in) a[out][i]
10         ⇨ /= a[out][in];
11     a[out][in] = 1; // 理论上是 1 / a[out][in], 但这个系
12     ⇨ 数可以任取, 但也不要随便取。
13     for (int i = 1; i <= n; i++) {
14         if (i != out && dcmp(a[i][in])) {
15             b[i] -= a[i][in] * b[out];
16             for (int j = 1; j <= m; j++)
17                 if (j != in) a[i][j] -= a[i][in] *
18                     ⇨ a[out][j];
19             a[i][in] *= -a[out][in];
20         }
21     }
22     db res = c[in] * b[out];
23     for (int i = 1; i <= m; i++) if (i != in) c[i] -=
24         ⇨ a[out][i] * c[in];
25     c[in] *= -a[out][in];

```

```

22     return res;
23 }
24 void simplex() {
25     db res = 0;
26     while (1) {
27         int in = 0;
28         for (int i = 1; i <= m; i++) {
29             if (c[i] > eps) {
30                 in = i;
31                 break;
32             }
33         }
34         if (!in) break; // simplex 完成, 找到最优解。
35         int out = 0;
36         db mn = inf;
37         for (int i = 1; i <= n; i++)
38             if (a[i][in] > eps && b[i] / a[i][in] < mn)
39                 ⇨ mn = b[i] / a[i][in], out = i;
40         if (!out) {
41             res = inf;
42             break;
43         } // 解为无穷大。
44         res += pivot(out, in);
45     }
46     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:

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

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:

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

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 $O(nV)$ 多重背包 (单调队列优化)

```

1 int ql,qr;
2 struct QUE
3 {
4     int num,val;
5 }que[Maxv];
6 void many_pack(int c,int w,int m)
7 {
8     if(!c) { add+=m*w; return; }
9     m=min(m,V/c);
10    for(int pos=0;s;pos<c;pos++)
11    {
12        ql=1,qr=0,s=(V-pos)/c;
13        for(int j=0;j<=s;j++)
14        {
15            while(ql<=qr &&
16                ↪ que[qr].val<=(dp[pos+j*c]-j*w)) qr--;
17            que[++qr]=(QUE){j,dp[pos+j*c]-j*w};
18            while(ql<=qr && (j-que[ql].num)>m) ql++;
19            ↪ dp[pos+j*c]=max(dp[pos+j*c],que[ql].val+j*w);
20        }
21    }
22 }

```

6.2 二项式反演

$$f(n) = \sum_{i=n}^m \binom{i}{n} g(i) \iff g(n) = \sum_{i=n}^m \binom{i}{n} (-1)^{i-n} f(i)$$

$f(n)$ 是钦定有 n 个满足, 其他随便的方案数, 而 $g(n)$ 是恰好有 n 个满足的方案数。

6.3 FFT

```

1 struct comp {
2     double x, y;
3     comp(double X = 0, double Y = 0) { x = X, y = Y; }
4     comp operator + (const comp &rhs) const { return
5         ↪ comp(x + rhs.x, y + rhs.y); }
6     comp operator - (const comp &rhs) const { return
7         ↪ comp(x - rhs.x, y - rhs.y); }
8     comp operator * (const comp &rhs) const { return
9         ↪ comp(x * rhs.x - y * rhs.y, x * rhs.y + y *
10            ↪ rhs.x); }
11    comp operator / (const comp &rhs) const {
12        double t = rhs.x * rhs.x + rhs.y * rhs.y;
13        return {(x * rhs.x + y * rhs.y) / t, (y * rhs.x
14            ↪ - x * rhs.y) / t};
15    }
16 };
17 const double pi = acos(-1);
18 int lim, tr[N];
19 void adjust(int n) { // n: 多项式的次数。
20     lim = 1;
21     while (lim <= n) lim <= 1;
22     for (int i = 0; i < lim; i++) tr[i] = (tr[i] >> 1) >>
23         ↪ 1 | ((i & 1) ? lim >> 1 : 0);
24 } // 准备蝶形变换。
25 void fft(comp *f, int op) { // op: 1 为 dft, -1 为 idft
26     ↪ 。
27     for (int i = 0; i < lim; i++) if (tr[i] < i)
28         ↪ swap(f[tr[i]], f[i]);
29     for (int l = 1; l < lim; l <= 1) {
30         comp w1(cos(2 * pi / (l << 1)), sin(2 * pi / (l
31             ↪ << 1)) * op);
32         for (int i = 0; i < lim; i += l << 1) {

```

```

24         comp w(1, 0);
25         for (int j = i; j < i + l; j++, w = w * w1)
26             ↪ {
27                 comp x = f[j], y = f[j + l] * w;
28                 f[j] = x + y, f[j + l] = x - y;
29             }
30     }
31     if (op == -1) {
32         for (int i = 0; i < lim; i++) f[i].x /= lim;
33     }
34 }

```

6.4 NTT

```

1 vector<int> omega[25];
2 // n 是 DFT 的最大长度, 如果有 2 个长为 m 的多项式相乘, n
3   ↪ 需要 >= 2m
4 void ntt_init(int n){
5     for(int k=2,d=0;k<=n;k*=2,d++){
6         omega[d].resize(k+1);
7         int wn=qpow(3,(p-1)/k),tmp=1;
8         for(int i=0;i<=k;i++){
9             omega[d][i]=tmp;
10            tmp=(LL)tmp*wn%p;
11        }
12    }
13 }
14 // 传入的必须在 [0,p) 范围内, 不能有负的
15 // 否则要把 d==16 改成 d%8==0 之类, 多取几次模
16 void ntt(int *c,int n,int tp){
17     static ULL a[N];
18     for(int i=0;i<n;i++)a[i]=c[i];
19     for(int i=1,j=0;i<n-1;i++){
20         int k=n;
21         do j^=(k>>=1); while(j<k);
22         if(i<j)swap(a[i],a[j]);
23     }
24     for(int k=1,d=0;k<n;k*=2,d++){
25         if(d==16)for(int i=0;i<n;i++)a[i]%=p;
26         for(int i=0;i<n;i+=k*2){
27             for(int j=0;j<k;j++){
28                 int w = omega[d][tp>0 ? j : k*2-j];
29                 ULL u = a[i+j], v = w*a[i+j+k]%p;
30                 a[i+j]=u+v;
31                 a[i+j+k]=u-v+p;
32             }
33         }
34     }
35     if(tp>0){for(int i=0;i<n;i++)c[i]=a[i]%p;}
36     else{
37         int inv = qpow(n, p-2);
38         for(int i=0;i<n;i++)c[i]=a[i]*inv%p;
39     }
40 }

```

6.5 拉格朗日插值

$$f(x) = \sum_i f(x_i) \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

6.6 集合幂级数

6.6.1 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。

6.7 多项式全家桶

```

1 void fmt(int *f, int op) { // op: 1 为 dft, -1 为 idft。
    ↪
2     for (int l = 1; l < lim; l <= 1) {
3         for (int i = 0; i < lim; i += l << 1) {
4             for (int j = i; j < i + l; j++) {
5                 f[j + l] += f[j] * op;
6             }
7         }
8     }
9 }

```

而集合交卷积则对应后缀和。

```

1 void fmt(int *f, int op) { // op: 1 为 dft, -1 为 idft。
    ↪
2     for (int l = 1; l < lim; l <= 1) {
3         for (int i = 0; i < lim; i += l << 1) {
4             for (int j = i; j < i + l; j++) {
5                 f[j] += f[j + l] * op;
6             }
7         }
8     }
9 }

```

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。

```

1 void multi(int *f, int *g, int *h) { // 对 f 和 g 做子集
    ↪ 卷积, 答案为 h。
2     static int a[n][lim], b[n][lim], c[n][lim]; // lim =
    ↪ 1 << n
3     memset(a, 0, sizeof a);
4     memset(b, 0, sizeof b);
5     memset(c, 0, sizeof c);
6     for (int i = 0; i < lim; i++) a[pcnt[i]][i] = f[i];
    ↪ // pcnt[i] = popcount(i)
7     for (int i = 0; i < lim; i++) b[pcnt[i]][i] = g[i];
8     for (int i = 0; i <= n; i++) fmt(a[i], 1), fmt(b[i],
    ↪ -1);
9     for (int i = 0; i <= n; i++)
10        for (int j = 0; j <= i; j++)
11            for (int k = 0; k < lim; k++)
12                c[i][k] += a[j][k] * b[i - j][k];
13     for (int i = 0; i <= n; i++) fmt(c[i], -1);
14     for (int i = 0; i < lim; i++) h[i] = c[pcnt[i]][i];
15 }

```

特别的,子集卷积等价于 n 元保留到一次项的线性卷积。

6.6.2 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 and n 元 FFT 没有什么区别。

```

1 void fwt(int *f, int op) { // op: 1 为 dft, -1 为 idft。
    ↪
2     for (int l = 1; l < lim; l <= 1) {
3         for (int i = 0; i < lim; i += l << 1) {
4             for (int j = i; j < i + l; j++) {
5                 int x = f[j], y = f[j + l];
6                 f[j] = x + y, f[j + l] = x - y;
7                 if (op == -1) f[j] /= 2, f[j + l] /= 2;
    ↪ // 模意义下改成乘逆元。
8             }
9         }
10     }
11 }
12 }

```

```

1 const int N=5000005;
2 const long long mod=998244353;
3 #define int long long
4 namespace poly{
5
6 const long long G=3, iG=332748118;
7 int tr[N], lim;
8 int qpow(int a, int b=mod-2){
9     int res=1;
10    while (b){
11        if (b & 1) (res*=a)%=mod;
12        (a*=a)%=mod, b >>= 1;
13    }
14    return res;
15 }
16 void adjust(int n){
17     lim=1;
18     while (lim <= n) lim <= 1;
19     for(int i=0; i<lim; i++)
20         tr[i]=(tr[i >> 1] >> 1) | ((i & 1) ? lim >> 1 : 0);
21 } // 准备蝶形变换
22 void copy(int *f, int *g, int n){
23     for(int i=0; i <= n; i++) f[i]=g[i];
24 void clear(int *f, int n){
25     for(int i=0; i <= n; i++) f[i]=0;
26 void erase(int *f, int l, int r){
27     for(int i=l; i <= r; i++) f[i]=0;
28 void reverse(int *f, int n){
29     for(int i=0; i <= n/2; i++) swap(f[i], f[n-i]);
30 void integral(int *f, int n){
31     for(int i=n+1; i >= 1; i--) f[i]=f[i-1]*qpow(i) % mod;
32     f[0]=0;
33 } // 对 n 次多项式 f 求积分, 默认常数项为 0。
34 void dif(int *f, int n){
35     for(int i=0; i<n; i++) f[i]=f[i+1]*(i+1) % mod;
36     f[n]=0;
37 } // 对 n 次多项式 f 求导。
38 void ntt(int *f, int op){ // op: 1 为 dft, -1 为 idft。
39     for(int i=0; i<lim; i++)
40         if (tr[i]<i) swap(f[tr[i]], f[i]);
41     for(int l=1; l<lim; l <= 1){
42         int w1=qpow(op == 1 ? G : iG, (mod-1)/(l << 1));
43         for(int i=0; i<lim; i += l << 1){
44             for(int j=i, w=1; j<i+l; j++, (w*=w1)%=mod){
45                 int x=f[j], y=f[j+l]*w % mod;
46                 f[j]=(x+y) % mod, f[j+l]=(x-y) % mod;
47             }
48         }
49     }
50     if (op == -1){
51         int iv=qpow(lim); // 算逆元。
52         for(int i=0; i<lim; i++) (f[i]*=iv)%=mod;
53     }
54 }
55 void multiply(int *h, int *f, int n, int *g, int m){
56     static int a[N], b[N];
57     copy(a, f, n), copy(b, g, m);
58     adjust(n+m);
59     ntt(a, 1), ntt(b, 1);
60     for(int i=0; i<lim; i++) h[i]=a[i]*b[i] % mod;
61     ntt(h, -1);
62     clear(a, lim-1), clear(b, lim-1);
63 } // 计算 f 与 g 的积, 存放在 h 中, f 与 g 不变。
64 void inverse(int *f, int *g, int n){
65     static int t[N];
66     if (!n) return f[0]=qpow(g[0]), void();
67     inverse(f, g, n >> 1);
68     adjust(2*n), copy(t, g, n);
69     ntt(t, 1), ntt(f, 1);
70     for(int i=0; i<lim; i++)
71         f[i]=f[i]*(2-f[i]*t[i] % mod) % mod;
72     ntt(f, -1), erase(f, n+1, lim-1), clear(t, lim-1);
73 } // 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, 要求 f 都是 0。

```



```

74 void ln(int *f, int *g, int n){
75     static int t[N];
76     copy(t, g, n), inverse(f, g, n), dif(t, n);
77     adjust(n*2), ntt(t, 1), ntt(f, 1);
78     for(int i=0; i<lim; i++) (f[i]*=t[i])%=mod;
79     ntt(f, -1), integral(f, n);
80     erase(f, n+1, lim-1), clear(t, lim-1);
81 } // 要求 g[0]=1。
82 void exp(int *f, int *g, int n){
83     static int t[N];
84     if (!n) return f[0]=1, void();
85     exp(f, g, n >> 1), ln(t, f, n);
86     for(int i=0; i <= n; i++) t[i]=(g[i]-t[i]) % mod;
87     t[0]++;
88     adjust(n*2), ntt(f, 1), ntt(t, 1);
89     for(int i=0; i<lim; i++) (f[i]*=t[i])%=mod;
90     ntt(f, -1), clear(t, lim-1), erase(f, n+1, lim-1);
91 } // 要求 g[0]=0。
92 void pow(int *f, int *g, int n, int k){
93     static int t[N];
94     ln(t, g, n);
95     for(int i=0; i <= n; i++) (t[i]*=k) % mod;
96     exp(f, t, n);
97 } // 要求 g[0]=1。
98 void divide(int *q, int *r, int *f, int n, int *g, int
    ↪ m){
99     static int a[N], b[N], c[N];
100     copy(a, f, n), copy(b, g, m);
101     reverse(a, n), reverse(b, m);
102     inverse(c, b, n-m);
103     multiply(q, a, n-m, c, n-m);
104     reverse(q, n-m);
105     multiply(a, g, m, q, n-m);
106     for(int i=0; i<m; i++) r[i]=(f[i]-a[i]+mod) % mod;
107 } // 多项式带余除法, 其中 q 为商, r 为余数。
108 }
109 using namespace poly;

```

7 数论

7.1 exgcd

```
1 ll exgcd(ll a, ll b, ll& x, ll& y) {
2     if (!b) { x = 1, y = 0; return a; }
3     ll g = exgcd(b, a % b, y, x);
4     y -= a / b * x;
5     return g;
6 }
```

7.2 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 $(0, 0), (\frac{n}{a}, 0), (0, \frac{n}{b})$ 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 $O(\log \max(a, b, c))$ 的时间内解决此类问题。

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

```
1 ll solve(ll a, ll b, ll c, ll n) {
2     if (a == 0) return (n + 1) * (b / c) % mod;
3     if (a >= c || b >= c)
4         return (n * (n + 1) % mod * inv2 % mod * (a / c)
5             ↪ % mod + (n + 1) * (b / c) % mod + solve(a % c, b % c, c, n)) % mod;
6     ll m = (a * n + b) / c;
7     return (n * (m % mod) % mod - solve(c, c - b - 1, a,
8         ↪ m - 1) + mod) % mod;
9 }
```

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：

```
1 struct Euclid {
2     ll f, g, h;
3 };
4 Euclid solve(ll a, ll b, ll c, ll n) {
5     Euclid ans, tmp;
6     if (a == 0) {
7         ans.f = (n + 1) * (b / c) % mod;
8         ans.g = n * (n + 1) % mod * (b / c) % mod * inv2
9             ↪ % mod;
10        ans.h = (n + 1) * (b / c) % mod * (b / c) % mod;
11        return ans;
12    }
13    if (a >= c || b >= c) {
14        tmp = solve(a % c, b % c, c, n);
15        ans.f = (n * (n + 1) % mod * inv2 % mod * (a / c)
16            ↪ % mod + (n + 1) * (b / c) % mod + tmp.f) % mod;
17        ans.g = (n * (n + 1) % mod * (2 * n + 1) % mod *
18            ↪ (a / c) % mod * inv6 % mod + n * (n + 1) % mod *
19            ↪ mod * (b / c) % mod * inv2 % mod + tmp.g) % mod;
20        ans.h = ((a / c) * (a / c) % mod * n % mod * (n
21            ↪ + 1) % mod * (n * 2 + 1) % mod * inv6 % mod
22            ↪ +
23            ↪ (b / c) * (b / c) % mod * (n + 1) % mod
24            ↪ + (a / c) * (b / c) % mod * n % mod
25            ↪ + (n + 1) % mod *
26            ↪ 2 * (b / c) % mod * tmp.f % mod + 2 *
27            ↪ (a / c) % mod * tmp.g % mod +
28            ↪ tmp.h) % mod;
29    }
```

```
19        mod;
20        return ans;
21    }
22    ll m = (a * n + b) / c;
23    tmp = solve(c, c - b - 1, a, m - 1);
24    ans.f = (n * (m % mod) % mod + mod - tmp.f) % mod;
25    ans.g = (n * (m % mod) % mod * (n + 1) % mod - tmp.f
26        ↪ - tmp.f) % mod * inv2 % mod;
27    ans.h = (n * (m % mod) % mod * (m + 1) % mod - 2 *
28        ↪ tmp.g - 2 * tmp.f - ans.f) % mod;
29    return ans;
30 }
31 ans.f = (ans.f % mod + mod) % mod;
32 ans.g = (ans.g % mod + mod) % mod;
33 ans.h = (ans.h % mod + mod) % mod;
```

7.3 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。

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

7.4 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 $\text{lcm}(m_1, m_2)$ ，上式仍然成立，即当前的解是在 $\text{lcm}(m_1, m_2)$ 意义下的。这样我们得出新的同余式：

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

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。

7.5 BSGS

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

```
1 inline int BSGS(int a, int mod, int b) {
2     int sq = ceil(sqrt(mod));
3     mp.clear();
4     int powa = 1;
5     for (int B = 0, x = 1; B <= sq; B++)
```

```

6      x = 1ll * b * powa % mod, mp[x] = B,
7      powa = 1ll * powa * ((B == sq) ? 1 : a) % mod;
8      for (int A = 0, B, x = 1; A <= sq; A++) {
9          if (mp.find(x) != mp.end()) {
10             B = mp[x];
11             if (A * sq - B >= 0)
12                 return A * sq - B;
13         }
14         x = 1ll * x * powa % mod;
15     }
16     return -1; // 无解
17 }

```

7.6 扩展 BSGS

不要求 a, p 互质。

```

1 inline int inv(int a, int mod) {
2     int x, y;
3     exgcd(a, mod, x, y);
4     return (x % mod + mod) % mod;
5 }
6 inline int exBSGS(int a, int mod, int b) {
7     b %= mod;
8     if (b == 1 || mod == 1)
9         return 0;
10    int k = 0, D = 1, U = 1, tmod = mod, tb = b;
11    for (int x; (x = gcd(tmod, a)) > 1;) {
12        if (tb % x)
13            return -1;
14        tmod /= x, tb /= x, D *= x, k++;
15        U = 1ll * U * a / x % tmod;
16        if (U == tb)
17            return k;
18    }
19    tb = 1ll * tb * inv(U, tmod) % tmod;
20    int ret = BSGS(a, tmod, tb);
21    if (ret == -1)
22        return -1;
23    return ret + k;
24 }

```

7.7 Lucas 定理

```

1 ll C(ll x, ll y) {
2     if (x < y)
3         return 0;
4     if (x < mod && y < mod)
5         return mi[x] * invmi[y] % mod * invmi[x - y] %
6             mod;
7     return C(x / mod, y / mod) * C(x % mod, y % mod) %
8         mod;
9 } // mod 为一素数。

```

7.8 扩展 Lucas 定理

```

1 // 代码是题解里拉的
2 #include <bits/stdc++.h>
3 #define ll long long
4 using namespace std;
5 #ifndef Fading
6 inline char gc() {
7     static char now[1 << 16], *S, *T;
8     if (T == S) {
9         T = (S = now) + fread(now, 1, 1 << 16, stdin);
10        if (T == S)
11            return EOF;
12    }
13    return *S++;
14 }

```

```

15 #endif
16 #ifndef Fading
17 #define gc getchar
18 #endif
19 void exgcd(ll a, ll b, ll& x, ll& y) {
20     if (!b)
21         return (void)(x = 1, y = 0);
22     exgcd(b, a % b, x, y);
23     ll tmp = x;
24     x = y;
25     y = tmp - a / b * y;
26 }
27 ll gcd(ll a, ll b) {
28     if (b == 0)
29         return a;
30     return gcd(b, a % b);
31 }
32 inline ll INV(ll a, ll p) {
33     ll x, y;
34     exgcd(a, p, x, y);
35     return (x + p) % p;
36 }
37 inline ll lcm(ll a, ll b) {
38     return a / gcd(a, b) * b;
39 }
40 inline ll mabs(ll x) {
41     return (x > 0 ? x : -x);
42 }
43 inline ll fast_mul(ll a, ll b, ll p) {
44     ll t = 0;
45     a %= p;
46     b %= p;
47     while (b) {
48         if (b & 1LL)
49             t = (t + a) % p;
50         b >>= 1LL;
51         a = (a + a) % p;
52     }
53     return t;
54 }
55 inline ll fast_pow(ll a, ll b, ll p) {
56     ll t = 1;
57     a %= p;
58     while (b) {
59         if (b & 1LL)
60             t = (t * a) % p;
61         b >>= 1LL;
62         a = (a * a) % p;
63     }
64     return t;
65 }
66 inline ll read() {
67     ll x = 0, f = 1;
68     char ch = gc();
69     while (!isdigit(ch)) {
70         if (ch == '-')
71             f = -1;
72         ch = gc();
73     }
74     while (isdigit(ch))
75         x = x * 10 + ch - '0', ch = gc();
76     return x * f;
77 }
78 inline ll F(ll n, ll P, ll PK) {
79     if (n == 0)
80         return 1;
81     ll rou = 1; // 循环节
82     ll rem = 1; // 余项
83     for (ll i = 1; i <= PK; i++) {
84         if (i % P)
85             rou = rou * i % PK;
86     }
87     rou = fast_pow(rou, n / PK, PK);
88     for (ll i = PK * (n / PK); i <= n; i++) {
89         if (i % P)
90             rem = rem * (i % PK) % PK;
91     }
92 }

```

```

91     }
92     return F(n / P, P, PK) * rou % PK * rem % PK;
93 }
94 inline ll G(ll n, ll P) {
95     if (n < P)
96         return 0;
97     return G(n / P, P) + (n / P);
98 }
99 inline ll C_PK(ll n, ll m, ll P, ll PK) {
100     ll fz = F(n, P, PK), fm1 = INV(F(m, P, PK), PK),
101     fm2 = INV(F(n - m, P, PK), PK);
102     ll mi = fast_pow(P, G(n, P) - G(m, P) - G(n - m, P),
103     ↪ PK);
104     return fz * fm1 % PK * fm2 % PK * mi % PK;
105 }
106 ll A[1001], B[1001];
107 // x=B(mod A)
108 inline ll exLucas(ll n, ll m, ll P) {
109     ll ljc = P, tot = 0;
110     for (ll tmp = 2; tmp * tmp <= P; tmp++) {
111         if (!(ljc % tmp)) {
112             ll PK = 1;
113             while (!(ljc % tmp)) {
114                 PK *= tmp;
115                 ljc /= tmp;
116             }
117             A[++tot] = PK;
118             B[tot] = C_PK(n, m, tmp, PK);
119         }
120     }
121     if (ljc != 1) {
122         A[++tot] = ljc;
123         B[tot] = C_PK(n, m, ljc, ljc);
124     }
125     ll ans = 0;
126     for (ll i = 1; i <= tot; i++) {
127         ll M = P / A[i], T = INV(M, A[i]);
128         ans = (ans + B[i] * M % P * T % P) % P;
129     }
130     return ans;
131 }
132 signed main() {
133     ll n = read(), m = read(), P = read();
134     printf("%lld\n", exLucas(n, m, P));
135     return 0;
136 }

```

7.9 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:

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

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 $O(n^{\frac{2}{3}})$ 。

7.10 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。

第一步

求 $g(n) = \sum_{p \leq n} f(p)$ 。

设 $g(n, j) = \sum_{i=1}^n [i \text{ 是质数} \vee i \text{ 的最小质因子} > p_j] f(i)$ 。

因此, 能得到递推式:

$$g(n, j) = g(n, j-1) - f(p_j) \left(g\left(\left\lfloor \frac{n}{p_j} \right\rfloor, j-1\right) - g(p_{j-1}, j-1) \right)$$

第二步

做法一 设 $S(n, j) = \sum_{i=2}^n [i \text{ 的最小质因子} > p_j] f(i)$ 。

最后答案就是 $S(n, 0) + f(1)$ 。

把 $S(n, j)$ 分成质数部分和合数部分进行计算。

$$\begin{aligned}
 S(n, j) &= g(n) - g(p_j) \\
 &+ \sum_{j < k, p_k \leq \sqrt{n}, p_k^{e+1} \leq n} \left(f(p_k^e) S\left(\left\lfloor \frac{n}{p_k^e} \right\rfloor, k\right) - f(p_k^{e+1}) \right)
 \end{aligned}$$

然后暴力递归计算。

做法二 设 $S(n, j) = \sum_{i=2}^n [i \text{ 是质数} \vee i \text{ 的最小质因子} > p_j] f(i)$ 。

写出递推式:

$$\begin{aligned}
 S(n, j) &= S(n, j+1) \\
 &+ \sum_{p_{j+1} \leq \sqrt{n}, p_{j+1}^{e+1} \leq n} f(p_{j+1}^e) \left(S\left(\left\lfloor \frac{n}{p_{j+1}^e} \right\rfloor, j+1\right) - g(p_{j+1}) \right) \\
 &+ f(p_{j+1}^{e+1})
 \end{aligned}$$

和第一步中一样, 滚动数组 dp 即可, **边界条件** $S(x, \cdot) = g(x)$ 。

注意到做法二可以求出所有 $S(\lfloor \frac{n}{x} \rfloor)$, 这是做法一无法完成的。

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:

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 using ll=long long;
5
6 const ll mod = 1e9+7;
7 constexpr ll ksm(ll a,ll b){
8     ll r = 1;
9     while(b){
10         if(b&1)r=r*a%mod;
11         a=a*a%mod;
12         b>>=1;
13     }

```

```

14     return r;
15 }
16 constexpr ll inv2 = ksm(2, mod-2);
17 constexpr ll inv6 = ksm(6, mod-2);
18
19 constexpr ll sq(ll n){n%=mod;return n*n%mod;}
20 constexpr ll sum(ll n){n%=mod;return
    ↪ n*(n+1)%mod*inv2%mod;}
21 constexpr ll sum_sq(ll n){n%=mod;return
    ↪ n*(n+1)%mod*(2*n+1)%mod*inv6%mod;}
22
23 const int N = 2e5+5;
24
25 ll p[N],tot; bool vis[N];
26 void init_sieve(){
27     for(int i=2;i<N;i++){
28         if(!vis[i])p[++tot]=i;
29         for(int j=1;j<=tot && i*p[j]<N;j++){
30             vis[i*p[j]]=1;
31             if(i%p[j]==0)break;
32         }
33     }
34 }
35
36 ll v[N],m;
37 int id1[N],id2[N];
38 ll g1[N],g2[N];
39 ll n;
40 ll get(ll x){
41     return x<N?id1[x]:id2[n/x];
42 }
43
44 void init_sqrt(){
45     for(ll l=1,r;l<=n;l=r+1){
46         r=n/(n/l);
47         v[++m]=n/l;
48         if(v[m]<N)id1[v[m]]=m;
49         else id2[n/v[m]]=m;
50         g1[m]=(sum_sq(v[m])+mod-1)%mod,
            ↪ g2[m]=(sum(v[m])+mod-1)%mod;
51     }
52 }
53
54 void calc_g(){
55     for(int j=1;j<=tot;j++){
56         for(int i=1;i<=m && p[j]<=v[i]/p[j];i++){
57             g1[i] = (g1[i] -
                ↪ sq(p[j])*g1[get(v[i]/p[j])])%mod +
                ↪ sq(p[j])*g1[get(p[j-1])] + mod) % mod;
58             g2[i] = (g2[i] - p[j]*g2[get(v[i]/p[j])])%mod
                ↪ + p[j]*g2[get(p[j-1])] + mod) % mod;
59         }
60     }
61 }
62
63 ll f(ll x){x%=mod;return (x*x%mod-x+mod)%mod;}
64
65 /* 做法一部分 */
66 ll S(ll x,int j){
67     if(p[j] >= x) return 0;
68     ll res = (g1[get(x)] - g2[get(x)] - g1[get(p[j])]) +
        ↪ g2[get(p[j])] + mod + mod) % mod;
69     for(int k=j+1;k<=tot && p[k]<=x/p[j];k++){
70         ll w = p[k];
71         for(;w<=x/p[k];w*=p[k]){
72             res = (res + f(w) * S(x/w, k))%mod +
                ↪ f(w*p[k]))%mod;
73         }
74     }
75     return res;
76 }
77 /* 做法一部分 */
78
79 /* 做法二部分 */
80 ll g[N],S[N];
81 /* 做法二部分 */
82

```

```

83 int main(){
84     ios::sync_with_stdio(0);cin.tie(0);
85     cin>>n;
86     init_sieve();
87     init_sqrt();
88     calc_g();
89
90     /* 做法一部分 */
91     cout << ((S(n, 0) + 1)%mod+mod)%mod << '\n';
92     /* 做法一部分 */
93
94     /* 做法二部分 */
95     for(int
        ↪ i=1;i<=m;i++)S[i]=g[i]=(g1[i]-g2[i]+mod)%mod;
96     for(int j=tot-1;j>=0;j--){
97         for(int i=1;i<=m && p[j+1]<=v[i]/p[j+1];i++){
98             for(ll w=p[j+1];w<=v[i]/p[j+1];w*=p[j+1]){
99                 S[i] = (S[i] + f(w) * (S[get(v[i]/w)] -
                    ↪ g[get(p[j+1]]) + mod) % mod +
                    ↪ f(w*p[j+1]))%mod;
100             }
101         }
102     }
103     cout << S[get(n)]+1 << '\n';
104     /* 做法二部分 */
105     return 0;
106 }

```

7.11 没有精度问题的整除

Remark: $y < 0$ 需将 x, y 都取反才能得到期望结果。

```

1 int ceil(int x, int y) {
2     return (x>0 ? (x+y-1)/y : x/y);
3 }
4 int floor(int x, int y) {
5     return (x>0 ? x/y : (x-y+1)/y);
6 }

```

8 计算几何

8.1 声明与宏

```

1 #define cp const P&
2 #define cl const L&
3 #define cc const C&
4 #define vp vector<P>
5 #define cvp const vector<P>&
6 #define D double
7 #define LD long double
8 std::mt19937 rnd(time(0));
9 const LD eps = 1e-12;
10 const LD pi = std::numbers::pi;
11 const LD INF = 1e9;
12 int sgn(LD x) {
13     return x > eps ? 1 : (x < -eps ? -1 : 0);
14 }

```

8.2 点与向量

```

1 struct P {
2     LD x, y;
3
4     P(const LD &x = 0, const LD &y = 0) : x(x), y(y) {}
5     P(cp a) : x(a.x), y(a.y) {}
6
7     P operator=(cp a) {
8         x = a.x, y = a.y;
9         return *this;
10    }
11    P rot(LD t) const {
12        LD c = cos(t), s = sin(t);
13        return P(x * c - y * s, x * s + y * c);
14    } // counterclockwise
15    P rot90() const { return P(-y, x); } //
16    ↪ counterclockwise
17    P _rot90() const { return P(y, -x); } // clockwise
18    LD len() const { return sqrtl(x * x + y * y); }
19    LD len2() const { return x * x + y * y; }
20    P unit() const {
21        LD d = len();
22        return P(x / d, y / d);
23    }
24    void read() { scanf("%Lf%Lf", &x, &y); }
25    void print() const { printf("%.9Lf, %.9Lf", x, y); }
26    ↪ }
27 };
28
29 bool operator<(cp a, cp b) { return a.x == b.x ? a.y <
30     ↪ b.y : a.x < b.x; }
31 bool operator>(cp a, cp b) { return a.x == b.x ? a.y >
32     ↪ b.y : a.x > b.x; }
33 bool operator==(cp a, cp b) { return !sgn(a.x - b.x) &&
34     ↪ !sgn(a.y - b.y); }
35 P operator+(cp a, cp b) { return P(a.x + b.x, a.y +
36     ↪ b.y); }
37 P operator-(cp a, cp b) { return P(a.x - b.x, a.y -
38     ↪ b.y); }
39 P operator*(const LD &a, cp b) { return P(a * b.x, a *
40     ↪ b.y); }
41 P operator*(cp b, const LD &a) { return P(a * b.x, a *
42     ↪ b.y); }
43 P operator/(cp b, const LD &a) { return P(b.x / a, b.y /
44     ↪ a); }
45 LD operator*(cp a, cp b) { return a.x * b.x + a.y * b.y;
46     ↪ }
47     ↪ // 点积
48 LD operator^(cp a, cp b) { return a.x * b.y - a.y * b.x;
49     ↪ }
50     ↪ // 叉积
51 LD rad(cp a, cp b) { return acosl((a.x == 0 && a.y == 0
52     ↪ || b.x == 0 && b.y == 0) ? 0 : (a * b / a.len() /
53     ↪ b.len())); } // 夹角 其中有零向量时返回 0

```

```

38 bool left(cp a, cp b) { return sgn(a ^ b) > 0; }
    ↪ // 没什么用

```

8.3 线

```

1 struct L {
2     P s, t;
3
4     L(cp a, cp b) : s(a), t(b) {}
5     L(cl l) : s(l.s), t(l.t) {}
6     void read() { s.read(), t.read(); }
7 };
8
9 bool point_on_line(cp a, cl l) { return sgn((a - l.s) ^
10     ↪ (l.t - l.s)) == 0; }
11 bool point_on_segment(cp a, cl l) { return sgn((a - l.s)
12     ↪ ^ (l.t - l.s)) == 0 && sgn((a - l.s) * (a - l.t)) <=
13     ↪ 0; }
14 bool two_side(cp a, cp b, cl l) { return sgn((a - l.s) ^
15     ↪ (a - l.t)) * sgn((b - l.s) ^ (b - l.t)) < 0; }
16 bool intersection_judge_strict(cl a, cl b) { return
17     ↪ two_side(a.s, a.t, b) && two_side(b.s, b.t, a); }
18 bool intersection_judge(cl a, cl b) { return
19     ↪ point_on_segment(a.s, b) || point_on_segment(a.t, b)
20     ↪ || point_on_segment(b.s, a) || point_on_segment(b.t,
21     ↪ a) || intersection_judge_strict(a, b); }
22 P ll_intersection(cl a, cl b) {
23     LD s1 = (a.t - a.s) ^ (b.s - a.s), s2 = (a.t - a.s)
24     ↪ ^ (b.t - a.s);
25     return (s2 * b.s - s1 * b.t) / (s2 - s1);
26 }
27 bool point_on_ray(cp a, cl b) { return sgn((a - b.s) ^
28     ↪ (b.t - b.s)) == 0 && sgn((a - b.s) * (b.t - b.s)) >=
29     ↪ 0; }
30 bool ray_intersection_judge(cl a, cl b) {
31     P p(ll_intersection(a, b));
32     return sgn((p - a.s) * (a.t - a.s)) >= 0 && sgn((p -
33     ↪ b.s) * (b.t - b.s)) >= 0;
34 } // 似乎有更快的做法, 但是看不懂
35 LD point_to_line(cp a, cl b) { return fabs((b.t - b.s) ^
36     ↪ (a - b.s)) / (b.t - b.s).len(); } // 距离
37 P project_to_line(cp a, cl b) { return b.s + (a - b.s) *
38     ↪ (b.t - b.s) / (b.t - b.s).len2() * (b.t - b.s); } //
39     ↪ 垂足
40 LD point_to_segment(cp a, cl b) {
41     if (b.s == b.t) return (a - b.s).len();
42     if (sgn((a - b.s) * (b.t - b.s)) * sgn((a - b.t) *
43     ↪ (b.t - b.s)) <= 0)
44         return fabs((b.t - b.s) ^ (a - b.s)) / (b.t -
45     ↪ b.s).len();
46     return min((a - b.s).len(), (a - b.t).len());
47 }

```

8.4 圆

```

1 struct C {
2     P c;
3     LD r;
4
5     C(cp c, const LD &r = 0) : c(c), r(r) {}
6     C(cc a) : c(a.c), r(a.r) {};
7     C(cp a, cp b) {
8         c = (a + b) / 2;
9         r = (a - c).len();
10    }
11    C(cp x, cp y, cp z) { // 注意三点共线
12        P p(y - x), q(z - x);
13        P s(p * p / 2, q * q / 2);
14        LD d = p ^ q;
15        p = P(s ^ P(p.y, q.y), P(p.x, q.x) ^ s) / d;
16        c = x + p, r = p.len();
17    }

```

```

18 };
19
20 bool in_circle(cp a, cc b) { return sgn((b.c - a).len()
    ↪ - b.r) <= 0; }
21 C min_circle(vp p) // 0(n)
22 {
23     shuffle(p.begin(), p.end(), rnd);
24     int n = p.size(), i, j, k;
25     C ret(p[0], 0);
26     for (i = 1; i < n; ++i)
27         if (!in_circle(p[i], ret))
28             for (ret = C(p[i], p[0]), j = 1; j < i; ++j)
29                 if (!in_circle(p[j], ret))
30                     for (ret = C(p[i], p[j]), k = 0; k <
    ↪ j; ++k)
31                         if (!in_circle(p[k], ret))
32                             ret = C(p[i], p[j], p[k]);
33     return ret;
34 }
35
36 vp lc_intersection(cl l, cc c) {
37     LD x = point_to_line(c.c, l);
38     if (sgn(x - c.r) > 0) return vp();
39     x = sqrtl(c.r * c.r - x * x);
40     P p(project_to_line(c.c, l));
41     if (sgn(x) == 0) return vp({p});
42     return vp({p + x * (l.t - l.s).unit(), p - x * (l.t
    ↪ - l.s).unit()});
43 }
44
45 LD cc_intersection_area(cc a, cc b) {
46     LD d = (a.c - b.c).len();
47     if (sgn(d - (a.r + b.r)) >= 0) return 0;
48     if (sgn(d - fabs(a.r - b.r)) <= 0) {
49         LD r = min(a.r, b.r);
50         return pi * r * r;
51     }
52     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)));
53     return a.r * a.r * t1 + b.r * b.r * t2 - d * a.r *
    ↪ sinl(t1);
54 }
55
56 vp cc_intersection(cc a, cc b) {
57     LD d = (a.c - b.c).len();
58     if (a.c == b.c || sgn(d - a.r - b.r) > 0 || sgn(d -
    ↪ fabs(a.r - b.r)) < 0) return vp();
59     P r = (b.c - a.c).unit();
60     LD x = (d * d + a.r * a.r - b.r * b.r) / (d * 2);
61     LD h = sqrtl(a.r * a.r - x * x);
62     if (sgn(h) == 0) return vp({a.c + r * x});
63     return vp({a.c + r * x + r.rot90() * h, a.c + r * x
    ↪ - r.rot90() * h});
64 }
65
66 vp tangent(cp a, cc b) { return cc_intersection(b, C(a,
    ↪ b.c)); } // 点到圆的切点
67
68 vector<L> tangent(cc a, cc b, int f) // f = 1
    ↪ extangent; f = -1 intangent
69 {
70     D d = (b.c - a.c).len();
71     int sg = sgn(fabs((b.r - f * a.r) / d) - 1);
72     if (sg == 1)
73         return {};
74     else if (sg == 0) {
75         P p = a.c - sgn(b.r - f * a.r) * f * a.r * (b.c
    ↪ - a.c).unit();
76         return {p, p + (b.c - a.c).rot90()};
77     } // 内切/外切
78     D theta = asin(min((D)1, max((D)-1, f * (b.r - f *
    ↪ a.r) / d)));
79     P du = (b.c - a.c).unit(), du1 = du.rot(theta + pi /
    ↪ 2), du2 = du.rot(-theta - pi / 2);
80     return {{a.c + a.r * du1, b.c + f * b.r * du1}, {a.c
    ↪ + a.r * du2, b.c + f * b.r * du2}};

```

```

81 }

```

8.5 凸包

```

1 vp convex(vp a) {
2     if (a.size() < 2) return a;
3     sort(a.begin(), a.end());
4     int n = a.size(), cnt = 0;
5     vp con({p[0]});
6     for (int i = 0; i < n; ++i) {
7         while (cnt > 0 && sgn(((p[i] - con[cnt - 1]) ^
    ↪ (con[cnt] - con[cnt - 1]))) > 0) //>=
8             --cnt, con.pop_back();
9         ++cnt, con.push_back(p[i]);
10    }
11    int fixed = cnt;
12    for (int i = n - 2; ~i; --i) {
13        while (cnt > fixed && sgn(((p[i] - con[cnt - 1])
    ↪ ^ (con[cnt] - con[cnt - 1]))) > 0) //>=
14            --cnt, con.pop_back();
15        ++cnt, con.push_back(p[i]);
16    }
17    con.pop_back();
18    return con;
19 }
20
21 vp minkowski(vp p1, vp p2) {
22     if (p1.empty() || p2.empty()) return vp();
23     int n1 = p1.size(), n2 = p2.size();
24     vp ret;
25     if (n1 == 1) {
26         for (int i = 0; i < n2; ++i)
27             ret.push_back(p1[0] + p2[i]);
28         return ret;
29     }
30     if (n2 == 1) {
31         for (int i = 0; i < n1; ++i)
32             ret.push_back(p1[i] + p2[0]);
33         return ret;
34     }
35     p1.push_back(p1[0]), p1.push_back(p1[1]),
    ↪ p2.push_back(p2[0]), p2.push_back(p2[1]);
36     ret.push_back(p1[0] + p2[0]);
37     int i1 = 0, i2 = 0;
38     while (i1 < n1 || i2 < n2) {
39         if (((p1[i1 + 1] - p1[i1]) ^ (p2[i2 + 1] -
    ↪ p2[i2])) > 0)
40             ret.push_back(p1[++i1] + p2[i2]);
41         else
42             ret.push_back(p2[++i2] + p1[i1]);
43     } //
    ↪ p1.pop_back(), p1.pop_back(), p2.pop_back(), p2.pop_back();
44     ret.pop_back();
45     return ret;
46 }
47
48 struct Con {
49     int n;
50     vp a, upper, lower;
51
52     Con(cvp _a) : a(_a) {
53         n = a.size();
54         int k = 0;
55         for (int i = 1; i < n; ++i)
56             if (a[i] > a[k]) k = i;
57         for (int i = 0; i <= k; ++i)
58             lower.push_back(a[i]);
59         for (int i = k; i < n; ++i)
60             upper.push_back(a[i]);
61         upper.push_back(a[0]);
62     }
63 }
64 }
65

```



```

66 // Below is from Nemesis
67
68 // Convex
69 int n;
70 vector<point> a; // 可以封装成一个 struct
71 bool inside(cp u) { // 点在凸包内
72     int l = 1, r = n - 1;
73     while (l < r) {
74         int mid = (l + r + 1) / 2;
75         if (turn(a[0], a[mid], u) >= 0)
76             l = mid;
77         else
78             r = mid - 1;
79     }
80     return turn(a[0], a[l], u) >= 0 && turn(a[l], a[l + 1], u) >= 0 && turn(a[l + 1], a[0], u) >= 0;
81 }
82 int search(auto f) { // 凸包求极值, 需要 C++17
83     int l = 0, r = n - 1;
84     int d = f(a[r], a[l]) ? (swap(l, r), -1) : 1;
85     while (d * (r - l) > 1) {
86         int mid = (l + r) / 2;
87         if (f(a[mid], a[l]) && f(a[mid], a[mid - d]))
88             l = mid;
89         else
90             r = mid;
91     }
92     return l;
93 }
94 pair<int, int> get_tan(cp u) { // 求切线
95     return // 严格在凸包外; 需要边界上时, 特判 a[n-1] ->
96         {search([&](cp x, cp y) { return turn(u, y, x) >
97             0; }},
98         search([&](cp x, cp y) { return turn(u, x, y) >
99             0; }));
100 }
101 point at(int i) { return a[i % n]; }
102 int inter(cp u, cp v, int l, int r) {
103     int sl = turn(u, v, at(l));
104     while (l + 1 < r) {
105         int m = (l + r) / 2;
106         if (sl == turn(u, v, at(m)))
107             l = m;
108         else
109             r = m;
110     }
111     return l % n;
112 }
113 bool get_inter(cp u, cp v, int &i, int &j) { // 求直线交
114     点
115     int p0 = search([&](cp x, cp y) { return det(v - u,
116         x - u) < det(v - u, y - u); }},
117     p1 = search([&](cp x, cp y) { return det(v - u,
118         x - u) > det(v - u, y - u); }));
119     if (turn(u, v, a[p0]) * turn(u, v, a[p1]) < 0) {
120         if (p0 > p1) swap(p0, p1);
121         i = inter(u, v, p0, p1);
122         j = inter(u, v, p1, p0 + n);
123         return true;
124     } else
125         return false;
126 }
127 LD near(cp u, int l, int r) {
128     if (l > r) r += n;
129     int sl = sgn(dot(u - at(l), at(l + 1) - at(l)));
130     LD ret = p2s(u, {at(l), at(l + 1)});
131     while (l + 1 < r) {
132         int m = (l + r) / 2;
133         if (sl == sgn(dot(u - at(m), at(m + 1) -
134             at(m))))
135             l = m;
136         else
137             r = m;
138     }
139     return min(ret, p2s(u, {at(l), at(l + 1)}));
140 }

```

```

135 LD get_near(cp u) { // 求凸包外点到凸包最近点
136     if (inside(u)) return 0;
137     auto [x, y] = get_tan(u);
138     return min(near(u, x, y), near(u, y, x));
139 }
140
141 // Dynamic convex hull
142 struct hull { // upper hull, left to right
143     set<point> a;
144     LL tot;
145     hull() { tot = 0; }
146     LL calc(auto it) {
147         auto u = it == a.begin() ? a.end() : prev(it);
148         auto v = next(it);
149         LL ret = 0;
150         if (u != a.end()) ret += det(*u, *it);
151         if (v != a.end()) ret += det(*it, *v);
152         if (u != a.end() && v != a.end()) ret -= det(*u,
153             *v);
154         return ret;
155     }
156     void insert(point p) {
157         if (!a.size()) {
158             a.insert(p);
159             return;
160         }
161         auto it = a.lower_bound(p);
162         bool out;
163         if (it == a.begin())
164             out = (p < *it); // special case
165         else if (it == a.end())
166             out = true;
167         else
168             out = turn(*prev(it), *it, p) > 0;
169         if (!out) return;
170         while (it != a.begin()) {
171             auto o = prev(it);
172             if (o == a.begin() || turn(*prev(o), *o, p)
173                 < 0)
174                 break;
175             else
176                 erase(o);
177         }
178         while (it != a.end()) {
179             auto o = next(it);
180             if (o == a.end() || turn(p, *it, *o) < 0)
181                 break;
182             else
183                 erase(it), it = o;
184         }
185         tot += calc(a.insert(p).first);
186     }
187     void erase(auto it) {
188         tot -= calc(it);
189         a.erase(it);
190     }
191 };

```

8.6 三角形

```

1 // From Nemesis
2 point incenter(cp a, cp b, cp c) { // 内心
3     double p = dis(a, b) + dis(b, c) + dis(c, a);
4     return (a * dis(b, c) + b * dis(c, a) + c * dis(a,
5         b)) / p;
6 }
7 point circumcenter(cp a, cp b, cp c) { // 外心
8     point p = b - a, q = c - a, s(dot(p, p) / 2, dot(q,
9         q) / 2);
10     double d = det(p, q);
11     return a + point(det(s, {p.y, q.y}), det({p.x, q.x},
12         s)) / d;
13 }
14 point orthocenter(cp a, cp b, cp c) { // 重心

```

```

12     return a + b + c - circumcenter(a, b, c) * 2.0;
13 }
14 point ferma_point(cp a, cp b, cp c) { // 费马点
15     if (a == b) return a;
16     if (b == c) return b;
17     if (c == a) return c;
18     double ab = dis(a, b), bc = dis(b, c), ca = dis(c,
19         ↪ a);
20     double cosa = dot(b - a, c - a) / ab / ca;
21     double cosb = dot(a - b, c - b) / ab / bc;
22     double cosc = dot(b - c, a - c) / ca / bc;
23     double sq3 = PI / 3.0;
24     point mid;
25     if (sgn(cosa + 0.5) < 0)
26         mid = a;
27     else if (sgn(cosb + 0.5) < 0)
28         mid = b;
29     else if (sgn(cosc + 0.5) < 0)
30         mid = c;
31     else if (sgn(det(b - a, c - a)) < 0)
32         mid = line_inter({a, b + (c - b).rot(sq3)}, {b,
33             ↪ c + (a - c).rot(sq3)});
34     else
35         mid = line_inter({a, c + (b - c).rot(sq3)}, {c,
36             ↪ b + (a - b).rot(sq3)});
37     return mid;
38 } // minimize(|A-x|+|B-x|+|C-x|)

```

8.7 多边形

```

1 vp Poly_cut(vp p, cl l) // 直线切多边形, 返回一侧的图形端
2   ↪ 点, 0(n)
3 {
4     if (p.empty()) return vp();
5     vp ret;
6     int n = p.size();
7     p = push_back(p[0]);
8     for (int i = 0; i < n; ++i) {
9         if (((p[i] - l.s) ^ (l.t - l.s)) <= 0)
10             ret.push_back(p[i]);
11         if (two_size(p[i], p[i + 1], l))
12             ret.push_back(l.intersection(l, L(p[i], p[i
13                 ↪ + 1])));
14     } // p.pop_back();
15     return ret;
16 }
17 LD Poly_area(vp p) {
18     LD ar = 0;
19     int n = p.size();
20     for (int i = 0, j = n - 1; i < n; j = i++)
21         ar += p[i] ^ p[j];
22     return fabs(ar) / 2;
23 }
24 // Below is from Nemesis
25
26 // 多边形与圆交
27 LD angle(cp u, cp v) {
28     return 2 * asin(dis(u.unit(), v.unit()) / 2);
29 }
30 LD area(cp s, cp t, LD r) { // 2 * area
31     LD theta = angle(s, t);
32     LD dis = p2s({0, 0}, {s, t});
33     if (sgn(dis - r) >= 0) return theta * r * r;
34     auto [u, v] = line_circle_inter({s, t}, {{0, 0},
35         ↪ r});
36     point lo = sgn(det(s, u)) >= 0 ? u : s;
37     point hi = sgn(det(v, t)) >= 0 ? v : t;
38     return det(lo, hi) + (theta - angle(lo, hi)) * r *
39         ↪ r;
40 }
41 LD solve(vector<point> &p, cc c) {
42     LD ret = 0;
43     for (int i = 0; i < (int)p.size(); ++i) {

```

```

42     auto u = p[i] - c.c;
43     auto v = p[(i + 1) % p.size()] - c.c;
44     int s = sgn(det(u, v));
45     if (s > 0)
46         ret += area(u, v, c.r);
47     else if (s < 0)
48         ret -= area(v, u, c.r);
49     }
50     return abs(ret) / 2;
51 } // ret 在 p 逆时针时为正

```

8.8 半平面交

```

1 // From Nemesis
2 int half(cp a) { return a.y > 0 || (a.y == 0 && a.x > 0)
3   ↪ ? 1 : 0; }
4 bool turn_left(cl a, cl b, cl c) {
5     return turn(a.s, a.t, line_inter(b, c)) > 0;
6 }
7 bool is_para(cl a, cl b) { return !sgn(det(a.t - a.s,
8     ↪ b.t - b.s)); }
9 bool cmp(cl a, cl b) {
10     int sign = half(a.t - a.s) - half(b.t - b.s);
11     int dir = sgn(det(a.t - a.s, b.t - b.s));
12     if (!dir && !sign)
13         return turn(a.s, a.t, b.t) < 0;
14     else
15         return sign ? sign > 0 : dir > 0;
16 }
17 vector<point> hpi(vector<line> h) { // 半平面交
18     sort(h.begin(), h.end(), cmp);
19     vector<line> q(h.size());
20     int l = 0, r = -1;
21     for (auto &i : h) {
22         while (l < r && !turn_left(i, q[r - 1], q[r]))
23             --r;
24         while (l < r && !turn_left(i, q[l], q[l + 1]))
25             ++l;
26         if (l <= r && is_para(i, q[r])) continue;
27         q[++r] = i;
28     }
29     while (r - l > 1 && !turn_left(q[l], q[r - 1],
30         ↪ q[r]))
31         --r;
32     while (r - l > 1 && !turn_left(q[r], q[l], q[l +
33         ↪ 1]))
34         ++l;
35     if (r - l < 2) return {};
36     vector<point> ret(r - l + 1);
37     for (int i = l; i <= r; i++)
38         ret[i - l] = line_inter(q[i], q[i == r ? l : i +
39             ↪ 1]);
40     return ret;
41 }
42 // 空集会在队列里留下一个开区间; 开区间会被判定为空集。
43 // 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。
44 ↪
45 // 实在需要零面积区域边缘, 需要仔细考虑 turn_left 的实现。

```

9 其它工具

9.1 编译命令

```
1 g++ X.cpp -Wall -Wshadow -Wextra -Wconversion -O2
  ↳ -fsanitize=undefined -fsanitize=address X
2 # -fsanitize=undefined: 检测未定义行为
3 # -fsanitize=address: 检测内存溢出
```

9.2 快读

```
1 char *p1, *p2, buf[100000];
2 #define gc() (p1 == p2 && (p2 = (p1 = buf) + fread(buf,
  ↳ 1, 100000, stdin), p1 == p2) ? EOF : *p1++)
3 int read() {
4     int x = 0, f = 1;
5     char ch = gc();
6     while (!isdigit(ch)) { if (ch == '-') f = -1; ch =
  ↳ gc(); }
7     while (isdigit(ch)) x = x * 10 + ch - '0', ch =
  ↳ gc();
8     return x * f;
9 }
```

9.3 Python Hints

itertools 库

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

random 库

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

列表操作

```
1 # 列表操作
2 l = sample(range(100000), 10)
3 l.sort() # 原地排序
4 l.sort(key=lambda x:x%10) # 按末尾排序
5 from functools import cmp_to_key
6 l.sort(key=cmp_to_key(lambda x,y:y-x)) # 比较函数, 小于返
  ↳ 回负数
7 sorted(l) # 非原地排序
8 l.reverse() reversed(l)
```

字典操作

```
1 from collections import defaultdict
2 # 提供一个函数返回缺省值
3 d = defaultdict(list)
4 d["a"].append(2)
5 d["a"].append(3)
6 d["b"].append(4)
7 print(d) # {'a': [2, 3], 'b': [4]}
8 # 用 lambda 可以快速构造出需要的默认值
9 d = defaultdict(lambda: 2)
10 # 遍历键值对
11 for k,v in d.items():
12     print(k, v)
```

复数

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

高精度小数

```
1 from decimal import Decimal, getcontext, FloatOperation,
  ↳ ROUND_HALF_EVEN
2 getcontext().prec = 100 # 设置有效位数
3 getcontext().rounding = getattr(ROUND_HALF_EVEN) # 四舍
  ↳ 六入五成双
4 getcontext().traps[FloatOperation] = True # 禁止 float
  ↳ 混合运算
5 a = Decimal("114514.1919810")
6 print(a, f"{a:.2f}")
7 a.ln() a.log10() a.sqrt() a**2
```

记忆化搜索

```
1 from functools import cache
2 # 记忆化搜索, 还可以记忆化元组, 只要参数满足 Hashable 即可
3 @cache
4 def fib(n):
5     if n<=2:
6         return 1
7     return fib(n-1)+fib(n-2)
```

9.4 对拍器

```
1 import os
2
3 while True:
4     os.system("python3 data.py > in")
5     os.system("/usr/bin/time -f 'test Time=%es' ./test <
  ↳ in > out")
6     os.system("/usr/bin/time -f 'std Time=%es' ./std <
  ↳ in > ans")
7     if os.system("diff out ans >/dev/null"):
8         print("WA")
9         exit(1)
10    print("AC")
```

9.5 常数表

| n | $\log_{10} n$ | $n!$ | $C(n, n/2)$ | $\text{lcm}(1\dots n)$ |
|-----|---------------|-----------|-------------|------------------------|
| 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 |

| $n \leq$ | 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 |
| $n \leq$ | 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 |
| $n \leq$ | 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 |
| $\pi(n)$ | $\pi(x) \sim x/\ln(x)$ | | | | | |

9.6 试机赛

- 检查身份证件: 护照、学生证、胸牌以及现场所需通行证。
- 确认什么东西能带进场。特别注意: 手机、智能手表、金属(钥匙)等等。
- 测试鼠标、键盘、显示器和座椅。如果有问题, 立刻联系工作人员。
- 测试编译器版本。

```
- #include<bits/stdc++.h>
- 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`
- 测试本地性能与提交性能

```
1 const int N=1<<20;
2 for(int T=4;T;T--){
3     int a[N],b[N],c[N];
4     for(int i=0;i<N;i++)a[i]=i,b[i]=N-i;
5     ntt_init(N);ntt(a,N,1);ntt(b,N,1);
6     for(int i=0;i<N;i++)c[i]=a[i]*b[i]; // 注意不
        ↳ 要取模谢谢
7     ntt(c,N,-1);
8     assert(c[0]==0xad223d);
9 }
10 // 本地: 291 ms
11 // Q0J: 340+-40 ms
12 // CF: 1000+-100 ms
```

- 测试 `make` 命令。

9.7 Bashrc

用 `code ~/.bashrc` 编辑。

```
1 export CXXFLAGS='-g -Wall -Wextra -Wconversion -Wshadow
   ↳ -std=gnu++20 -fsanitize=undefined'
2 ulimit -s 1048576
```

之后 `source ~/.bashrc`