The template

for ACM ICPC

Edited by

heyuhhh

Team 懂的都懂 at Sicau

2020.10

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。

不过在二分图中，用 Dinic 求最大匹配时间复杂度为 $O(n\sqrt{n})$。

```
#define INF 0x3f3f3f3f
template <class T>
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() {}
```

```
5   edge(int to, int _capacity, int _cost, int _rev) :to(to), capacity(_capacity),
        cost(_cost), rev(_rev) {}
6 };
7 struct Min_Cost_Max_Flow {
8   int V, H[N << 1], dis[N << 1], PreV[N << 1], PreE[N << 1];
9   vector<edge> G[N << 1];
10  void Init(int n) {
11    V = n;
12    for (int i = 0; i <= V; ++i)G[i].clear();
13  }
14  void Add_Edge(int from, int to, int cap, int cost) {
15    G[from].push_back(edge(to, cap, cost, G[to].size()));
16    G[to].push_back(edge(from, 0, -cost, G[from].size() - 1));
17  }
18 //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
19  int Min_cost_max_flow(int s, int t, int f, int& flow) {
20    int res = 0; fill(H, H + 1 + V, 0);
21    while (f) {
22      priority_queue <pair<int, int>, vector<pair<int, int>>, greater<pair<int,
            int>> > q;
23      fill(dis, dis + 1 + V, INF);
24      dis[s] = 0; q.push(pair<int, int>(0, s));
25      while (!q.empty()) {
26        pair<int, int> now = q.top(); q.pop();
27        int v = now.second;
28        if (dis[v] < now.first)continue;
29        for (int i = 0; i < G[v].size(); ++i) {
30          edge& e = G[v][i];
31          if (e.capacity > 0 && dis[e.to] > dis[v] + e.cost + H[v] - H[e.to]) {
32            dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];
33            PreV[e.to] = v;
34            PreE[e.to] = i;
35            q.push(pair<int, int>(dis[e.to], e.to));
36          }
37        }
38      }
39      if (dis[t] == INF)break;
40      for (int i = 0; i <= V; ++i)H[i] += dis[i];
41      int d = f;
42      for (int v = t; v != s; v = PreV[v])d = min(d, G[PreV[v]][PreE[v]].capacity
            );
43      f -= d; flow += d; res += d*H[t];
44      for (int v = t; v != s; v = PreV[v]) {
45        edge& e = G[PreV[v]][PreE[v]];
46        e.capacity -= d;
47        G[v][e.rev].capacity += d;
48      }
```

```
    }
    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;
```

不过之前的板子在某些情况下可能会存在问题，一般对时间要求不是很严格的情况可以用下面的板子：

```
struct E {
    int from, to, cp, v;
    E() {}
```

```
4       E(int f, int t, int cp, int v) : from(f), to(t), cp(cp), v(v) {}
5   };
6
7   struct MCMF {
8       int n, m, s, t;
9       vector<E> edges;
10      vector<int> G[N];
11      bool inq[N];
12      int d[N], p[N], a[M];
13
14      void init(int _n, int _s, int _t) {
15          n = _n; s = _s; t = _t;
16          for(int i = 0; i <= n; i++) G[i].clear();
17          edges.clear(); m = 0;
18      }
19
20      void addedge(int from, int to, int cap, int cost) {
21          edges.emplace_back(from, to, cap, cost);
22          edges.emplace_back(to, from, 0, -cost);
23          G[from].push_back(m++);
24          G[to].push_back(m++);
25      }
26
27      bool BellmanFord(int &flow, int &cost) {
28          for(int i = 0; i <= n; i++) d[i] = INF;
29          memset(inq, 0, sizeof inq);
30          d[s] = 0, a[s] = INF, inq[s] = true;
31          queue<int> Q; Q.push(s);
32          while (!Q.empty()) {
33              int u = Q.front(); Q.pop();
34              inq[u] = false;
35              for (int& idx: G[u]) {
36                  E &e = edges[idx];
37                  if (e.cp && d[e.to] > d[u] + e.v) {
38                      d[e.to] = d[u] + e.v;
39                      p[e.to] = idx;
40                      a[e.to] = min(a[u], e.cp);
41                      if (!inq[e.to]) {
42                          Q.push(e.to);
43                          inq[e.to] = true;
44                      }
45                  }
46              }
47          }
48          if (d[t] == INF) return false;
49          flow += a[t];
50          cost += a[t] * d[t];
```

```
        int u = t;
        while (u != s) {
            edges[p[u]].cp -= a[t];
            edges[p[u] ^ 1].cp += a[t];
            u = edges[p[u]].from;
        }
        return true;
    }

    int go() {
        int flow = 0, cost = 0;
        while (BellmanFord(flow, cost));
        return cost;
    }
} MM;
```

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];
```

```
32                  }
33                  if(vis[u]!=i || u==root) continue;      //判断是否形成环
34                  for(int v=pre[u];v!=u;v=pre[v] )
35                      id[v]=idx;
36                  id[u] = idx++;
37              }
38          }
39          if(idx==0) break;
40          for(int i=0;i<n;i++){
41              if(id[i]==-1) id[i]=idx++;
42          }
43          for(int i=1;i<=m;i++){
44              e[i].w-=in[e[i].v];
45              e[i].u=id[e[i].u];
46              e[i].v=id[e[i].v];
47          }
48          n = idx;
49          root = id[root];//给根新的标号
50      }
51      return ans;
52  }
```

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，注意这里的邻接矩阵为广义的邻接矩阵，重边会计算多次。

```
1   ll b[N][N];
2   int g[N][N];
3   ll Det(int n){
4       int i,j,k;
5       ll ret = 1;
6       for(i=2;i<=n;i++){
7           for(j = i+1;j <= n;j++){
8               while(b[j][i]){
9                   ll tmp=b[i][i]/b[j][i];//不存在除不尽的情况
10                  for(k = i;k <= n;k++){
11                      b[i][k] = (b[i][k] - tmp*b[j][k])%MOD;
12                      if(b[i][k]<0) b[i][k]+=MOD;
13                  }
14                  swap(b[i],b[j]);
15                  ret = -ret;
16              }
17          }
18          if(!b[i][i]) return -1;
19          ret = ret * b[i][i]%MOD;
20      }
```

```
21      if(ret < 0) ret += MOD;
22      return ret;
23  }
```

1.5 割点及桥

1.5.1 求出割点及桥

```
1   void init(){
2       T=0;tot=0;
3       memset(head,-1,sizeof(head));
4       memset(cut,0,sizeof(cut));
5       memset(dfn,0,sizeof(dfn));
6       memset(bri,0,sizeof(bri));
7   }
8   void Tarjan(int u,int pre){
9       dfn[u]=low[u]=++T;
10      int son=0;
11      for(int i=head[u];i!=-1;i=e[i].next){
12          int v=e[i].v;
13          if(v==pre) continue ;
14          if(!dfn[v]){
15              son++;//起点有效儿子
16              Tarjan(v,u);
17              low[u]=min(low[u],low[v]);
18              if(low[v]>=dfn[u]&&u!=pre)cut[u]=1;
19              if(low[v]>dfn[u]){
20                  bri[i]=1;bri[i^1]=1;
21              }
22          }else{
23              low[u]=min(low[u],dfn[v]);
24          }
25      }
26      if(u==pre && son>1) cut[u]=1;
27  }
```

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。

```
1   int T,tot,cnt;
2   int dfn[N],low[N],cut[N],num[N],f[N];
3   void adde(int u,int v){
4       e[tot].u=u;e[tot].v=v;e[tot].next=head[u];head[u]=tot++;
5   }
6   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{
            low[u]=min(low[u],dfn[v]);
        }
        if(low[v]>dfn[u]){
            cut[v]=1;//割点
        }else Union(u,v);
    }
}
```

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，时间复杂度为 $O(nm)$，也可以网络流做，效率更高：

```
struct MaxMatch {
    int n, m;
    vector<int> G[N];
```

```
    int vis[N], Match[N], clk;

    void init(int n, int m) {
        this->n = n;
        this->m = m;
        for (int i = 1; i <= n; i++) G[i].clear();
        fill(vis + 1, vis + m + 1, -1);
        fill(Match + 1, Match + m + 1, -1);
    }

    void adde(int u, int v) {
        G[u].push_back(v);
    }

    bool dfs(int u) {
        for (int v: G[u])
            if (vis[v] != clk) {
                vis[v] = clk;
                if (Match[v] == -1 || dfs(Match[v])) {
                    Match[v] = u;
                    return true;
                }
            }
        return false;
    }

    int solve() {
        int res = 0;
        for (int i = 1; i <= n; i++, ++clk) {
            res += dfs(i);
        }
        return res;
    }
} MM;
```

1.6.2 最大权匹配

KM 算法，复杂度为 $O(n^3)$。

以防到时候看不太懂，贴上两份代码供参考。

```
int cx, cy, ct;// ct为左右两部点数量的最大值
int vx[N], vy[N], w[N][N];
int lnk[N], pre[N], slk[N];
bool vis[N];

void bfs(int x, int y = 0) {
    lnk[0] = x, memset(pre, 0, (ct + 1) * sizeof(int));
```

```
    memset(slk, 63, (ct + 1) * sizeof(int));
    memset(vis, false, (ct + 1) * sizeof(bool));
    for(int ny, mi; lnk[y]; y = ny) {
        x = lnk[y], mi = INF, vis[y] = true;
        for(int i = 1; i <= ct; i++) {
            if(vis[i]) continue;
            if(slk[i] > vx[x] + vy[i] - w[x][i])
                slk[i] = vx[x] + vy[i] - w[x][i], pre[i] = y;
            if(slk[i] < mi) mi = slk[i], ny = i;
        }
        for(int i = 0; i <= ct; i++)
            vis[i] ? vx[lnk[i]] -= mi, vy[i] += mi : slk[i] -= mi;
    }
    for(; y; y = pre[y]) lnk[y] = lnk[pre[y]];
}

ll km() {
    for(int i = 1; i <= ct; i++) bfs(i);
    ll ans = 0;
    for(int i = 1; i <= ct; i++) ans += vx[i] + vy[i];
    return ans;
}
```

```
namespace R {
    int n; // n为左边点数和右边点数的最大值，点的标号从1开始。
    int w[N][N], kx[N], ky[N], py[N], vy[N], slk[N], pre[N];
    ll KM() {
        fill(kx, kx + n + 1, 0);
        fill(ky, ky + n + 1, 0);
        fill(py, py + n + 1, 0);
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                kx[i] = max(kx[i], w[i][j]);

        for(int i = 1; i <= n; i++) {
            fill(vy, vy + n + 1, 0);
            fill(slk, slk + n + 1, INF);
            fill(pre, pre + n + 1, 0);
            int k = 0, p = -1;
            for(py[k = 0] = i; py[k]; k = p) {
                int d = INF;
                vy[k] = 1;
                int x = py[k];
                for(int j = 1; j <= n; j++)
                    if (!vy[j]) {
                        int t = kx[x] + ky[j] - w[x][j];
                        if (t < slk[j]) { slk[j] = t; pre[j] = k; }
```

```
                        if (slk[j] < d) { d = slk[j]; p = j; }
                    }
                for(int j = 0; j <= n; j++)
                    if (vy[j]) { kx[py[j]] -= d; ky[j] += d; }
                    else slk[j] -= d;
            }
            for (; k; k = pre[k]) py[k] = py[pre[k]];
        }
        ll ans = 0;
        for(int i = 1; i <= n; i++) ans += kx[i] + ky[i];
        return ans;
    }
}
```

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。

```
int link[N][N],vis[N],group[N],cnt[N];
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;
    int semi[N], idom[N], dfn[N], idx[N], f[N];
    void init() {
        T = 0;
        for(int i = 1; i <= n; i++) semi[i] = f[i] = best[i] = i;
        for(int i = 1; i <= n; i++) dt[i].clear();
    }
    void dfs(int u) {
        dfn[u] = ++T; idx[T] = u;;
        for(auto v : G[u]) {
            if(!dfn[v]) {
                fa[v] = u; dfs(v);
            }
        }
    }
    int find(int x) {
        if(f[x] == x) return x;
        int fx = find(f[x]);
        if(dfn[semi[best[f[x]]]] < dfn[semi[best[x]]]) best[x] = best[f[x]];
        return f[x] = fx;
    }
    void Tarjan(int rt) {
        dfs(rt);
        for(int i = T; i >= 2; i--) {
            int x = idx[i];
            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;
57  int sz[N];
58  void dfs(int u, int fa) {
59      sz[u] = 1;
60      for(auto v : LT::dt[u]) {
61          if(v == fa) continue;
62          dfs(v, u);
63          sz[u] += sz[v];
64      }
65  }
66  int main() {
67      ios::sync_with_stdio(false); cin.tie(0);
68      cin >> n >> m;
69      LT::n = n;
70      LT::init();
71      for(int i = 1; i <= m; i++) {
72          int u, v; cin >> u >> v;
73          LT::G[u].push_back(v);
74          LT::rG[v].push_back(u);
75      }
76      LT::Tarjan(1);
77      dfs(1, -1);
78      for(int i = 1; i <= n; i++) cout << sz[i] << ' ';
79      return 0;
80  }
```

1.9 三元环计数

```
auto cmp = [&](int x, int y) {
    return deg[x] != deg[y] ? deg[x] > deg[y] : x < y;
}
for (int i = 1; i <= m; i++) {
    if(cmp(u[i], v[i])) G[u[i]].push_back(v[i]);
    else G[v[i]].push_back(u[i]);
}
int ans = 0;
for (int i = 1; i <= nl i++) {
    for (auto u : G[i]) {
        vis[u] = i;
    }
    for (auto u : G[i]) {
        for (auto v : G[u]) {
            if (vis[v] == i) {
                ++ans;
            }
        }
    }
}
```

1.10 带花树算法

用于解决一般图匹配。

```
struct mf {
    // Time complexity: O(n^3)
    // 1-based Vertex index
    // match[x]: vertex matched with x
    // N: numbers of vertex
    int vis[N], par[N], orig[N], match[N], aux[N], t, n;
    vector<int> conn[N];
    queue<int> Q;
    void add_edge(int u, int v) {
        conn[u].push_back(v);
        conn[v].push_back(u);
    }
    void init(int _n) {
        n = _n;
        t = 0;
        for (int i = 0; i <= n; ++i) {
            conn[i].clear();
            match[i] = aux[i] = par[i] = 0;
        }
    }
```

```
    void augment(int u, int v) {
        int pv = v, nv;
        do {
            pv = par[v];
            nv = match[pv];
            match[v] = pv;
            match[pv] = v;
            v = nv;
        } while (u != pv);
    }
    int lca(int v, int w) {
        ++t;
        while (true) {
            if (v) {
                if (aux[v] == t) return v;
                aux[v] = t;
                v = orig[par[match[v]]];
            }
            swap(v, w);
        }
    }
    void blossom(int v, int w, int a) {
        while (orig[v] != a) {
            par[v] = w;
            w = match[v];
            if (vis[w] == 1) Q.push(w), vis[w] = 0;
            orig[v] = orig[w] = a;
            v = par[w];
        }
    }
    bool bfs(int u) {
        fill(vis + 1, vis + 1 + n, -1);
        iota(orig + 1, orig + n + 1, 1);
        Q = queue<int>();
        Q.push(u);
        vis[u] = 0;
        while (!Q.empty()) {
            int v = Q.front();
            Q.pop();
            for (int x : conn[v]) {
                if (vis[x] == -1) {
                    par[x] = v;
                    vis[x] = 1;
                    if (!match[x]) return augment(u, x), true;
                    Q.push(match[x]);
                    vis[match[x]] = 0;
                } else if (vis[x] == 0 && orig[v] != orig[x]) {
```

```
                    int a = lca(orig[v], orig[x]);
                    blossom(x, v, a);
                    blossom(v, x, a);
                }
            }
        }
        return false;
    }
    int Match() {
        int ans = 0;
        // find random matching (not necessary, constant improvement)
        vector<int> V(n - 1);
        iota(V.begin(), V.end(), 1);
        shuffle(V.begin(), V.end(), mt19937(61471));
        for (auto x : V)
            if (!match[x]) {
                for (auto y : conn[x])
                if (!match[y]) {
                    match[x] = y, match[y] = x;
                    ++ans;
                    break;
                }
            }
        for (int i = 1; i <= n; ++i)
            if (!match[i] && bfs(i)) ++ans;
        return ans;
    }
} mf;
```

1.11 虚树

只提取树上的“关键点”以及某些关键点的 lca 出来建树，并且不改变他们在原树中 dfs 序的相对大小关系。如果题目答案只与关键点有关的话，那么这样能很好地帮助我们减小问题的规模来计算答案。

```
int in[N], out[N], T;
int f[N][20], deep[N];
vector <int> G[N];
void dfs(int u, int fa) {
    in[u] = ++T;
    deep[u] = deep[fa] + 1;
    f[u][0] = fa;
    for(int i = 1; i < 20; i++) {
        f[u][i] = f[f[u][i - 1]][i - 1];
    }
    for(auto v : G[u]) if(v != fa) {
```

```
12          dfs(v, u);
13      }
14      out[u] = T;
15  }
16  int LCA(int x, int y) {
17      if(deep[x] < deep[y]) swap(x, y);
18      for(int i = 19; i >= 0; i--) {
19          if(deep[f[x][i]] >= deep[y]) x = f[x][i];
20      }
21      if(x == y) return x;
22      for(int i = 19; i >= 0; i--) {
23          if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
24      }
25      return f[x][0];
26  }
27  int dis(int x, int y) {
28      int z = LCA(x, y);
29      return deep[x] + deep[y] - 2 * deep[z];
30  }
31
32  int V[N << 1], tot;
33  vector <int> vt[N];
34  void clear() {
35      for(int i = 1; i <= tot; i++) vt[V[i]].clear();
36  }
37  int buildVT(vector<int>& nodes) {
38      static int st[N], top;
39      auto cmp = [&](int x, int y) {
40          return in[x] < in[y];
41      };
42      auto chk = [&](int x, int y) {
43          return in[y] >= in[x] && in[y] <= out[x];
44      };
45
46      tot = 0;
47      for(auto it : nodes) V[++tot] = it;
48      sort(V + 1, V + 1 + tot, cmp);
49      for(int i = 1, tmp = tot; i < tmp; i++) V[++tot] = LCA(V[i], V[i + 1]);
50      sort(V + 1, V + 1 + tot, cmp);
51      tot = unique(V + 1, V + 1 + tot) - V - 1;
52
53      st[top = 1] = V[1];
54      for(int i = 2; i <= tot; i++) {
55          while(top > 1 && !chk(st[top], V[i])) --top;
56          vt[st[top]].push_back(V[i]);
57          st[++top] = V[i];
58      }
```

```
    return V[1];
}
```

1.12 Kruskal 重构树

核心思想是将边拆为点当作祖先，那么两个点的 lca 的点权值就为他们简单路径中经过的边权的最大/最小值。能够将某些树上边权问题转化为点权问题。

```
vector <int> G[N << 1];
int fa[N << 1], val[N << 1];
int find (int x) {
    return fa[x] == x ? fa[x] : fa[x] = find(fa[x]);
}

int f[N << 1][20], deep[N << 1], dfn[N << 1], T;
void dfs(int u, int fa) {
    dfn[u] = ++T;
    deep[u] = deep[fa] + 1;
    f[u][0] = fa;
    for(int i = 1; i < 20; i++) {
        f[u][i] = f[f[u][i - 1]][i - 1];
    }
    for(auto v : G[u]) if(v != fa) {
        dfs(v, u);
    }
}
int LCA(int x, int y) {
    if(deep[x] < deep[y]) swap(x, y);
    for(int i = 19; i >= 0; i--) {
        if(deep[f[x][i]] >= deep[y]) x = f[x][i];
    }
    if(x == y) return x;
    for(int i = 19; i >= 0; i--) {
        if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
    }
    return f[x][0];
}

void Kruskal (vector <pair<int, pii>>& edges) {
    for (int i = 1; i <= n << 1; i++) {
        fa[i] = i;
    }
    sort(all(edges), [&] (pair<int, pii> A, pair<int, pii> B) {
        return A.fi > B.fi;
    });
    int cnt = n;
```

```
39      for (int i = 0; i < sz(edges); i++) {
40          int u = edges[i].se.fi, v = edges[i].se.se, w = edges[i].fi;
41          int x = find(u), y = find(v);
42          if (x != y) {
43              val[++cnt] = w;
44              fa[x] = fa[y] = cnt;
45              G[cnt].push_back(x);
46              G[cnt].push_back(y);
47          }
48      }
49      dfs(cnt, 0);
50  }
```

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 $O(n)$。

也能用来筛积性函数，只需要知道函数的递推关系就行。

```
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

用来测试一个数是否为素数。

```
//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);
        a = mul(a, a, p);
        b >>= 1;
    }
    return ans;
}
bool check(ll a, ll n, ll x, ll t) {
    ll ans = qp(a, x, n);
    ll last = ans;
    for(int i = 1; i <= t; i++) {
        ans = mul(ans, ans, n);
        if(ans == 1 && last != 1 && last != n - 1) return true;
        last = ans;
    }
    if(ans != 1) return true;
    return false;
}
bool Miller_Rabin(ll n) {
    if(n == 1 || (n & 1) == 0) return false;
    if(n == 2) return true;
    ll x = n - 1, t = 0;
    while((x & 1) == 0) {x >>= 1, ++t;}
    srand(time(NULL));
    for(int i = 0; i < 8; i++) {
        ll a = rand() % (n - 1) + 1;
        if(check(a, n, x, t)) return false;
    }
    return true;
}
```

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 $x=x*c/g, y=y*c/g$。

```
//扩展欧几里得
//求解最小正整数解
//x_0,y_0为方程a'x+b'y=c'的解(除以gcd过后)
//x=c'*(x_0+k*b'),y=c'*(y_0-k*a')
//注意最后要乘上c/gcd
void exgcd(ll a, ll b, ll &x, ll &y) {
    if(b == 0) {
        x = 1, y = 0;
        return ;
```

```
    }
    exgcd(b,a%b,x,y);
    ll z = x ;
    x = y;
    y = z - y * (a / b);
}
ll calc(ll a, ll b, ll c) {
    ll x, y;
    ll g = __gcd(a, b);
    if(c % g != 0) return -1;
    a /= g, b /= g, c /= g;
    exgcd(a, b, x, y);
    // x *= c 如果在前面乘以c的话可能得到更小的正整数解
    x = (x % b + b) % b;
    x *= c; //!!!
    return x;
}
```

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：$inv[i]=(p-p/i)*inv[p\%i]\%p$

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++;
}
```

以及浮点数的高斯消元。

```
// 高斯消元浮点数
// a: 系数矩阵 b: 常数列 x: 答案列
double x[N];
void Gauss(double a[][N], double* b, int n) {
    for (int i = 1; i <= n; i++) {
        int p = i;
        for (int k = i + 1; k <= n; k++) {
            if (fabs(a[k][i]) > fabs(a[p][i])) p = k;
        }
```

```
            if (i != p) {
                swap(a[i], a[p]), swap(b[i], b[p]);
            }
            for (int k = i + 1; k <= n; k++) {
                double d = a[k][i] / a[i][i];
                b[k] -= d * b[i];
                for (int j = 1; j <= n; j++) {
                    a[k][j] -= d * a[i][j];
                }
            }
        }
        // 回代
        for (int i = n; i >= 1; i--) {
            for (int j = i + 1; j <= n; j++) b[i] -= x[j] * a[i][j];
            x[i] = b[i] / a[i][i];
        }
    }
```

2.5 欧拉函数

$\varphi(x)$ 表示 $1..x$ 中与 x 互质的数，如果 i 与 x 互质，那么 $x-i$ 与 x 也互质。

$\varphi(x) = x\prod_{i=1}^{n}(1-\frac{1}{p_i})$

$\varphi * I = id$

$phi(p^k) = p^k - p^{k-1}$

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 $S(n,1)+f(1)$。

upd：

用于解决一类积性函数前缀和的问题，积性函数 F 满足下面条件：* F 为完全积性函数，且 $F(p^c)$ 易求；；* 对于素数 p，能够构造完全积性函数使得 $f(p)=F(p)$，即在素数位置的取值相同；* f 函数的前缀和能在 $O(1)$ 快速求解

代码以 $F(p^k)=p^k(p^k-1)$ 为例，规定 F 为积性函数。显然 F 满足以上条件，并且 $f(p)=p^2-p$，其前缀和也易求。之后将其拆为两部分进行考虑，每个单独算最后相减就行，详见代码：

```
1  //min25筛
2  //以f(p^k)=p^k(p^k-1)为例
3  ll n;
4  ll sum1[N], sum2[N], prime[N];
5  ll w[N], ind1[N], ind2[N];
6  ll g1[N], g2[N];
7  bool chk[N];
8  int tot, cnt;
9  void pre(int n) { //  \sqrt
10     chk[1] = 1;
11     for(int i = 1; i <= n; i++) {
12         if(!chk[i]) {
13             prime[++tot] = i;
14             // 预处理 \sum f(p)，这里只预处理了根号范围内
15             sum1[tot] = (sum1[tot - 1] + i) % MOD;
16             sum2[tot] = (sum2[tot - 1] + 1ll * i * i % MOD) % MOD;
17         }
18         for(int j = 1; j <= tot && prime[j] * i <= n; j++) {
19             chk[i * prime[j]] = 1;
20             if(i % prime[j] == 0) break;
21         }
22     }
23 }
24 // 筛得\sum f(p)，求出2~n范围内的素数和，注意要去掉1
25 void calc_g() {
26     int z = sqrt(n);
27     // 预处理，首先将质数的取值扩展到所有数，并且建立映射数组，只有O(sqrt(n))个位
       置有用
28     // ind数组能对于一个数x，快速找到对应的下标，利用了整除的性质
29     for(ll i = 1, j; i <= n; i = j + 1) {
30         j = n / (n / i);
31         w[++cnt] = n / i;
32         g1[cnt] = w[cnt] % MOD;
33         g2[cnt] = g1[cnt] * (g1[cnt] + 1) / 2 % MOD * (2 * g1[cnt] + 1) % MOD *
               inv3 % MOD - 1;
34         g1[cnt] = g1[cnt] * (g1[cnt] + 1) / 2 % MOD - 1;
```

```
35          if(n / i <= z) ind1[n / i] = cnt;
36          else ind2[n / (n / i)] = cnt;
37      }
38      // 开始筛法，g[j]-=f(p[i])*(g[k]-sum[i-1])
39      for(int i = 1; i <= tot; i++) {
40          for(int j = 1; j <= cnt && prime[i] * prime[i] <= w[j]; j++) {
41              ll tmp = w[j] / prime[i], k;
42              if(tmp <= z) k = ind1[tmp]; else k = ind2[n / tmp];
43              (g1[j] -= prime[i] * (g1[k] - sum1[i - 1] + MOD) % MOD) %= MOD;
44              (g2[j] -= prime[i] * prime[i] % MOD * (g2[k] - sum2[i - 1] + MOD) %
                    MOD) %= MOD;
45              if(g1[j] < 0) g1[j] += MOD;
46              if(g2[j] < 0) g2[j] += MOD;
47          }
48      }
49  }
50  // 因为F为积性函数，所以先分两种情况：质数和合数进行考虑
51  ll S(ll x, int y) { // 2~x >= P_y
52      if(x <= 1 || prime[y] > x) return 0;
53      ll z = sqrt(n);
54      ll k = x <= z ? ind1[x] : ind2[n / x];
55      // 求这部分的质数答案，可扩展性较高
56      ll ans = (g2[k] - g1[k] + MOD - (sum2[y - 1] - sum1[y - 1]) + MOD) % MOD;
57      // 枚举最小质因子及次方，然后递归求解
58      // ans+=F(p[i]^e) * S(n/p[i],i+1) + F(p[i]^{e+1})
59      for(int i = y; i <= tot && prime[i] * prime[i] <= x ; i++) {
60          ll pe = prime[i], pe2 = prime[i] * prime[i];
61          for(int e = 1; pe2 <= x; ++e, pe = pe2, pe2 *= prime[i]) {
62              ans = (ans + pe % MOD * ((pe - 1) % MOD) % MOD * S(x / pe, i + 1) +
                    pe2 % MOD * ((pe2 - 1) % MOD) % MOD) % MOD;
63          }
64      }
65      return ans % MOD;
66  }
```

其余示例，求最大质因子不超过 k 的 min25 筛部分（只需要修改统计质数答案部分即可，本质考察对 min25 筛和积性函数的理解）：

```
1   ll n, k;
2   ll sum1[N], sum2[N], prime[N];
3   ll w[N], ind1[N], ind2[N];
4   ll g1[N], g2[N];
5   bool chk[N];
6   int tot, cnt;
7   void pre(int n) { //  \sqrt
8       chk[1] = 1;
9       for(int i = 1; i <= n; i++) {
10          if(!chk[i]) {
```

```
            prime[++tot] = i;
            sum1[tot] = sum1[tot - 1] + 1;
        }
        for(int j = 1; j <= tot && prime[j] * i <= n; j++) {
            chk[i * prime[j]] = 1;
            if(i % prime[j] == 0) break;
        }
    }
}

int f(int x, int y) {
    return x <= k;
}

void calc_g(ll n) {
    int z = sqrt(n + 0.5);
    for(ll i = 1, j; i <= n; i = j + 1) {
        j = n / (n / i);
        w[++cnt] = n / i;

        g1[cnt] = w[cnt] - 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] -= (g1[k] - sum1[i - 1]);
        }
    }
}

ll num;

ll S(ll x, int y) { // 2~x >= P_y
    if(x <= 1 || prime[y] > x) return 0;
    int z = sqrt(n + 0.5);
    ll ans;
    if (k <= x) {
        ans = num - sum1[y - 1];
    } else {
        ll t = (x <= z ? ind1[x] : ind2[n / x]);
        ans = g1[t] - sum1[y - 1];
    }
    ans = max(ans, 0ll);
```

```
58        ll tmp = ans;
59        for(int i = y; i <= tot && prime[i] * prime[i] <= x ; i++) {
60            ll pe = prime[i];
61            for(int e = 1; pe * prime[i] <= x; ++e, pe = pe * prime[i]) {
62                ans += f(prime[i], e) * S(x / pe, i + 1) + f(prime[i], e + 1);
63            }
64        }
65        return ans;
66    }
```

2.10 中国剩余定理

一般用于求解一元一次同余方程组的合法解，要求模数为质数。

扩展中国剩余定理可以处理模数不互质的情况，这时的大致思想是两两方程列式，通过 exgcd 来求得合法解，然后不断合并。

```
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;
15         for(int i = 1; i <= n; i++) M *= m[i];
16         for(int i = 1; i <= n; i++) {
17             Mi = M / m[i];
18             exgcd(Mi, m[i], d, X, Y);
19             ans = (ans + Mi * X * a[i]) % M;
20         }
21         if(ans < 0) ans += M;
22         return ans;
23     }
24 }crt;
```

2.11 Lucas 定理

```
1  // Lucas定理，用于快速求解大组合数在模意义下的答案
2
```

```
3   // 当模数为质数时，组合数那么可以拆分为十进制下每一位对应组合数的乘积
4   long long Lucas(long long n, long long m, long long p) {
5     if (m == 0) return 1;
6     return (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
7   }
8
9   // 当模数不为质数时，先将模数拆分为若干个质因子的乘积，然后只考虑组合数在模p^t
        下的答案，最后通过excrt合并
10  // 之后考虑 n!,m!,(n-m)! 在模 p^t 意义下的答案即可
11  // 以 n! 为例，那么把 p 的倍数单独拿出来，并都提取一个 p 出来，那么后面相当于 (
        n/p)! 的一个子问题
12  // 前面剩下的则是以 p^t(也可以说是p)为周期的，那么可以O(p^t)预处理后直接计算。
13  // 注意：预处理时不用再把p的倍速给算进去。
14  ll calc(ll n, ll x, ll P) {
15    if (!n) return 1;
16    ll s = 1;
17    for (int i = 1; i <= P; i++)
18      if (i % x) s = s * i % P;
19    s = qpow(s, n / P, P);
20    for (int i = n / P * P + 1; i <= n; i++)
21      if (i % x) s = s * i % P;
22    // s：前半部分的结果，忽略x的贡献
23    return s * calc(n / x, x, P) % P;
24  }
25  ll multilucas(ll m, ll n, ll x, ll P) {
26    int cnt = 0;
27    for (int i = m; i; i /= x) cnt += i / x;
28    for (int i = n; i; i /= x) cnt -= i / x;
29    for (int i = m - n; i; i /= x) cnt -= i / x;
30    // 将质因子x的贡献单独拿到外面计算，之后就不考虑x的贡献了
31    return qpow(x, cnt, P) % P * calc(m, x, P) % P * inverse(calc(n, x, P), P) %
32           P * inverse(calc(m - n, x, P), P) % P;
33  }
34  ll exlucas(ll m, ll n, ll P) {
35    int cnt = 0;
36    ll p[20], a[20];
37    for (ll i = 2; i * i <= P; i++) {
38      if (P % i == 0) {
39        p[++cnt] = 1;
40        while (P % i == 0) p[cnt] = p[cnt] * i, P /= i;
41        a[cnt] = multilucas(m, n, i, p[cnt]);
42      }
43    }
44    if (P > 1) p[++cnt] = P, a[cnt] = multilucas(m, n, P, P);
45    // 中国剩余定理合并得到答案
46    return CRT(cnt, a, p);
47  }
```

2.12 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;
```

2.13 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 $p|a$。

```
const int moder = (int) 1e9 + 7;
const int inv2 = (moder + 1) / 2;

struct field2{
    int x, y, a, p;

    field2():x(0), y(0), a(0), p(0){}
    field2(int x, int y, int a, int p):x(x), y(y), a(a), p(p){}

    field2 operator *(const field2 &f)const{
        int retx = (1ll * x * f.x + 1ll * y * f.y % p * a) % p;
        int rety = (1ll * x * f.y + 1ll * y * f.x) % p;
        return field2(retx, rety, a, p);
    }

    field2 powermod(int exp)const{
        field2 ret(1, 0, a, p), aux = *this;
        for ( ; exp > 0; exp >>= 1){
            if (exp & 1){
                ret = ret * aux;
            }
            aux = aux * aux;
        }
        return ret;
    }
};

int powermod(int a, int exp, int moder){
    int ret = 1;
    for ( ; exp; exp >>= 1){
        if (exp & 1){
            ret = 1ll * ret * a % moder;
        }
        a = 1ll * a * a % moder;
    }
    return ret;
}

int randint(int n){
    return rand() % n;
}
vector <int> remain2(int a, int p){ //x^2 = a (mod p)
```

```
43        if (!a || p == 2){ //特判
44            return {a, a};
45        }
46        if (powermod(a, p - 1 >> 1, p) != 1){ //欧拉准则
47            return {};
48        }
49        while (true){
50            field2 f(randint(p - 1) + 1, randint(p - 1) + 1, a, p);
51            f = f.powermod(p - 1 >> 1);
52            if (f.x){
53                continue;
54            }
55            int ret = powermod(f.y, p - 2, p);
56            return {ret, p - ret};
57        }
58    }
```

2.14 欧拉降幂

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

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)

```
1  ll Mod(ll a, ll b) { //可以直接避免分类讨论
2      return a < b ? a : a % b + b;
3  }
4  int qp(ll a, ll b, ll p) {
5      int ans = 1;
6      while(b) {
7          if(b & 1) ans = Mod(ans * a, p);
8          a = Mod(a * a, p);
9          b >>= 1;
10     }
11     return ans;
12 }
13 int calc(ll a, ll b, ll m) {
14     if(m == 1 || !b) return 1;
15     int p = phi[m];
16     int x = calc(a, b - 1, p);
17     return qp(a, x, m);
18 }
```

2.15 多项式全家桶

vector 进行封装。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector <int> Poly;
#define ri register int
#define cs const
#define sz(a) (a).size()
#define all(a) (a).begin(), (a).end()
const int MOD = 998244353, inv2 = (MOD + 1) >> 1;
const int N = 4e5 + 5, M = (1 << 19); //四倍空间,M=2^x且M>=N
typedef vector <int> Poly;
#define ri register int
#define cs const
inline int add(int a, int b) {return a + b >= MOD ? a + b - MOD : a + b;}
inline int dec(int a, int b) {return a < b ? a - b + MOD : a - b;}
inline int mul(int a, int b) {return 1ll * a * b % MOD;}
inline int qpow(int a, int b) {int res = 1; while(b) {if(b & 1) res = mul(res,
    a); a = mul(a, a); b >>= 1;} return res;}
int rev[N], inv[N], pw[M], W[2][M];
inline void init() {
    inv[0] = inv[1] = 1;
    for(ri i = 2; i < N; i++) inv[i] = mul(inv[MOD % i], MOD - MOD / i);
    int w0 = qpow(3, (MOD - 1) / M), w1 = qpow((MOD + 1) / 3, (MOD - 1) / M);
    W[0][0] = W[1][0] = 1;
    for(ri i = 1; i < M; i++) {
        W[0][i] = mul(W[0][i - 1], w0);
        W[1][i] = mul(W[1][i - 1], w1);
    }
}
inline void init_rev(int len) {
    for(ri i = 0; i < len; i++) rev[i] = rev[i >> 1] >> 1 | ((i & 1) * (len >>
        1));
}
inline void NTT(Poly &a, int n, int op) {
    for(ri i = 0; i < n; i++) if(i < rev[i]) swap(a[i], a[rev[i]]);
    for(ri i = 1; i < n; i <<= 1) {
        int t = M / (i << 1), t2 = (op == -1 ? 1 : 0);
        for(ri j = 0; j < i; ++j) pw[j] = W[t2][j * t];
        int wn = (op == -1 ? W[1][i] : W[0][i]);
        for(ri j = 0; j < n; j += (i << 1)) {
            for(ri k = 0, x, y; k < i; ++k) {
                x = a[j + k], y = mul(pw[k], a[j + k + i]);
                a[j + k] = add(x, y);
                a[j + k + i] = dec(x, y);
```

```
43                  }
44              }
45          }
46          if(op == -1) for(ri i = 0; i < n; i++) a[i] = mul(a[i], inv[n]);
47      }
48
49      struct cn{
50          int x, y, w;
51          cn operator * (cn a){
52              cn ans;
53              ans.x = (1ll * x * a.x % MOD + 1ll * y * a.y % MOD * w % MOD) % MOD;
54              ans.y = (1ll * x * a.y % MOD + 1ll * y * a.x % MOD) % MOD;
55              ans.w = w;
56              return ans;
57          }
58          int operator ^ (int b){
59              cn ans, x = *this;
60              ans.x = 1, ans.y = 0, ans.w = w;
61              while(b) {
62                  if(b & 1) ans = ans * x;
63                  x = x * x;
64                  b >>= 1;
65              }
66              return ans.x;
67          }
68      };
69      //求解n的二次剩余
70      int sqrt_mod(int n) {
71          if(n == 0) return 0;
72          if(qpow(n, (MOD - 1) / 2) == MOD - 1) return -1;
73          int a, w;
74          while(true) {
75              a = rand() % MOD;
76              w = (1ll * a * a - n + MOD) % MOD;
77              if(qpow(w, (MOD - 1) / 2) == MOD - 1) break;
78          }
79          cn x;
80          x.x = a, x.y = 1, x.w = w;
81          return x ^ ((MOD + 1) / 2);
82      }
83      inline print(cs Poly &a) {
84          for(ri i = 0; i < sz(a); i++) cout << a[i] << ' ';
85          cout << '\n';
86      }
87
88      inline Poly operator + (cs Poly &a, cs Poly &b) {
89          Poly c = a; c.resize(max(sz(a), sz(b)));
```

```
90      for(ri i = 0; i < sz(b); i++) c[i] = add(c[i], b[i]);
91      return c;
92  }
93  inline Poly operator - (cs Poly &a, cs Poly &b) {
94      Poly c = a; c.resize(max(sz(a), sz(b)));
95      for(ri i = 0; i < sz(b); i++) c[i] = dec(c[i], b[i]);
96      return c;
97  }
98  inline Poly operator * (Poly a, Poly b) {
99      int n = sz(a), m = sz(b), l = 1;
100     while(l < n + m - 1) l <<= 1;
101     init_rev(l);
102     a.resize(l), NTT(a, l, 1);
103     b.resize(l), NTT(b, l, 1);
104     for(ri i = 0; i < l; i++) a[i] = mul(a[i], b[i]);
105     NTT(a, l, -1);
106     a.resize(n + m - 1);
107     return a;
108 }
109 inline Poly operator * (Poly a, int b) {
110     for(ri i = 0; i < sz(a); i++) a[i] = mul(a[i], b);
111     return a;
112 }
113
114 //求导
115 inline Poly Deriv(Poly a) {
116     for(ri i = 0; i + 1 < sz(a); i++) a[i] = mul(a[i + 1], i + 1);
117     a.pop_back();
118     return a;
119 }
120
121 //积分
122 inline Poly Integ(Poly a) {
123     a.push_back(0);
124     for(ri i = sz(a) - 1; i; i--) a[i] = mul(a[i - 1], inv[i]);
125     a[0] = 0;
126     return a;
127 }
128
129 //多项式取逆
130 inline Poly Inv(cs Poly &a, int lim) {
131     Poly c, b(1, qpow(a[0], MOD - 2));
132     for(ri l = 4; (l >> 2) < lim; l <<= 1) {
133         init_rev(l);
134         c = a, c.resize(l >> 1);
135         c.resize(l), NTT(c, l, 1);
136         b.resize(l), NTT(b, l, 1);
```

```
137             for(ri i = 0; i < l; i++) b[i] = mul(b[i], dec(2, mul(c[i], b[i])));
138             NTT(b, l, -1);
139             b.resize(l >> 1);
140         }
141         b.resize(lim);
142         return b;
143     }
144     inline Poly Inv(cs Poly &a) {return Inv(a, sz(a));}
145 
146     //多项式开根
147     inline Poly Sqrt(cs Poly &a, int lim) {
148         Poly c, d, b(1, sqrt_mod(a[0]));
149         for(ri l = 4; (l >> 2) < lim; l <<= 1) {
150             init_rev(l);
151             c = a, c.resize(l >> 1);
152             d = Inv(b, l >> 1);
153             c.resize(l), NTT(c, l, 1);
154             d.resize(l), NTT(d, l, 1);
155             for(ri j = 0; j < l; j++) c[j] = mul(c[j], d[j]);
156             NTT(c, l, -1);
157             b.resize(l >> 1);
158             for(ri j = 0; j < (l >> 1); j++) b[j] = mul(add(c[j], b[j]), inv2);
159         }
160         b.resize(lim);
161         return b;
162     };
163     inline Poly Sqrt(cs Poly &a) {return Sqrt(a, sz(a));}
164 
165     //多项式ln
166     inline Poly Ln(Poly a, int lim){
167         a = Integ(Deriv(a) * Inv(a, lim));
168         a.resize(lim);
169         return a;
170     }
171     inline Poly Ln(cs Poly &a) {return Ln(a, sz(a));}
172 
173     //多项式exp
174     inline Poly Exp(cs Poly &a, int lim){
175         Poly c, b(1, 1);
176         for(ri i = 2; (i >> 1) < lim; i <<= 1) {
177             c = Ln(b, i);
178             for(ri j = 0; j < i; j++) c[j] = dec(j < sz(a) ? a[j] : 0, c[j]);
179             c[0] = add(c[0], 1);
180             b = b * c;
181             b.resize(i);
182         }
183         b.resize(lim);
```

```
184         return b;
185     }
186     inline Poly Exp(cs Poly &a) {return Exp(a, sz(a));}
187
188     //三角函数
189     cs int w4 = qpow(3, (MOD - 1) / 4);
190     inline Poly Cos(cs Poly &a, int lim) {
191         Poly c = a; c.resize(lim);
192         c = (Exp(c * w4) + Exp(c * (MOD - w4))) * inv2;
193         return c;
194     }
195     inline Poly Cos(cs Poly &a) {return Cos(a, sz(a));}
196     inline Poly Sin(cs Poly &a, int lim){
197         Poly c = a; c.resize(lim);
198         c = (Exp(c * w4) - Exp(c * (MOD - w4))) * mul(inv2, qpow(w4, MOD - 2));
199         return c;
200     }
201     inline Poly Sin(cs Poly &a) {return Sin(a, sz(a));}
202
203     //多项式除法
204     inline Poly operator / (Poly a, Poly b) {
205         ri len = 1, deg = sz(a) - sz(b) + 1;
206         reverse(all(a)), reverse(all(b));
207         while(len <= deg) len <<= 1;
208         b = Inv(b, len), b.resize(deg);
209         a = a * b, a.resize(deg);
210         reverse(all(a));
211         return a;
212     }
213     inline Poly operator % (const Poly &a, const Poly &b) {
214         Poly c = a - (a / b) * b;
215         c.resize(sz(b) - 1);
216         return c;
217     }
218
219     //多项式k次方
220     inline Poly Ksm(Poly a, int k, int lim){
221         a = Exp(Ln(a) * k);
222         a.resize(lim);
223         return a;
224     }
225     inline Poly Ksm(Poly &a, int k) {
226         int t, x;
227         for(ri i = 0; i < sz(a); i++) if(a[i] > 0) {
228             t = a[i], x = i; break;
229         }
230         if(t == 1 && x == 0) return Ksm(a, k, sz(a));
```

```
231         Poly b(x + 1, 0); b[x] = t;
232         a = a / b;
233         a = Ksm(a, k, sz(a));
234         a.resize(sz(a) + x * k);
235         for(int i = sz(a) - 1; i >= 0; i--) {
236             if(i - x * k < 0) a[i] = 0;
237             else a[i] = a[i - x * k];
238         }
239         a = a * qpow(t, k);
240         return a;
241     }
242
243     //分治FFT
244     //f_i=\sum_{j=0}^{i-1}f_j*g_{i-j},f[0]=1
245     void DC_FFT(int l, int r, Poly &f, const Poly &g) {
246         if(l >= r) return;
247         int mid = (l + r) >> 1;
248         DC_FFT(l, mid, f, g);
249         Poly a(r - l), b(r - l);
250         Poly c = f;
251         for(int i = mid + 1; i <= r; i++) c[i] = 0;
252         for(int i = 0; i < r - l; i++) a[i] = c[i + l], b[i] = g[i + 1];
253         c = a * b;
254         for(int i = mid + 1; i <= r; i++) f[i] = (f[i] + c[i - l - 1]) % MOD;
255         DC_FFT(mid + 1, r, f, g);
256     }
257
258     //多点求值
259     //已知a[i],f，求解b[i] = f(a[i])
260     Poly P[N];
261     void DC_NTT(int o, int l, int r, const Poly& a) {
262         if(l == r) {
263             P[o].resize(2);
264             P[o][0] = MOD - a[l], P[o][1] = 1;
265             return;
266         }
267         int mid = (l + r) >> 1;
268         DC_NTT(o << 1, l, mid, a), DC_NTT(o << 1|1, mid + 1, r, a);
269         P[o] = P[o << 1] * P[o << 1|1];
270     }
271     void DC_MOD(const Poly &f, const Poly &a, Poly &b, int o, int l, int r) {
272         if(r - l <= 400) {
273             for(int i = l; i <= r; i++) {
274                 int res = 0;
275                 for(int j = sz(f) - 1; j >= 0; j--) res = add(mul(res, a[i]), f[j])
                        ;
276                 b[i] = res;
```

```
277             }
278             return;
279         }
280         int mid = (l + r) >> 1;
281         Poly lf = f, rf = f;
282         if(sz(lf) >= sz(P[o << 1])) lf = lf % P[o << 1];
283         if(sz(rf) >= sz(P[o << 1|1])) rf = rf % P[o << 1|1];
284         DC_MOD(lf, a, b, o << 1, l, mid), DC_MOD(rf, a, b, o << 1|1, mid + 1, r);
285     }
286     void getval(const Poly &f, const Poly &a, Poly &b) {
287         DC_NTT(1, 0, sz(a) - 1, a);//插值时省略避免重复运算
288         DC_MOD(f, a, b, 1, 0, sz(a) - 1);
289     }
290 
291     //多项式快速插值
292     //已知点对(x_i,y_i)，求解插值多项式
293     inline Poly DC_NTT2(int o, int l, int r, const Poly& v) {
294         if(r == l) {
295             Poly t(1, v[l]);
296             return t;
297         }
298         int mid = (l + r) >> 1;
299         Poly lf = DC_NTT2(o << 1, l, mid, v), rf = DC_NTT2(o << 1|1, mid + 1, r, v)
                ;
300         return lf * P[o << 1|1] + rf * P[o << 1];
301     }
302 
303     inline Poly fast_interpolation(const Poly &x, const Poly &y) {
304         int n = sz(x);
305         DC_NTT(1, 0, n - 1, x);
306         Poly M = P[1], val(n);
307         M = Deriv(M);
308         getval(M, x, val);
309         for(ri i = 0; i < n; i++) val[i] = mul(y[i], qpow(val[i], MOD - 2));
310         return DC_NTT2(1, 0, n - 1, val);
311     }
312 
313     int main() {
314         init(); //记住要init
315         return 0;
316     }
```

2.16 蔡勒公式

快速判断一天为星期几。

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

2.17 原根

```
ll qp(ll a, ll b) {
    ll ans = 1;
    while(b) {
        if(b & 1) ans = ans * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return ans;
}
int work(ll P) {   //p - 1
    vector <int> div;
    for(int i = 2; 1ll * i * i <= P; i++) {
        if(P % i == 0) {
            div.push_back(i);
            while(P % i == 0) P /= i;
        }
    }
    if(P > 1) div.push_back(P);
    for(int i = 2;; i++) {
        bool ok = true;
        for(auto x : div) {
            if(qp(i, (p - 1) / x) % p == 1) {
                ok = false; break;
            }
        }
        if(ok) return i;
    }
}
```

2.18 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 $O(k)$。

```
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;
```

2.19 拉格朗日求系数

```
//拉格朗日求d次多项式系数，a[i]为系数
ll qpow(ll a, ll b) {
    ll ans = 1;
    while(b) {
        if(b & 1) ans = ans * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return ans;
}
struct Lagrange {
    ll f[N], a[N], b[N];
    int d;
    void init(int _d) {
        d = _d;
        //y_i
        for(int i = 1; i <= d + 1; i++) f[i] = (f[i - 1] + qpow(i, d - 1)) %
            MOD;
        b[0] = 1;
    }
    void work() {
        for(int i = 0; i <= d; i++) {
            for(int j = i + 1; j; j--) b[j] = (b[j - 1] + MOD - 1ll * b[j] * (i
                + 1) % MOD) % MOD;
            b[0] = 1ll * b[0] * (MOD - i - 1) % MOD;
        }
        for(int i = 0; i <= d; i++) {
            int s = f[i + 1], inv = qpow(i + 1, MOD - 2);
            for(int j = 0; j <= d; j++) if(i != j) s = 1ll * s * qpow((i - j +
                MOD) % MOD, MOD - 2) % MOD;
            b[0] = 1ll * b[0] * (MOD - inv) % MOD;
            for(int j = 1; j <= d + 1; j++) b[j] = (MOD - 1ll * (b[j] + MOD - b
                [j - 1]) * inv % MOD) % MOD;
            for(int j = 0; j <= d + 1; j++) a[j] = (a[j] + 1ll * s * b[j]) %
                MOD;
            for(int j = d + 1; j; j--) b[j] = (b[j - 1] + MOD - 1ll * b[j] * (i
                + 1) % MOD) % MOD;
            b[0] = 1ll * b[0] * (MOD - i - 1) % MOD;
        }
    }
}A;
```

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 $O(klogn)$。

```cpp
#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;
typedef long long ll;

ll powMOD(ll a, ll b) {
    ll ans = 1;
    for (; b; b >>= 1, a = a * a%MOD)if (b & 1)ans = ans * a%MOD;
    return ans;
}
ll n;
namespace linear_seq {
    const int N = 10010;
    ll res[N], base[N], _c[N], _md[N];

    vector<int> Md;
    void mul(ll *a, ll *b, int k) {
        rep(i, 0, k + k) _c[i] = 0;
        rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j])
            % MOD;
        for (int i = k + k - 1; i >= k; i--) if (_c[i])
            rep(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] *
                _md[Md[j]]) % MOD;
        rep(i, 0, k) a[i] = _c[i];
    }
    int solve(ll n, VI a, VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
        ll ans = 0, pnt = 0;
        int k = SZ(a);
        assert(SZ(a) == SZ(b));
        rep(i, 0, k) _md[k - 1 - i] = -a[i]; _md[k] = 1;
        Md.clear();
        rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
        rep(i, 0, k) res[i] = base[i] = 0;
        res[0] = 1;
        while ((1ll << pnt) <= n) pnt++;
        for (int p = pnt; p >= 0; p--) {
            mul(res, res, k);
            if ((n >> p) & 1) {
                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 }
```

2.21 快速 gcd

预处理 gcd。

```
1   for (int i = 1; i <= a; i++) {
2       for (int j = 1; j <= b; j++) {
3           if (!Gcd[i][j]) {
4               for (int k = 1; k * i <= a && k * j <= b; k++)
5                   Gcd[k * i][k * j] = k, Lcm[k * i][k * j] = i * j * k;
6           }
7       }
8   }
```

2.22 卡特兰数

注意通过网格中非降路径来证明的方法，应用很多，包括最简单的出入栈，以及多边形的三角划分、n 个结点的二叉树个数、圆上不相交的线段数等。

前面几项为：1，1，2，5，14，42，132，429

```
1   //卡特兰数递推式：cat[i]=cat[i-1]*(4i-2)/(i+1)
2   //通项：cat[n]=C(2n,n)/(n+1)
3   int inv[N], cat[N];
4   void init(int n) {
5       inv[0] = inv[1] = 1;
```

```
    for (int i = 2; i <= n + 1; i++) {
        inv[i] = 1ll * inv[MOD % i] * (MOD - MOD / i) % MOD;
    }
    cat[0] = cat[1] = 1;
    for (int i = 2; i <= n; i++) {
        cat[i] = 1ll * cat[i - 1] * (4 * i - 2) % MOD * inv[i + 1] % MOD;
    }
}
```

2.23 FMT

FMT: 快速莫比乌斯变换，即 $f(\hat{S}) = \sum_{s \subset S} f(s)$

这样可以快速求 $h(S) = \sum_{L \subset S, R \subset S} [L \cup R = S] f(L) * g(R)$，只需要通过 FMT 变化过后，直接乘起来，再 IFMT 变换回去即可。类似于 FWT 的思想。

```
// 快速莫比乌斯变化,即f变为f的子集和
// 之后f'*g'再通过逆变化回去就是子集并的卷积
void FMT(int *f, int n, int opt) {
    for (int j = 0; j < n; ++j)
        for (int i = 0; i < (1 << n); ++i) if (i >> j & 1)
            f[i] += opt * f[i ^ (1 << j)];
}
```

2.24 子集卷积

子集卷积要解决 $h(S) = \sum_{T \subset S} f(T) * g(S - T)$，比 [[FMT| 卷积并]] 要求更严，要求两个集合交集为空。

实际上可以借助于 [[FMT]]。通过 FMT 过后，我们只需要再加一维状态表示二进制中 1 的个数，那么 $f[|s|][s]$ 才是真实合法的状态（其余状态是假的状态，并没有什么用），然后直接做卷积即可。原理在于如果两个集合相交为空集，那么 $|S| + |T| = |S \cup T|$，后者直接卷积并来搞。

时间复杂度为 $O(n^2 2^n)$。

模板如下：

```
// 子集卷积, 计算 h(S)=\sum_{s\in S,t=S-s}f(s)*g(t)
// 通过集合并卷积(FMT)记录二进制中1的个数得到, 时间复杂度为O(n^2 * 2^n)
const int Len = 1 << 21, Base = 21;

int add(int x, int y) {return x + y >= MOD ? x + y - MOD : x + y;}
int dec(int x, int y) {return x - y < 0 ? x - y + MOD : x - y;}
int mul(int x, int y) {return 1ll * x * y % MOD;}

inline void fmt_or(vector<int>& A, int dmt = 1, int base = Base) {
```

```
10      int len = 1 << base;
11      for (int i = 1; i < len; i <<= 1)
12          for (int j = 0, t = i << 1; j < len; j += t)
13              for (int k = j, K = j + i; k < K; ++k)
14                  A[k + i] = (dmt > 0 ? add(A[k + i], A[k]) : dec(A[k + i], A[k])
                        );
15  }
16
17  // 计算二进制中1的个数
18  vector<int> pc;
19  void get_top(int sz) {
20      pc.resize(sz, 0);
21      for (int i = 1; i < sz; ++i) pc[i] = pc[i >> 1] + (i & 1);
22  }
23
24  vector<int> subset_conv(const vector<int>& A, const vector<int>& B, int base) {
25      int len = 1 << base;
26      vector<int> H(len);
27      vector<vector<int> > siga(base + 1, vector<int>(len, 0)), sigb = siga, sigh
            = siga;
28      for (int S = 0; S < len; ++S) siga[pc[S]][S] = A[S], sigb[pc[S]][S] = B[S];
            //初始化
29      for (int i = 0; i <= base; ++i) {
30          fmt_or(siga[i], 1, base);
31          fmt_or(sigb[i], 1, base);
32          for (int S = 0; S < len; ++S)
33              for (int j = 0; j <= i; ++j) sigh[i][S] = add(sigh[i][S], mul(siga[
                    j][S], sigb[i - j][S]));
34          fmt_or(sigh[i], -1, base);
35      }
36      for (int S = 0; S < len; ++S) H[S] = sigh[pc[S]][S]; // 将真实答案赋值
37      return H;
38  }
```

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;
```

```
15                  p = i + (Next[pos = i] = j) - 1;
16              }
17          }
18      }
19      void work(char *s, char *t) {
20          get_next(t);
21          int lens = strlen(s + 1), lent = strlen(t + 1), p = 1, pos;
22          while(p <= lent && s[p] == t[p]) ++p;
23          p = extend[pos = 1] = p - 1;
24          for(int i = 2; i <= lens; i++) {
25              int len = Next[i - pos + 1];
26              if(len + i < p + 1) extend[i] = len;
27              else {
28                  int j = max(p - i + 1, 0);
29                  while(i + j <= lens && j <= lent && t[j + 1] == s[i + j]) j++;
30                  p = i + (extend[pos = i] = j) - 1;
31              }
32          }
33      }
34  }exkmp;
```

3.3 马拉车算法

```
1   struct Manacher{
2       char ch[N << 1];
3       int p[N << 1];
4       void work(char *s) {
5           int l = 0;
6           ch[l++] = '&'; ch[l++] = '#';
7           for(int i = 0; s[i]; i++) {
8               ch[l++] = s[i];
9               ch[l++] = '#';
10          }
11          ch[l] = '\0';
12          int mx = 0, id = 0;
13          for(int i = 0; i < l; i++) {
14              p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
15              while(ch[i + p[i]] == ch[i - p[i]]) p[i]++;
16              if(i + p[i] > mx) mx = i + p[i], id = i;
17          }
18      }
19  }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;
```

01trie 一般也比较常用，是用来维护二进制相关信息的，可以用来求异或最大值。

01trie 是只有两个儿子的 trie 树，一般用于各种与二进制有关的问题中，比如异或最大、维护异或值等。

01trie 维护异或值十分方便，并且支持全局 +1 操作，即一颗字典树上所有的值都 +1；并且 01trie 还可以很方便地合并结点。

下面给出一道例题的代码，用到了全局 +1 以及 trie 树合并等操作，注意代码里面是动态开点不然空间存不下：

```
//维护异或和
void pushup(int o) {
    w[o] = w[ls] + w[rs];
    sum[o] = 0;
    if (ls) sum[o] = (sum[ls] << 1);
    if (rs) sum[o] ^= ((sum[rs] << 1) | (w[rs] & 1));
}

void insert(int& o, int v, int d, int op) {
    if (!o) o = ++tot;
    if (d == MAXH) {
        w[o] += op;
        return;
    }
    insert(ch[o][v & 1], v >> 1, d + 1, op);
    pushup(o);
}
```

```
18  // 全局+1操作，本质是交换左右两个儿子
19  void add(int o) {
20      swap(ls, rs);
21      if (ls) add(ls);
22      pushup(o);
23  }
24  //字典树合并
25  int merge(int x, int y) {
26      if (!x) return y;
27      if (!y) return x;
28      w[x] += w[y];
29      sum[x] ^= sum[y];
30      ch[x][0] = merge(ch[x][0], ch[y][0]);
31      ch[x][1] = merge(ch[x][1], ch[y][1]);
32      return x;
33  }
34
35  ll ans;
36
37  void dfs(int u) {
38      int sons = 0;
39      for (auto v : G[u]) {
40          dfs(v);
41          add(rt[v]);
42          if (++sons > 1) {
43              rt[u] = merge(rt[u], rt[v]);
44          } else {
45              rt[u] = rt[v];
46          }
47      }
48      insert(rt[u], a[u], 0, 1);
49      ans += sum[rt[u]];
50  }
51
52  void run() {
53      cin >> n;
54      for (int i = 1; i <= n; i++) {
55          cin >> a[i];
56      }
57      for (int i = 2; i <= n; i++) {
58          int f;
59          cin >> f;
60          G[f].push_back(i);
61      }
62      dfs(1);
63      cout << ans << '\n';
64  }
```

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;
            else {
                int nq = ++tot; dian[nq] = dian[q];
                dian[nq].len = dian[p].len + 1;
                dian[q].fa = dian[np].fa = nq;
                for(; p && dian[p].ch[c] == q; p = dian[p].fa) dian[p].ch[c] = nq
                    ;
            }
        }
    }
    void withdraw(int lens) { //越短个数越多
        while(now && dian[dian[now].fa].len >= lens) now = dian[now].fa;
        if(now == 0) now = 1;
    }
    void trans(int t, int lens) {
        now = dian[now].ch[t];
        withdraw(lens);
    }
    bool match(int t) {
        return dian[now].ch[t];
    }
}A;
```

```
ll dp[N];
int main() {
    ios::sync_with_stdio(false); cin.tie(0);
    while(cin >> s + 1) {
        int n = strlen(s + 1);
        cin >> p >> q;
        A.init(n);
        int l = 2, r = 1;
        A.add(s[1] - 'a'); dp[1] = p;
        for(int i = 2; i <= n; i++) {
            ++r; int tmp = s[i] - 'a';
            dp[i] = dp[i - 1] + p;
            while((!A.match(tmp) || r - l + 1 > i / 2) && l <= r) {
                A.add(s[l++] - 'a');
                A.withdraw(r - l);
            }
            A.trans(tmp, r - l + 1);
            dp[i] = min(dp[i], dp[l - 1] + q);
        }
        cout << dp[n] << '\n';
    }
    return 0;
}
```

3.10 广义后缀自动机

```
//广义后缀自动机，分在线和离线两种算法
//离线需要先构造出trie树，之后直接在trie树上bfs添加结点即可，一般比较麻烦，必须建出显式trie树
//在线算法直接插入即可，注意1：需要调整last，从last开始插入；注意2：要加上SAM中被注释的部分，这就是普通sam和广义sam的区别
//N:结点个数，M:字符集大小，sam需要乘以一个2
//其它操作和普通sam类似
struct SAM {
    const static int MAXNODE = N * M * 2;
    const static int M = 10;  //
    int go[MAXNODE][M], link[MAXNODE], len[MAXNODE];
    int last, sz, root;

    int newnode() {
        ++sz;
        memset(go[sz], 0, sizeof(go[sz]));
        return sz;
    }

    void init() {
```

```
19          sz = 0;
20          root = last = newnode();
21          len[root] = link[root] = 0;
22      }
23
24      int split(int p, int q, int ch) {
25          int clone = newnode();
26          memcpy(go[clone], go[q], sizeof(go[q]));
27          link[clone] = link[q];
28          link[q] = clone;
29          len[clone] = len[p] + 1;
30          for (int i = p; i && go[i][ch] == q; i = link[i]) {
31              go[i][ch] = clone;
32          }
33          return clone;
34      }
35
36      void insert(int ch) {
37          // if (go[last][ch]) {
38          //     int q = go[last][ch];
39          //     last = len[last] + 1 == len[q] ? q : split(last, q, ch);
40          //     return;
41          // }
42          // ----
43          int cur = newnode();
44          len[cur] = len[last] + 1;
45          int p = last;
46          for (; p && !go[p][ch]; p = link[p]) {
47              go[p][ch] = cur;
48          }
49          if (!p) {
50              link[cur] = root;
51          } else {
52              int q = go[p][ch];
53              link[cur] = len[p] + 1 == len[q] ? q : split(p, q, ch);
54          }
55          last = cur;
56      }
57
58      ll solve() {
59          ll ans = 0;
60          for (int i = root + 1; i <= sz; ++i) {
61             ans += len[i] - len[link[i]];
62          }
63          return ans;
64      }
65  }sam;
```

```
66  int sam_id[N * M];
67  struct Trie {
68      int ch[N * M][M];
69      int root, cnt, last;
70
71      int newnode() {
72          memset(ch[++cnt], 0, sizeof(ch[cnt]));
73          return cnt;
74      }
75
76      void init() {
77          cnt = 0;
78          root = newnode();
79      }
80
81      int insert(int last, int x) {
82          if (!ch[last][x]) {
83              ch[last][x] = newnode();
84          }
85          return ch[last][x];
86      }
87
88      void bfs() {
89          int p = root;
90          sam.init();
91          sam_id[root] = sam.root;
92          queue <int> q;
93          q.push(p);
94          while (!q.empty()) {
95              int u = q.front();
96              q.pop();
97              for (int i = 0; i < M; i++) {
98                  if (ch[u][i]) {
99                      sam.last = sam_id[u];
100                     sam.insert(i);
101                     sam_id[ch[u][i]] = sam.last;
102                     q.push(ch[u][i]);
103                 }
104             }
105         }
106     }
107 }trie;
```

3.11 最小表示法

```
1 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;
}
```

3.12 lydon 分解

```
//Lydon分解的duval算法
//Lydon串本质是当前串就为最小这个串的最小表示法，所以Lydon分解就是将整个串划分为若干个最小表示法的串，并且这种划分具有唯一性。
//Lydon分解：将串$s$分解为若干个Lydon串$s_1,s_2,\cdots,s_k$，并且满足$s_1>=s_2>=...>=s_k$。
//duval算法核心思想就是通过两个指针来操作，然后根据大小关系来进行指针移位。
//若$k$指针移到$i$，说明找到了一个Lydon串，然后开启一个新的循环；否则就说明继续跑循环节或者Lydon串已经截止。
vector<string> duval(string const& s) {
  int n = s.size(), i = 0;
  vector<string> factorization;
  while (i < n) {
    int j = i + 1, k = i;
    while (j < n && s[k] <= s[j]) {
      if (s[k] < s[j])
        k = i;
      else
        k++;
      j++;
    }
    while (i <= k) {
      factorization.push_back(s.substr(i, j - k));
      i += j - k;
    }
  }
  return factorization;
}
```

第四章　数据结构

4.1　树状数组

```
struct BIT {
    int c[N];
    int lowbit(int x) {return x & (-x);}
    void add(int x, int v = 1) {
        for(; x < N; x += lowbit(x)) c[x] += v;
    }
    int query(int x) {
        int res = 0;
        for(; x; x -= lowbit(x)) res += c[x];
        return res;
    }
    int query(int l, int r) {
        return query(r) - query(l - 1);
    }
    //MAX=2^t < N
    int kth(int k) {
      int p = 0;
      for (int i = MAX >> 1; i; i >>= 1) {
          if (c[p + i] < k) {
              k -= c[p + i];
              p += i;
          }
      }
      return p + 1;
    }
}bit;
```

4.2　笛卡尔树

```
// 每个结点由二元组(k,w)组成，k满足二叉搜索树的性质，w满足堆的性质
// 构建的过程中用栈来维护，有点像单调栈？
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;
```

4.3 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];
            }
        }
        return x == 0;
    }
    int calc(ui x) {
        int ans = 0;
        for(int i = 31; i >= 0; i--) {
            if(x >> i) {
                x ^= r[i];
                ans ^= f[i];
            }
        }
        return ans;
    }
};
```

```
node _merge(node u, node v) {
    node tmp, res; res.clear();
    tmp = u;
    for(int i = 31; i >= 0; i--) {
        ui x = v.r[i];
        if(tmp.find(x)) {
            res.ins(x ^ tmp.calc(x));
        } else tmp.ins2(x);
    }
    return res;
}
ui a[N][33];
node b[N << 2];
void build(int o, int l, int r) {
    if(l == r) {
        b[o].clear();
        for(int j = 1; j <= 32; j++) b[o].ins(a[l][j]);
        return ;
    }
    int mid = (l + r) >> 1;
    build(o << 1, l, mid); build(o << 1|1, mid + 1, r);
    b[o] = _merge(b[o << 1], b[o << 1|1]);
}
bool query(int o, int l, int r, int L, int R, ui v) {
    if(L <= l && r <= R) {
        return b[o].find(v);
    }
    int mid = (l + r) >> 1;
    bool ans = 1;
    if(L <= mid) ans &= query(o << 1, l, mid, L, R, v);
    if(R > mid) ans &= query(o << 1|1, mid + 1, r, L, R, v) ;
    return ans;
}
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;
```

```
}
```

4.4 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;
    }
    int mid = (l + r) >> 1;
    double x = (tr[o].b - b) / (k - tr[o].k);
    if(x <= mid) {
        if(l1 > l2) update(o << 1, l, mid, tr[o].id, tr[o].k, tr[o].b), tr[o].upd
            (id, k, b);
        else update(o << 1, l, mid, id, k, b);
    } else {
        if(l1 > l2) update(o << 1|1, mid + 1, r, id, k, b);
        else update(o << 1|1, mid + 1, r, tr[o].id, tr[o].k, tr[o].b), tr[o].upd(
            id, k, b);
    }
}
```

```
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);
71     int lastans = 0, cnt = 0;
72     for(int i = 1; i <= n; i++) {
73         int op; scanf("%d", &op);
74         if(op == 0) {
75             int k; scanf("%d", &k);
76             int x = (k + lastans - 1) % 39989 + 1;
77             int o = query(1, 1, MAX, x);
78             lastans = tr[o].id;
79             printf("%d\n", lastans);
80         } else {
81             int x1, x2, y1, y2; ++cnt;
82             scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
83             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;
}
```

4.5 长链剖分

一般用于处理与深度有关的 dp 问题，因为长链剖分本质上是按照深度来进行剖分的，对于重儿子直接转移。不过转移的过程用到了指针来优化。

```
ll *f[N], *g[N], ans;
ll tmp[N << 2], *id = tmp;

int len[N], bson[N];
void dfs(int u, int fa) {
    int Max = 0;
    for (auto v : G[u]) if (v != fa) {
        dfs(v, u);
        if (len[v] > Max) {
            Max = len[v];
            bson[u] = v;
        }
    }
    len[u] = len[bson[u]] + 1;
}
void dfs2(int u, int fa) {
    f[u][0] = 1;
    if (bson[u]) {
        //处理重链
        int v = bson[u];
        f[v] = f[u] + 1;
        g[v] = g[u] - 1;
        dfs2(v, u);
    }
    ans += g[u][0];
    for (auto v : G[u]) {
        if (v == fa || v == bson[u]) continue;
        //分配空间
        f[v] = id, id += (len[v] << 1);
        g[v] = id, id += (len[v] << 1);
```

```
31          dfs2(v, u);
32          //从轻链转移
33          for (int i = 0; i < len[v]; i++) {
34              ans += f[v][i] * g[u][i + 1];
35              if (i) {
36                  ans += f[u][i - 1] * g[v][i];
37              }
38          }
39          for (int i = 1; i <= len[v]; i++) {
40              if (i < len[v]) {
41                  g[u][i - 1] += g[v][i];
42              }
43              g[u][i] += f[u][i] * f[v][i - 1];
44              f[u][i] += f[v][i - 1];
45          }
46      }
47  }
48
49  dfs(1, 0);
50  f[1] = id, id += (len[1] << 1);
51  g[1] = id, id += (len[1] << 1);
52  dfs2(1, 0);
```

4.6 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。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int N = 50005;
5
6  int T, q;
7
8  struct node{
9      int x, y, z, op, id, part;
10 }a[N * 10], b[N * 10], d[N * 10];
11
12 int hs[N << 2];
13
14 bool isq[N];
15 int ans[N];
16
17 int c[N << 2];
```

```
18 int lowbit(int x) {return x & (-x);}
19
20 void update(int x, int v) {
21     for(; x < N << 2; x += lowbit(x)) c[x] += v;
22 }
23
24 int query(int x) {
25     int ans = 0;
26     for(; x; x -= lowbit(x)) ans += c[x];
27     return ans;
28 }
29
30 void cdq2(int l, int r) {
31     if(l == r) return ;
32     int mid = (l + r) >> 1;
33     cdq2(l, mid); cdq2(mid + 1, r);
34     int t1 = l, t2 = mid + 1;
35     for(int i = l; i <= r; i++) {
36         if(t2 > r || (t1 <= mid && b[t1].y <= b[t2].y)) {
37             if(b[t1].part == 0 && b[t1].op == 0) {
38                 update(b[t1].z, 1);
39             }
40             d[i] = b[t1++];
41         } else {
42             if(b[t2].part && b[t2].op != 0) {
43                 ans[b[t2].id] += b[t2].op * query(b[t2].z);
44             }
45             d[i] = b[t2++];
46         }
47     }
48     for(int i = l; i <= mid; i++) {
49         if(b[i].part == 0 && b[i].op == 0) update(b[i].z, -1);
50     }
51     for(int i = l; i <= r; i++) b[i] = d[i];
52 }
53
54 void cdq(int l, int r) {
55     if(l == r) return ;
56     int mid = (l + r) >> 1;
57     cdq(l, mid); cdq(mid + 1, r);
58     int t1 = l, t2 = mid + 1;
59     for(int i = l; i <= r; i++) {
60         if(t2 > r || (t1 <= mid && a[t1].x <= a[t2].x)) {
61             b[i] = a[t1++];
62             b[i].part = 0;
63         } else {
64             b[i] = a[t2++];
```

```
65              b[i].part = 1;
66          }
67      }
68      for(int i = l; i <= r; i++) a[i] = b[i];
69      cdq2(l, r);
70  }
71
72  int main() {
73  //    freopen("input.in", "r", stdin);
74      ios::sync_with_stdio(false); cin.tie(0);
75      cin >> T;
76      while(T--) {
77          cin >> q;
78          for(int i = 1; i <= q; i++) ans[i] = 0, isq[i] = false;
79          int cnt = 0; hs[0] = 0;
80          for(int i = 1; i <= q; i++) {
81              int op; cin >> op;
82              if(op == 1) {
83                  int x, y, z; cin >> x >> y >> z;
84                  a[++cnt] = {x, y, z, 0, i, -1};
85                  hs[++hs[0]] = z;
86              } else {
87                  isq[i] = true;
88                  int x1, y1, z1, x2, y2, z2;
89                  cin >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
90                  --x1, --y1, --z1;
91                  a[++cnt] = {x1, y1, z1, -1, i, -1};
92                  a[++cnt] = {x1, y2, z1, 1, i, -1};
93                  a[++cnt] = {x2, y2, z1, -1, i, -1};
94                  a[++cnt] = {x2, y1, z1, 1, i, -1};
95                  a[++cnt] = {x1, y1, z2, 1, i, -1};
96                  a[++cnt] = {x1, y2, z2, -1, i, -1};
97                  a[++cnt] = {x2, y2, z2, 1, i, -1};
98                  a[++cnt] = {x2, y1, z2, -1, i, -1};
99                  hs[++hs[0]] = z1, hs[++hs[0]] = z2;
100             }
101         }
102         sort(hs + 1, hs + hs[0] + 1);
103         hs[0] = unique(hs + 1, hs + hs[0] + 1) - hs - 1;
104         for(int i = 1; i <= cnt; i++) a[i].z = lower_bound(hs + 1, hs + hs[0] +
                1, a[i].z) - hs;
105         cdq(1, cnt);
106         for(int i = 1; i <= q; i++) {
107             if(isq[i]) cout << ans[i] << '\n';
108         }
109     }
110     return 0;
```

```
}
```

4.7 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。

```
#include <bits/stdc++.h>
#define MP make_pair
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int N = 2e5 + 5;
int n, m;
struct node{
    int u, v, l, r;
}a[N];
int b[N], D;
vector <int> c[N << 2];
vector <pair<int, int> > d[100];
ll ans;
void insert(int o, int l, int r, int L, int R, int id) {
    if(L <= l && r <= R) {
        c[o].push_back(id);
        return;
    }
    int mid = (l + r) >> 1;
    if(L <= mid) insert(o << 1, l, mid, L, R, id);
    if(R > mid) insert(o << 1|1, mid + 1, r, L, R, id);
}
int f[N], sz[N];
int find(int x) {
    return f[x] == x ? x : find(f[x]);
}
void merge(int x, int y, int dep) {
    int fx = find(x), fy = find(y);
    if(fx == fy) return;
    if(sz[fx] > sz[fy]) swap(fx, fy);
    int tmp = 0;
    f[fx] = fy;
    if(sz[fx] == sz[fy]) tmp++;
    sz[fy] += tmp;
    d[dep].push_back(MP(fx, tmp));
}
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;
}
```

4.8 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;
        }
```

```
    }
    int build(int l, int r, int now) {
        D = now;
        int mid = (l + r) >> 1;
        nth_element(p + l, p + mid, p + r + 1);
        tr[mid].t = p[mid];
        tr[mid].lz = false;
        tr[mid].col = 1;
        if(l < mid) tr[mid].l = build(l, mid - 1, now ^ 1);
        else tr[mid].l = 0;
        if(r > mid) tr[mid].r = build(mid + 1, r, now ^ 1);
        else tr[mid].r = 0;
        push_up(mid);
        return mid;
    }
    int query(int o, Point T, int now) {
        if(o == 0) return 0;
        if(tr[o].t == T) return tr[o].col;
        push_down(o);
        D = now;
        if(T.d[D] < tr[o].t.d[D]) return query(tr[o].l, T, now ^ 1);
        else return query(tr[o].r, T, now ^ 1);
    }
    void update(int o, int l, int r, int d, int u, int c) {
        if(tr[o].mn[0] >= l && tr[o].mx[0] <= r && tr[o].mn[1] >= d && tr[o].mx
            [1] <= u) {
            tr[o].col = c; tr[o].lz = true;
            return;
        }
        if(tr[o].mn[0] > r || tr[o].mx[0] < l || tr[o].mn[1] > u || tr[o].mx[1] <
             d) return;
        push_down(o);
        if(tr[o].t[0] >= l && tr[o].t[0] <= r && tr[o].t[1] >= d && tr[o].t[1] <=
             u) {
            tr[o].col = c;
        }
        if(tr[o].l) update(tr[o].l, l, r, d, u, c);
        if(tr[o].r) update(tr[o].r, l, r, d, u, c);
    }
}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。

```
int n;
int D;
struct Point {
    int d[2], val;
}tmp[N], T;
```

```
6  struct Node {
7      int mn[2], mx[2];
8      int l, r, sumv, sz;
9      Point t;
10 }tr[N];
11 bool operator < (const Point &A, const Point &B) {
12     return A.d[D] < B.d[D];
13 }
14 int rt;
15 int rub[N], top, tot;
16 struct kdtree {
17     const double E = 0.75;
18     int ans;
19     int new_node() {
20         if(top) return rub[top--];
21         return ++tot;
22     }
23     void push_up(int o) {
24         int ls = tr[o].l, rs = tr[o].r;
25         for(int i = 0; i < 2; i++) {
26             tr[o].mn[i] = tr[o].mx[i] = tr[o].t.d[i];
27             if(ls) {
28                 tr[o].mn[i] = min(tr[o].mn[i], tr[ls].mn[i]);
29                 tr[o].mx[i] = max(tr[o].mx[i], tr[ls].mx[i]);
30             }
31             if(rs) {
32                 tr[o].mn[i] = min(tr[o].mn[i], tr[rs].mn[i]);
33                 tr[o].mx[i] = max(tr[o].mx[i], tr[rs].mx[i]);
34             }
35         }
36         tr[o].sumv = tr[ls].sumv + tr[rs].sumv + tr[o].t.val;
37         tr[o].sz = 1 + tr[ls].sz + tr[rs].sz;
38     }
39     void pia(int o, int num) {
40         int ls = tr[o].l, rs = tr[o].r;
41         if(ls) pia(ls, num);
42         tmp[tr[ls].sz + num + 1] = Point{tr[o].t.d[0], tr[o].t.d[1], tr[o].t.val
               };
43         rub[++top] = o;
44         if(rs) pia(rs, tr[ls].sz + num + 1);
45     }
46     int rebuild(int l, int r, int now) {
47         if(l > r) return 0;
48         D = now;
49         int mid = (l + r) >> 1;
50         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;
```

4.9 树链剖分

```
1   // 树链剖分
```

```
// 轻边个数为O(logn)，重链个数为O(logn)，top数组记录的为重链顶点
// 注意bson的初始化，其余可以自动初始化
// 注意每个点实际值为dfn[x]
// 处理一条链的信息，如果两条链就先跳到一条链上去，然后再操作一次就行
int sz[N], deep[N], bson[N], ff[N];
int top[N], dfn[N], T;
void dfs(int u, int fa) {
    deep[u] = deep[fa] + 1;
    sz[u] = 1;
    ff[u] = fa;
    int Max = -1;
    for (auto v : G[u]) {
        if (v != fa) {
            dfs(v, u);
            sz[u] += sz[v];
            if (sz[v] > Max) {
                Max = sz[v];
                bson[u] = v;
            }
        }
    }
}

void dfs(int u, int fa, int topf) {
    dfn[u] = ++T;
    top[u] = topf;
    if (bson[u] != 0) {
        dfs(bson[u], u, topf);
    }
    for (auto v : G[u]) {
        if (v != fa && v != bson[u]) {
            dfs(v, u, v);
        }
    }
}
```

4.10 Splay

核心思想是通过 splay 操作以及上旋操作以保证树随时平衡并且始终满足二叉搜索树的性质。

其余的 insert、delete、kth 等操作都和普通的二叉搜索树类似，只是多了一个上旋操作。

* 注意查找某个信息（比如值为 x 的排名、x 的前驱等），若当前权值不存在，先插入并且旋转为根，那么就可以直接开始往下搜答案，最后再删除这个结点即可

* 能实现众多功能，亮点在于区间翻转，缺点为常数较大

第一份代码是动态插入、删除，第二种是一开始根据序列建一颗二叉搜索树，同时维护结点

信息，一般第二种比较常用。第二种我们删除、插入结点只需要通过 splay 操作找到对应的位置进行操作即可。

模板如下：

```
struct Splay {
    // key: 结点的权值  cnt: 结点的个数  size: 子树大小  sz: 结点总数
    int f[N], ch[N][2], key[N], cnt[N], size[N], sz = 0, root = 0;
    Splay() {sz = root = 0;}
    // 清空结点信息
    inline void clear(int x) {
        ch[x][0] = ch[x][1] = f[x] = key[x] = cnt[x] = size[x] = 0;
    }
    // 判断当前结点是否为左右儿子
    inline int get(int x) {
        return ch[f[x]][1] == x;
    }
    // 更新结点信息
    inline void update(int x) {
        if (!x) return;
        size[x] = cnt[x];
        if (ch[x][0]) size[x] += size[ch[x][0]];
        if (ch[x][1]) size[x] += size[ch[x][1]];
    }
    // 上旋操作
    inline void rot(int x) {
        int old = f[x], oldf = f[old], tp = get(x);
        ch[old][tp] = ch[x][tp ^ 1]; f[ch[old][tp]] = old;
        f[old] = x; ch[x][tp ^ 1] = old;
        f[x] = oldf;
        if (oldf) ch[oldf][ch[oldf][1] == old] = x;
        update(old); update(x);
    }
    // 将结点x伸展到根结点以维持整棵树平衡
    inline void splay(int x) {
        for (int fa; fa = f[x], fa; rot(x))
            if (f[fa]) rot(get(x) == get(fa) ? fa : x);
        root = x;
    }
    // 插入结点并且上旋为根结点
    inline int insert(int v) {
        if (root == 0) {
            ++sz; ch[sz][0] = ch[sz][1] = f[sz] = 0;
            key[sz] = v; cnt[sz] = 1; size[sz] = 1;
            root = sz;
            update(root);
            return 1;
        }
```

```
44          int tot = root, fa = 0;
45          while (1) {
46              if (key[tot] == v) {
47                  cnt[tot]++; update(tot), update(fa);
48                  splay(tot);
49                  return cnt[tot];
50              }
51              fa = tot;
52              tot = ch[tot][v > key[tot]];
53              if (tot == 0) {
54                  ++sz; ch[sz][0] = ch[sz][1] = 0;
55                  key[sz] = v; cnt[sz] = size[sz] = 1;
56                  f[sz] = fa; ch[fa][v > key[fa]] = sz;
57                  update(sz); update(fa); splay(sz);
58                  break;
59              }
60          }
61          return 1;
62      }
63      // 查找v的排名，即比v小的数的个数+1，并将其伸展为根结点
64      inline int rk(int v) {
65          int ans = 0, tot = root;
66          while (1) {
67              if (v < key[tot]) tot = ch[tot][0];
68              else {
69                  ans += (ch[tot][0] ? size[ch[tot][0]] : 0);
70                  if (v == key[tot]) {
71                      splay(tot); return ans + 1;
72                  }
73                  ans += cnt[tot];
74                  tot = ch[tot][1];
75              }
76          }
77          return 0;
78      }
79      // 查找第k大的值
80      inline int kth(int x) {
81          int tot = root;
82          while (1) {
83              if (ch[tot][0] && x <= size[ch[tot][0]]) tot = ch[tot][0];
84              else {
85                  int t = (ch[tot][0] ? size[ch[tot][0]] : 0) + cnt[tot];
86                  if (x <= t) return key[tot];
87                  x -= t; tot = ch[tot][1];
88              }
89          }
90          return -1;
```

```
91          }
92          // 返回根节点前驱所在位置
93          inline int prev() {
94              int tot = ch[root][0];
95              while (ch[tot][1]) tot = ch[tot][1];
96              return tot;
97          }
98          // 返回根节点后驱所在位置
99          inline int succ() {
100             int tot = ch[root][1];
101             while (ch[tot][0]) tot = ch[tot][0];
102             return tot;
103         }
104         // 删除一个结点，若有两个儿子则取一个前驱/后继作为根，另一个直接拼上
105         inline void del(int v) {
106             rk(v);
107             if (cnt[root] > 1) {
108                 --cnt[root]; update(root); return;
109             }
110             if (!ch[root][0] && !ch[root][1]) {
111                 clear(root); root = 0; return;
112             }
113             if (!ch[root][0]) {
114                 int old = root; root = ch[root][1]; f[root] = 0;
115                 clear(old); return;
116             } else if (!ch[root][1]) {
117                 int old = root; root = ch[root][0]; f[root] = 0;
118                 clear(old); return;
119             }
120             int x = prev(), old = root;
121             splay(x); f[ch[old][1]] = x; ch[x][1] = ch[old][1];
122             clear(old); update(root);
123         }
124     }tr;
```

下面是实现区间翻转的代码，有几个区别：

1. 注意 rev 标记的下传

2. 头尾哨兵结点的建立，这样方便找 l-1 和 r+1 的位置，这样就容易得到 [l,r] 区间的子树

3.rotate 操作到指定结点，并且 rotate 操作时也会下传对应标记以保证正确

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   #define INF 0x3f3f3f3f
4   const int N = 1e5 + 5;
5
6   int a[N];
7   struct Splay{
8       int f[N], ch[N][2], size[N], rev[N] = {0}, key[N], root, sz;
```

```
9       Splay(){
10          root = sz = size[0] = rev[0] = f[0] = 0; key[0] = INF;
11      }
12      inline int get(int x){return ch[f[x]][1] == x;}
13      inline void update(int x){
14          size[x] = size[ch[x][0]] + size[ch[x][1]] + 1;
15      }
16      inline void pushdown(int x){
17          if (!rev[x]) return;
18          swap(ch[x][0], ch[x][1]);
19          rev[ch[x][0]] ^= 1; rev[ch[x][1]] ^= 1; rev[x] = 0;
20      }
21      // 建二叉搜索树
22      inline int build(int l, int r, int rt){
23          if (l > r) return 0;
24          int m = (l + r) >> 1, tot = ++sz;
25          key[tot] = a[m]; f[tot] = rt; rev[tot] = 0;
26          ch[tot][0] = build(l, m - 1, tot);
27          ch[tot][1] = build(m + 1, r, tot);
28          update(tot);
29          return tot;
30      }
31      inline void rot(int x){
32          int old = f[x], oldf = f[old], tp = get(x);
33          pushdown(old); pushdown(x); // 注意这里的区别
34          ch[old][tp] = ch[x][tp ^ 1]; f[ch[old][tp]] = old;
35          ch[x][tp ^ 1] = old; f[old] = x;
36          f[x] = oldf;
37          if (oldf) ch[oldf][ch[oldf][1] == old] = x;
38          update(old); update(x);
39      }
40      inline void splay(int x, int tar){ // 旋转到对应位置
41          for (int fa; (fa = f[x]) != tar; rot(x))
42              if (f[fa] != tar)
43                  rot(get(fa) == get(x) ? fa : x);
44          if (!tar) root = x;
45      }
46      inline int rk(int k){
47          int tot = root;
48          while (1){
49              pushdown(tot); // !!
50              if (k <= size[ch[tot][0]]) tot = ch[tot][0];
51              else {
52                  k -= size[ch[tot][0]] + 1;
53                  if (!k) return tot;
54                  tot = ch[tot][1];
55              }
```

```
56          }
57      }
58      inline void reverse(int l, int r){
59          int a = rk(l), b = rk(r + 2); // 找到对应位置
60          splay(a, 0); splay(b, a); // 分别旋转至对应位置
61          rev[ch[b][0]] ^= 1; // 打上翻转标记
62      }
63      void print(int p){
64          pushdown(p);
65          if (ch[p][0]) print(ch[p][0]);
66          if (key[p] != INF && key[p] > 0) printf("%d ", key[p]);
67          if (ch[p][1]) print(ch[p][1]);
68      }
69  }tr;
70
71  int n, m;
72  void run() {
73      cin >> n >> m;
74      for (int i = 1; i <= n; i++) {
75          a[i + 1] = i;
76      }
77      a[1] = 0, a[n + 2] = INF;
78      tr.root = tr.build(1, n + 2, 0);
79      while (m--) {
80          int l, r;
81          cin >> l >> r;
82          tr.reverse(l, r);
83      }
84      tr.print(tr.root);
85  }
86  int main() {
87      run();
88      return 0;
89  }
```

4.11 LCT

伸缩自如、变化万千的高级数据结构!

跟 splay 类似，通过几个核心函数就能操作整个世界。

lct 的思想基于实链剖分，每个点最多一条实边连接其儿子，而这个实边是我们自己决定的，所以就很灵活。多条相邻实边形成一条实链，每条实链我们用一个 splay 去维护，满足这个 splay 的中序遍历为这条链自顶向下进行遍历的序列。并且还有一点：认父不认子。所以虚实边转化时原来的实边直接丢掉，直接将一条虚边变为实边即可。具体来说就是添加儿子、修改父亲。

核心就为 access 操作和 makeroot 操作，一个是将 x 到根节点的边全拉实，一个是换树根。

* 注意 pushup 和 pushdown 的使用，稍微不注意就可能会有大问题。

* 注意 lct 的懒标记思想，每次是在一个 splay 的顶点打上标记就行，然后标记会自动下传不需要关心。具体原理就是每次实边转化为虚边时懒标记会提前下传，并且我们访问一个连通块时（splay、access 操作）都会自顶向下传。所以其实只用关心懒标记如何打就行，最后直接询问顶点信息即可。

* 遇事不决 makeroot。

* 可以动态维护生成树，但因为 lct 不删除处理边权的信息，所以可以将每条边拆为两个没有点权的点连接一个有点权的点，那么每次删边、添边做对应的两次操作就行。

```
// LCT 维护路径异或值
// 虚实相转化的数据结构，配合splay更加灵活与强大
// 可支持动态添边、删边、换根等操作，也可代替树剖，因为树是动态的
// 特点：1.认父不认子，与儿子的边可以随便断开，但是父亲始终知道是哪个；
// 2.中序遍历为树上的一条深度严格递增的实链，每条实链都由一个splay维护；
// 3.实链虚链可以相互转化，我们自己确定
struct LCT {
    #define ls ch[x][0]
    #define rs ch[x][1]
    int ch[N][2], fa[N], sum[N], val[N];
    int tag[N], siz[N];
    inline void init(int n) {
        for (int i = 1; i <= n; i++) {
            ch[i][0] = ch[i][1] = 0;
            fa[i] = sum[i] = val[i] = 0;
            tag[i] = siz[i] = 0;
        }
    }
    // 注意pushup在哪里用，一般是进行上旋操作时用到
    inline void pushup(int x) {
        siz[x] = siz[ls] + siz[rs] + 1;
        sum[x] = sum[ls] ^ sum[rs] ^ val[x];
    }
    inline void pushdown(int x) {
        if (tag[x]) {
            if (ls) swap(ch[ls][0], ch[ls][1]), tag[ls] ^= tag[x];
            if (rs) swap(ch[rs][0], ch[rs][1]), tag[rs] ^= tag[x];
            tag[x] = 0;
        }
    }
    // 标记要先下传，避免奇奇怪怪的错误
    void update(int x) {
        if (!isroot(x)) update(fa[x]);
        pushdown(x);
    }
    inline int get(int x) {
        return x == ch[fa[x]][1];
```

```
38      }
39      inline bool isroot(int x) {
40          return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
41      }
42      // 与splay类似，多了一个isroot的判断，rotate时会变换父亲
43      inline void rotate(int x) {
44          int y = fa[x], z = fa[y], k = get(x);
45          if (!isroot(y)) ch[z][ch[z][1] == y] = x;
46          ch[y][k] = ch[x][k ^ 1]; fa[ch[y][k]] = y;
47          ch[x][k ^ 1] = y; fa[y] = x; fa[x] = z;
48          pushup(y), pushup(x);
49      }
50      inline void splay(int x) {
51          update(x);
52          for (int f = fa[x]; !isroot(x); f = fa[x]) {
53              if (!isroot(f)) rotate(get(f) == get(x) ? f : x);
54              rotate(x);
55          }
56      }
57      // 核心操作，将x到当前树中的根这条路径上的边全转化为实边，这样x和根节点在一
            颗splay里面
58      // 返回最后得到splay的根节点，如果连用两次，那么就是两个点的lca
59      inline int access(int x) {
60          int p;
61          for (p = 0; x; p = x, x = fa[x]) {
62              splay(x); ch[x][1] = p; pushup(x);
63          }
64          return p;
65      }
66      // 重要操作，将整棵树的根变为x，实际上就是一段路径的边方向发生翻转，这样才
            能使得任意一条链都满足深度递增
67      inline void makeroot(int x) {
68          x = access(x); swap(ls, rs);
69          tag[x] ^= 1;
70      }
71      // 链接两个点，如果已联通则不链接
72      inline void link(int x, int y) {
73          makeroot(x), splay(x);
74          makeroot(y), splay(y);
75          if (fa[x] || fa[y]) return; // 如果在一连通块中，那么有个必然不能为根，
                所以存在父亲
76          fa[y] = x;
77      }
78      // 提取x~y这条链，并且使得y为splay树根
79      inline void split(int x, int y) {
80          makeroot(x); access(y); splay(y);
81      }
```

```
82      // 切断一条边，注意不存在边的判断
83      inline void cut(int x, int y) {
84          makeroot(y); access(x); splay(x);
85          if (siz[x] != 2 || ls != y) return;
86          ls = fa[y] = 0;
87      }
88      // 寻找x所在连通块的根节点，深度最小那么一直往左子树走即可，最后会使得根节点为splay的树根
89      inline int find(int x) {
90          access(x), splay(x);
91          while (ls) pushdown(x), x = ls;
92          splay(x); return x;
93      }
94      inline void modify(int x, int v) {
95          splay(x); val[x] = v; pushup(x);
96      }
97      inline int query(int x, int y) {
98          makeroot(x); access(y); splay(y);
99          return sum[y];
100     }
101 }lct;
```

打乘法和加法标记的 lct:

```
1   struct LCT {
2       #define ls ch[x][0]
3       #define rs ch[x][1]
4       int ch[N][2], fa[N], sum[N], val[N];
5       int add[N], mul[N], tag[N], siz[N];
6       inline void tagadd(int x, int v) {
7           sum[x] = (sum[x] + 1ll * v * siz[x] % MOD) % MOD;
8           val[x] = (val[x] + v) % MOD;
9           add[x] = (add[x] + v) % MOD;
10      }
11      inline void tagmul(int x, int v) {
12          sum[x] = 1ll * sum[x] * v % MOD;
13          val[x] = 1ll * val[x] * v % MOD;
14          add[x] = 1ll * add[x] * v % MOD;
15          mul[x] = 1ll * mul[x] * v % MOD;
16      }
17      inline void init(int n) {
18          for (int i = 1; i <= n; i++) {
19              ch[i][0] = ch[i][1] = 0;
20              fa[i] = sum[i] = 0;
21              siz[i] = 0;
22              add[i] = 0, mul[i] = val[i] = 1;
23          }
24      }
```

```
25      inline void pushup(int x) {
26          siz[x] = siz[ls] + siz[rs] + 1;
27          sum[x] = (sum[ls] + sum[rs] + val[x]) % MOD;
28      }
29      inline void pushdown(int x) {
30          if (tag[x]) {
31              if (ls) swap(ch[ls][0], ch[ls][1]), tag[ls] ^= tag[x];
32              if (rs) swap(ch[rs][0], ch[rs][1]), tag[rs] ^= tag[x];
33              tag[x] = 0;
34          }
35          if (mul[x] != 1) {
36              if (ls) {
37                  tagmul(ls, mul[x]);
38              }
39              if (rs) {
40                  tagmul(rs, mul[x]);
41              }
42              mul[x] = 1;
43          }
44          if (add[x] > 0) {
45              if (ls) {
46                  tagadd(ls, add[x]);
47              }
48              if (rs) {
49                  tagadd(rs, add[x]);
50              }
51              add[x] = 0;
52          }
53      }
54      // 对于一个连通块，下放懒标记
55      void update(int x) {
56          if (!isroot(x)) update(fa[x]);
57          pushdown(x);
58      }
59      inline int get(int x) {
60          return x == ch[fa[x]][1];
61      }
62      inline bool isroot(int x) {
63          return ch[fa[x]][0] != x && ch[fa[x]][1] != x;
64      }
65      inline void rotate(int x) {
66          int y = fa[x], z = fa[y], k = get(x);
67          if (!isroot(y)) ch[z][ch[z][1] == y] = x;
68          ch[y][k] = ch[x][k ^ 1]; fa[ch[y][k]] = y;
69          ch[x][k ^ 1] = y; fa[y] = x; fa[x] = z;
70          pushup(y), pushup(x);
71      }
```

```
72      // 每次splay时都会提前下传上面连通块的懒标记，然后虚实转化就没问题
73      inline void splay(int x) {
74          update(x);
75          for (int f = fa[x]; !isroot(x); f = fa[x]) {
76              if (!isroot(f)) rotate(get(f) == get(x) ? f : x);
77              rotate(x);
78          }
79      }
80      inline int access(int x) {
81          int p;
82          for (p = 0; x; p = x, x = fa[x]) {
83              splay(x); ch[x][1] = p; pushup(x);
84          }
85          return p;
86      }
87      inline void makeroot(int x) {
88          x = access(x); swap(ls, rs);
89          tag[x] ^= 1;
90      }
91      inline void link(int x, int y) {
92          makeroot(x), splay(x);
93          makeroot(y), splay(y);
94          if (fa[x] || fa[y]) return;
95          fa[y] = x;
96      }
97      inline void split(int x, int y) {
98          makeroot(x); access(y); splay(y);
99      }
100     inline void cut(int x, int y) {
101         makeroot(y); access(x); splay(x);
102         if (siz[x] != 2 || ls != y) return;
103         ls = fa[y] = 0;
104     }
105     inline int find(int x) {
106         access(x), splay(x);
107         while (ls) pushdown(x), x = ls;
108         splay(x); return x;
109     }
110     inline int query(int x, int y) {
111         makeroot(x); access(y); splay(y);
112         return sum[y];
113     }
114     void update(int x, int y, int op, int v) {
115         split(y, x);
116         if (op) tagmul(x, v);
117         else tagadd(x, v);
118     }
```

```
119 | }lct;
```

第五章　其它

5.1　高维前缀和

其实就是 FMT。

```
//子集
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(i >> j & 1) f[i] += f[i ^ (1 << j)];

//超集
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(!(i >> j & 1)) f[i] += f[i ^ (1 << j)];
```

5.2　正整数拆分

```
// 广义五边形数 q[i]=(3i^2[+-]i) / 2

// 正整数拆分的生成函数 等于 欧拉函数的倒数
// p[n]=\sum_i (-1)^{i-1} p(n-q_i)
// p[n]=p[n-1]+p[n-2]-p[n-5]-p[n-7]...
// p[n]表示n的正整数拆分数
// 时间复杂度为 O(n\sqrt{n})
void init_p(int n, int* p) {
    for (int i = 0; i <= n; i++) {
        p[i] = 0;
    }
    auto P = [&] (int i) {
        return MP(i * (3 * i - 1) / 2, i * (3 * i + 1) / 2);
    };
    p[0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1;; j++) {
            pii val = P(j);
            if (val.fi > i) break;
            int t = p[i - val.fi];
```

```
21                if (val.se <= i) {
22                    t += p[i - val.se];
23                    if (t >= MOD) t -= MOD;
24                }
25                if (!(j & 1)) p[i] = (p[i] + MOD - t) % MOD;
26                else p[i] = (p[i] + t) % MOD;
27            }
28        }
29    }
```

5.3 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：

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

5.4 二维 rmq

```
1   struct RMQ {
2       int f[N][N][9][9], g[N][N][9][9];
3       int mm[N];
4       void init(int n, int m, int a[][N]) {
5           mm[0] = -1;
6           for (int i = 1; i < N; ++i) {
7               mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
8           }
9           for (int i = 1; i <= n; ++i) {
10              for (int j = 1; j <= m; ++j) {
11                  f[i][j][0][0] = a[i][j];
12                  g[i][j][0][0] = a[i][j];
13              }
14          }
15          for (int ii = 0; ii <= mm[n]; ++ii) {
16              for (int jj = 0; jj <= mm[m]; ++jj) {
17                  if (ii + jj) {
18                      for (int i = 1; i + (1 << ii) - 1 <= n; ++i) {
19                          for (int j = 1; j + (1 << jj) - 1 <= m; ++j) {
20                              if (ii) {
21                                  f[i][j][ii][jj] = max(f[i][j][ii - 1][jj], f[i
                                        + (1 << (ii - 1))][j][ii - 1][jj]);
```

```
                                g[i][j][ii][jj] = min(g[i][j][ii - 1][jj], g[i
                                    + (1 << (ii - 1))][j][ii - 1][jj]);
                            } else {
                                f[i][j][ii][jj] = max(f[i][j][ii][jj - 1], f[i
                                    ][j + (1 << (jj - 1))][ii][jj - 1]);
                                g[i][j][ii][jj] = min(g[i][j][ii][jj - 1], g[i
                                    ][j + (1 << (jj - 1))][ii][jj - 1]);
                            }
                        }
                    }
                }
            }
        }
        void Max(int &x, short y) {
            if (x < y) x = y;
        }
        void Min(int &x, short y) {
            if (x > y) x = y;
        }
        pII query(int x1, int y1, int x2, int y2) {
            int k1 = mm[x2 - x1 + 1];
            int k2 = mm[y2 - y1 + 1];
            x2 = x2 - (1 << k1) + 1;
            y2 = y2 - (1 << k2) + 1;
            pII res = pII(-INF, INF);
            Max(res.fi, f[x1][y1][k1][k2]);
            Max(res.fi, f[x1][y2][k1][k2]);
            Max(res.fi, f[x2][y1][k1][k2]);
            Max(res.fi, f[x2][y2][k1][k2]);
            Min(res.se, g[x1][y1][k1][k2]);
            Min(res.se, g[x1][y2][k1][k2]);
            Min(res.se, g[x2][y1][k1][k2]);
            Min(res.se, g[x2][y2][k1][k2]);
            return res;
        }
    }rmq;
```

5.5 点分治

点分治一般用来解决树上路径问题，不断处理重心其实就相当于枚举经过每个点的路径。时间复杂度为 $O(nlogn)$。

点分树是将点分治的过程保存下来，即每一层的重心与上一层的重心相连，这样可以做一些待修改的点分治问题，只用考虑当前点到根节点这条链上的结点就行。并且点分树的树高为 $O(logn)$ 的，所以可以直接暴力修改信息。

```cpp
// 不要忘了dfs前先findroot一次
int n, m;
vector <pii> G[N];
int tsz, rt;
int sz[N], Max[N], father[N];
bool vis[N];
void getrt(int u, int fa) {
    sz[u] = 1; Max[u] = 0;
    for(auto it : G[u]) {
        int v = it.fi, w = it.se;
        if(v == fa || vis[v]) continue;
        getrt(v, u);
        sz[u] += sz[v];
        if(sz[v] > Max[u]) Max[u] = sz[v];
    }
    Max[u] = max(Max[u], tsz - sz[u]);
    if(Max[u] < Max[rt]) rt = u;
}
void findrt(int u, int fa) {
    tsz = (sz[u] == 0 ? n : sz[u]);
    Max[rt = 0] = INF;
    getrt(u, fa);
}
void dfs(int u, int fa) {
    vis[u] = true;
    for(auto it : G[u]) {
        int v = it.fi;
        if(v != fa && !vis[v]) {
            findrt(v, u);
            father[rt] = u;
            dfs(rt, 0);
        }
    }
}

int f[N][20], deep[N], d[N];
void dfs2(int u, int fa) {
    deep[u] = deep[fa] + 1;
    f[u][0] = fa;
    for(int i = 1; i < 20; i++) {
        f[u][i] = f[f[u][i - 1]][i - 1];
    }
    for(auto it : G[u]) {
        int v = it.fi, w = it.se;
        if(v != fa) {
            d[v] = d[u] + w;
            dfs2(v, u);
```

```
48          }
49      }
50  }
51  int LCA(int x, int y) {
52      if(deep[x] < deep[y]) swap(x, y);
53      for(int i = 19; i >= 0; i--) {
54          if(deep[f[x][i]] >= deep[y]) x = f[x][i];
55      }
56      if(x == y) return x;
57      for(int i = 19; i >= 0; i--) {
58          if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
59      }
60      return f[x][0];
61  }
62
63  int dis(int x, int y) {
64      int z = LCA(x, y);
65      return d[x] + d[y] - 2 * d[z];
66  }
```

5.6 Floyd 判圈法

```
1   // Floyd判圈法
2   // 用于快速判断一个函数的循环节相关信息
3   ll f(ll x) {
4       //函数
5       return (x + (x >> 20) + 12345) % MOD;
6   }
7   //first为起点处的值, len为循环节长度, id为循环节起始处坐标, val为循环节起始处的
        值
8   bool FloydCycle(ll first, ll& len, ll& id, ll& val) {
9       ll slow, fast;
10      slow = f(first);
11      fast = f(f(first));
12      int cnt = 1;
13      while(slow != fast && cnt <= 1000000000)
14      {
15          //快指针的移动速度是慢指针的2倍
16          slow = f(slow);
17          fast = f(f(fast));
18          cnt++;
19      }
20      if(slow != fast) return false;//无环
21
22      len = 1;//环的长度
23      slow = f(slow);
```

```
24        while(slow != fast)
25        {
26            slow = f(slow);
27            len++;
28        }
29
30        id = 0;
31        slow = first;
32        while(slow != fast)
33        {
34            slow = f(slow);
35            fast = f(fast);
36            id++;
37        }
38        val = slow;
39
40        return true;
41    }
```

5.7 随机数

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

5.8 最大子矩阵

```
1  int a[N][N];
2  int n, m;
3  #define mp make_pair
4  #define se second
5  #define fi first
6  namespace max_matrix{
7      int vis[N], f[N], sz[N];
8      int t[N][N];
9      int mx = 0, mx2 = 0;
10     pair<int, int> h[N];
11     int find(int x) {
12         return f[x] == x ? x : f[x] = find(f[x]) ;
13     }
14     void Union(int x, int y) {
15         int fx = find(x), fy = find(y);
16         f[fx] = fy;
17         sz[fy] += sz[fx];
18     }
19     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;
    }
};
```

5.9 快速读入/输出

```
template <class T>
inline void read(T& x) {
    static char c;
    x = 0;
    bool sign = 0;
    while (!isdigit(c = getchar()))
        if (c == '-')
            sign = 1;
    for (; isdigit(c); x = x * 10 + c - '0', c = getchar())
        ;
    if (sign)
        x = -x;
}
```

再来个更快的:

```
  //快读
```

```
2   //记得FO输出字符，最后要Flush
3   #define FI(n) FastIO::read(n)
4   #define FO(n) FastIO::write(n)
5   #define Flush FastIO::Fflush()
6   namespace FastIO {
7     const int SIZE = 1 << 16;
8     char buf[SIZE], obuf[SIZE], str[60];
9     int bi = SIZE, bn = SIZE, opt;
10    double D[] = {0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001,
          0.00000001, 0.000000001, 0.0000000001};
11    int read(char *s) {
12      while (bn) {
13        for (; bi < bn && buf[bi] <= ' '; bi++);
14        if (bi < bn) break;
15        bn = fread(buf, 1, SIZE, stdin);
16        bi = 0;
17      }
18      int sn = 0;
19      while (bn) {
20        for (; bi < bn && buf[bi] > ' '; bi++) s[sn++] = buf[bi];
21        if (bi < bn) break;
22        bn = fread(buf, 1, SIZE, stdin);
23        bi = 0;
24      }
25      s[sn] = 0;
26      return sn;
27    }
28    bool read(int& x) {
29      int n = read(str), bf = 0;
30      if (!n) return 0;
31      int i = 0; if (str[i] == '-') bf = 1, i++; else if (str[i] == '+') i++;
32      for (x = 0; i < n; i++) x = x * 10 + str[i] - '0';
33      if (bf) x = -x;
34      return 1;
35    }
36    bool read(long long& x) {
37      int n = read(str), bf;
38      if (!n) return 0;
39      int i = 0; if (str[i] == '-') bf = -1, i++; else bf = 1;
40      for (x = 0; i < n; i++) x = x * 10 + str[i] - '0';
41      if (bf < 0) x = -x;
42      return 1;
43    }
44    void write(int x) {
45      if (x == 0) obuf[opt++] = '0';
46      else {
47        if (x < 0) obuf[opt++] = '-', x = -x;
```

```
48        int sn = 0;
49        while (x) str[sn++] = x % 10 + '0', x /= 10;
50        for (int i = sn - 1; i >= 0; i--) obuf[opt++] = str[i];
51      }
52      if (opt >= (SIZE >> 1)) {
53        fwrite(obuf, 1, opt, stdout);
54        opt = 0;
55      }
56    }
57    void write(long long x) {
58      if (x == 0) obuf[opt++] = '0';
59      else {
60        if (x < 0) obuf[opt++] = '-', x = -x;
61        int sn = 0;
62        while (x) str[sn++] = x % 10 + '0', x /= 10;
63        for (int i = sn - 1; i >= 0; i--) obuf[opt++] = str[i];
64      }
65      if (opt >= (SIZE >> 1)) {
66        fwrite(obuf, 1, opt, stdout);
67        opt = 0;
68      }
69    }
70    void write(unsigned long long x) {
71      if (x == 0) obuf[opt++] = '0';
72      else {
73        int sn = 0;
74        while (x) str[sn++] = x % 10 + '0', x /= 10;
75        for (int i = sn - 1; i >= 0; i--) obuf[opt++] = str[i];
76      }
77      if (opt >= (SIZE >> 1)) {
78        fwrite(obuf, 1, opt, stdout);
79        opt = 0;
80      }
81    }
82    void write(char x) {
83      obuf[opt++] = x;
84      if (opt >= (SIZE >> 1)) {
85        fwrite(obuf, 1, opt, stdout);
86        opt = 0;
87      }
88    }
89    void Fflush() { if (opt) fwrite(obuf, 1, opt, stdout); opt = 0;}
90  };
```

5.10 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
```

5.11 头文件加速

```
#pragma comment(linker, "/STACK:25600000,25600000")
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
```

```
41  #pragma GCC optimize("inline-small-functions")
42  #pragma GCC optimize("-finline-small-functions")
43  #pragma GCC optimize("-ftree-switch-conversion")
44  #pragma GCC optimize("-foptimize-sibling-calls")
45  #pragma GCC optimize("-fexpensive-optimizations")
46  #pragma GCC optimize("-funsafe-loop-optimizations")
47  #pragma GCC optimize("inline-functions-called-once")
48  #pragma GCC optimize("-fdelete-null-pointer-checks")
```