The template

for ACM ICPC

诶，我和你讲这比赛超休闲的

Edited by

heyuhhh

Team 万物静默如谜 at Sicau

2019.10

NANCHONG

目录

第一章　图论

1.1　最短路

1.1.1　堆优化 Dijkstra

```
void Dijkstra(int s){
    priority_queue <node> q;
  memset(d, INF, sizeof(d));
    memset(vis, 0, sizeof(vis)); d[s]=0;
    q.push(node{0, s});
    while(!q.empty()){
        node cur = q.top(); q.pop();
        int u = cur.u;
        if(vis[u]) continue ;
        vis[cur.u] = 1;
        for(int i = head[u]; i != -1; i = e[i].next){
            int v = e[i].v;
            if(d[v] > d[u] + e[i].w){
                d[v] = d[u] + e[i].w;
                q.push(node{d[v], v});
            }
        }
    }
}
```

1.1.2　spfa

```
//c:判断有无负环
int spfa(int s){
    queue <int> q;
    memset(d, INF, sizeof(d));
    memset(vis, 0, sizeof(vis));
    memset(c, 0, sizeof(c));
    q.push(s);
    vis[s] = 1; d[s] = 0; c[s] = 1;
    while(!q.empty()){
        int u = q.front(); q.pop(); vis[u]=0;
```

```
        if(c[u] > n){
            return -1;
        }
        for(int i = head[u];i != -1;i = e[i].next){
            int v = e[i].v;
            if(d[v] > d[u] + e[i].w){
                d[v] = d[u] + e[i].w;
                fa[v] = u;
                if(!vis[v]){
                    vis[v] = 1;
                    q.push(v);
                    c[v]++;
                }
            }
        }
    }
    return d[n];
}
```

1.2 网络流

1.2.1 Dinic

复杂度玄学

```
#define INF 0x3f3f3f3f
template <class T> //模板，可处理double类型
struct Dinic{
    struct Edge{
        int v, next;
        T flow;
        Edge(){}
        Edge(int v, int next, T flow) : v(v), next(next), flow(flow) {}
    }e[N << 1];
    int head[N], tot;
    int dep[N];
    void init() {
        memset(head, -1, sizeof(head)); tot = 0;
    }
    void adde(int u, int v, T w, T rw = 0) {
        e[tot] = Edge(v, head[u], w);
        head[u] = tot++;
        e[tot] = Edge(u, head[v], rw);
        head[v] = tot++;
    }
    bool BFS(int _S, int _T) {
        memset(dep, 0, sizeof(dep));
```

```
        queue <int> q; q.push(_S); dep[_S] = 1;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for(int i = head[u]; ~i; i = e[i].next) {
                int v = e[i].v;
                if(!dep[v] && e[i].flow > 0) {
                    dep[v] = dep[u] + 1;
                    q.push(v);
                }
            }
        }
        return dep[_T] != 0;
    }
    T dfs(int _S, int _T, T a) {
        T flow = 0, f;
        if(_S == _T || a == 0) return a;
        for(int i = head[_S]; ~i; i = e[i].next) {
            int v = e[i].v;
            if(dep[v] != dep[_S] + 1) continue;
            f = dfs(v, _T, min(a, e[i].flow));
            if(f) {
                e[i].flow -= f;
                e[i ^ 1].flow += f;
                flow += f;
                a -= f;
                if(a == 0) break;
            }
        }
        if(!flow) dep[_S] = -1;
        return flow;
    }
    T dinic(int _S, int _T) {
        T max_flow = 0;
        while(BFS(_S, _T)) max_flow += dfs(_S, _T, INF);
        return max_flow;
    }
};
```

1.2.2 费用流

势优化过后的 dijkstra，可以处理负边权。

```
#define INF 0x3f3f3f3f
struct edge {
  int to, capacity, cost, rev;
  edge() {}
  edge(int to, int _capacity, int _cost, int _rev) :to(to), capacity(_capacity),
```

```
      cost(_cost), rev(_rev) {}
};
struct Min_Cost_Max_Flow {
  int V, H[N << 1], dis[N << 1], PreV[N << 1], PreE[N << 1];
  vector<edge> G[N << 1];
  void Init(int n) {
    V = n;
    for (int i = 0; i <= V; ++i)G[i].clear();
  }
  void Add_Edge(int from, int to, int cap, int cost) {
    G[from].push_back(edge(to, cap, cost, G[to].size()));
    G[to].push_back(edge(from, 0, -cost, G[from].size() - 1));
  }
//flow是自己传进去的变量，就是最后的最大流，返回的是最小费用，f=INF
  int Min_cost_max_flow(int s, int t, int f, int& flow) {
    int res = 0; fill(H, H + 1 + V, 0);
    while (f) {
      priority_queue <pair<int, int>, vector<pair<int, int>>, greater<pair<int,
          int>> > q;
      fill(dis, dis + 1 + V, INF);
      dis[s] = 0; q.push(pair<int, int>(0, s));
      while (!q.empty()) {
        pair<int, int> now = q.top(); q.pop();
        int v = now.second;
        if (dis[v] < now.first)continue;
        for (int i = 0; i < G[v].size(); ++i) {
          edge& e = G[v][i];
          if (e.capacity > 0 && dis[e.to] > dis[v] + e.cost + H[v] - H[e.to]) {
            dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];
            PreV[e.to] = v;
            PreE[e.to] = i;
            q.push(pair<int, int>(dis[e.to], e.to));
          }
        }
      }
      if (dis[t] == INF)break;
      for (int i = 0; i <= V; ++i)H[i] += dis[i];
      int d = f;
      for (int v = t; v != s; v = PreV[v])d = min(d, G[PreV[v]][PreE[v]].capacity
          );
      f -= d; flow += d; res += d*H[t];
      for (int v = t; v != s; v = PreV[v]) {
        edge& e = G[PreV[v]][PreE[v]];
        e.capacity -= d;
        G[v][e.rev].capacity += d;
      }
    }
```

```
    return res;
  }
  int Max_cost_max_flow(int s, int t, int f, int& flow) {
    int res = 0;
    fill(H, H + 1 + V, 0);
    while (f) {
      priority_queue <pair<int, int>> q;
      fill(dis, dis + 1 + V, -INF);
      dis[s] = 0;
      q.push(pair<int, int>(0, s));
      while (!q.empty()) {
        pair<int, int> now = q.top(); q.pop();
        int v = now.second;
        if (dis[v] > now.first)continue;
        for (int i = 0; i < G[v].size(); ++i) {
          edge& e = G[v][i];
          if (e.capacity > 0 && dis[e.to] < dis[v] + e.cost + H[v] - H[e.to]) {
            dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];
            PreV[e.to] = v;
            PreE[e.to] = i;
            q.push(pair<int, int>(dis[e.to], e.to));
          }
        }
      }
      if (dis[t] == -INF)break;
      for (int i = 0; i <= V; ++i)H[i] += dis[i];
      int d = f;
      for (int v = t; v != s; v = PreV[v])d = min(d, G[PreV[v]][PreE[v]].capacity
          );
      f -= d; flow += d;
      res += d*H[t];
      for (int v = t; v != s; v = PreV[v]) {
        edge& e = G[PreV[v]][PreE[v]];
        e.capacity -= d;
        G[v][e.rev].capacity += d;
      }
    }
    return res;
  }
}sol;
```

1.3 连通性

1.3.1 有向图强连通分量

Tarjan 算法，复杂度 $O(n)$，能够求出有向图的极大强连通分量。

```
stack <int> s;
int T, num;
int scc[N], dfn[N], low[N], vis[N];
void Tarjan(int u){
    dfn[u] = low[u] = ++T; vis[u] = 1;
    s.push(u);
    for(int i = head[u]; i != -1;i = e[i].next){
        int v = e[i].v;
        if(!vis[v]){
            Tarjan(v);
            low[u] = min(low[u], low[v]);
        }else if(!scc[v]){
            low[u] = min(low[u], dfn[v]);
        }
    }
    if(low[u] == dfn[u]){
        num++; int now;
        do{
            now = s.top(); s.pop();
            scc[now] = num;
        }while(!s.empty() && now!=u);
    }
}
```

1.3.2 2-sat

通过在反图上求强连通分量，优先选择序号较大者。

```
vector<int> G[N], rG[N], vs;
int used[N], bel[N];

void adde(int from, int to) {
    G[from].push_back(to);
    rG[to].push_back(from);
}

void dfs(int v) {
    used[v] = true;
    for(int u: G[v]) {
        if(!used[u])
            dfs(u);
    }
    vs.push_back(v);
}

void rdfs(int v, int k) {
    used[v] = true;
```

```
    bel[v] = k;
    for(int u: rG[v])
        if(!used[u])
            rdfs(u, k);
}

int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    for(int v = 0; v < n; ++v)
        if(!used[v]) dfs(v);
    memset(used, 0, sizeof(used));
    int k = 0;
    for(int i = (int) vs.size() - 1; i >= 0; --i)
        if(!used[vs[i]]) rdfs(vs[i], k++);
    return k;
}
```

1.4 生成树

1.4.1 Kruskal 算法

```
struct Edge{
    int u,v,w;
    bool operator < (const Edge &A)const{
        return w<A.w;
    }
}e[N*N];
int f[N];
int find(int x){
    return f[x]==x?f[x]:f[x]=find(f[x]);
}
int Kruskal(){
    int ans=0;
    for(int i=0;i<=n+1;i++) f[i]=i;
    for(int i=1;i<=m;i++){
        int fx=find(e[i].u),fy=find(e[i].v);
        if(fx==fy) continue ;
        f[fx]=fy;
        ans+=e[i].w;
    }
    return ans ;
}
```

1.4.2 次小生成树

复杂度为 $O(n^2)$，貌似可以通过树上倍增优化到 $O(nlogn)$。

严格次小生成树则需要同时维护最大值和次大值来比较。

```
struct Edge{
    int u,v,w;
    bool operator < (const Edge &A)const{
        return w<A.w;
    }
}e[N*N];
int f[N],check[N];
int d[N][N],dis[N][N],mp[N][N];
int find(int x){
    return f[x]==x?f[x]:f[x]=find(f[x]);
}
int Kruskal(){
    int ans=0;
    for(int i=0;i<=n+1;i++) f[i]=i;
    for(int i=1;i<=m;i++){
        int u=e[i].u,v=e[i].v;
        int fx=find(u),fy=find(v);
        if(fx==fy) continue ;
        f[fx]=fy;
        mp[u][v]=mp[v][u]=1;
        ans+=e[i].w;
    }
    return ans ;
}
void dfs(int u,int fa){
    for(int x=1;x<=n;x++){
        if(check[x]) d[x][u]=d[u][x]=max(d[x][fa],dis[u][fa]);
    }
    check[u]=1;
    for(int v=1;v<=n;v++){
        if(mp[v][u] && v!=fa) dfs(v,u);
    }
}
```

1.4.3 最小有向生成树

朱刘算法，复杂度 $O(nm)$。

```
struct Edge{
    int u,v,w;
}e[M];
int pre[N]; //记录前驱.
int id[N],vis[N],in[N];
```

```
int dirMst(int root){
    int ans=0;
    while(1){
        memset(in,INF,sizeof(in));
        memset(id,-1,sizeof(id));
        memset(vis,-1,sizeof(vis));
        for(int i=1;i<=m;i++){
            int u=e[i].u,v=e[i].v,w=e[i].w;
            if(w<in[v] && v!=u){
                pre[v]=u;
                in[v]=w;
            }
        }           //求最小入边集
        in[root]=0;
        pre[root]=root;
        for(int i=0;i<n;i++){
            if(in[i]==INF) return -1;
            ans+=in[i];
        }
        int idx = 0; //新标号
        for(int i=0;i<n;i++){
            if(vis[i] == -1 ){
                int u = i;
                while(vis[u] == -1){
                    vis[u] = i;
                    u = pre[u];
                }
                if(vis[u]!=i || u==root) continue;     //判断是否形成环
                for(int v=pre[u];v!=u;v=pre[v] )
                    id[v]=idx;
                id[u] = idx++;
            }
        }
        if(idx==0) break;
        for(int i=0;i<n;i++){
            if(id[i]==-1) id[i]=idx++;
        }
        for(int i=1;i<=m;i++){
            e[i].w-=in[e[i].v];
            e[i].u=id[e[i].u];
            e[i].v=id[e[i].v];
        }
        n = idx;
        root = id[root];//给根新的标号
    }
    return ans;
}
```

1.4.4 最小生成树计数

b 为基尔霍夫矩阵，即度数矩阵-邻接矩阵，注意要考虑重边，重复的边只算一次。

```
ll Det(int n){
    int i,j,k;
    ll ret = 1;
    for(i=2;i<=n;i++){
        for(j = i+1;j <= n;j++){
            while(b[j][i]){
                ll tmp=b[i][i]/b[j][i];//不存在除不尽的情况
                for(k = i;k <= n;k++)
                    b[i][k] -= tmp*b[j][k];
                for(k=i;k<=n;k++)
                    swap(b[i][k],b[j][k]);
                ret = -ret;
            }
        }
        if(!b[i][i]) return 0;
        ret *= b[i][i];
    }
    if(ret < 0) ret = -ret;
    return ret;
}
```

1.5 割点及桥

1.5.1 求出割点及桥

```
void init(){
    T=0;tot=0;
    memset(head,-1,sizeof(head));
    memset(cut,0,sizeof(cut));
    memset(dfn,0,sizeof(dfn));
    memset(bri,0,sizeof(bri));
}
void Tarjan(int u,int pre){
    dfn[u]=low[u]=++T;
    int son=0;
    for(int i=head[u];i!=-1;i=e[i].next){
        int v=e[i].v;
        if(v==pre) continue ;
        if(!dfn[v]){
            son++;//起点有效儿子
            Tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]&&u!=pre)cut[u]=1;
```

```
            if(low[v]>dfn[u]){
                bri[i]=1;bri[i^1]=1;
            }
        }else{
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(u==pre && son>1) cut[u]=1;
}
```

1.5.2 无向图缩点成树

核心思想，保留割点，其余用并查集联通，最终树上的点一定由割点和环构成。

```
int T,tot,cnt;
int dfn[N],low[N],cut[N],num[N],f[N];
void adde(int u,int v){
    e[tot].u=u;e[tot].v=v;e[tot].next=head[u];head[u]=tot++;
}
void init(){
    T=0;tot=0;cnt=0;
        return f[x]==x?f[x]:f[x]=find(f[x]);
    }
    void Union(int x,int y){
        int fx=find(x),fy=find(y);
        if(fx!=fy) f[fx]=fy;
    }
    void Tarjan(int u,int pre){
        dfn[u]=low[u]=++T;
        int k=0;
        for(int i=head[u];i!=-1;i=e[i].next){
    memset(head,-1,sizeof(head));
    memset(cut,0,sizeof(cut));
    memset(dfn,0,sizeof(dfn));
    memset(num,0,sizeof(num));
    for(int i=0;i<=n+1;i++) f[i]=i;
}
int find(int x){
        int v=e[i].v;
        if(v==pre && !k){//处理重边，重边会考虑
            k=1;
            continue ;
        }
        if(!dfn[v]){
            Tarjan(v,u);
            low[u]=min(low[u],low[v]);
        }else{
```

```
34              low[u]=min(low[u],dfn[v]);
35          }
36          if(low[v]>dfn[u]){
37              cut[v]=1;//割点
38          }else Union(u,v);
39      }
40  }
```

1.6 二分图

1.6.1 最大匹配

最大匹配 = 最小点覆盖 = 最小路径覆盖 = 顶点数-最大独立集
二分图的最大团 = 补图的最大独立集（图不一定是二分图）
以下是匈牙利算法，也可以网络流做，效率更高：

```
1   int match[N],check[N];
2   int dfs(int x,int nown){
3       for(int i=1;i<=nown;i++){
4           if(!check[i] && link[x][i]){
5               check[i]=1;
6               if(match[i]==-1 || dfs(match[i],nown)){
7                   match[i]=x;
8                   return 1;
9               }
10          }
11      }
12      return 0;
13  }
14  int hungry(int n1,int m1){
15      memset(match,-1,sizeof(match));
16      int ans=0;
17      for(int i=1;i<=n1;i++){
18          memset(check,0,sizeof(check));
19          ans+=dfs(i,m1);
20      }
21      return ans ;
22  }
```

1.7 最大团

一般图的做法就是爆搜 + 减枝，这是个 NPC 问题。
注意独立集和团的转换。

```
1   int link[N][N],vis[N],group[N],cnt[N];
2   int ans ;
```

```
int dfs(int x,int tot){
    for(int i=x+1;i<=n;i++){
        if(cnt[i]+tot<=ans) return 0;//剪枝1
        if(link[x][i]){
            int flag = 0;
            for(int j=0;j<tot;j++){
                if(!link[i][vis[j]]) flag=1;
            }
            if(!flag){
                vis[tot]=i;
                if(dfs(i,tot+1)) return 1;//剪枝2
            }
        }
    }
    if(tot>ans){
        ans=tot;
        for(int i=0;i<tot;i++) group[i]=vis[i];
        return 1;
    }
    return 0;
}
void maxclique(){
    ans=-1;
    memset(cnt,0,sizeof(cnt));
    for(int i=n;i>=1;i--){
        vis[0]=i;
        dfs(i,1);
        cnt[i]=ans;
    }
}
```

1.8 支配树

在 DAG 中，可以直接利用性质来构造，一个点的支配点就为所有能到达它的点在支配树上的 LCA，比较好理解。

下面给出一般图构造支配树的算法

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 2e5 + 5, M = 3e5 + 5;

namespace LT{
    vector <int> G[N], rG[N];
    vector <int> dt[N];     //dominant tree
    int fa[N], best[N], T, n;
```

```
10      int semi[N], idom[N], dfn[N], idx[N], f[N];
11      void init() {
12          T = 0;
13          for(int i = 1; i <= n; i++) semi[i] = f[i] = best[i] = i;
14          for(int i = 1; i <= n; i++) dt[i].clear();
15      }
16      void dfs(int u) {
17          dfn[u] = ++T; idx[T] = u;;
18          for(auto v : G[u]) {
19              if(!dfn[v]) {
20                  fa[v] = u; dfs(v);
21              }
22          }
23      }
24      int find(int x) {
25          if(f[x] == x) return x;
26          int fx = find(f[x]);
27          if(dfn[semi[best[f[x]]]] < dfn[semi[best[x]]]) best[x] = best[f[x]];
28          return f[x] = fx;
29      }
30      void Tarjan(int rt) {
31          dfs(rt);
32          for(int i = T; i >= 2; i--) {
33              int x = idx[i];
34              for(int &u : rG[x]) {
35                  if(!dfn[u]) continue; //可能原图不能到达
36                  find(u);
37                  if(dfn[semi[x]] > dfn[semi[best[u]]]) semi[x] = semi[best[u]];
38              }
39              f[x] = fa[x];
40              dt[semi[x]].push_back(x);
41              x = fa[x];
42              for(int &u : dt[x]) {
43                  find(u);
44                  if(semi[best[u]] != x) idom[u] = best[u];
45                  else idom[u] = x;
46              }
47              dt[x].clear();
48          }
49          for(int i = 2; i <= T; i++) {
50              int x = idx[i];
51              if(idom[x] != semi[x]) idom[x] = idom[idom[x]];
52              dt[idom[x]].push_back(x);
53          }
54      }
55  }
56  int n, m;
```

```cpp
int sz[N];
void dfs(int u, int fa) {
    sz[u] = 1;
    for(auto v : LT::dt[u]) {
        if(v == fa) continue;
        dfs(v, u);
        sz[u] += sz[v];
    }
}
int main() {
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    LT::n = n;
    LT::init();
    for(int i = 1; i <= m; i++) {
        int u, v; cin >> u >> v;
        LT::G[u].push_back(v);
        LT::rG[v].push_back(u);
    }
    LT::Tarjan(1);
    dfs(1, -1);
    for(int i = 1; i <= n; i++) cout << sz[i] << ' ';
    return 0;
}
```

第二章　数学

2.1　素数

2.1.1　线性筛

线性筛，每个数都只被其最小质因子筛一次，复杂度 $O(n)$。

```
for(int i = 2; i <= n; i++) {
    if(!chk[i]) {
        prime[++tot] = i;
    }
    for(int j = 1; j <= tot && 1ll * i * prime[j] <= n; j++) {
        chk[i * prime[j]] = 1;
        if(i % prime[j] == 0) break;
    }
}
```

2.1.2　Miller-Rabin 素数测试

```
//记得初始化随机种子
ll mul(ll a, ll b, ll p) {
    a %= p, b %= p;
    ll ans = 0;
    while(b) {
        if(b & 1) {
            ans = ans + a;
            if(ans > p) ans -= p;
        }
        a = a + a;
        if(a > p) a -= p;
        b >>= 1;
    }
    return ans;
}
ll qp(ll a, ll b, ll p) {
    ll ans = 1; a %= p;
    while(b) {
        if(b & 1) ans = mul(ans, a, p);
```

```
20          a = mul(a, a, p);
21          b >>= 1;
22      }
23      return ans;
24  }
25  bool check(ll a, ll n, ll x, ll t) {
26      ll ans = qp(a, x, n);
27      ll last = ans;
28      for(int i = 1; i <= t; i++) {
29          ans = mul(ans, ans, n);
30          if(ans == 1 && last != 1 && last != n - 1) return true;
31          last = ans;
32      }
33      if(ans != 1) return true;
34      return false;
35  }
36  bool Miller_Rabin(ll n) {
37      if(n == 1 || (n & 1) == 0) return false;
38      if(n == 2) return true;
39      ll x = n - 1, t = 0;
40      while((x & 1) == 0) {x >>= 1, ++t;}
41      srand(time(NULL));
42      for(int i = 0; i < 8; i++) {
43          ll a = rand() % (n - 1) + 1;
44          if(check(a, n, x, t)) return false;
45      }
46      return true;
47  }
```

2.2 欧几里得算法

2.2.1 扩展欧几里得

求解 $ax+by=c, gcd(a,b)|c$ 的整数解 x,y, 设 $g=gcd(a,b)$, 通解为 $x=x+k*\frac{b}{g}, y=y-k*\frac{a}{g}$, 最后的解 $x=x*c/g, y=y*c/g$。

```
1   void exgcd(ll a, ll b, ll &x, ll &y) {
2       if(b == 0) {
3           x = 1, y = 0;
4           return ;
5       }
6       exgcd(b,a%b,x,y);
7       ll z = x ;
8       x = y;
9       y = z - y * (a / b);
10  }
```

2.2.2 类欧几里得

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

形象来说，问题转换为求直线下方整点数，然后不断坐标变换，让直线斜率变得更平缓。

复杂度为 $O(logn)$。

```
const int MOD = 1000000007, inv2 = (MOD + 1) / 2;
ll f(ll n, ll a, ll b, ll c) {
  //sum_{i=0}^n (ai+b)/c
    if(a <= 0) return 0;
    if(a >= c || b >= c) {
        return (n * (n + 1) % MOD * inv2 % MOD * (a / c) % MOD
        + (n + 1) * (b / c) % MOD + f(n, a % c, b % c, c)) % MOD;
    }
    ll m = (a * n + b) / c;
    return (m * n % MOD - f(m - 1, c, c - b - 1, a) + MOD) % MOD;
}
```

求解最小的 x,y，使得 $\frac{a}{b}\le\frac{x}{y}\le\frac{c}{d}$。

类欧几里得的一个应用，详细证明见:https://www.cnblogs.com/heyuhhh/p/11310342.html

```
void gao(ll a, ll b, ll c, ll d, ll &x, ll &y) { // a/b < x/y < c/d
    ll t = (a + b - 1) / b;
    if(c / d >= t) {
        y = 1; x = t;
        return;
    }
    a -= (t - 1) * b;
    c -= (t - 1) * d;
    gao(d, c, b, a, y, x);
    x += (t - 1) * y;
}
```

2.3 线性递推求逆元

求 $1,\ldots,p-1$ 模 p 的所有逆元，有公式：$inv[i]=(p-p/i)*inv[p\%i]\%m$

证明：设 $t=\frac{p}{i},k=p\%i$，就有 $t*i+k\equiv 0(\%p)$ 即 $t*i\equiv -k(\%p)$，两边同时除以 $i*k$ 就出来了。

代码略。

2.4 高斯消元

若秩小于 n，则有无穷多组解，但首先排除无解的情况。

以下给出高斯消元求同余模方程组的代码。

```
int now = 1;
for(int i = 1 ; i <= n ; i++) {
    int j = now ;
    while(j <= m && !a[j][i]) j++;
    if(j > m) continue ;
    if(j != now) {
        for(int k = 1 ; k <= n + 1 ; k++) {
            swap(a[now][k] , a[j][k]) ;
        }
    }
    for(int j = now + 1 ; j <= m ; j++)
        if(a[j][i]) {
            int t = a[j][i] * inv[a[now][i]] % MOD;
            for(int k = i ; k <= n + 1 ; k++) {
                a[j][k] = (((a[j][k] - t * a[now][k]) % MOD) + MOD) % MOD;
            }
        }
    now++;
}
```

2.5 欧拉函数

$\varphi(x) = x\prod_{i=1}^{n}(1-\frac{1}{p_i})$
$\varphi * I = id$
$phi(p^k) = p^k - p^{k-1}$

2.5.1 求 $\varphi(n)$

```
int getphi(int x) {
    int ans = x;
    for(int i = 2; 1ll * i * i <= x; i++) {
        if(x % i == 0) {
            ans = ans / i * (i - 1);
            while(x % i == 0) x /= i;
        }
    }
    if(x > 1) ans = ans / x * (x - 1);
    return ans;
}
```

2.5.2 线性筛

```
int p[N], phi[N], tot;
bool chk[N];
void Euler() {
```

```
    phi[1] = 1;
    for(int i = 2; i < N; i++) {
        if(!chk[i]) p[++tot] = i, phi[i] = i - 1;
        for(int j = 1; j <= tot && i * p[j] < N; j++) {
            chk[i * p[j]] = 1;
            if(i % p[j] == 0) {
                phi[i * p[j]] = phi[i] * p[j];
                break;
            }
            phi[i * p[j]] = phi[i] * (p[j] - 1);
        }
    }
}
```

2.6 莫比乌斯函数

$$\mu = \begin{cases} 0, \text{存在平方因子} \\ -1, \text{奇数个质因子} \\ 1, \text{偶数个质因子} \end{cases}$$

$\sum_{d|n} \mu(d) = [n = 1]$

莫比乌斯反演：

形式一：已知 $g(n) = \sum_{d|n} f(d)$，则 $f(n) = \sum_{d|n} \mu(d) \cdot g(\frac{n}{d})$

形式二：已知 $g(n) = \sum_{n|d} f(d)$，则 $f(n) = \sum_{d|n} \mu(\frac{d}{n}) \cdot g(d)$ 线性筛:

```
int mu[N], p[N];
bool chk[N];
void init() {
    mu[1] = 1;
    int cnt = 0;
    for(int i = 2; i < N; i++) {
        if(!chk[i]) p[++cnt] = i, mu[i] = -1;
        for(int j = 1; j <= cnt && i * p[j] <= k; j++) {
            chk[i * p[j]] = 1;
            if(i % p[j] == 0) {mu[i * p[j]] = 0; break;}
            mu[i * p[j]] = -mu[i];
        }
    }
}
```

2.7 整除分块

求解 $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$。

```
int ans = 0;
for(int i = 1, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ans += (j - i + 1) * (n / i);
}
```

2.8 杜教筛

求解 $S(n)=\sum_{i=1}^{n} f(i), f$ 为积性函数。

构造 $h=f*g$，最后有 $g(1)\cdot S(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{d=2}^{n} g(d)S(\lfloor\frac{n}{d}\rfloor)$ 暴力递归即可。

可预处理前 $\frac{1}{3}$ 项，复杂度 $O(n^{\frac{2}{3}})$，注意记忆化。

下面给出筛欧拉函数和莫比乌斯函数的代码。

```
ll djs_mu(int n) {
    if(n <= 5000000) return mu[n];
    if(mp1[n]) return mp1[n];
    ll ans = 1;
    for(int i = 2, j; i <= n; i = j + 1) {
        j = n / (n / i);
        ans -= (j - i + 1) * djs_mu(n / i);
    }
    return mp1[n] = ans;
}
ll djs_phi(int n) {
    if(n <= 5000000) return phi[n];
    if(mp2[n]) return mp2[n];
    ll ans = 1ll * (n + 1) * n / 2;
    for(int i = 2, j; i <= n; i = j + 1) {
        j = n / (n / i);
        ans -= (j - i + 1) * djs_phi(n / i);
    }
    return mp2[n] = ans;
}
```

2.9 min25 筛

求解一类积性函数的前缀和，此类积性函数满足在 p^k 处的值容易计算，并且 $f(p)$ 为一个与 p 有关的简单多项式。

$$g(n,j)=g(n,j-1)-f(p_j)*(g(\frac{n}{p_j},j-1)-sum_{j-1})$$

$$S(i,j)=g(i,|P|)-sum_{j-1}+\sum_{k\geq j}\sum_{e}F(p_k^e)(S(\frac{i}{p_k^e},k+1)+[e\neq 1])$$

$g(n,j):1\cdots n$ 中，全为质数或者最小质因子大于第 j 个质数的数之和；$S(i,j):1\cdots i$ 中，最小质因子大于等于第 j 个质数的数之和。

最终答案为 $S(n,1)+f(1)$。

```
//注意1单独考虑
//g应为完全积性函数，在质数处与f相等
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 2e6 + 5, MOD = 1e9 + 7, inv3 = 333333336;
ll n;
ll sum1[N], sum2[N], prime[N];
ll w[N], ind1[N], ind2[N];
ll g1[N], g2[N];
bool chk[N];
int tot, cnt;
void pre(int n) { //  \sqrt
    chk[1] = 1;
    for(int i = 1; i <= n; i++) {
        if(!chk[i]) {
            prime[++tot] = i;
            sum1[tot] = (sum1[tot - 1] + i) % MOD;
            sum2[tot] = (sum2[tot - 1] + 1ll * i * i % MOD) % MOD;
        }
        for(int j = 1; j <= tot && prime[j] * i <= n; j++) {
            chk[i * prime[j]] = 1;
            if(i % prime[j] == 0) break;
        }
    }
}
void calc_g() {
    int z = sqrt(n);
    for(ll i = 1, j; i <= n; i = j + 1) {
        j = n / (n / i);
        w[++cnt] = n / i;
        g1[cnt] = w[cnt] % MOD;
        g2[cnt] = g1[cnt] * (g1[cnt] + 1) / 2 % MOD * (2 * g1[cnt] + 1) % MOD *
            inv3 % MOD - 1;
        g1[cnt] = g1[cnt] * (g1[cnt] + 1) / 2 % MOD - 1;
        if(n / i <= z) ind1[n / i] = cnt;
        else ind2[n / (n / i)] = cnt;
    }
    for(int i = 1; i <= tot; i++) {
        for(int j = 1; j <= cnt && prime[i] * prime[i] <= w[j]; j++) {
            ll tmp = w[j] / prime[i], k;
            if(tmp <= z) k = ind1[tmp]; else k = ind2[n / tmp];
            (g1[j] -= prime[i] * (g1[k] - sum1[i - 1] + MOD) % MOD) %= MOD;
```

```
43              (g2[j] -= prime[i] * prime[i] % MOD * (g2[k] - sum2[i - 1] + MOD) %
                    MOD) %= MOD;
44              if(g1[j] < 0) g1[j] += MOD;
45              if(g2[j] < 0) g2[j] += MOD;
46          }
47      }
48  }
49  ll S(ll x, int y) { // 2~x >= P_y
50      if(x <= 1 || prime[y] > x) return 0;
51      ll z = sqrt(n);
52      ll k = x <= z ? ind1[x] : ind2[n / x];
53      ll ans = (g2[k] - g1[k] + MOD - (sum2[y - 1] - sum1[y - 1]) + MOD) % MOD;
54      for(int i = y; i <= tot && prime[i] * prime[i] <= x ; i++) {
55          ll pe = prime[i], pe2 = prime[i] * prime[i];
56          for(int e = 1; pe2 <= x; ++e, pe = pe2, pe2 *= prime[i]) {
57              ans = (ans + pe % MOD * ((pe - 1) % MOD) % MOD * S(x / pe, i + 1) +
                    pe2 % MOD * ((pe2 - 1) % MOD) % MOD) % MOD;
58          }
59      }
60      return ans % MOD;
61  }
62  int main() {
63      cin >> n;
64      int tmp = sqrt(n);
65      pre(tmp);
66      calc_g();
67      cout << (S(n, 1) + 1) % MOD ;
68      return 0;
69  }
```

2.10 中国剩余定理

```
1   struct CRT{
2       void exgcd(ll a, ll b, ll &g, ll &x, ll &y) {
3           if(b == 0) {
4               x = 1, y = 0, g = a;
5               return ;
6           }
7           exgcd(b, a % b, g, x, y);
8           int t = x;
9           x = y;
10          y = t - (a / b) * y;
11      }
12      ll china(ll m[], ll a[], int n) {
13          ll M, Mi, d, X, Y, ans;
14          M = 1; ans = 0;
```

```
        for(int i = 1; i <= n; i++) M *= m[i];
        for(int i = 1; i <= n; i++) {
            Mi = M / m[i];
            exgcd(Mi, m[i], d, X, Y);
            ans = (ans + Mi * X * a[i]) % M;
        }
        if(ans < 0) ans += M;
        return ans;
    }
}crt;
```

扩展中国剩余定理可以处理模数不互质的情况，直接类似于数学归纳法合并解即可。

2.11 BSGS

求解高次同余方程 $a^x \equiv b \mod m$

令 $x = i * t + j, t = \sqrt{m}$，分块处理即可。

```
//预处理$a^j$，之后将逆元乘过去
//v=$\frac{1}{a^t}$
struct B{
    const int mod = 524287; // (1 << 19) - 1;
    int tot;
    int h[524288], next[524288], L[524288], v[524288];
    int Find(ll x) {
        int k = h[x & mod];
        while(k != 0) {
            if(L[k] == x) return v[k];
            else k = next[k];
        }
        return -1;
    }
    void Add(int e, int i) {
        tot++;
        next[tot] = h[e & mod];
        L[tot] = e; v[tot] = i;
        h[e & mod] = tot;
    }
    void init(int a, int n) {
        memset(h, 0, sizeof(h)); memset(L, 0, sizeof(L));tot = 0;
        memset(next, 0, sizeof(next)); memset(v, 0, sizeof(v));
        ll t, e = 1;
        t = (int)sqrt(n) + 1;
        for(int i = 0; i < t; i++) {
            if(Find(e) == -1) Add(e, i);
            e = e * a % n;
        }
```

```
    }
    ll BSGS(int a, int b, int n, ll v, ll t) { // a ^ x = b (mod n)
        for(int i = 0; i < t; i++) {
            if(Find(b) != -1) return i * t + Find(b);
            b = b * v % n;
        }
        return -1;
    }
}S;
```

2.12 二次剩余

若 $a^{\frac{p-1}{2}} \equiv 1 \mod p$，那么 a 为模 p 的二次剩余。若 $a^{\frac{p-1}{2}} \equiv -1 \mod p$，则 a 为模 p 的非二次剩余。若 $a^{\frac{p-1}{2}} \equiv 0 \mod p$，则 $p|a$。

2.13 欧拉降幂

$$a^b \equiv \begin{cases} a^{b\%\varphi(p)}, & gcd(a,p)=1 \\ a^b, & gcd(a,p)\neq 1, b<\varphi(p) \\ a^{\varphi(p)+b\%\varphi(p)}, & gcd(a,p)\neq 1, b>=\varphi(p) \end{cases} (mod\ p)$$

求 $a^{a^{a^{\cdots^a}}}\%p$ 代码如下：(注意快速幂部分，如果指数大于 p 了，应当保留一个 p)

```
ll Mod(ll a, ll b) { //可以直接避免分类讨论
    return a < b ? a : a % b + b;
}
int qp(ll a, ll b, ll p) {
    int ans = 1;
    while(b) {
        if(b & 1) ans = Mod(ans * a, p);
        a = Mod(a * a, p);
        b >>= 1;
    }
    return ans;
}
int calc(ll a, ll b, ll m) {
    if(m == 1 || !b) return 1;
    int p = phi[m];
    int x = calc(a, b - 1, p);
    return qp(a, x, m);
}
```

2.14 FFT

```
const double pi = acos(-1.0);
struct C{
    double x, y;
    C (double xx = 0, double yy = 0) {x = xx, y = yy;}
}a[N], b[N], c[N];
C operator + (C a, C b) {return C(a.x + b.x, a.y + b.y);}
C operator - (C a, C b) {return C(a.x - b.x, a.y - b.y);}
C operator * (C a, C b) {return C(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);}
int l, r[N];
int lim = 1;
void solve(C *A, int type) {
    for(int i = 0; i < lim; i++) if(i < r[i]) swap(A[i], A[r[i]]);
    for(int mid = 1; mid < lim; mid <<= 1) {
        C Wn(cos(pi / mid), type * sin(pi / mid)) ;
        for(int R = mid << 1, j = 0; j < lim; j += R) {
            C w(1, 0);
            for(int k = 0; k < mid; k++, w = w * Wn) {
                C x = A[j + k], y = w * A[j + mid + k];
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}
void FFT(C *a, C *b) {
    while(lim <= n + m) lim <<= 1, l++;
    for(int i = 0; i < lim; i++) {
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    }
    for(int i = n + 1; i < lim; i++) a[i] = C();
    for(int i = m + 1; i < lim; i++) b[i] = C();
    solve(a, 1); solve(b, 1);
    for(int i = 0; i <= lim; i++) c[i] = a[i] * b[i];
    solve(c, -1);
    for(int i = 0; i <= n + m; i++) c[i].x = (c[i].x / lim + 0.5);
}
```

2.15 NTT

有模数版 FFT，模数一般为 998244353。

```
const int N = 4e5 + 5, M = 2e6 + 5, P = 998244353, G = 3, Gi = 332748118;
int n, m, lim = 1, L, r[N];
ll a[N], b[N];
ll qp(ll a, ll k) {
```

```
    ll ans = 1;
    while(k) {
        if(k & 1) ans = (ans * a ) % P;
        a = (a * a) % P;
        k >>= 1;
    }
    return ans;
}
void NTT(ll *A, int type) {
    for(int i = 0; i < lim; i++)
        if(i < r[i]) swap(A[i], A[r[i]]);
    for(int mid = 1; mid < lim; mid <<= 1) {
        ll Wn = qp( type == 1 ? G : Gi , (P - 1) / (mid << 1)); //Wn = g ^ ((p -
            1) / n)  (mod p)
        for(int j = 0; j < lim; j += (mid << 1)) {
            ll w = 1;
            for(int k = 0; k < mid; k++, w = (w * Wn) % P) {
                 int x = A[j + k], y = w * A[j + k + mid] % P;
                 A[j + k] = (x + y) % P,
                 A[j + k + mid] = (x - y + P) % P;
            }
        }
    }
}
void solve(ll *a, ll *b) {
    while(lim <= n + m) lim <<= 1, L++;
    for(int i = 0; i < lim; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (L - 1));
    for(int i = n + 1; i < lim; i++) a[i] = 0;  //a,b need init
    for(int i = m + 1; i < lim; i++) b[i] = 0;
    NTT(a, 1); NTT(b, 1);
    for(int i = 0; i < lim; i++) a[i] = (a[i] * b[i]) % P;
    NTT(a, -1);
    ll inv = qp(lim, P - 2);
    for(int i = 0; i < lim; i++) a[i] = a[i] * inv % P;
}
```

2.16 蔡勒公式

快速判断一天为星期几。

```
bool check(int k) {
    int y = 0, m = 0, d = 0;
    for(int i = 0; i < 4; i++) y = y * 10 + (a[s[k][i] - 'A']);
    for(int i = 5; i < 7; i++) m = m * 10 + (a[s[k][i] - 'A']);
    for(int i = 8; i < 10; i++) d = d * 10 + (a[s[k][i] - 'A']);
    if(y < 1600 || y > 9999 || m > 12 || d > 31) return false;
```

```
7       if((m == 2 && d > 28 + (y % 4 == 0 && (y % 100 != 0 || y % 400 == 0))))
            return false;
8       if(m != 2 && d > days[m]) return false;
9       if(m < 3) y--, m += 12;
10      return (y + y / 4 - y / 100 + y / 400 + d + 1 + 2 * m + 3 * (m + 1) / 5) % 7
            == 5; //年份大于1582
11    return (y + y / 4 + 5 + 3 * (m + 1) / 5 + d + 2 * m) % 7 //年份小于1582
12  }
```

2.17 原根

```
1   ll qp(ll a, ll b) {
2       ll ans = 1;
3       while(b) {
4           if(b & 1) ans = ans * a % p;
5           a = a * a % p;
6           b >>= 1;
7       }
8       return ans;
9   }
10  int work(ll P) {   //p - 1
11      vector <int> div;
12      for(int i = 2; 1ll * i * i <= P; i++) {
13          if(P % i == 0) {
14              div.push_back(i);
15              while(P % i == 0) P /= i;
16          }
17      }
18      if(P > 1) div.push_back(P);
19      for(int i = 2;; i++) {
20          bool ok = true;
21          for(auto x : div) {
22              if(qp(i, (p - 1) / x) % p == 1) {
23                  ok = false; break;
24              }
25          }
26          if(ok) return i;
27      }
28  }
```

2.18 二次剩余

```
1   //求解x^2=n(mod p)
2   const int moder = (int) 1e9 + 7;
3   const int inv2 = (moder + 1) / 2;
```

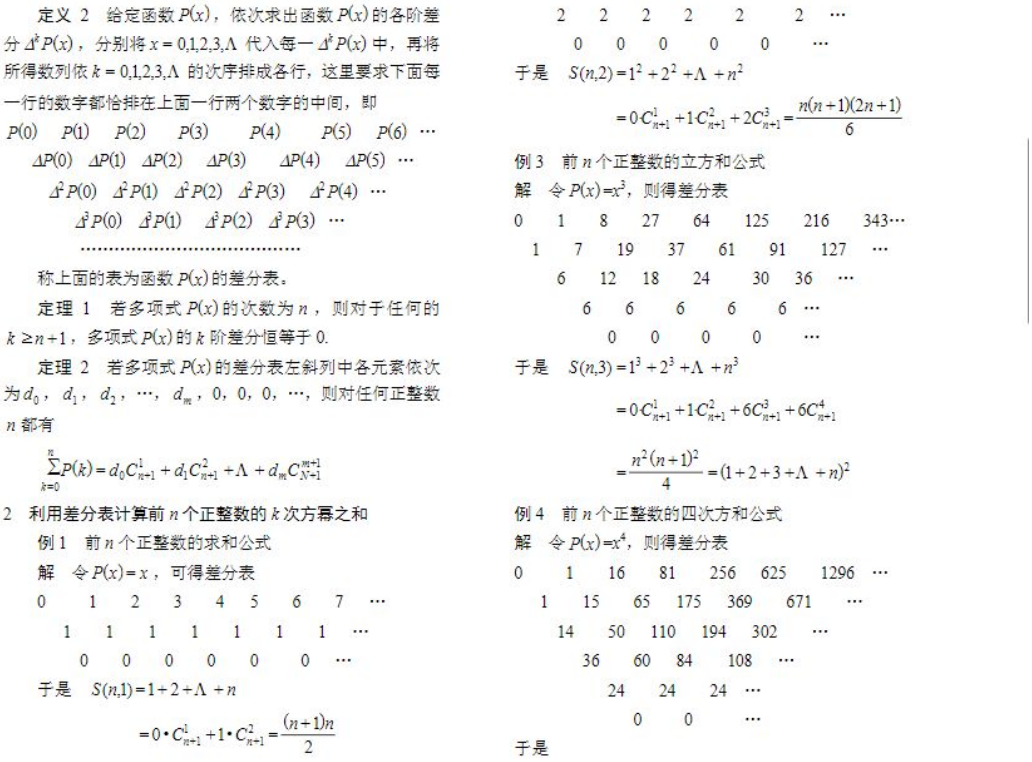
```
4
5   struct field2{
6       int x, y, a, p;
7
8       field2():x(0), y(0), a(0), p(0){}
9       field2(int x, int y, int a, int p):x(x), y(y), a(a), p(p){}
10
11      field2 operator *(const field2 &f)const{
12          int retx = (1ll * x * f.x + 1ll * y * f.y % p * a) % p;
13          int rety = (1ll * x * f.y + 1ll * y * f.x) % p;
14          return field2(retx, rety, a, p);
15      }
16
17      field2 powermod(int exp)const{
18          field2 ret(1, 0, a, p), aux = *this;
19          for ( ; exp > 0; exp >>= 1){
20              if (exp & 1){
21                  ret = ret * aux;
22              }
23              aux = aux * aux;
24          }
25          return ret;
26      }
27  };
28
29  int powermod(int a, int exp, int moder){
30      int ret = 1;
31      for ( ; exp; exp >>= 1){
32          if (exp & 1){
33              ret = 1ll * ret * a % moder;
34          }
35          a = 1ll * a * a % moder;
36      }
37      return ret;
38  }
39
40  int randint(int n){
41      return rand() % n;
42  }
43  vector <int> remain2(int a, int p){ //x^2 = a (mod p)
44      if (!a || p == 2){ //特判
45          return {a, a};
46      }
47      if (powermod(a, p - 1 >> 1, p) != 1){ //欧拉准则
48          return {};
49      }
50      while (true){
```

```
        field2 f(randint(p - 1) + 1, randint(p - 1) + 1, a, p);
        f = f.powermod(p - 1 >> 1);
        if (f.x){
            continue;
        }
        int ret = powermod(f.y, p - 2, p);
        return {ret, p - ret};
    }
}
```

2.19 拉格朗日插值

对于一个 n 次多项式，如果已知其 $n+1$ 项，那么就可以快速求解其他项，若那 $n+1$ 项连续，则可以通过预处理阶乘，复杂度达到 $O(n)$。

常见应用就为求解 $\sum_{i=1}^{n} i^k$，这是一个以 n 为自变量的 $k+1$ 次多项式，应用拉格朗日插值复杂度为 $O(k)$。



定义 2　给定函数 $P(x)$，依次求出函数 $P(x)$ 的各阶差分 $\Delta^k P(x)$，分别将 $x=0,1,2,3,\Lambda$ 代入每一 $\Delta^k P(x)$ 中，再将所得数列依 $k=0,1,2,3,\Lambda$ 的次序排成各行，这里要求下面每一行的数字都恰排在上面一行两个数字的中间，即

$P(0)\quad P(1)\quad P(2)\quad P(3)\quad P(4)\quad P(5)\quad P(6)\ \cdots$

$\Delta P(0)\quad \Delta P(1)\quad \Delta P(2)\quad \Delta P(3)\quad \Delta P(4)\quad \Delta P(5)\ \cdots$

$\Delta^2 P(0)\quad \Delta^2 P(1)\quad \Delta^2 P(2)\quad \Delta^2 P(3)\quad \Delta^2 P(4)\ \cdots$

$\Delta^3 P(0)\quad \Delta^3 P(1)\quad \Delta^3 P(2)\quad \Delta^3 P(3)\ \cdots$

……

称上面的表为函数 $P(x)$ 的差分表。

定理 1　若多项式 $P(x)$ 的次数为 n，则对于任何的 $k\geq n+1$，多项式 $P(x)$ 的 k 阶差分恒等于 0.

定理 2　若多项式 $P(x)$ 的差分表左斜列中各元素依次为 d_0，d_1，d_2，…，d_m，0，0，0，…，则对任何正整数 n 都有

$$\sum_{k=0}^{n} P(k) = d_0 C_{n+1}^1 + d_1 C_{n+1}^2 + \Lambda + d_m C_{N+1}^{m+1}$$

2　利用差分表计算前 n 个正整数的 k 次方幂之和

例 1　前 n 个正整数的求和公式

解　令 $P(x)=x$，可得差分表

0　1　2　3　4　5　6　7　…

1　1　1　1　1　1　1　…

0　0　0　0　0　0　…

于是　$S(n,1)=1+2+\Lambda+n$

$$=0\cdot C_{n+1}^1+1\cdot C_{n+1}^2=\frac{(n+1)n}{2}$$

2　2　2　2　2　2　…

0　0　0　0　0　…

于是　$S(n,2)=1^2+2^2+\Lambda+n^2$

$$=0C_{n+1}^1+1C_{n+1}^2+2C_{n+1}^3=\frac{n(n+1)(2n+1)}{6}$$

例 3　前 n 个正整数的立方和公式

解　令 $P(x)=x^3$，则得差分表

0　1　8　27　64　125　216　343…

1　7　19　37　61　91　127　…

6　12　18　24　30　36　…

6　6　6　6　6　…

0　0　0　0　…

于是　$S(n,3)=1^3+2^3+\Lambda+n^3$

$$=0C_{n+1}^1+1C_{n+1}^2+6C_{n+1}^3+6C_{n+1}^4$$

$$=\frac{n^2(n+1)^2}{4}=(1+2+3+\Lambda+n)^2$$

例 4　前 n 个正整数的四次方和公式

解　令 $P(x)=x^4$，则得差分表

0　1　16　81　256　625　1296　…

1　15　65　175　369　671　…

14　50　110　194　302　…

36　60　84　108　…

24　24　24　…

0　0　…

于是

```
struct Lagrange {
  static const int SIZE = 110;
  ll f[SIZE], fac[SIZE], inv[SIZE], pre[SIZE], suf[SIZE];
  int n;
  inline void add(int &x, int y) {
    x += y;
    if(x >= MOD) x -= MOD;
```

```
  }
  void init(int _n) {
    n = _n;
    fac[0] = 1;
    for (int i = 1; i < SIZE; ++i) fac[i] = fac[i - 1] * i % MOD;
      inv[SIZE - 1] = qp(fac[SIZE - 1], MOD - 2);
    for (int i = SIZE - 1; i >= 1; --i) inv[i - 1] = inv[i] * i % MOD;
    //设置f初值，可以根据需要修改
    f[0] = 0;
    for (int i = 1; i <= n; ++i)
      f[i] = (f[i - 1] + qp(i, K)) % MOD;
  }
  ll calc(ll x) {
    if (x <= n) return f[x];
    pre[0] = x % MOD;
    for (int i = 1; i <= n; ++i) pre[i] = pre[i - 1] * ((x - i) % MOD) % MOD;
    suf[n] = (x - n) % MOD;
    for (int i = n - 1; i >= 0; --i) suf[i] = suf[i + 1] * ((x - i) % MOD) % MOD;
    ll res = 0;
    for (int i = 0; i <= n; ++i) {
      ll tmp = f[i] * inv[n - i] % MOD * inv[i] % MOD;
      if (i) tmp = tmp * pre[i - 1] % MOD;
      if (i < n) tmp = tmp * suf[i + 1] % MOD;
      if ((n - i) & 1) tmp = MOD - tmp;
      add(res, tmp);
    }
    return res;
  }
}lagrange;
```

2.20 BM 线性递推

若递推式由前 k 项推出，我们只需要求出前 $2k$ 项，然后传入就行，能够快速求解第 n 项，复杂度为 $O(klogn)$。

```
#include<bits/stdc++.h>
#define rep(i,a,n) for (int i=a;i<n;i++)
#define SZ(x) ((int)(x).size())
const int MAXN = 2e5 + 5, INF = 0x3f3f3f3f, MOD = 1e9 + 7;
using namespace std;
#define lson o<<1,l,m
#define rson o<<1|1,m+1,r
#define mid l + ((r-l)>>1)
#define pb push_back

typedef vector<int> VI;
```

```
12 typedef long long ll;
13
14 ll powMOD(ll a, ll b) {
15     ll ans = 1;
16     for (; b; b >>= 1, a = a * a%MOD)if (b & 1)ans = ans * a%MOD;
17     return ans;
18 }
19 ll n;
20 namespace linear_seq {
21     const int N = 10010;
22     ll res[N], base[N], _c[N], _md[N];
23
24     vector<int> Md;
25     void mul(ll *a, ll *b, int k) {
26         rep(i, 0, k + k) _c[i] = 0;
27         rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j])
               % MOD;
28         for (int i = k + k - 1; i >= k; i--) if (_c[i])
29             rep(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] *
                   _md[Md[j]]) % MOD;
30         rep(i, 0, k) a[i] = _c[i];
31     }
32     int solve(ll n, VI a, VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
33         ll ans = 0, pnt = 0;
34         int k = SZ(a);
35         assert(SZ(a) == SZ(b));
36         rep(i, 0, k) _md[k - 1 - i] = -a[i]; _md[k] = 1;
37         Md.clear();
38         rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
39         rep(i, 0, k) res[i] = base[i] = 0;
40         res[0] = 1;
41         while ((1ll << pnt) <= n) pnt++;
42         for (int p = pnt; p >= 0; p--) {
43             mul(res, res, k);
44             if ((n >> p) & 1) {
45                 for (int i = k - 1; i >= 0; i--) res[i + 1] = res[i]; res[0] = 0;
46                 rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] * _md[Md[j]])
                       % MOD;
47             }
48         }
49         rep(i, 0, k) ans = (ans + res[i] * b[i]) % MOD;
50         if (ans < 0) ans += MOD;
51         return ans;
52     }
53     VI BM(VI s) {
54         VI C(1, 1), B(1, 1);
55         int L = 0, m = 1, b = 1;
```

```
56          rep(n, 0, SZ(s)) {
57              ll d = 0;
58              rep(i, 0, L + 1) d = (d + (ll)C[i] * s[n - i]) % MOD;
59              if (d == 0) ++m;
60              else if (2 * L <= n) {
61                  VI T = C;
62                  ll c = MOD - d * powMOD(b, MOD - 2) % MOD;
63                  while (SZ(C) < SZ(B) + m) C.pb(0);
64                  rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % MOD;
65                  L = n + 1 - L; B = T; b = d; m = 1;
66              }
67              else {
68                  ll c = MOD - d * powMOD(b, MOD - 2) % MOD;
69                  while (SZ(C) < SZ(B) + m) C.pb(0);
70                  rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % MOD;
71                  ++m;
72              }
73          }
74          return C;
75      }
76      int gao(VI a, ll n) {
77          VI c = BM(a);
78          c.erase(c.begin());
79          rep(i, 0, SZ(c)) c[i] = (MOD - c[i]) % MOD;
80          return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
81      }
82  };
83  inline void add(int &x, int y) {
84      x += y;
85      if (x >= MOD)x -= MOD;
86  }
87  int t, k, dp[MAXN];
88  vector<int> v;
89  int main() {
90      ios::sync_with_stdio(false); cin.tie(0);
91      cin >> t;
92      while (t--) {
93          v.clear();
94          cin >> k >> n;
95          if (n == -1) {
96              cout << 2 * powMOD(k + 1, MOD - 2) % MOD << '\n';
97              continue;
98          }
99          ll inv = powMOD(k, MOD - 2);
100         dp[1] = 1; v.push_back(1);
101         for (int i = 2; i <= 2 * k; i++) {
102             dp[i] = 0;
```

```
103             for (int j = max(1, i - k); j < i; j++) {
104                 add(dp[i], dp[j]);
105             }
106             dp[i] = dp[i] * inv%MOD;
107             v.push_back(dp[i]);
108         }
109         cout << linear_seq::gao(v, n) << '\n';
110     }
111     return 0;
112 }
```

第三章　字符串

3.1　KMP

```
void Get_next(char *s) {
    int j, L = strlen(s + 1);
    nxt[1] = j = 0;
    for(int i = 2; i <= L; i++) {
        while(j && s[i] != s[j + 1]) j = nxt[j] ;
        if(s[i] == s[j + 1]) j++;
        nxt[i] = j;
    }
}
int L1 = strlen(s1 + 1), L2 = strlen(s2 + 1) ;
for(int i = 1, j = 0; i <= L1; i++) {
    while(j > 0 && (j == L2 || s1[i] != s2[j + 1])) j = nxt[j] ;
    if(s1[i] == s2[j + 1]) j++;
    //if(j == L2) 此时位于末尾
}
```

3.2　扩展 KMP

求出一个串的所有后缀与另外一个串的最长公共前缀。

```
struct ExKmp{
    int Next[N];
    int extend[N];
    void get_next(char *s) {
        int len = strlen(s + 1), p = 1, pos;
        Next[1] = len;
        while(p + 1 <= len && s[p] == s[p + 1]) p++;
        Next[pos = 2] = p - 1;
        for(int i = 3; i <= len; i++) {
            int l = Next[i - pos + 1];
            if(i + l < p + 1) Next[i] = l;
            else {
                int j = max(p - i + 1, 0);
                while(i + j <= len && s[j + 1] == s[i + j]) ++j;
```

```
                p = i + (Next[pos = i] = j) - 1;
            }
        }
    }
    void work(char *s, char *t) {
        get_next(t);
        int lens = strlen(s + 1), lent = strlen(t + 1), p = 1, pos;
        while(p <= lent && s[p] == t[p]) ++p;
        p = extend[pos = 1] = p - 1;
        for(int i = 2; i <= lens; i++) {
            int len = Next[i - pos + 1];
            if(len + i < p + 1) extend[i] = len;
            else {
                int j = max(p - i + 1, 0);
                while(i + j <= lens && j <= lent && t[j + 1] == s[i + j]) j++;
                p = i + (extend[pos = i] = j) - 1;
            }
        }
    }
}exkmp;
```

3.3 马拉车算法

```
struct Manacher{
    char ch[N << 1];
    int p[N << 1];
    void work(char *s) {
        int l = 0;
        ch[l++] = '&'; ch[l++] = '#';
        for(int i = 0; s[i]; i++) {
            ch[l++] = s[i];
            ch[l++] = '#';
        }
        ch[l] = '\0';
        int mx = 0, id = 0;
        for(int i = 0; i < l; i++) {
            p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
            while(ch[i + p[i]] == ch[i - p[i]]) p[i]++;
            if(i + p[i] > mx) mx = i + p[i], id = i;
        }
    }
}Man;
```

3.4 Trie 树

```
struct Trie{
    int ch[N * 30][2], cnt[N * 30];
    int tot;
    void init() {
        tot = 0;
        ch[tot][0] = ch[tot][1] = 0;
    }
    int New_node() {
        ch[++tot][0] = ch[tot][1] = 0;
        return tot;
    }
    void Insert(int x) {
        int u = 0;
        for(register int i = 29; ~i; --i) {
            int p = ((x >> i & 1) ? 1 : 0);
            if(!ch[u][p]) ch[u][p] = New_node();
            u = ch[u][p];
            ++cnt[u];
        }
    }
}t1, t2;
```

3.5 AC 自动机

```
queue <int> q;
struct ACAM{
    int sz;
    int ch[N][MAX];
    int cnt[N], fail[N];
    void init() {
        sz = -1;
        newnode();
    }
    int newnode() {
        memset(ch[++sz], 0, sizeof(ch[sz]));
        cnt[sz] = fail[sz] = 0;
        return sz;
    }
    void insert(char *s) {
        int u = 0;
        for(int i = 1; s[i]; i++) {
            int idx = s[i] - 'a';
            if(!ch[u][idx]) ch[u][idx] = newnode();
            u = ch[u][idx];
        }
        cnt[u]++;
```

```
    }
    void build() {
        while(!q.empty()) q.pop();
        for(int i = 0; i < 26; i++) {
            if(ch[0][i]) q.push(ch[0][i]);
        }
        while(!q.empty()) {
            int cur = q.front(); q.pop();
            for(int i = 0; i < 26; i++) {
                if(ch[cur][i]) {
                    fail[ch[cur][i]] = ch[fail[cur]][i];
                    q.push(ch[cur][i]);
                } else {
                    ch[cur][i] = ch[fail[cur]][i];
                }
            }
        }
    }
}ac;
```

3.6 字符串 hash

hash 方式有多种，可以视情况来设计 hash 函数，下面给出最常见的一种：

```
typedef unsigned long long ull;
template <unsigned mod, unsigned base>
struct rolling_hash {
    unsigned int pg[N], val[N]; // val:1,2...n
    rolling_hash() {
        pg[0] = 1;
        for(int i = 1; i < N; i++) pg[i] = 1ull * pg[i - 1] * base % mod;
        val[0] = 0;
    }
    void build(const char *str) {
        for(int i = 0; str[i]; i++) {
            val[i + 1] = (str[i] + 1ull * val[i] * base) % mod;
        }
    }
    unsigned int operator() (int l, int r) {
        ++r; //
        return (val[r] - 1ull * val[l] * pg[r - l] % mod + mod) % mod;
    }
};
struct dm_hasher {
    //str:0,1...len-1
    rolling_hash<997137961, 753> h1;
    rolling_hash<1003911991, 467> h2;
```

```
    void build(const char *str) {
        h1.build(str); h2.build(str);
    }
    ull operator() (int l, int r) {
        return ull(h1(l, r)) << 32 | h2(l, r);
    }
}hasher;
```

3.7 回文自动机

一般用来求解本质不同的回文串相关问题，每个结点到根都代表一个回文后缀。与 AC 自动机有点类似。

```
namespace PAM{
    int ch[N][26], fail[N], len[N], st[N], cnt[N];
    int sz, n, last;
    int New(int l, int f) {
        memset(ch[++sz], 0, sizeof(ch[sz]));
        len[sz] = l, fail[sz] = f;
        return sz;
    }
    void init() {
        sz = -1;
        New(0, 1); last = New(-1, 0);
        st[n = 0] = -1;
        memset(cnt, 0, sizeof(cnt));
    }
    int getf(int x) {
        while(st[n - len[x] - 1] != st[n]) x = fail[x];
        return x;
    }
    bool Insert(int c) { //int
        st[++n] = c;
        int x = getf(last);
        bool F = 0;
        if(!ch[x][c]) {
            F = 1;
            int f = getf(fail[x]);
            ch[x][c] = New(len[x] + 2, ch[f][c]);
        }
        last = ch[x][c];
        cnt[last]++;
        return F;
    }
    void count() {
        for(int i = sz; i >= 1; i--) cnt[fail[i]] += cnt[i];
```

```
    }
};
```

3.8 后缀数组

```
struct SA{                                          //sa:1...n  Rank:0...n-1
    int x[N], y[N], sa[N], c[N], height[N], Rank[N];
    int f[N][20], lg[N];
    int n;                                          //length
    void da(char *s, int m){
        n++;
        for(int i = 0; i < m; i++) c[i] = 0;
        for(int i = 0; i < n; i++) c[x[i] = s[i]]++;
        for(int i = 1; i < m; i++) c[i] += c[i - 1] ;
        for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
        for(int k = 1; k <= n; k <<= 1) {
            int p = 0 ;
            for(int i = n - k; i < n; i++) y[p++] = i ;
            for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] =sa[i] - k;
            for(int i = 0; i < m; i++) c[i] = 0;
            for(int i = 0; i < n; i++) c[x[y[i]]]++;
            for(int i = 1; i < m; i++) c[i] += c[i - 1];
            for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i] ;
            swap(x , y); p = 1; x[sa[0]] = 0;
            for(int i = 1; i < n; i++)
                x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i-1] + k] == y[sa[i]
                    + k] ? p - 1 : p++;
            if(p >= n) break ;
            m = p;
        }
        n--;
        int k = 0;
        for(int i = 0; i <= n; i++) Rank[sa[i]] = i;
        for(int i = 0; i < n; i++) {
            if(k) k--;
            int j = sa[Rank[i] - 1];
            while(s[i + k] == s[j + k]) k++;
            height[Rank[i]] = k;
        }
    }
    ll count() {
        ll ans = 0;
        for(int i = 1; i <= n; i++) ans += n - sa[i] - height[i];
        return ans;
    }
    void init() {
```

```
        for(int i = 2; i < N; i++) lg[i] = lg[i >> 1] + 1;
        for(int i = 2; i <= n; i++) f[i][0] = height[i];
        for(int j = 1; j < 20; j++)
            for(int i = 2; i + (1 << j) - 1 <= n; i++)
                f[i][j] = min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]) ;
    }
    int get_lcp(int l, int r) {
        if(Rank[l] > Rank[r]) swap(l, r);
        l = Rank[l] + 1, r = Rank[r];
        int k = lg[r - l + 1];
        return min(f[l][k], f[r - (1 << k) + 1][k]);
    }
}
```

3.9 后缀自动机

以一道例题为例，做法是 $dp+$ 后缀自动机，用后缀自动机维护最远的那个位置来进行转移。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 4e5 + 5;
int p, q;
char s[N];
struct SAM{
    struct node{
        int ch[26];
        int len, fa;
        node(){memset(ch, 0, sizeof(ch)), len = 0;}
    }dian[N];
    int last, tot, now; //now为在后缀自动机上跑的一个指针
    void init(int n) {
        last = tot = now = 1;
        for(int i = 1; i <= 2 * n; i++) {
            for(int j = 0; j < 26; j++) dian[i].ch[j] = 0;
            dian[i].len = 0;
        }
    }
    void add(int c) { //核心部分
        int p = last;
        int np = last = ++tot;
        dian[np].len = dian[p].len + 1;
        for(; p && !dian[p].ch[c]; p = dian[p].fa) dian[p].ch[c] = np;
        if(!p) dian[np].fa = 1;
        else {
            int q = dian[p].ch[c];
            if(dian[q].len == dian[p].len + 1) dian[np].fa = q;
```

```
30              else {
31                  int nq = ++tot; dian[nq] = dian[q];
32                  dian[nq].len = dian[p].len + 1;
33                  dian[q].fa = dian[np].fa = nq;
34                  for(; p && dian[p].ch[c] == q; p = dian[p].fa) dian[p].ch[c] = nq
                        ;
35              }
36          }
37      }
38      void withdraw(int lens) { //越短个数越多
39          while(now && dian[dian[now].fa].len >= lens) now = dian[now].fa;
40          if(now == 0) now = 1;
41      }
42      void trans(int t, int lens) {
43          now = dian[now].ch[t];
44          withdraw(lens);
45      }
46      bool match(int t) {
47          return dian[now].ch[t];
48      }
49  }A;
50  ll dp[N];
51  int main() {
52      ios::sync_with_stdio(false); cin.tie(0);
53      while(cin >> s + 1) {
54          int n = strlen(s + 1);
55          cin >> p >> q;
56          A.init(n);
57          int l = 2, r = 1;
58          A.add(s[1] - 'a'); dp[1] = p;
59          for(int i = 2; i <= n; i++) {
60              ++r; int tmp = s[i] - 'a';
61              dp[i] = dp[i - 1] + p;
62              while((!A.match(tmp) || r - l + 1 > i / 2) && l <= r) {
63                  A.add(s[l++] - 'a');
64                  A.withdraw(r - l);
65              }
66              A.trans(tmp, r - l + 1);
67              dp[i] = min(dp[i], dp[l - 1] + q);
68          }
69          cout << dp[n] << '\n';
70      }
71      return 0;
72  }
```

3.10 最小表示法

```
int getmin(char *s){
    int n=strlen(s);
    int i=0,j=1,k=0,t;
    while(i<n && j<n && k<n){
        t=s[(i+k)%n]-s[(j+k)%n];
        if(!t) k++;
        else{
            if(t>0) i+=k+1;
            else j+=k+1;
            if(i==j) j++;
            k=0;
        }
    }
    return i<j?i:j;
}
```

第四章　数据结构

4.1　笛卡尔树

```
pii b[N];
struct Cartesian_Tree{
    struct node{
        int id, val, fa;
        int son[2];
        node(){}
        node(int id, int val, int fa) : id(id), val(val), fa(fa) {
            son[0] = son[1] = 0;
        }
    }tr[N];
    int rt;
    void init() {
        tr[0] = node(0, 1e9, 0);
    }
    void build(int n, int *a) {
        for(int i = 1; i <= n; i++) tr[i] = (i, a[i], 0);
        for(int i = 1; i <= n; i++) {
            int k = i - 1;
            while(tr[k].val < tr[i].val) k = tr[k].fa;
            tr[i].son[0] = tr[k].son[1];
            tr[k].son[1] = i;
            tr[i].fa = k;
            tr[tr[i].son[0]].fa = i;
        }
        rt = tr[0].son[1];
    }
    int dfs(int u) {
        if(!u) return 0;
        int lsz = dfs(tr[u].son[0]);
        int rsz = dfs(tr[u].son[1]);
        b[tr[u].id].fi = lsz;
        b[tr[u].id].se = rsz;
        return lsz + rsz + 1;
    }
}CT;
```

4.2 线性基

普通线性基就不说了，给出线性基求交集的模板。

求交的话大概就是对于两个基集合 B_1, B_2, 枚举 B_2 中的基，如果与 B_1 线性无关，那么就插在 B_1 里面去；否则就对于当前的基，异或掉 B_1 中之前插进去的 B_2 的基，然后将其插入交集里面就行了。

线段树维护线性基交：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned int ui;
const int N = 50005;
int n, m;
struct node{
    ui r[32], f[32];
    bool ins(ui x) {
        for(int i = 31; i >= 0; i--) {
            if(x >> i) {
                if(!r[i]) {r[i] = x; return 1;}
                else x ^= r[i];
            }
        }
        return 0;
    }
    bool ins2(ui x) {
        ui tmp = x;
        for(int i = 31; i >= 0; i--) {
            if(x >> i) {
                if(!r[i]) {r[i] = x; f[i] = tmp; return 1;}
                else {
                    x ^= r[i]; tmp ^= f[i];
                }
            }
        }
        return 0;
    }
    void clear() {
        for(int i = 0; i <= 31; i++) r[i] = f[i] = 0;
    }
    bool find(ui x) {
        for(int i = 31; i >= 0; i--) {
            if(x >> i) {
                if(!r[i]) return 0;
                x ^= r[i];
            }
        }
```

```
40          return x == 0;
41      }
42      int calc(ui x) {
43          int ans = 0;
44          for(int i = 31; i >= 0; i--) {
45              if(x >> i) {
46                  x ^= r[i];
47                  ans ^= f[i];
48              }
49          }
50          return ans;
51      }
52  };
53  node _merge(node u, node v) {
54      node tmp, res; res.clear();
55      tmp = u;
56      for(int i = 31; i >= 0; i--) {
57          ui x = v.r[i];
58          if(tmp.find(x)) {
59              res.ins(x ^ tmp.calc(x));
60          } else tmp.ins2(x);
61      }
62      return res;
63  }
64  ui a[N][33];
65  node b[N << 2];
66  void build(int o, int l, int r) {
67      if(l == r) {
68          b[o].clear();
69          for(int j = 1; j <= 32; j++) b[o].ins(a[l][j]);
70          return ;
71      }
72      int mid = (l + r) >> 1;
73      build(o << 1, l, mid); build(o << 1|1, mid + 1, r);
74      b[o] = _merge(b[o << 1], b[o << 1|1]);
75  }
76  bool query(int o, int l, int r, int L, int R, ui v) {
77      if(L <= l && r <= R) {
78          return b[o].find(v);
79      }
80      int mid = (l + r) >> 1;
81      bool ans = 1;
82      if(L <= mid) ans &= query(o << 1, l, mid, L, R, v);
83      if(R > mid) ans &= query(o << 1|1, mid + 1, r, L, R, v) ;
84      return ans;
85  }
86  int main() {
```

```
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        int k; cin >> k;
        for(int j = 1; j <= k; j++) cin >> a[i][j];
    }
    build(1, 1, n);
    for(int i = 1, l, r; i <= m; i++) {
        ui x; cin >> l >> r >> x;
        if(query(1, 1, n, l, r, x)) cout << "YES" << '\n';
        else cout << "NO" << '\n';
    }
    return 0;
}
```

4.3 李超树

题意：现在需要维护两种操作，一种是插入一条线段，另一种是询问 $x = k$ 时，最上方线段的编号，如有多个线段处于最大值状态，那么就输出编号最小的。强制在线。

李超树模板题，注意其标记可持久化。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 5, MAX = 50005;
int n;

struct node{
    bool sign; int id;
    double k, b;
    void upd(int _id, double _k, double _b) {
        id = _id, k = _k, b = _b;
    }
}tr[MAX << 3];

void update(int o, int l, int r, int id, double k, double b) {
    if(!tr[o].sign) {
        tr[o].sign = 1;
        tr[o].upd(id, k, b);
        return;
    }
    double l1 = l * tr[o].k + tr[o].b, l2 = l * k + b;
    double r1 = r * tr[o].k + tr[o].b, r2 = r * k + b;
    if(l2 <= l1 && r2 <= r1) return;
    if(l2 > l1 && r2 > r1) {
        tr[o].upd(id, k, b); return;
```

```
26      }
27      int mid = (l + r) >> 1;
28      double x = (tr[o].b - b) / (k - tr[o].k);
29      if(x <= mid) {
30          if(l1 > l2) update(o << 1, l, mid, tr[o].id, tr[o].k, tr[o].b), tr[o].upd
                (id, k, b);
31          else update(o << 1, l, mid, id, k, b);
32      } else {
33          if(l1 > l2) update(o << 1|1, mid + 1, r, id, k, b);
34          else update(o << 1|1, mid + 1, r, tr[o].id, tr[o].k, tr[o].b), tr[o].upd(
                id, k, b);
35      }
36  }
37
38  void update(int o, int l, int r, int L, int R, int id, double k, double b) {
39      if(L <= l && r <= R) {
40          update(o, l, r, id, k, b); return;
41      }
42      int mid = (l + r) >> 1;
43      if(L <= mid) update(o << 1, l, mid, L, R, id, k, b);
44      if(R > mid) update(o << 1|1, mid + 1, r, L, R, id, k, b);
45  }
46
47  void chk(int &res, int o1, int o2, int x) {
48      double y1, y2;
49      y1 = x * tr[o1].k + tr[o1].b;
50      y2 = x * tr[o2].k + tr[o2].b;
51      if(y1 > y2 && tr[o1].sign) res = o1;
52      else if(y1 < y2 && tr[o2].sign) res = o2;
53      else if(y1 == y2){
54          if(tr[o1].id < tr[o2].id && tr[o1].sign) res = o1;
55          else if(tr[o1].id > tr[o2].id && tr[o2].sign) res = o2;
56      }
57      return;
58  }
59
60  int query(int o, int l, int r, int p) {
61      if(l == r) return o;
62      int mid = (l + r) >> 1, res = 0;
63      if(p <= mid) chk(res, query(o << 1, l, mid, p), o, p);
64      else chk(res, query(o << 1|1, mid + 1, r, p), o, p);
65      return res;
66  }
67
68  int main() {
69  //    freopen("input.in", "r", stdin);
70      scanf("%d", &n);
```

```
    int lastans = 0, cnt = 0;
    for(int i = 1; i <= n; i++) {
        int op; scanf("%d", &op);
        if(op == 0) {
            int k; scanf("%d", &k);
            int x = (k + lastans - 1) % 39989 + 1;
            int o = query(1, 1, MAX, x);
            lastans = tr[o].id;
            printf("%d\n", lastans);
        } else {
            int x1, x2, y1, y2; ++cnt;
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            x1 = (x1 + lastans - 1) % 39989 + 1;
            y1 = (y1 + lastans - 1) % 1000000000 + 1;
            x2 = (x2 + lastans - 1) % 39989 + 1;
            y2 = (y2 + lastans - 1) % 1000000000 + 1;
            if(x1 > x2) swap(x1, x2), swap(y1, y2);
            double k = 1.0 * (y1 - y2) / (x1 - x2), b = y1 - k * x1;
            update(1, 1, MAX, x1, x2, cnt, k, b);
        }
    }
    return 0;
}
```

4.4 CDQ 分治

题意：现有两种操作，插入和查询，插入操作则插入一个点 (x, y, z)，查询操作给出两个点 $(x1, y1, z1), (x2, y2, z2)$，回答满足 $x1\ x\ x2, y1\ y\ y2, z1\ z\ z2$ 的 (x,y,z) 的个数为多少。

带修改的四维偏序，四维分别为时间、x、y、z，直接 cdq 套 cdq 就行。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 50005;

int T, q;

struct node{
    int x, y, z, op, id, part;
}a[N * 10], b[N * 10], d[N * 10];

int hs[N << 2];

bool isq[N];
int ans[N];

```

```
int c[N << 2];
int lowbit(int x) {return x & (-x);}

void update(int x, int v) {
    for(; x < N << 2; x += lowbit(x)) c[x] += v;
}

int query(int x) {
    int ans = 0;
    for(; x; x -= lowbit(x)) ans += c[x];
    return ans;
}

void cdq2(int l, int r) {
    if(l == r) return ;
    int mid = (l + r) >> 1;
    cdq2(l, mid); cdq2(mid + 1, r);
    int t1 = l, t2 = mid + 1;
    for(int i = l; i <= r; i++) {
        if(t2 > r || (t1 <= mid && b[t1].y <= b[t2].y)) {
            if(b[t1].part == 0 && b[t1].op == 0) {
                update(b[t1].z, 1);
            }
            d[i] = b[t1++];
        } else {
            if(b[t2].part && b[t2].op != 0) {
                ans[b[t2].id] += b[t2].op * query(b[t2].z);
            }
            d[i] = b[t2++];
        }
    }
    for(int i = l; i <= mid; i++) {
        if(b[i].part == 0 && b[i].op == 0) update(b[i].z, -1);
    }
    for(int i = l; i <= r; i++) b[i] = d[i];
}

void cdq(int l, int r) {
    if(l == r) return ;
    int mid = (l + r) >> 1;
    cdq(l, mid); cdq(mid + 1, r);
    int t1 = l, t2 = mid + 1;
    for(int i = l; i <= r; i++) {
        if(t2 > r || (t1 <= mid && a[t1].x <= a[t2].x)) {
            b[i] = a[t1++];
            b[i].part = 0;
        } else {
```

```
            b[i] = a[t2++];
            b[i].part = 1;
        }
    }
    for(int i = l; i <= r; i++) a[i] = b[i];
    cdq2(l, r);
}

int main() {
//    freopen("input.in", "r", stdin);
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> T;
    while(T--) {
        cin >> q;
        for(int i = 1; i <= q; i++) ans[i] = 0, isq[i] = false;
        int cnt = 0; hs[0] = 0;
        for(int i = 1; i <= q; i++) {
            int op; cin >> op;
            if(op == 1) {
                int x, y, z; cin >> x >> y >> z;
                a[++cnt] = {x, y, z, 0, i, -1};
                hs[++hs[0]] = z;
            } else {
                isq[i] = true;
                int x1, y1, z1, x2, y2, z2;
                cin >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
                --x1, --y1, --z1;
                a[++cnt] = {x1, y1, z1, -1, i, -1};
                a[++cnt] = {x1, y2, z1, 1, i, -1};
                a[++cnt] = {x2, y2, z1, -1, i, -1};
                a[++cnt] = {x2, y1, z1, 1, i, -1};
                a[++cnt] = {x1, y1, z2, 1, i, -1};
                a[++cnt] = {x1, y2, z2, -1, i, -1};
                a[++cnt] = {x2, y2, z2, 1, i, -1};
                a[++cnt] = {x2, y1, z2, -1, i, -1};
                hs[++hs[0]] = z1, hs[++hs[0]] = z2;
            }
        }
        sort(hs + 1, hs + hs[0] + 1);
        hs[0] = unique(hs + 1, hs + hs[0] + 1) - hs - 1;
        for(int i = 1; i <= cnt; i++) a[i].z = lower_bound(hs + 1, hs + hs[0] +
            1, a[i].z) - hs;
        cdq(1, cnt);
        for(int i = 1; i <= q; i++) {
            if(isq[i]) cout << ans[i] << '\n';
        }
    }
```

```
110     return 0;
111 }
```

4.5 并查集

线段树维护可撤销并查集。

因为可撤销，所以不能路径压缩，注意启发式合并。

```
1  #include <bits/stdc++.h>
2  #define MP make_pair
3  using namespace std;
4  typedef long long ll;
5  typedef pair<int, int> pii;
6  const int N = 2e5 + 5;
7  int n, m;
8  struct node{
9      int u, v, l, r;
10 }a[N];
11 int b[N], D;
12 vector <int> c[N << 2];
13 vector <pair<int, int> > d[100];
14 ll ans;
15 void insert(int o, int l, int r, int L, int R, int id) {
16     if(L <= l && r <= R) {
17         c[o].push_back(id);
18         return;
19     }
20     int mid = (l + r) >> 1;
21     if(L <= mid) insert(o << 1, l, mid, L, R, id);
22     if(R > mid) insert(o << 1|1, mid + 1, r, L, R, id);
23 }
24 int f[N], sz[N];
25 int find(int x) {
26     return f[x] == x ? x : find(f[x]);
27 }
28 void merge(int x, int y, int dep) {
29     int fx = find(x), fy = find(y);
30     if(fx == fy) return;
31     if(sz[fx] > sz[fy]) swap(fx, fy);
32     int tmp = 0;
33     f[fx] = fy;
34     if(sz[fx] == sz[fy]) tmp++;
35     sz[fy] += tmp;
36     d[dep].push_back(MP(fx, tmp));
37 }
38 void del(int dep) {
```

```
    for(auto it : d[dep]) {
        sz[f[it.first]] -= it.second;
        f[it.first] = it.first;
    }
    d[dep].clear();
}
void dfs(int o, int l, int r, int dep) {
    for(auto it : c[o]) {
        merge(a[it].u, a[it].v, dep);
    }
    if(find(1) == find(n)) {
        ans += b[r + 1] - b[l];
    } else if(l < r) {
        int mid = (l + r) >> 1;
        dfs(o << 1, l, mid, dep + 1);
        dfs(o << 1|1, mid + 1, r, dep + 1);
    }
    del(dep);
}
int main() {
    ios::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        cin >> a[i].u >> a[i].v >> a[i].l >> a[i].r;
        b[++D] = a[i].l, b[++D] = a[i].r + 1;
    }
    sort(b + 1, b + D + 1);
    D = unique(b + 1, b + D + 1) - b - 1;
    for(int i = 1; i <= m; i++) {
        a[i].l = lower_bound(b + 1, b + D + 1, a[i].l) - b;
        a[i].r = lower_bound(b + 1, b + D + 1, a[i].r + 1) - b - 1;
        insert(1, 1, D, a[i].l, a[i].r, i);
    }
    for(int i = 1; i <= n; i++) f[i] = i, sz[i] = 1;
    dfs(1, 1, D, 1);
    cout << ans;
    return 0;
}
```

4.6 kd-tree

一种多维二叉搜索树，每一层有不同的排序规则。

寻找最近点、最远点之类的复杂度为 $O(logn)$ $O(n)$，而矩阵查询、修改之类的复杂度一般为 $O(n\sqrt{n})$。

矩阵查询，单点修改，复杂度都为 $O(\sqrt{n})$

```
int D;
struct Point {
    int d[2];
    Point(int x = 0, int y = 0) {
        d[0] = x, d[1] = y;
    }
    int& operator[] (int x) {return d[x];}
}p[N];
struct Node{
    int mn[2], mx[2];
    int l, r, col;
    bool lz;
    Point t;
}tr[N];
bool operator < (const Point &A, const Point &B) {
    return A.d[D] < B.d[D];
}
bool operator == (const Point &A, const Point &B) {
    return A.d[0] == B.d[0] && A.d[1] == B.d[1];
}
int rt;
struct kdtree {
    void push_up(int o) {
        int ls = tr[o].l, rs = tr[o].r;
        for(int i = 0; i < 2; i++) {
            tr[o].mn[i] = tr[o].mx[i] = tr[o].t[i];
            if(ls) {
                tr[o].mn[i] = min(tr[o].mn[i], tr[ls].mn[i]);
                tr[o].mx[i] = max(tr[o].mx[i], tr[ls].mx[i]);
            }
            if(rs) {
                tr[o].mn[i] = min(tr[o].mn[i], tr[rs].mn[i]);
                tr[o].mx[i] = max(tr[o].mx[i], tr[rs].mx[i]);
            }
        }
    }
    void push_down(int o) {
        if(tr[o].lz) {
            if(tr[o].l) {
                tr[tr[o].l].lz = true;
                tr[tr[o].l].col = tr[o].col;
            }
            if(tr[o].r) {
                tr[tr[o].r].lz = true;
                tr[tr[o].r].col = tr[o].col;
            }
            tr[o].lz = false;
```

```
48          }
49      }
50      int build(int l, int r, int now) {
51          D = now;
52          int mid = (l + r) >> 1;
53          nth_element(p + l, p + mid, p + r + 1);
54          tr[mid].t = p[mid];
55          tr[mid].lz = false;
56          tr[mid].col = 1;
57          if(l < mid) tr[mid].l = build(l, mid - 1, now ^ 1);
58          else tr[mid].l = 0;
59          if(r > mid) tr[mid].r = build(mid + 1, r, now ^ 1);
60          else tr[mid].r = 0;
61          push_up(mid);
62          return mid;
63      }
64      int query(int o, Point T, int now) {
65          if(o == 0) return 0;
66          if(tr[o].t == T) return tr[o].col;
67          push_down(o);
68          D = now;
69          if(T.d[D] < tr[o].t.d[D]) return query(tr[o].l, T, now ^ 1);
70          else return query(tr[o].r, T, now ^ 1);
71      }
72      void update(int o, int l, int r, int d, int u, int c) {
73          if(tr[o].mn[0] >= l && tr[o].mx[0] <= r && tr[o].mn[1] >= d && tr[o].mx
                [1] <= u) {
74              tr[o].col = c; tr[o].lz = true;
75              return;
76          }
77          if(tr[o].mn[0] > r || tr[o].mx[0] < l || tr[o].mn[1] > u || tr[o].mx[1] <
                 d) return;
78          push_down(o);
79          if(tr[o].t[0] >= l && tr[o].t[0] <= r && tr[o].t[1] >= d && tr[o].t[1] <=
                 u) {
80              tr[o].col = c;
81          }
82          if(tr[o].l) update(tr[o].l, l, r, d, u, c);
83          if(tr[o].r) update(tr[o].r, l, r, d, u, c);
84      }
85  }kd;
```

矩阵查询，支持修改和树的重构（设立一个阀值），区间查询，动态开点。

```
1   int n;
2   int D;
3   struct Point {
4       int d[2], val;
```

```
}tmp[N], T;
struct Node {
    int mn[2], mx[2];
    int l, r, sumv, sz;
    Point t;
}tr[N];
bool operator < (const Point &A, const Point &B) {
    return A.d[D] < B.d[D];
}
int rt;
int rub[N], top, tot;
struct kdtree {
    const double E = 0.75;
    int ans;
    int new_node() {
        if(top) return rub[top--];
        return ++tot;
    }
    void push_up(int o) {
        int ls = tr[o].l, rs = tr[o].r;
        for(int i = 0; i < 2; i++) {
            tr[o].mn[i] = tr[o].mx[i] = tr[o].t.d[i];
            if(ls) {
                tr[o].mn[i] = min(tr[o].mn[i], tr[ls].mn[i]);
                tr[o].mx[i] = max(tr[o].mx[i], tr[ls].mx[i]);
            }
            if(rs) {
                tr[o].mn[i] = min(tr[o].mn[i], tr[rs].mn[i]);
                tr[o].mx[i] = max(tr[o].mx[i], tr[rs].mx[i]);
            }
        }
        tr[o].sumv = tr[ls].sumv + tr[rs].sumv + tr[o].t.val;
        tr[o].sz = 1 + tr[ls].sz + tr[rs].sz;
    }
    void pia(int o, int num) {
        int ls = tr[o].l, rs = tr[o].r;
        if(ls) pia(ls, num);
        tmp[tr[ls].sz + num + 1] = Point{tr[o].t.d[0], tr[o].t.d[1], tr[o].t.val
            };
        rub[++top] = o;
        if(rs) pia(rs, tr[ls].sz + num + 1);
    }
    int rebuild(int l, int r, int now) {
        if(l > r) return 0;
        D = now;
        int mid = (l + r) >> 1;
        nth_element(tmp + l, tmp + mid, tmp + r + 1);
```

```
51          int node = new_node();
52          tr[node].t = tmp[mid];
53          tr[node].l = rebuild(l, mid - 1, now ^ 1);
54          tr[node].r = rebuild(mid + 1, r, now ^ 1);
55          push_up(node);
56          return node;
57      }
58      void chk(int &o, int now) {
59          if(tr[o].sz * E <= tr[tr[o].l].sz || tr[o].sz * E <= tr[tr[o].r].sz) {
60              pia(o, 0);
61              o = rebuild(1, tr[o].sz, now);
62          }
63      }
64      void insert(int &o, int now) {
65          if(!o) {
66              tr[o = new_node()].t = T;
67              tr[o].l = tr[o].r = 0;
68              push_up(o);
69              return;
70          }
71          D = now;
72          if(tr[o].t.d[D] < T.d[D]) insert(tr[o].r, now ^ 1);
73          else insert(tr[o].l, now ^ 1);
74          push_up(o);
75          chk(o, now);
76      }
77      bool in(int x, int y, int x1, int y1, int x2, int y2) {
78          return x >= x1 && x <= x2 && y >= y1 && y <= y2;
79      }
80      void query(int o, int x1, int y1, int x2, int y2) {
81          if(o == 0) return;
82          if(tr[o].mn[0] >= x1 && tr[o].mx[0] <= x2 && tr[o].mn[1] >= y1 && tr[o].
                mx[1] <= y2) {
83              ans += tr[o].sumv;
84              return;
85          }
86          if(tr[o].mn[0] > x2 || tr[o].mx[0] < x1 || tr[o].mn[1] > y2 || tr[o].mx
                [1] < y1) return;
87          if(in(tr[o].t.d[0], tr[o].t.d[1], x1, y1, x2, y2)) ans += tr[o].t.val;
88          query(tr[o].l, x1, y1, x2, y2);
89          query(tr[o].r, x1, y1, x2, y2);
90      }
91  }kd;
```

第五章　其它

5.1　莫队算法

对区间问题离线处理，按照一定规则排序，之后指针暴力移动就行。核心代码：

```
for(; r < q[i].r; r++) add(r + 1, 1);
for(; r > q[i].r; r--) add(r, -1);
for(; l < q[i].l; l++) add(l, -1);
for(; l > q[i].l; l--) add(l - 1, 1);
for(; t < q[i].k; t++) Update(upd[t + 1]);
for(; t > q[i].k; t--) Update(upd[t]);
```

5.2　点分治

优雅的暴力，对于一颗无根树每次以树的重心为根容斥计算答案，本质相当于枚举每一个结点作为路径上面的必经点然后来考虑。复杂度一般为 $O(nlogn)$。

寻找两点之间路径为 3 的倍数的代码：

```
#include <bits/stdc++.h>
#define MP make_pair
#define fi first
#define se second
#define INF 0x3f3f3f3f
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int N = 20005;
int n;
vector <pii> g[N];
int sz[N], mx[N], tsz;
int d[N];
int rt, ans, cnt;
bool vis[N];
void getrt(int u, int fa) {
    sz[u] = 1; mx[u] = 0;
    for(auto it : g[u]) {
        int v = it.fi;
        if(v == fa || vis[v]) continue;
```

```
21          getrt(v, u);
22          sz[u] += sz[v];
23          if(sz[v] > mx[u]) mx[u] = sz[v];
24      }
25      mx[u] = max(mx[u], tsz - sz[u]);
26      if(mx[u] < mx[rt]) rt = u;
27  }
28  void dfs2(int u, int D, int fa) {
29      d[++cnt] = D;
30      for(auto it : g[u]) {
31          int v = it.fi, w = it.se;
32          if(vis[v] || v == fa) continue;
33          dfs2(v, D + w, u);
34      }
35  }
36  int calc() {
37      for(int i = 1; i <= cnt; i++) d[i] %= 3;
38      int tmp[3] = {0, 0, 0};
39      for(int i = 1; i <= cnt; i++) ++tmp[d[i]];
40      int ans = tmp[0] * (tmp[0] - 1) + 2 * (tmp[1] * tmp[2]);
41      return ans;
42  }
43  void dfs(int u) {
44      vis[u] = 1;
45      cnt = 0; dfs2(u, 0, 0);
46      int tmp = calc();
47      ans += tmp;
48      for(auto it : g[u]) {
49          int v = it.fi, w = it.se;
50          if(vis[v]) continue;
51          cnt = 0, dfs2(v, w, 0);
52          ans -= calc();
53          tsz = sz[v], rt = 0, getrt(v, u);
54          dfs(rt);
55      }
56  }
57  int gcd(int a, int b) {
58      return b == 0 ? a : gcd(b, a % b);
59  }
60  int main() {
61      ios::sync_with_stdio(false); cin.tie(0);
62      cin >> n;
63      for(int i = 1; i < n; i++) {
64          int u, v, w; cin >> u >> v >> w;
65          g[u].push_back(MP(v, w)); g[v].push_back(MP(u, w));
66      }
67      tsz = n, mx[0] = INF, getrt(1, 0);
```

```
    dfs(rt);
    ans += n;
    int tot = n * n;
    int g = gcd(tot, ans);
    ans /= g, tot /= g;
    cout << ans << '/' << tot;
    return 0;
}
```

5.3 随机数

```
mt19937 rnd(time(NULL));
rnd()%n;
```

5.4 最大子矩阵

```
int a[N][N];
int n, m;
#define mp make_pair
#define se second
#define fi first
namespace max_matrix{
    int vis[N], f[N], sz[N];
    int t[N][N];
    int mx = 0, mx2 = 0;
    pair<int, int> h[N];
    int find(int x) {
        return f[x] == x ? x : f[x] = find(f[x]) ;
    }
    void Union(int x, int y) {
        int fx = find(x), fy = find(y);
        f[fx] = fy;
        sz[fy] += sz[fx];
    }
    void solve(int a, int b) {
        int area = a * b;
        if(area > mx) mx2 = mx, mx = area;
        else if(area > mx2) mx2 = area;
    }
    int work(int n, int m, int a[][N]) {
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++) {
                t[i][j] = (a[i][j] == 1) ? t[i - 1][j] + 1 : 0;
            }
        for(int i = 1; i <= n; i++) {
```

```
            for(int j = 1; j <= m; j++) {
                f[j] = j, vis[j] = 0, h[j] = mp(t[i][j], j), sz[j] = 1;
            }
            sort(h + 1, h + m + 1);
            for(int j = m; j >= 1; j--) {
                int k = h[j].se; vis[k] = 1;
                if(vis[k - 1]) Union(k - 1, k);
                if(vis[k + 1]) Union(k + 1, k);
                int len = sz[find(k)];
                solve(len, h[j].fi);
                solve(len - 1, h[j].fi);
                solve(len, h[j].fi - 1);
            }
        }
        return mx2;
    }
};
```

5.5 快速读入/输出

```
struct Istream {
  template <class T>
  Istream &operator >>(T &x) {
    static char ch;static bool neg;
    for(ch=neg=0;ch<'0' || '9'<ch;neg|=ch=='-',ch=getchar());
    for(x=0;'0'<=ch && ch<='9';(x*=10)+=ch-'0',ch=getchar());
    x=neg?-x:x;
    return *this;
  }
}fin;

struct Ostream {
  template <class T>
  Ostream &operator <<(T x) {
    x<0 && (putchar('-'),x=-x);
    static char stack[233];static int top;
    for(top=0;x;stack[++top]=x%10+'0',x/=10);
    for(top==0 && (stack[top=1]='0');top;putchar(stack[top--]));
    return *this;
  }

  Ostream &operator <<(char ch) {
    putchar(ch);
    return *this;
  }
}fout;
```

5.6 二进制函数

```
__builtin_popcount(); //1的个数
__builtin_parity(); //1个数的奇偶
__builtin_ffs(); //最后一个1的位置
```

5.7 第三章

5.7.1 第一节 新民主主义到社会主义的转变

kkkk

latex