

SOUTH CHINA UNIVERSITY OF TECHNOLOGY

SCUT\_GUGUGU

# TEMPLATE



**0 error(s), 0 warning(s)**

Last build at April 13, 2019

# Contents

|          |                                      |           |
|----------|--------------------------------------|-----------|
| <b>1</b> | <b>Graph Theory</b>                  | <b>4</b>  |
| 1.1      | Shortest Path . . . . .              | 4         |
| 1.1.1    | Dijkstra . . . . .                   | 4         |
| 1.1.2    | SPFA . . . . .                       | 4         |
| 1.2      | Network Flow . . . . .               | 5         |
| 1.2.1    | ISAP . . . . .                       | 5         |
| 1.2.2    | HLPP . . . . .                       | 6         |
| 1.2.3    | Dinic . . . . .                      | 7         |
| 1.2.4    | MCMF . . . . .                       | 8         |
| 1.3      | Tree Related . . . . .               | 9         |
| 1.3.1    | Kruskal . . . . .                    | 9         |
| 1.3.2    | Prim . . . . .                       | 10        |
| 1.3.3    | Tree Divide and Conquer . . . . .    | 10        |
| 1.4      | LCA . . . . .                        | 12        |
| 1.4.1    | Tree Decomposition LCA . . . . .     | 12        |
| 1.4.2    | Tarjan LCA . . . . .                 | 12        |
| 1.5      | Tarjan . . . . .                     | 13        |
| 1.5.1    | SCC . . . . .                        | 13        |
| 1.5.2    | BCC . . . . .                        | 14        |
| <b>2</b> | <b>Data Structures</b>               | <b>16</b> |
| 2.1      | Basic Structures . . . . .           | 16        |
| 2.1.1    | RMQ . . . . .                        | 16        |
| 2.1.2    | Divide Blocks . . . . .              | 16        |
| 2.2      | Tree Structures . . . . .            | 16        |
| 2.2.1    | Tree Decomposition . . . . .         | 16        |
| 2.2.2    | Link-Cut Tree . . . . .              | 19        |
| 2.3      | Sequence Structures . . . . .        | 20        |
| 2.3.1    | Segment Tree . . . . .               | 20        |
| 2.3.2    | Splay Tree . . . . .                 | 21        |
| 2.4      | Persistent Data Structures . . . . . | 23        |
| 2.4.1    | Chairman Tree . . . . .              | 23        |
| 2.4.2    | Persistent Trie . . . . .            | 24        |
| <b>3</b> | <b>String</b>                        | <b>25</b> |
| 3.1      | Basics . . . . .                     | 25        |
| 3.1.1    | Hash . . . . .                       | 25        |
| 3.1.2    | KMP && exKMP . . . . .               | 25        |
| 3.1.3    | AC Automaton . . . . .               | 26        |
| 3.2      | Suffix Related . . . . .             | 27        |
| 3.2.1    | Suffix Array . . . . .               | 27        |

|          |   |           |
|----------|---|-----------|
| 3.2.2    | Suffix Automaton . . . . .                  | 28        |
| 3.3      | Palindrome Related . . . . .                | 28        |
| 3.3.1    | Manacher . . . . .                          | 28        |
| 3.3.2    | Palindromic Tree . . . . .                  | 28        |
| <b>4</b> | <b>Math</b>                                 | <b>30</b> |
| 4.1      | Algebra . . . . .                           | 30        |
| 4.1.1    | FFT . . . . .                               | 30        |
| 4.1.2    | NTT . . . . .                               | 30        |
| 4.1.3    | Linear Basis . . . . .                      | 31        |
| 4.2      | Math Theory . . . . .                       | 32        |
| 4.2.1    | Inverse . . . . .                           | 32        |
| 4.2.2    | Lucas . . . . .                             | 32        |
| 4.2.3    | CRT && exCRT . . . . .                      | 33        |
| 4.2.4    | BSGS . . . . .                              | 34        |
| 4.2.5    | Miller-Rabin && PollardRho . . . . .        | 34        |
| 4.2.6    | $\Phi(n)$ . . . . .                         | 35        |
| 4.2.7    | Euler Sieve . . . . .                       | 35        |
| 4.2.8    | Möbius Inversion . . . . .                  | 36        |
| <b>5</b> | <b>Geometry</b>                             | <b>38</b> |
| 5.1      | Commonly Definition and Functions . . . . . | 38        |
| 5.1.1    | Const and Functions . . . . .               | 38        |
| 5.1.2    | Point Definition . . . . .                  | 38        |
| 5.1.3    | Line Definition . . . . .                   | 38        |
| 5.1.4    | Get Area . . . . .                          | 39        |
| 5.1.5    | Get Circumference . . . . .                 | 39        |
| 5.2      | Convex Hull . . . . .                       | 39        |
| 5.3      | Half Plane Intersection . . . . .           | 40        |
| 5.4      | Min Circle Cover . . . . .                  | 40        |
| 5.5      | Circle Union Area . . . . .                 | 41        |
| 5.6      | Simpson Integrate . . . . .                 | 43        |
| <b>6</b> | <b>Others</b>                               | <b>44</b> |
| 6.1      | Sample . . . . .                            | 44        |
| 6.1.1    | vimrc . . . . .                             | 44        |
| 6.1.2    | FastIO . . . . .                            | 44        |
| 6.1.3    | Java BigInt . . . . .                       | 45        |
| 6.2      | Offline Algorithm . . . . .                 | 45        |
| 6.2.1    | CDQ Divide and Conquer . . . . .            | 45        |
| 6.2.2    | Mo's Algorithm . . . . .                    | 46        |
| 6.2.3    | Mo's Algorithm On Tree . . . . .            | 47        |
| 6.3      | Randomized Algorithm . . . . .              | 49        |
| 6.3.1    | Simulated Annealing . . . . .               | 49        |

|       |   |    |
|-------|---|----|
| 6.4   | Other Method . . . . .  | 49 |
| 6.4.1 | Enumerate Subset . . . . .  | 49 |
| 6.4.2 | Enumerate $\lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$ . . . . . | 50 |
| 6.5   | Formula . . . . .   | 50 |
| 6.5.1 | Euler's theorem . . . . .   | 50 |
| 6.5.2 | Dirichlet Convolution . . . . .   | 50 |
| 6.5.3 | Möbius Inversion Formula . . . . .  | 50 |

# 1 Graph Theory

## 1.1 Shortest Path

### 1.1.1 Dijkstra

```

1  typedef pair<int, int> P;
2  struct Edge {
3      int to, nxt;
4      LL w;
5  }e[MAXM];
6  int head[MAXN], ecnt;
7  LL d[MAXN];
8  priority_queue<P, vector<P>, greater<P> > q;
9  inline void addEdge(int x, int y, LL w) {
10     e[++ecnt] = (Edge) {y, head[x], w}; head[x] = ecnt;
11 }
12 void dijkstra(int st) {
13     memset(d, 0x3f, sizeof(d));
14     d[st] = 0;
15     q.push(make_pair(0, st));
16     while(!q.empty()) {
17         P x = q.top(); q.pop();
18         int u = x.second;
19         for(int i = head[u], v; i; i = e[i].nxt) {
20             v = e[i].to;
21             if(d[v] > d[u] + e[i].w) {
22                 d[v] = d[u] + e[i].w;
23                 q.push(make_pair(d[v], v));
24             }
25         }
26     }
27 }

```

### 1.1.2 SPFA

```

1  struct Edge {
2      int to, nxt;
3      LL w;
4  }e[MAXE];
5  int head[MAXN], ecnt;
6  LL d[MAXN];
7  bool exist[MAXN];
8  queue<int> q;
9  inline void addEdge(int x, int y, LL w) {
10     e[++ecnt] = (Edge) {y, head[x], w}; head[x] = ecnt;
11 }
12 void SPFA(int st) {
13     memset(d, 0x3f, sizeof(d));
14     d[st] = 0;
15     q.push(st);
16     exist[st] = 1;
17     while(!q.empty()) {
18         int u = q.front(); q.pop();
19         exist[u] = 0;
20         for(int i = head[u], v; i; i = e[i].nxt) {
21             v = e[i].to;
22             if(d[v] > d[u] + e[i].w) {

```

```

23         d[v] = d[u] + e[i].w;
24         //pre[v] = u;
25         if(!exist[v]) {
26             q.push(v);
27             exist[v] = 1;
28         }
29     }
30 }
31 }
32 }

```

## 1.2 Network Flow

### 1.2.1 ISAP

```

1 namespace NWF {
2     struct Edge{
3         int to, nxt; LL f;
4     }e[MAXM << 1];
5     int S, T, tot;
6     int ecnt, head[MAXN], cur[MAXN], pre[MAXN], num[MAXN], dis[MAXN];
7     queue<int> q;
8     void init(int _S, int _T, int _tot){
9         ecnt = 1; S = _S; T = _T; tot = _tot;
10        memset(num, 0, (tot + 1) * sizeof(int));
11        memset(head, 0, (tot + 1) * sizeof(int));
12    }
13    inline void addEdge(int u, int v, LL f) {
14        e[++ecnt] = (Edge) {v, head[u], f}; head[u] = ecnt;
15        e[++ecnt] = (Edge) {u, head[v], 0}; head[v] = ecnt;
16    }
17    void bfs() {
18        memset(dis, 0, (tot + 1) * sizeof(int));
19        q.push(T);
20        dis[T] = 1;
21        while(!q.empty()) {
22            int u = q.front(), v; q.pop();
23            num[dis[u]]++;
24            for(int i = cur[u] = head[u]; i; i = e[i].nxt) {
25                if(!dis[v = e[i].to]) {
26                    dis[v] = dis[u] + 1;
27                    q.push(v);
28                }
29            }
30        }
31    }
32    LL augment() {
33        LL flow = INF;
34        for(int i = S; i != T; i = e[cur[i]].to)
35            flow = min(flow, e[cur[i]].f);
36        for(int i = S; i != T; i = e[cur[i]].to) {
37            e[cur[i]].f -= flow;
38            e[cur[i] ^ 1].f += flow;
39        }
40        return flow;
41    }
42    LL isap() {
43        bfs();
44        int u = S, v;

```

```

45     LL flow = 0;
46     while(dis[S] <= tot) {
47         if(u == T) {
48             flow += augment();
49             u = S;
50         }
51         bool fg = 0;
52         for(int i = cur[u]; i; i = e[i].nxt) {
53             if(e[i].f && dis[u] > dis[v = e[i].to]) {
54                 pre[v] = u;
55                 cur[u] = i;
56                 u = v;
57                 fg = 1;
58                 break;
59             }
60         }
61         if(fg) continue;
62         if(!--num[dis[u]]) break;
63         int maxDis = tot;
64         for(int i = head[u]; i; i = e[i].nxt) {
65             if(e[i].f && maxDis > dis[v = e[i].to]) {
66                 maxDis = dis[v];
67                 cur[u] = i;
68             }
69         }
70         num[dis[u] = maxDis + 1]++;
71         if(u != S) u = pre[u];
72     }
73     return flow;
74 }
75 }

```

### 1.2.2 HLPP

```

1  namespace NWF{
2      struct Edge{
3          int to,nxt;LL f;
4      }e[MAXM << 1];
5      int S, T, tot;
6      int ecnt, head[MAXN], dis[MAXN], num[MAXN];
7      LL sumf[MAXN];
8      queue<int> q;
9      list<int> dep[MAXN];
10     void init(int _S,int _T,int _tot){
11         ecnt = 1;S = _S;T = _T;tot = _tot;
12         memset(num, 0, (tot + 1) * sizeof(int));
13         memset(head, 0, (tot + 1) * sizeof(int));
14         memset(sumf, 0, (tot + 1) * sizeof(LL));
15     }
16     void addEdge(int u,int v,LL f){
17         e[++ecnt] = (Edge) {v, head[u], f};head[u] = ecnt;
18         e[++ecnt] = (Edge) {u, head[v], 0};head[v] = ecnt;
19     }
20     void bfs(){
21         memset(dis, 0, (tot + 1) * sizeof(int));
22         q.push(T); dis[T] = 1;
23         while(!q.empty()){
24             int u=q.front(), v; q.pop();
25             for(int i = head[u]; i; i = e[i].nxt)
26                 if(!dis[v = e[i].to]){

```

```

27         dis[v] = dis[u] + 1;
28         q.push(v);
29     }
30 }
31 }
32 LL hlpp(){
33     bfs();
34     dis[S] = tot + 1;
35     for(int i = 1; i <= tot; ++i) num[dis[i]]++;
36     for(int i = tot + 1; ~i; --i) dep[i].clear();
37     int maxd = dis[S]; LL f;
38     dep[maxd].push_back(S); sumf[S] = INF;
39     for(;;){
40         while(maxd && dep[maxd].empty()) maxd--;
41         if(!maxd) break;
42         int u = dep[maxd].back(), v; dep[maxd].pop_back();
43         int minDis = tot + 1;
44         for(int i = head[u]; i; i = e[i].nxt)
45             if(e[i].f){
46                 if(dis[u] > dis[v = e[i].to]){
47                     f = min(sumf[u], e[i].f);
48                     e[i].f -= f; e[i^1].f += f;
49                     if(sumf[u] != INF) sumf[u] -= f;
50                     if(sumf[v] != INF) sumf[v] += f;
51                     if(v != S && v != T && sumf[v] == f){
52                         maxd = max(maxd, dis[v]);
53                         dep[dis[v]].push_back(v);
54                     }
55                     if(!sumf[u]) break;
56                 } else minDis = min(minDis, dis[v] + 1);
57             }
58         if(sumf[u]){
59             if(!--num[dis[u]]){
60                 for(int i = dis[u]; i <= maxd; ++i){
61                     while(!dep[i].empty()){
62                         --num[i];
63                         dis[dep[i].back()] = tot + 1;
64                         dep[i].pop_back();
65                     }
66                 }
67                 maxd = dis[u] - 1; dis[u] = tot + 1;
68             } else {
69                 dis[u] = minDis;
70                 if(minDis > tot) continue;
71                 num[minDis]++;
72                 maxd = max(maxd, minDis);
73                 dep[minDis].push_back(u);
74             }
75         }
76     }
77     return sumf[T];
78 }
79 }

```

### 1.2.3 Dinic

```

1 namespace NWF {
2     struct Edge {
3         int to, nxt; LL f;
4     } e[MAXM << 1];

```



```

5  int S, T, tot;
6  int ecnt, head[MAXN], cur[MAXN], dis[MAXN];
7  queue<int> q;
8  void init(int _S, int _T, int _tot){
9      ecnt = 1; S = _S; T = _T; tot = _tot;
10     memset(head, 0, (tot + 1) * sizeof(int));
11 }
12 void addEdge(int u, int v, LL f) {
13     e[++ecnt] = (Edge) {v, head[u], f}; head[u] = ecnt;
14     e[++ecnt] = (Edge) {u, head[v], 0}; head[v] = ecnt;
15 }
16 bool bfs() {
17     memset(dis, 0, (tot + 1) * sizeof(int));
18     q.push(S); dis[S] = 1;
19     while (!q.empty()) {
20         int u = q.front(), v; q.pop();
21         for (int i = cur[u] = head[u]; i; i = e[i].nxt) {
22             if (e[i].f && !dis[v = e[i].to]) {
23                 q.push(v);
24                 dis[v] = dis[u] + 1;
25             }
26         }
27     }
28     return dis[T];
29 }
30 LL dfs(int u, LL maxf) {
31     if (u == T) return maxf;
32     LL sumf = maxf;
33     for (int &i = cur[u]; i; i = e[i].nxt) {
34         if (e[i].f && dis[e[i].to] > dis[u]) {
35             LL tmpf = dfs(e[i].to, min(sumf, e[i].f));
36             e[i].f -= tmpf; e[i ^ 1].f += tmpf;
37             sumf -= tmpf;
38             if (!sumf) return maxf;
39         }
40     }
41     return maxf - sumf;
42 }
43 LL dinic() {
44     LL ret = 0;
45     while (bfs()) ret += dfs(S, INF);
46     return ret;
47 }
48 }

```

#### 1.2.4 MCMF

```

1  namespace NWF{
2      struct Edge {
3          int to, nxt; LL f, c;
4      } e[MAXM << 1];
5      int S, T, tot;
6      int ecnt, head[MAXN], cur[MAXN]; LL dis[MAXN];
7      bool exist[MAXN];
8      queue<int> q;
9      void init(int _S, int _T, int _tot){
10         ecnt = 1; S = _S; T = _T; tot = _tot;
11         memset(head, 0, (tot + 1) * sizeof(int));
12     }
13     void addEdge(int u, int v, LL f, LL c) {

```

```

14     e[++ecnt] = (Edge) {v, head[u], f, c}; head[u] = ecnt;
15     e[++ecnt] = (Edge) {u, head[v], 0, -c}; head[v] = ecnt;
16 }
17 bool spfa() {
18     for(int i = 0; i <= tot; ++i){
19         dis[i] = INF; exist[i] = cur[i] = 0;
20     }
21     q.push(S); dis[S] = 0; exist[S] = 1;
22     while(!q.empty()) {
23         int u = q.front(), v; q.pop(); exist[u] = 0;
24         for(int i = head[u]; i; i = e[i].nxt) {
25             if(e[i].f && dis[v = e[i].to] > dis[u] + e[i].c) {
26                 dis[v] = dis[u] + e[i].c;
27                 cur[v] = i;
28                 if(!exist[v]) {
29                     q.push(v);
30                     exist[v] = 1;
31                 }
32             }
33         }
34     }
35     return dis[T] != INF;
36 }
37 LL mcmf() {
38     LL cost = 0;
39     while(spfa()) {
40         LL flow = INF;
41         for(int i = T; i != S; i = e[cur[i] ^ 1].to)
42             flow = min(flow, e[cur[i]].f);
43         for(int i = T; i != S; i = e[cur[i] ^ 1].to) {
44             e[cur[i]].f -= flow;
45             e[cur[i] ^ 1].f += flow;
46         }
47         cost += flow * dis[T];
48     }
49     return cost;
50 }
51 }

```

## 1.3 Tree Related

### 1.3.1 Kruskal

```

1 namespace MST{
2     struct Edge{
3         int u,v; LL w;
4         bool operator < (const Edge& x) const { return w < x.w; }
5     }e[MAXM];
6     int ecnt, fa[MAXN];
7     void addEdge(int u, int v, LL w) {
8         e[++ecnt] = (Edge){v, u, w}; headp[u] = ecnt;
9     }
10    int Find(int x) { return x == fa[x] ? x : fa[x] = Find(fa[x]); }
11    LL kruskal(int n) {
12        sort(e + 1, e + ecnt + 1);
13        for(int i = 1; i <= n; i++) fa[i] = i;
14        LL sum = 0;
15        for (int i = 1; i <= ecnt; i++){
16            int fu = Find(e[i].u), fv = Find(e[i].v);

```

```

17         if(fu != fv){
18             fa[fu] = fv;
19             sum += e[i].w;
20         }
21     }
22     return sum;
23 }
24 }

```

### 1.3.2 Prim

```

1 namespace MST {
2     struct Edge{
3         int to,nxt; LL w;
4     }e[MAXM];
5     int ecnt, head[MAXN], vis[MAXN]; // pre[MAXN];
6     LL dis[MAXN];
7     void addEdge(int u, int v, LL w){
8         e[++ecnt] = (Edge){v, head[u], w}; head[u] = ecnt;
9         e[++ecnt] = (Edge){u, head[v], w}; head[v] = ecnt;
10    }
11    LL Prim(int n){
12        for (int i = 1; i <= n; i++){
13            //pre[i] = 0;
14            vis[i] = 0;
15            dis[i] = INF;
16        }
17        vis[1] = 1;
18        LL sum = 0;
19        for (int i = head[1]; i; i = e[i].nxt)
20            dis[e[i].to] = min(dis[e[i].to],e[i].w);
21        for (int j = 1; j < n; j++){
22            int u; LL minDis = INF;
23            for (int i = 1; i <= n; ++i)
24                if (!vis[i] && dis[i] < minDis){
25                    minDis = dis[i];
26                    u = i;
27                }
28            if (minDis == INF) return -1;
29            vis[u] = 1;
30            sum += minDis;
31            for (int i = head[u], v; i; i = e[i].nxt)
32                if (!vis[v = e[i].to] && e[i].w < dis[v]){
33                    //pre[u] = v;
34                    dis[v] = e[i].w;
35                }
36        }
37        return sum;
38    }
39 }

```

### 1.3.3 Tree Divide and Conquer

```

1 struct Edge {
2     int to, nxt, w;
3 }e[MAXM];
4 int head[MAXN], ecnt;
5 int sz[MAXN];

```

```

6  int d[MAXN], t[5], ans;
7  bool vis[MAXN];
8  inline void add_edge(int u, int v, int w) {
9      e[++ecnt] = (Edge) {v, head[u], w}; head[u] = ecnt;
10     e[++ecnt] = (Edge) {u, head[v], w}; head[v] = ecnt;
11 }
12 int getsz(int x, int fa) {
13     sz[x] = 1;
14     for(int i = head[x]; i; i = e[i].nxt) {
15         int y = e[i].to;
16         if(vis[y] || y == fa) continue;
17         sz[x] += getsz(y, x);
18     }
19     return sz[x];
20 }
21 int getrt(int x) {
22     int tot = getsz(x, 0) >> 1;
23     while(1) {
24         int u = -1;
25         for(int i = head[x]; i; i = e[i].nxt) {
26             int y = e[i].to;
27             if(vis[y] || sz[y] > sz[x]) continue;
28             if(u == -1 || sz[y] > sz[u]) u = y;
29         }
30         if(~u && sz[u] > tot) x = u;
31         else break;
32     }
33     return x;
34 }
35 void getdep(int x, int fa) {
36     t[d[x]]++;
37     for(int i = head[x]; i; i = e[i].nxt) {
38         int y = e[i].to;
39         if(vis[y] || y == fa) continue;
40         d[y] = (d[x] + e[i].w) % 3;
41         getdep(y, x);
42     }
43 }
44 int cal(int x, int v) {
45     t[0] = t[1] = t[2] = 0;
46     d[x] = v % 3;
47     getdep(x, 0);
48     return t[0] * t[0] + t[1] * t[2] * 2;
49 }
50 void solve(int x) {
51     vis[x] = 1;
52     ans += cal(x, 0);
53     for(int i = head[x]; i; i = e[i].nxt) {
54         int y = e[i].to;
55         if(vis[y]) continue;
56         ans -= cal(y, e[i].w);
57         solve(getrt(y));
58     }
59 }
60 int main() {
61     solve(getrt(1));
62 }

```

## 1.4 LCA

### 1.4.1 Tree Decomposition LCA

```

1  int sz[MAXN], dep[MAXN], top[MAXN], fa[MAXN], son[MAXN], num[MAXN], totw;
2  struct Edge {
3      int to, nxt;
4  } e[MAXN << 1];
5  int head[MAXN], ecnt;
6  inline void add_edge(int x, int y) {
7      e[++ecnt] = (Edge) {y, head[x]}; head[x] = ecnt;
8  }
9  void dfs1(int x) {
10     sz[x] = 1; son[x] = 0;
11     for(int i = head[x]; i; i = e[i].nxt) {
12         int v = e[i].to;
13         if(v == fa[x]) continue;
14         fa[v] = x;
15         dep[v] = dep[x] + 1;
16         dfs1(v);
17         sz[x] += sz[v];
18         if(sz[v] > sz[son[x]]) son[x] = v;
19     }
20 }
21 void dfs2(int x) {
22     B[num[x]] = A[x];
23     if(son[x]) {
24         top[son[x]] = top[x];
25         num[son[x]] = ++totw;
26         dfs2(son[x]);
27     }
28     for(int i = head[x]; i; i = e[i].nxt) {
29         int v = e[i].to;
30         if(v == fa[x] || v == son[x]) continue;
31         top[v] = v;
32         num[v] = ++totw;
33         dfs2(v);
34     }
35 }
36 int lca(int u, int v) {
37     if(u == v) return u;
38     while(top[u] != top[v]) {
39         if(dep[top[u]] > dep[top[v]]) swap(u, v);
40         v = fa[top[v]];
41     }
42     if(dep[u] > dep[v]) swap(u, v);
43     return u;
44 }
45 inline void init() {
46     memset(head, 0, sizeof(head)); ecnt = 0;
47     fa[1] = 0; dep[1] = 1; top[1] = 1; num[1] = 1; totw = 1;
48 }
49 inline void pre() {
50     dfs1(1); dfs2(1);
51 }

```

### 1.4.2 Tarjan LCA

```

1  vector< pair<int,int> > G[MAXN], ask[MAXN];

```

```

2  int fa[MAXN], ans[MAXN], vis[MAXN], dis[MAXN];
3  int Find(int x){
4      return x == fa[x] ? x : fa[x] = Find(fa[x]);
5  }
6  void init(int n){
7      memset(ans, 0, sizeof ans);
8      memset(vis, 0, sizeof vis);
9      for(int i = 0; i <= n; i++){
10         G[i].clear();
11         ask[i].clear();
12     }
13 }
14 void LCA(int u){
15     int v;
16     fa[u] = u;
17     vis[u] = true;
18     for(auto it : ask[u])
19         if(vis[v = it.first])
20             ans[it.second] = dis[u] + dis[v] - 2 * dis[Find(it.first)];
21     for(auto it : G[u])
22         if(!vis[v = it.first]){
23             dis[v] = dis[u] + it.second;
24             LCA(v);
25             fa[v] = u;
26         }
27 }

```

## 1.5 Tarjan

### 1.5.1 SCC

```

1  namespace SCC{
2      vector<int> G[MAXN];
3      int dfs_clock, scc_cn, dfn[MAXN], low[MAXN], sccno[MAXN];
4      stack<int> S;
5      //int sum_b; //Bridge
6      //int sum_d; //Dot
7      void addEdge(int u, int v) {
8          G[u].push_back(v);
9      }
10     void tarjan(int u) {
11         dfn[u] = low[u] = ++dfs_clock;
12         S.push(u);
13         for(auto v : G[u]) {
14             if(!dfn[v]) {
15                 tarjan(v);
16                 low[u] = min(low[u], low[v]);
17                 //if(low[v] > dfn[u]) sum_b++;
18                 //if(low[v] >= dfn[u]) vis[u]++;
19             }else if(!sccno[v]) {
20                 low[u] = min(low[u], dfn[v]);
21             }
22         }
23         //if((u == rt && vis[u] > 2) || (u != rt && vis[u] > 1)) sum_d++;
24         if(dfn[u] == low[u]) {
25             scc_cnt++;
26             for(;;) {
27                 int v = S.top(); S.pop();
28                 sccno[v] = scc_cnt;

```

```

29         if(v == u) break;
30     }
31 }
32 }
33 void findSCC(int n) {
34     for(int i = 1; i <= n; i++)
35         if(!dfn[i]) tarjan(i);
36 }
37 void init(int n){
38     dfs_clock = scc_cnt = 0;
39     for(int i = 0; i <= n; ++i){
40         dfn[i] = low[i] = sccno[i] = 0;
41         G[i].clear();
42     }
43 }
44 }

```

### 1.5.2 BCC

```

1 struct Edge {
2     int to, nxt;
3 }e[MAXE];
4 struct Node {
5     int u, v;
6 };
7 int head[MAXN], ecnt;
8 int pre[MAXN], low[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
9 vector<int> bcc[MAXN];
10 stack<Node> S;
11 inline void add_edge(int x, int y) {
12     e[++ecnt] = (Edge) {y, head[x]}; head[x] = ecnt;
13     e[++ecnt] = (Edge) {x, head[y]}; head[y] = ecnt;
14 }
15 inline void init() {
16     memset(pre, 0, sizeof(pre));
17     memset(low, 0, sizeof(low));
18     memset(bccno, 0, sizeof(bccno));
19     memset(iscut, 0, sizeof(iscut));
20     memset(head, 0, sizeof(head)); ecnt = 0;
21     dfs_clock = bcc_cnt = 0;
22 }
23 void tarjan(int u, int fa) {
24     low[u] = pre[u] = ++dfs_clock;
25     int ch = 0;
26     for(int i = head[u]; i; i = e[i].nxt) {
27         int v = e[i].to;
28         if(!pre[v]) {
29             S.push((Node) {u, v});
30             ch++;
31             tarjan(v, u);
32             low[u] = min(low[u], low[v]);
33             if(low[v] >= pre[u]) {
34                 iscut[u] = 1;
35                 bcc[bcc_cnt++].clear();
36                 for(;;) {
37                     Node x = S.top(); S.pop();
38                     if(bccno[x.u] != bcc_cnt) {
39                         bcc[bcc_cnt].push_back(x.u);
40                         bccno[x.u] = bcc_cnt;
41                     }

```

```
42         if(bccno[x.v] != bcc_cnt) {
43             bcc[bcc_cnt].push_back(x.v);
44             bccno[x.v] = bcc_cnt;
45         }
46         if(x.u == u && x.v == v) break;
47     }
48 }
49 }
50 else if(pre[v] < pre[u] && v != fa) {
51     S.push((Node) {u, v});
52     low[u] = min(low[u], pre[v]);
53 }
54 }
55 if(u == fa && ch <= 1) iscut[u] = 0;
56 }
```



## 2 Data Structures

### 2.1 Basic Structures

#### 2.1.1 RMQ

```

1 struct RMQ {
2     int d[MAXN][S + 3];
3     inline void init(int *a, int n) {
4         for(int i = 0; i < n; i++) d[i][0] = a[i];
5         for(int k = 1; (1 << k) < n; k++)
6             for(int i = 0; i + (1 << k) - 1 < n; i++)
7                 d[i][k] = min(d[i][k - 1], d[i + (1 << (k - 1))][k - 1]);
8     }
9     inline int query(int l, int r) {
10        if(l > r) swap(l, r);
11        int k = 0;
12        while((1 << (k + 1)) <= r - l + 1) k++;
13        return min(d[l][k], d[r - (1 << k) + 1][k]);
14    }
15 }rmq;

```

#### 2.1.2 Divide Blocks

```

1 int belong[MAXN], l[MAXN], r[MAXN];
2 int sz, num;
3 void build(int n) {
4     sz = sqrt(n);
5     num = n / sz; if(n % sz) num++;
6     for(int i = 1; i <= num; i++) {
7         l[i] = (i - 1) * sz + 1;
8         r[i] = i * sz;
9     }
10    r[num] = n;
11    for(int i = 1; i <= n; i++) {
12        belong[i] = (i - 1) / sz + 1;
13    }
14 }

```

### 2.2 Tree Structures

#### 2.2.1 Tree Decomposition

```

1 int sz[MAXN], dep[MAXN], top[MAXN], fa[MAXN], son[MAXN], num[MAXN], totw;
2 struct Edge {
3     int to, nxt;
4 }e[MAXN << 1];
5 int head[MAXN], ecnt;
6 int n, m, Q;
7 #define Ls(x) (x << 1)
8 #define Rs(x) (x << 1 | 1)
9 struct Tree {
10    int l, r, lazy;
11    LL sum, mx;
12 }tree[MAXN << 2];
13 int A[MAXN], B[MAXN];

```

```

14 void push_up(int x) {
15     tree[x].sum = tree[Ls(x)].sum + tree[Rs(x)].sum;
16     tree[x].mx = max(tree[Ls(x)].mx, tree[Rs(x)].mx);
17 }
18 void push_down(int x) {
19     if(tree[x].lazy) {
20         tree[Ls(x)].sum += tree[x].lazy * (tree[Ls(x)].r - tree[Ls(x)].l + 1);
21         tree[Rs(x)].sum += tree[x].lazy * (tree[Rs(x)].r - tree[Rs(x)].l + 1);
22         tree[Ls(x)].mx += tree[x].lazy;
23         tree[Rs(x)].mx += tree[x].lazy;
24         tree[Ls(x)].lazy += tree[x].lazy;
25         tree[Rs(x)].lazy += tree[x].lazy;
26         tree[x].lazy = 0;
27     }
28 }
29 void build(int x, int L, int R) {
30     tree[x].lazy = 0;
31     tree[x].l = L; tree[x].r = R;
32     if(L == R) {
33         tree[x].sum = B[L];
34         tree[x].mx = B[L];
35         return;
36     }
37     int mid = (L + R) >> 1;
38     build(Ls(x), L, mid);
39     build(Rs(x), mid + 1, R);
40     push_up(x);
41 }
42 void update(int x, int L, int R, LL val) {
43     if(tree[x].l >= L && tree[x].r <= R) {
44         tree[x].lazy += val;
45         tree[x].sum += val * (tree[x].r - tree[x].l + 1);
46         tree[x].mx += val;
47         return;
48     }
49     push_down(x);
50     int mid = (tree[x].l + tree[x].r) >> 1;
51     if(L <= mid) update(Ls(x), L, R, val);
52     if(R > mid) update(Rs(x), L, R, val);
53     push_up(x);
54 }
55 LL query(int x, int L, int R) {
56     if(tree[x].l >= L && tree[x].r <= R)
57         return tree[x].sum;
58     push_down(x);
59     int mid = (tree[x].l + tree[x].r) >> 1;
60     LL res = 0;
61     if(L <= mid) res += query(Ls(x), L, R);
62     if(R > mid) res += query(Rs(x), L, R);
63     return res;
64 }
65 LL query2(int x, int L, int R) {
66     if(tree[x].l >= L && tree[x].r <= R)
67         return tree[x].mx;
68     push_down(x);
69     int mid = (tree[x].l + tree[x].r) >> 1;
70     LL res = -INF;
71     if(L <= mid) res = max(res, query2(Ls(x), L, R));
72     if(R > mid) res = max(res, query2(Rs(x), L, R));
73     return res;
74 }

```

```

75 inline void add_edge(int x, int y) {
76     e[++ecnt] = (Edge) {y, head[x]}; head[x] = ecnt;
77 }
78 void dfs1(int x) {
79     sz[x] = 1; son[x] = 0;
80     for(int i = head[x]; i; i = e[i].nxt) {
81         int v = e[i].to;
82         if(v == fa[x]) continue;
83         fa[v] = x;
84         dep[v] = dep[x] + 1;
85         dfs1(v);
86         sz[x] += sz[v];
87         if(sz[v] > sz[son[x]]) son[x] = v;
88     }
89 }
90 void dfs2(int x) {
91     B[num[x]] = A[x];
92     if(son[x]) {
93         top[son[x]] = top[x];
94         num[son[x]] = ++totw;
95         dfs2(son[x]);
96     }
97     for(int i = head[x]; i; i = e[i].nxt) {
98         int v = e[i].to;
99         if(v == fa[x] || v == son[x]) continue;
100         top[v] = v;
101         num[v] = ++totw;
102         dfs2(v);
103     }
104 }
105 void up(int a, int b, int c) {
106     int f1 = top[a], f2 = top[b];
107     while(f1 != f2) {
108         if(dep[f1] < dep[f2]) { swap(a, b); swap(f1, f2); }
109         update(1, num[f1], num[a], c);
110         a = fa[f1];
111         f1 = top[a];
112     }
113     if(dep[a] > dep[b]) swap(a, b);
114     update(1, num[a], num[b], c);
115 }
116 int qsum(int a, int b) {
117     if(a == b) return query(1, num[a], num[a]);
118     int f1 = top[a], f2 = top[b];
119     int res = 0;
120     while(f1 != f2) {
121         if(dep[f1] < dep[f2]) { swap(a, b); swap(f1, f2); }
122         res += query(1, num[f1], num[a]);
123         a = fa[f1];
124         f1 = top[a];
125     }
126     if(dep[a] > dep[b]) swap(a, b);
127     res += query(1, num[a], num[b]);
128     return res;
129 }
130 int qmax(int a, int b) {
131     if(a == b) return query2(1, num[a], num[a]);
132     int f1 = top[a], f2 = top[b];
133     int res = -1000000000;
134     while(f1 != f2) {
135         if(dep[f1] < dep[f2]) { swap(a, b); swap(f1, f2); }

```

```

136     res = max(res, query2(1, num[f1], num[a]));
137     a = fa[f1];
138     f1 = top[a];
139 }
140 if(dep[a] > dep[b]) swap(a, b);
141 res = max(res, query2(1, num[a], num[b]));
142 return res;
143 }
144 inline void init() {
145     memset(head, 0, sizeof(head)); ecnt = 0;
146     fa[1] = 0; dep[1] = 1; top[1] = 1; num[1] = 1; totw = 1;
147 }
148 inline void pre() {
149     dfs1(1); dfs2(1); build(1, 1, totw);
150 }

```

### 2.2.2 Link-Cut Tree

```

1 namespace LCT {
2     int fa[MAXN], rev[MAXN], tr[MAXN][2];
3     int s[MAXN], val[MAXN];
4     void push_up(int x) {
5         int l = tr[x][0], r = tr[x][1];
6         s[x] = s[l] + s[r] + val[x];
7     }
8     void Rev(int x) {
9         rev[x] ^= 1; swap(tr[x][0], tr[x][1]);
10    }
11    void push_down(int x) {
12        if(!rev[x]) return;
13        int l = tr[x][0], r = tr[x][1];
14        rev[x] = 0;
15        if(l) Rev(l); if(r) Rev(r);
16    }
17    bool isroot(int x) {
18        return tr[fa[x]][0] != x && tr[fa[x]][1] != x;
19    }
20    void pre(int x) {
21        if(!isroot(x)) pre(fa[x]);
22        push_down(x);
23    }
24    void rotate(int x) {
25        int y = fa[x]; int z = fa[y];
26        int l = tr[y][1] == x;
27        int r = l ^ 1;
28        if(!isroot(y)) tr[z][tr[z][1] == y] = x;
29        fa[x] = z; fa[y] = x; fa[tr[x][r]] = y;
30        tr[y][l] = tr[x][r]; tr[x][r] = y;
31        push_up(y);
32    }
33    void splay(int x) {
34        pre(x);
35        int y, z;
36        while(!isroot(x)) {
37            y = fa[x]; z = fa[y];
38            if(!isroot(y)) {
39                if((tr[z][0] == y) == (tr[y][0] == x)) rotate(y);
40                else rotate(x);
41            }
42            rotate(x);

```

```

43     }
44     push_up(x);
45 }
46 void access(int x) {
47     int y = 0;
48     while(x) {
49         splay(x); tr[x][1] = y;
50         push_up(x);
51         y = x; x = fa[x];
52     }
53 }
54 void makeroot(int x) {
55     access(x); splay(x); Rev(x);
56 }
57 void lnk(int x, int y) {
58     makeroot(x); fa[x] = y;
59 }
60 void cut(int x, int y) {
61     makeroot(x); access(y); splay(y);
62     tr[y][0] = fa[x] = 0; push_up(y);
63 }
64 void update(int x, int y) {
65     makeroot(x); val[x] = y; push_up(x);
66 }
67 int query(int x, int y) {
68     makeroot(x); access(y); splay(y);
69     return s[y];
70 }
71 bool check(int x, int y) {
72     int tmp = y;
73     makeroot(x); access(y); splay(x);
74     while(!isroot(y)) y = fa[y];
75     splay(tmp);
76     return x == y;
77 }
78 }

```

## 2.3 Sequence Structures

### 2.3.1 Segment Tree

```

1  #define Ls(x) (x << 1)
2  #define Rs(x) (x << 1 | 1)
3  struct Tree {
4      int l, r, lazy;
5      LL sum, mx;
6  } tree[MAXN << 2];
7  int A[MAXN];
8  void push_up(int x) {
9      tree[x].sum = tree[Ls(x)].sum + tree[Rs(x)].sum;
10     tree[x].mx = max(tree[Ls(x)].mx, tree[Rs(x)].mx);
11 }
12 void push_down(int x) {
13     if(tree[x].lazy) {
14         tree[Ls(x)].sum += tree[x].lazy * (tree[Ls(x)].r - tree[Ls(x)].l + 1);
15         tree[Rs(x)].sum += tree[x].lazy * (tree[Rs(x)].r - tree[Rs(x)].l + 1);
16         tree[Ls(x)].mx += tree[x].lazy;
17         tree[Rs(x)].mx += tree[x].lazy;
18         tree[Ls(x)].lazy += tree[x].lazy;

```

```

19     tree[Rs(x)].lazy += tree[x].lazy;
20     tree[x].lazy = 0;
21 }
22 }
23 void build(int x, int L, int R) {
24     tree[x].lazy = 0;
25     tree[x].l = L; tree[x].r = R;
26     if(L == R) {
27         tree[x].sum = A[L];
28         tree[x].mx = A[L];
29         return;
30     }
31     int mid = (L + R) >> 1;
32     build(Ls(x), L, mid);
33     build(Rs(x), mid + 1, R);
34     push_up(x);
35 }
36 void update(int x, int L, int R, LL val) {
37     if(tree[x].l >= L && tree[x].r <= R) {
38         tree[x].lazy += val;
39         tree[x].sum += val * (tree[x].r - tree[x].l + 1);
40         tree[x].mx += val;
41         return;
42     }
43     push_down(x);
44     int mid = (tree[x].l + tree[x].r) >> 1;
45     if(L <= mid) update(Ls(x), L, R, val);
46     if(R > mid) update(Rs(x), L, R, val);
47     push_up(x);
48 }
49 LL query(int x, int L, int R) {
50     if(tree[x].l >= L && tree[x].r <= R)
51         return tree[x].sum;
52     push_down(x);
53     int mid = (tree[x].l + tree[x].r) >> 1;
54     LL res = 0;
55     if(L <= mid) res += query(Ls(x), L, R);
56     if(R > mid) res += query(Rs(x), L, R);
57     return res;
58 }
59 LL query2(int x, int L, int R) {
60     if(tree[x].l >= L && tree[x].r <= R)
61         return tree[x].mx;
62     push_down(x);
63     int mid = (tree[x].l + tree[x].r) >> 1;
64     LL res = -INF;
65     if(L <= mid) res = max(res, query2(Ls(x), L, R));
66     if(R > mid) res = max(res, query2(Rs(x), L, R));
67     return res;
68 }

```

### 2.3.2 Splay Tree

```

1 namespace splay{
2     int n, m, sz, rt;
3     int val[MAXN], id[MAXN];
4     int tr[MAXN][2], size[MAXN], fa[MAXN], rev[MAXN], s[MAXN], lazy[MAXN];
5     void push_up(int x) {
6         int l = tr[x][0], r = tr[x][1];
7         s[x] = max(val[x], max(s[l], s[r]));

```

```

8      size[x] = size[l] + size[r] + 1;
9  }
10 void push_down(int x) {
11     int l = tr[x][0], r = tr[x][1];
12     if(lazy[x]) {
13         if(l) {
14             lazy[l] += lazy[x];
15             s[l] += lazy[x];
16             val[l] += lazy[x];
17         }
18         if(r) {
19             lazy[r] += lazy[x];
20             s[r] += lazy[x];
21             val[r] += lazy[x];
22         }
23         lazy[x] = 0;
24     }
25     if(rev[x]) {
26         rev[x] = 0;
27         rev[l] ^= 1; rev[r] ^= 1;
28         swap(tr[x][0], tr[x][1]);
29     }
30 }
31 void rotate(int x, int &k) {
32     int y = fa[x];
33     int z = fa[y];
34     int l, r;
35     if(tr[y][0] == x) l = 0;
36     else l = 1;
37     r = l ^ 1;
38     if(y == k) k = x;
39     else {
40         if(tr[z][0] == y) tr[z][0] = x;
41         else tr[z][1] = x;
42     }
43     fa[x] = z; fa[y] = x; fa[tr[x][r]] = y;
44     tr[y][l] = tr[x][r]; tr[x][r] = y;
45     push_up(y); push_up(x);
46 }
47 void splay(int x, int &k) {
48     int y, z;
49     while(x != k) {
50         y = fa[x];
51         z = fa[y];
52         if(y != k) {
53             if((tr[y][0] == x) ^ (tr[z][0] == y)) rotate(x, k);
54             else rotate(y, k);
55         }
56         rotate(x, k);
57     }
58 }
59 int find(int x, int rank) {
60     push_down(x);
61     int l = tr[x][0], r = tr[x][1];
62     if(size[l] + 1 == rank) return x;
63     else if(size[l] >= rank) return find(l, rank);
64     else return find(r, rank - size[l] - 1);
65 }
66 void update(int l, int r, int v) {
67     int x = find(rt, l), y = find(rt, r + 2);
68     splay(x, rt); splay(y, tr[x][1]);

```

```

69     int z = tr[y][0];
70     lazy[z] += v;
71     val[z] += v;
72     s[z] += v;
73 }
74 void reverse(int l, int r) {
75     int x = find(rt, l), y = find(rt, r + 2);
76     splay(x, rt); splay(y, tr[x][1]);
77     int z = tr[y][0];
78     rev[z] ^= 1;
79 }
80 void query(int l, int r) {
81     int x = find(rt, l), y = find(rt, r + 2);
82     splay(x, rt); splay(y, tr[x][1]);
83     int z = tr[y][0];
84     printf("%d\n", s[z]);
85 }
86 void build(int l, int r, int f) {
87     if(l > r) return;
88     int now = id[l], last = id[f];
89     if(l == r) {
90         fa[now] = last; size[now] = 1;
91         if(l < f) tr[last][0] = now;
92         else tr[last][1] = now;
93         return;
94     }
95     int mid = (l + r) >> 1; now = id[mid];
96     build(l, mid - 1, mid); build(mid + 1, r, mid);
97     fa[now] = last;
98     push_up(now);
99     if(mid < f) tr[last][0] = now;
100    else tr[last][1] = now;
101 }
102 void init() {
103     s[0] = -INF;
104     scanf("%d%d", &n, &m);
105     for(int i = 1; i <= n + 2; i++) id[i] = ++sz;
106     build(1, n + 2, 0); rt = (n + 3) >> 1;
107 }
108 }

```

## 2.4 Persistent Data Structures

### 2.4.1 Chairman Tree

```

1 struct Node {
2     int l, r;
3     LL sum;
4 } t[MAXN * 40];
5 int cnt, n;
6 int rt[MAXN];
7 void update(int pre, int &x, int l, int r, int v) {
8     x = ++cnt; t[x] = t[pre]; t[x].sum++;
9     if(l == r) return;
10    int mid = (l + r) >> 1;
11    if(v <= mid) update(t[pre].l, t[x].l, l, mid, v);
12    else update(t[pre].r, t[x].r, mid + 1, r, v);
13 }
14 int query(int x, int y, int l, int r, int v) {

```



```

15     if(l == r) return l;
16     int mid = (l + r) >> 1;
17     int sum = t[t[y].l].sum - t[t[x].l].sum;
18     if(sum >= v) return query(t[x].l, t[y].l, l, mid, v);
19     else return query(t[x].r, t[y].r, mid + 1, r, v - sum);
20 }

```

### 2.4.2 Persistent Trie

```

1  //区间异或最值查询
2  const int N=5e4+10;
3  int t[N];
4  int ch[N*32][2],val[N*32];
5  int cnt;
6  void init(){
7      mem(ch,0);
8      mem(val,0);
9      cnt=1;
10 }
11 int add(int root,int x){
12     int newroot=cnt++,ret=newroot;
13     for(int i=30;i>=0;i--){
14         ch[newroot][0]=ch[root][0];
15         ch[newroot][1]=ch[root][1];
16         int now=(x>>i)&1;
17         root=ch[root][now];
18         ch[newroot][now]=cnt++;
19         newroot=ch[newroot][now];
20         val[newroot]=val[root]+1;
21     }
22     return ret;
23 }
24 int query(int lt,int rt,int x){
25     int ans=0;
26     for(int i=30;i>=0;i--){
27         int now=(x>>i)&1;
28         if(val[ch[rt][now^1]]-val[ch[lt][now^1]]){
29             ans|=(1<<i);
30             rt=ch[rt][now^1];
31             lt=ch[lt][now^1];
32         } else{
33             rt=ch[rt][now];
34             lt=ch[lt][now];
35         }
36     }
37     return ans;
38 }

```

## 3 String

### 3.1 Basics

#### 3.1.1 Hash

```

1  const LL p1 = 201, p2 = 301, mod1 = 1200000319, mod2 = 2147483647;
2  struct Hash {
3      LL a, b;
4      void append(Hash pre, int v) {
5          a = (pre.a * p1 + v) % mod1;
6          b = (pre.b * p2 + v) % mod2;
7      }
8      void init(string S) {
9          a = b = 0;
10         for(int i = 0; i < S.size(); i++) append(*this, S[i]);
11     }
12     bool operator == (const Hash &x) const {
13         return a == x.a && b == x.b;
14     }
15     bool operator < (const Hash &x) const {
16         return a < x.a || (a == x.a && b < x.b);
17     }
18 };

```

#### 3.1.2 KMP && exKMP

```

1  namespace KMP {
2      int f[MAXN];
3      void get_fail(string A) {
4          f[0] = 0; f[1] = 0;
5          for(int i = 1; i < A.size(); i++) {
6              int j = f[i];
7              while(j && A[i] != A[j]) j = f[j];
8              f[i + 1] = A[i] == A[j] ? j + 1 : 0;
9          }
10     }
11
12     void kmp(string A, string B) {
13         get_fail(B);
14         int j = 0;
15         for(int i = 0; i < A.size(); i++) {
16             while(j && B[j] != A[i]) j = f[j];
17             if(B[j] == A[i]) j++;
18             if(j == B.size()) {
19                 ans++;
20                 j = f[j];
21             }
22         }
23     }
24 }
25 namespace exKMP {
26     int nxt[MAXN], ext[MAXN];
27     void get_nxt(string T) {
28         int j = 0, mx = 0;
29         int m = T.size();
30         nxt[0] = m;
31         for(int i = 1; i < m; i++) {

```

```

32         if(i >= mx || i + nxt[i - j] >= mx) {
33             if(i >= mx) mx = i;
34             while(mx < m && T[mx] == T[mx - i]) mx++;
35             nxt[i] = mx - i;
36             j = i;
37         }
38         else nxt[i] = nxt[i - j];
39     }
40 }
41 void exkmp(string S, string T) {
42     int j = 0, mx = 0;
43     get_nxt(T);
44     int n = S.size(), m = T.size();
45     for(int i = 0; i < n; i++) {
46         if(i >= mx || i + nxt[i - j] >= mx) {
47             if(i >= mx) mx = i;
48             while(mx < n && mx - i < m && S[mx] == T[mx - i]) mx++;
49             ext[i] = mx - i;
50             j = i;
51         }
52         else ext[i] = nxt[i - j];
53     }
54 }
55 }

```

### 3.1.3 AC Automaton

```

1 namespace AC {
2     int ch[MAXN][sigma_size], last[MAXN];
3     int val[MAXN], f[MAXN], sz;
4     inline void init() { sz = 1; memset(ch[0], 0, sizeof(ch[0])); }
5     inline int idx(char c) { return c - 'a'; }
6     void insert(string s, int v) {
7         int u = 0;
8         for(int i = 0; i < s.size(); i++) {
9             int c = idx(s[i]);
10            if(!ch[u][c]) {
11                memset(ch[sz], 0, sizeof(ch[sz]));
12                val[sz] = 0;
13                ch[u][c] = sz++;
14            }
15            u = ch[u][c];
16        }
17        val[u] = v;
18    }
19    void get_fail() {
20        queue<int> q;
21        f[0] = 0;
22        for(int c = 0; c < sigma_size; c++) {
23            int u = ch[0][c];
24            if(u) { f[u] = 0; q.push(u); last[u] = 0; }
25        }
26        while(!q.empty()) {
27            int r = q.front(); q.pop();
28            for(int c = 0; c < sigma_size; c++) {
29                int u = ch[r][c];
30                if(!u) { ch[r][c] = ch[f[r]][c]; continue; }
31                q.push(u);
32                int v = f[r];
33                while(v && !ch[v][c]) v = f[v];

```

```

34         f[u] = ch[v][c];
35         last[u] = val[f[u]] ? f[u] : last[f[u]];
36     }
37 }
38 }
39 inline void solve(int j) {
40     if(j) {
41         ans += val[j];
42         solve(last[j]);
43     }
44 }
45 void find(string T) {
46     int j = 0;
47     for(int i = 0; i < T.size(); i++) {
48         int c = idx(T[i]);
49         j = ch[j][c];
50         if(val[j]) solve(j);
51         else if(last[j]) solve(last[j]);
52     }
53 }
54 }

```

## 3.2 Suffix Related

### 3.2.1 Suffix Array

```

1 namespace SA {
2     char s[MAXN];
3     int sa[MAXN], rank[MAXN], height[MAXN];
4     int t[MAXN], t2[MAXN], c[MAXN], n;
5     void clear() { n = 0; memset(sa, 0, sizeof(sa)); }
6     void build(int m) {
7         int *x = t, *y = t2;
8         for(int i = 0; i < m; i++) c[i] = 0;
9         for(int i = 0; i < n; i++) c[x[i] = s[i]]++;
10        for(int i = 1; i < m; i++) c[i] += c[i - 1];
11        for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
12        for(int k = 1; k <= n; k <<= 1) {
13            int p = 0;
14            for(int i = n - k; i < n; i++) y[p++] = i;
15            for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
16            for(int i = 0; i < m; i++) c[i] = 0;
17            for(int i = 0; i < n; i++) c[x[y[i]]]++;
18            for(int i = 1; i < m; i++) c[i] += c[i - 1];
19            for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
20            swap(x, y);
21            p = 1; x[sa[0]] = 0;
22            for(int i = 1; i < n; i++)
23                x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k]
24                    ? p - 1 : p++;
25            if(p >= n) break;
26            m = p;
27        }
28    }
29    void buildHeight() {
30        int k = 0;
31        for(int i = 0; i < n; i++) rank[sa[i]] = i;
32        for(int i = 0; i < n; i++) {
33            if(k) k--;

```

```

33         int j = sa[rank[i] - 1];
34         while(s[i + k] == s[j + k]) k++;
35         height[rank[i]] = k;
36     }
37 }
38 void init() {
39     n = strlen(s) + 1;
40     build('z' + 1);
41     buildHeight();
42 }
43 }

```

### 3.2.2 Suffix Automaton

## 3.3 Palindrome Related

### 3.3.1 Manacher

```

1 namespace Palindrome {
2     char s1[MAXN], s2[MAXN];
3     int len1, len2, ans;
4     int p[MAXN]; //p[i] - 1
5     void init() {
6         len1 = strlen(s1);
7         s2[0] = '$';
8         s2[1] = '#';
9         for(int i = 0; i < len1; i++) {
10             s2[2 * i + 2] = s1[i];
11             s2[2 * i + 3] = '#';
12         }
13         len2 = len1 * 2 + 2;
14         s2[len2] = '&';
15     }
16     void manacher() {
17         int id = 0, mx = 0;
18         for(int i = 1; i < len2; i++) {
19             if(mx > i) p[i] = min(p[2 * id - i], mx - i);
20             else p[i] = 1;
21             while(s2[i + p[i]] == s2[i - p[i]]) p[i]++;
22             if(i + p[i] > mx) {
23                 mx = i + p[i];
24                 id = i;
25             }
26         }
27     }
28 }

```

### 3.3.2 Palindromic Tree

```

1 struct PalindromicTree {
2     int len[MAXN]; //节点i表示的回文串的长度
3     int fail[MAXN]; //fail指针
4     int cnt[MAXN]; //节点i表示的本质不同的串的个数(调用count())
5     int num[MAXN]; //以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
6     int nxt[MAXN][sigma_size]; //编号为i的节点表示的回文串在两边添加字符c以后变成的回文
    串的编号

```

```

7   int S[MAXN]; //存放添加的字符(S[0]=-1(or '#'))
8   int last, p, n;
9   int newnode(int l) {
10      for(int i = 0; i < sigma_size; i++) nxt[p][i] = 0;
11      cnt[p] = 0;
12      num[p] = 0;
13      len[p] = l;
14      return p++;
15  }
16  inline void init() {
17      p = 0;
18      newnode(0);
19      newnode(-1);
20      S[0] = -1;
21      n = 0;
22      fail[0] = 1;
23  }
24  int getfail(int x) {
25      while(S[n - len[x] - 1] != S[n]) x = fail[x];
26      return x;
27  }
28  void insert(int c) {
29      S[++n] = c;
30      int cur = getfail(last);
31      if(!nxt[cur][c]) {
32          int now = newnode(len[cur] + 2);
33          fail[now] = nxt[getfail(fail[cur])][c];
34          nxt[cur][c] = now;
35          num[now] = num[fail[now]] + 1;
36      }
37      last = nxt[cur][c];
38      cnt[last]++;
39  }
40  void count() {
41      for(int i = p - 1; i >= 0; i--) cnt[fail[i]] += cnt[i];
42  }
43  };

```

## 4 Math

### 4.1 Algebra

#### 4.1.1 FFT

```

1  const double pi = acos(-1.0);
2  const int MAXN = 300003;
3  struct comp {
4      double x, y;
5      comp operator + (const comp a) const { return (comp) {x + a.x, y + a.y}; }
6      comp operator - (const comp a) const { return (comp) {x - a.x, y - a.y}; }
7      comp operator * (const comp a) const { return (comp) {x * a.x - y * a.y, x * a.y + y
      * a.x}; }
8  };
9  int rev[MAXN], T;
10 comp tmp;
11 void fft(comp *a, int r) {
12     if(r == -1) for(int i = 0; i < T; i++) A[i] = A[i] * A[i];
13     for(int i = 0; i < T; i++) if(rev[i] > i) swap(a[rev[i]], a[i]);
14     for(int i = 2, mid = 1; i <= T; mid = i, i <= 1) {
15         comp step = (comp) {cos(pi / mid), r * sin(pi / mid)};
16         for(int j = 0; j < T; j += i) {
17             comp cur = (comp) {1, 0};
18             for(int k = j; k < j + mid; k++, cur = cur * step) {
19                 tmp = a[k + mid] * cur;
20                 a[k + mid] = a[k] - tmp;
21                 a[k] = a[k] + tmp;
22             }
23         }
24     }
25     if(r == -1) for(int i = 0; i < T; i++) a[i].y = (int)(a[i].y / T / 2 + 0.5);
26 }
27 int n, m;
28 comp A[MAXN];
29 void init() {
30     for(T = 1; T <= n + m; T <= 1);
31     for(int i = 1; i < T; i++) {
32         if(i & 1) rev[i] = (rev[i >> 1] >> 1) ^ (T >> 1);
33         else rev[i] = rev[i >> 1] >> 1;
34     }
35 }

```

#### 4.1.2 NTT

```

1  const int MAXN = 300005, G = 3, mod = 998244353; //or (479LL<<21) + 1
2  int rev[MAXN], T;
3  LL qpow(LL x, LL y) {
4      LL res = 1;
5      while(y) {
6          if(y & 1) res = res * x % mod;
7          x = x * x % mod;
8          y >>= 1;
9      }
10     return res;
11 }
12 void ntt(LL *a, int r) {
13     if(r == -1) for(int i = 0; i < T; i++) A[i] = A[i] * B[i] % mod;

```

```

14 for(int i = 0; i < T; i++) if(rev[i] > i) swap(a[rev[i]], a[i]);
15 for(int i = 2, mid = 1; i <= T; mid = i, i <= 1) {
16     LL gn = qpow(G, (mod - 1) / i);
17     if(r == -1) gn = qpow(gn, mod - 2);
18     for(int j = 0; j < T; j += i) {
19         LL cur = 1, tmp;
20         for(int k = j; k < j + mid; k++, cur = cur * gn % mod) {
21             tmp = a[k + mid] * cur % mod;
22             a[k + mid] = ((a[k] - tmp) % mod + mod) % mod;
23             a[k] = (a[k] + tmp) % mod;
24         }
25     }
26 }
27 if(r == -1) {
28     LL inv = qpow(T, mod - 2);
29     for(int i = 0; i < T; i++) a[i] = a[i] * inv % mod;
30 }
31 }
32 int n, m;
33 LL A[MAXN], B[MAXN];
34 void init() {
35     for(T = 1; T <= n + m; T <= 1);
36     for(int i = 0; i < T; i++) {
37         if(i & 1) rev[i] = (rev[i >> 1] >> 1) ^ (T >> 1);
38         else rev[i] = rev[i >> 1] >> 1;
39     }
40 }

```

#### 4.1.3 Linear Basis

```

1 int Gauss(int n, int m) {
2     int num = 1;
3     for(int x = 1; x <= n && x <= m; x++) {
4         int t = 0;
5         for(int j = x; j <= m; j++) if(g[j][x]) { t = j; break; }
6         if(t) {
7             swap(g[x], g[t]);
8             for(int i = x + 1; i <= n; i++) {
9                 if(g[i][x]) {
10                     for(int k = 1; k <= m; k++) g[i][k] ^= g[x][k];
11                 }
12             }
13             num++;
14         }
15     }
16     return --num;
17 }
18 //long long
19 int Gauss() {
20     int num = 1;
21     for(int k = 61; k >= 0; k--) {
22         int t = 0;
23         for(int j = num; j <= cnt; j++) if((A[j] >> k) & 1) { t = j; break; }
24         if(t) {
25             swap(A[t], A[num]);
26             for(int j = num + 1; j <= cnt; j++) if((A[j] >> k) & 1) A[j] ^= A[num];
27             num++;
28         }
29     }
30     return --num;

```



```
31 }
```

## 4.2 Math Theory

### 4.2.1 Inverse

```
1 //O(logn) 求n的逆元
2 const int mod = 1e6 + 3;
3 int exgcd(int a, int b, int &x, int &y) {
4     int d = a;
5     if(b != 0) {
6         d = exgcd(b, a % b, y, x);
7         y -= (a / b) * x;
8     }
9     else {
10        x = 1; y = 0;
11    }
12    return d;
13 }
14 int inverse(int a) {
15     int x, y;
16     exgcd(a, mod, x, y);
17     return (x % mod + mod) % mod;
18 }
19 int inverse(int a) { return qpow(a, mod - 2); }
20 //O(n) 求1~n的逆元
21 int inv[MAXN];
22 void init() {
23     inv[0] = inv[1] = 1;
24     for(int i = 2; i < MAXN; i++) inv[i] = (long long)(mod - mod / i) * inv[mod % i] %
        mod;
25 }
```

### 4.2.2 Lucas

```
1 //mod很小可以预处理逆元的情况
2 void init() {
3     fac[0] = 1;
4     for(int i = 1; i < mod; i++) fac[i] = (long long)fac[i - 1] * i % mod;
5     inv[0] = inv[1] = 1;
6     for(int i = 2; i < mod; i++) inv[i] = (long long)(mod - mod / i) * inv[mod % i] %
        mod;
7     for(int i = 1; i < mod; i++) inv[i] = (long long)inv[i] * inv[i - 1] % mod;
8 }
9 int C(int a, int b) {
10    if(b > a) return 0;
11    if(a < mod) return (long long)fac[a] * inv[b] % mod * inv[a - b] % mod;
12    return (long long)C(a / mod, b / mod) * C(a % mod, b % mod) % mod;
13 }
14 //mod过大不能预处理逆元的情况
15 LL qpow(LL x, LL y) {
16    LL res = 1;
17    while(y) {
18        if(y & 1) res = res * x % mod;
19        x = x * x % mod;
20        y >>= 1;
21    }
22    return res;
23 }
```

```

23 }
24 LL C(LL a, LL b) {
25     if(b > a) return 0;
26     if(b > a - b) b = a - b;
27     LL s1 = 1, s2 = 1;
28     for(LL i = 0; i < b; i++) {
29         s1 = s1 * (a - i) % mod;
30         s2 = s2 * (i + 1) % mod;
31     }
32     return s1 * qpow(s2, mod - 2) % mod;
33 }
34 LL lucas(LL a, LL b) {
35     if(a < mod) return C(a, b);
36     return lucas(a / mod, b / mod) * C(a % mod, b % mod);
37 }

```

#### 4.2.3 CRT && exCRT

```

1 namespace CRT {
2     LL m[MAXN], a[MAXN]; //x_i = a[i] (mod m[i])
3     LL exgcd(LL _a, LL _b, LL &x, LL &y) {
4         if(!_b) {
5             x = 1; y = 0;
6             return _a;
7         }
8         LL d = exgcd(_b, _a % _b, y, x);
9         y -= (_a / _b) * x;
10        return d;
11    }
12    LL crt(int n) {
13        LL M = 1, tmp, res = 0, x, y;
14        for(int i = 1; i <= n; i++) M *= m[i];
15        for(int i = 1; i <= n; i++) {
16            tmp = M / m[i];
17            exgcd(tmp, m[i], x, y);
18            x = (x + m[i]) % m[i];
19            res = (a[i] * x % M * tmp % M + res) % M;
20        }
21        return res;
22    }
23 }
24 namespace EXCRT {
25     LL m[MAXN], a[MAXN];
26     LL exgcd(LL _a, LL _b, LL &x, LL &y) {
27         if(!_b) {
28             x = 1; y = 0;
29             return _a;
30         }
31         LL d = exgcd(_b, _a % _b, y, x);
32         y -= (_a / _b) * x;
33         return d;
34     }
35     LL excrt(int n) {
36         LL M = m[1], A = a[1], x, y, d, tmp;
37         for(int i = 2; i <= n; i++) {
38             d = exgcd(M, m[i], x, y);
39             if((A - a[i]) % d) return -1; //No solution
40             tmp = M / d; M *= m[i] / d;
41             y = (A - a[i]) / d % M * y % M;
42             y = (y + tmp) % tmp;

```

```

43     A = (m[i] % M * y % M + a[i]) % M;
44     A = (A + M) % M;
45 }
46 return A;
47 }
48 }

```

#### 4.2.4 BSGS

```

1  const int MOD = 76543;
2  int hs[MOD + 5], head[MOD + 5], nxt[MOD + 5], id[MOD + 5], ecnt;
3  void insert(int x, int y) {
4      int k = x % MOD;
5      hs[ecnt] = x, id[ecnt] = y, nxt[ecnt] = head[k], head[k] = ecnt++;
6  }
7  int find(int x) {
8      int k = x % MOD;
9      for(int i = head[k]; i; i = nxt[i])
10         if(hs[i] == x)
11             return id[i];
12     return -1;
13 }
14 int BSGS(int a, int b, int c){
15     memset(head, 0, sizeof head); ecnt = 1;
16     if(b == 1) return 0;
17     int m = sqrt(c * 1.0), j;
18     LL x = 1, p = 1;
19     for(int i = 0; i < m; i++, p = p * a % c)
20         insert(p * b % c, i);
21     for(LL i = m; i <= c; i += m){
22         if((j = find(x = x * p % c)) != -1) return i - j;
23         if(i > c) break;
24     }
25     return -1;
26 }

```

#### 4.2.5 Miller-Rabin && PollardRho

```

1  LL ksc(LL a, LL n, LL mod){
2      LL ret=0;
3      for(;n;n>=1){
4          if(n&1){ret+=a;if(ret>=mod)ret-=mod;}
5          a<<=1;if(a>=mod)a-=mod;
6      }
7      return ret;
8  }
9  LL ksm(LL a, LL n, LL mod){
10     LL ret = 1;
11     for(;n;n>=1){
12         if(n&1)ret=ksc(ret,a,mod);
13         a=ksc(a,a,mod);
14     }
15     return ret;
16 }
17 int millerRabin(LL n){
18     if(n<2 || (n!=2 && !(n&1)))return 0;
19     LL d=n-1;for(;!(d&1);d>>=1);
20     for(int i=0;i<20;++i){

```

```

21         LL a=rand()%(n-1)+1;
22         LL t=d,m=ksm(a,d,n);
23         for(;t!=n-1 && m!=1 && m!=n-1;m=ksc(m,m,n),t<=1);
24         if(m!=n-1 && !(t&1)) return 0;
25     }
26     return 1;
27 }
28 LL cnt,fact[100];
29 LL gcd(LL a,LL b){return !b?a:gcd(b,a%b);}
30 LL pollardRho(LL n, int a){
31     LL x=rand()%n,y=x,d=1,k=0,i=1;
32     while(d==1){
33         ++k;
34         x=ksc(x,x,n)+a;if(x>=n)x-=n;
35         d=gcd(x>y?x-y:y-x,n);
36         if(k==i){y=x;i<=1;}
37     }
38     if(d==n)return pollardRho(n,a+1);
39     return d;
40 }
41 void findfac(LL n){
42     if(millerRabin(n)){fact[++cnt]=n;return;}
43     LL p=pollardRho(n,rand()%(n-1)+1);
44     findfac(p);
45     findfac(n/p);
46 }

```

#### 4.2.6 $\Phi(n)$

```

1 int phi(int x) {
2     int res = x;
3     for(int i = 2; i * i <= x; i++) {
4         if(x % i == 0) {
5             res = res / i * (i - 1);
6             while(x % i == 0) x /= i;
7         }
8     }
9     if(x > 1) res = res / x * (x - 1);
10    return res;
11 }

```

#### 4.2.7 Euler Sieve

```

1 int prime[MAXN], cnt, phi[MAXN], mu[MAXN];
2 bool isp[MAXN];
3
4 int min_pow[MAXN]; //最小质因子最高次幂
5 int min_sum[MAXN]; //1+p+p^2+...+p^k
6 int div_sum[MAXN]; //约数和
7
8 int min_index[MAXN]; //最小质因子的指数
9 int div_num[MAXN]; //约数个数
10 void Euler(int n) {
11     mu[1] = phi[1] = div_num[1] = div_sum[1] = 1;
12     for(int i = 2; i <= n; i++) {
13         if(!isp[i]) {
14             prime[++cnt] = min_pow[i] = i;
15             phi[i] = i - 1;

```

```

16     mu[i] = -1;
17     min_index[i] = 1; div_num[i] = 2;
18     div_sum[i] = min_sum[i] = i + 1;
19 }
20 for(int j = 1; j <= cnt && i * prime[j] <= n; j++) {
21     isp[i * prime[j]] = 1;
22     if(i % prime[j] == 0) {
23         phi[i * prime[j]] = phi[i] * prime[j];
24         mu[i * prime[j]] = 0;
25
26         min_index[i * prime[j]] = min_index[i] + 1;
27         div_num[i * prime[j]] = div_num[i] / (min_index[i] + 1) * (min_index[i] *
                prime[j] + 1);
28
29         min_sum[i * prime[j]] = min_sum[i] + min_pow[i] * prime[j];
30         div_sum[i * prime[j]] = div_sum[i] / min_sum[i] * min_sum[i * prime[j]];
31         min_pow[i * prime[j]] = min_pow[i] * prime[j];
32         break;
33     }
34     phi[i * prime[j]] = phi[i] * (prime[j] - 1);
35     mu[i * prime[j]] = -mu[i];
36
37     div_num[i * prime[j]] = div_num[i] << 1;
38     min_index[i * prime[j]] = 1;
39
40     div_sum[i * prime[j]] = div_sum[i] * (prime[j] + 1);
41     min_pow[i * prime[j]] = prime[j];
42     min_sum[i * prime[j]] = prime[j] + 1;
43 }
44 }
45 }

```

#### 4.2.8 Möbius Inversion

$$\sum_i^n \sum_j^m lcm(i, j) (mod\ p)$$

```

1  int mu[MAXN], prime[MAXN], sum[MAXN], cnt;
2  bool isp[MAXN];
3  void getmu(int n) {
4      mu[1] = 1;
5      for(int i = 2; i <= n; i++) {
6          if(!isp[i]) {
7              mu[i] = -1;
8              prime[++cnt] = i;
9          }
10         for(int j = 1; j <= cnt && i * prime[j] <= n; j++) {
11             isp[i * prime[j]] = 1;
12             if(i % prime[j] == 0) {
13                 mu[i * prime[j]] = 0;
14                 break;
15             }
16             mu[i * prime[j]] = -mu[i];
17         }
18     }
19 }
20 ll n, m, ans;
21 ll query(ll x, ll y) { return (x * (x + 1) / 2 % mod) * (y * (y + 1) / 2 % mod) % mod; }

```

```
22 ll F(ll x, ll y) {
23     ll res = 0, last;
24     for(ll i = 1; i <= min(x, y); i = last + 1) {
25         last = min(x / (x / i), y / (y / i));
26         res = (res + (sum[last] - sum[i - 1]) * query(x / i, y / i) % mod) % mod;
27     }
28     return res;
29 }
30 int main() {
31     cin >> n >> m;
32     getmu(min(n, m));
33     for(ll i = 1; i <= min(n, m); i++) sum[i] = (sum[i - 1] + (i * i * mu[i]) % mod) %
34         mod;
35     ll last;
36     for(ll d = 1; d <= min(n, m); d = last + 1) {
37         last = min(n / (n / d), m / (m / d));
38         ans = (ans + (last - d + 1) * (d + last) / 2 % mod * F(n / d, m / d) % mod) %
39             mod;
40     }
41     ans = (ans + mod) % mod;
42     cout << ans << endl;
43     return 0;
44 }
```

## 5 Geometry

### 5.1 Commonly Definition and Functions

#### 5.1.1 Const and Functions

```

1 namespace CG{
2     #define Point Vector
3     const double pi=acos(-1.0);
4     const double inf=1e100;
5     const double eps=1e-9;
6     template <typename T> inline T Abs(T x){return x>0?x:-x;}
7     template <typename T> inline bool operator == (T x,T y){return Abs(x-y)<eps;}
8     int sgn(double x){
9         if (Abs(x)<eps) return 0;
10        if (x>0) return 1;
11        else return -1;
12    }
13 }

```

#### 5.1.2 Point Definition

```

1 namespace CG{
2     struct Point{
3         double x,y;
4         Point(double x=0,double y=0):x(x),y(y){}
5     };
6     Vector operator + (const Vector a,const Vector b){return Vector(a.x+b.x,a.y+b.y);}
7     Vector operator - (const Vector a,const Vector b){return Vector(a.x-b.x,a.y-b.y);}
8     Vector operator * (const Vector a,const double k){return Vector(a.x*k,a.y*k);}
9     Vector operator / (const Vector a,const double k){return Vector(a.x/k,a.y/k);}
10    bool operator < (const Vector a,const Vector b) {return a.x==b.x?a.y<b.y:a.x<b.x;}
11    bool operator == (const Vector a,const Vector b) {return a.x==b.x && a.y==b.y;}
12    double Dot(const Vector a,const Vector b){return a.x*b.x+a.y*b.y;}
13    double Cross(const Vector a,const Vector b){return a.x*b.y-a.y*b.x;}
14    double Norm(const Vector a){return sqrt(Dot(a,a));}
15    double Angle(const Vector a,const Vector b){return acos(Dot(a,b)/Norm(a)/Norm(b));}
16    Vector Rotate(const Vector a,const double theta){return Vector(a.x*cos(theta)-a.y*
17        sin(theta),a.x*sin(theta)+a.y*cos(theta));}
18    boolToLeftTest(const Vector a,const Vector b){return Cross(a,b)<0;}
19    double DisPP(const Vector a,const Vector b){return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)
20        *(a.y-b.y));}
21 }

```

#### 5.1.3 Line Definition

```

1 namespace CG{
2     struct Line{
3         Point p0,v,p1;
4         double t,theta;
5         Line(Point _p0=0,Point _v=0,double _t=1):p0(_p0),v(_v),t(_t){p1=p0+v*t; theta=
6             atan2(v.y,v.x);}
7         // Line(Point _p0=0,Point _v=0,double _t=1):p0(_p0),p1(_v){v=(p1-p0)/t; theta=
8             atan2(v.y,v.x);}
9     };
10    bool operator < (const Line n,const Line m) {return n.theta<m.theta;}

```

```

9   Point GetIntersection(const Line n,const Line m){return n.p0+n.v*Cross(m.v,(n.p0-m.
    p0))/Cross(n.v,m.v);}
10  bool OnLine(const Vector a,const Line l){return Cross(l.p0-a,l.p1-a)==0;}
11  bool OnSegment(const Point a,const Line l){return sgn(Cross(l.p0-a,l.p1-a))==0 &&
    sgn(Dot(l.p0-a,l.p1-a))<0;}
12  double DisPL(const Point a,const Line l){return Abs(Cross(l.p1-l.p0,a-l.p0)/Norm(l.
    p1-l.p0));}
13  double DisPS(const Point a,const Line l){
14      if (l.p0==l.p1) return Norm(a-l.p0);
15      Vector v1=l.p1-l.p0,v2=a-l.p0,v3=a-l.p1;
16      if (sgn(Dot(v1,v2))<0) return Norm(v2);
17      if (sgn(Dot(v1,v3))>0) return Norm(v3);
18      return DisPL(a,l);
19  }
20  Point GetProjection(const Point a,const Line l){
21      Vector v=l.p1-l.p0;
22      return l.p0+v*(Dot(v,a-l.p0)/Dot(v,v));
23  }
24  bool SegmentIntersection(const Line n,const Line m,bool p){
25      double c1=Cross(n.p1-n.p0,m.p1-m.p0);
26      double c2=Cross(n.p1-n.p0,m.p1-n.p0);
27      double c3=Cross(m.p1-m.p0,n.p0-m.p0);
28      double c4=Cross(m.p1-m.p0,n.p1-m.p0);
29      if (p){
30          if (!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)){
31              return OnSegment(n.p0,m) || OnSegment(n.p1,m) || OnSegment(m.p0,n) ||
                OnSegment(m.p0,m);
32          }
33      }
34      return (sgn(c1)*sgn(c2)<0 && sgn(c3)*sgn(c4)<0);
35  }
36  }
37  }

```

#### 5.1.4 Get Area

```

1  namespace CG{
2      double GetArea(Point *p,int n){
3          double area=Cross(p[n],p[1]);
4          for (int i=2;i<=n;i++) area+=0.5*Cross(p[i-1],p[i]);
5          return Abs(area);
6      }
7  }

```

#### 5.1.5 Get Circumference

```

1  namespace CG{
2      double GetCircumference(Point *p,int n){
3          double Circumference=DisPP(p[n],p[1]);
4          for (int i=2;i<=n;i++) Circumference+=DisPP(p[i-1],p[i]);
5          return Circumference;
6      }
7  }

```

### 5.2 Convex Hull



```

1 namespace CG{
2     Point p[MAXN],s[MAXN];
3     int ConvexHull(Point *p,int n){
4         sort(p+1,p+1+n);
5         int m=0;
6         for (int i=1;i<=n;i++){
7             for (;m>=2 && !ToLeftTest(s[m]-s[m-1],p[i]-s[m-1]);m--);
8             s[++m]=p[i];
9         }
10        int k=m;
11        for (int i=n-1;i;i--){
12            for (;m>=k+1 && !ToLeftTest(s[m]-s[m-1],p[i]-s[m-1]);m--);
13            s[++m]=p[i];
14        }
15        return m-1;
16    }
17 }

```

### 5.3 Half Plane Intersection

```

1 namespace CG{
2     void HalfPlaneIntersection(Line l[],int n){
3         deque <Point> p;
4         sort(l+1,l+1+n);
5         deque <Line> q;
6         q.push_back(l[1]);
7         for (int i=2;i<=n;i++){
8             for (;!p.empty() && !ToLeftTest(p.back()-l[i].p0,l[i].v);q.pop_back(),p.
9                 pop_back());
10            for (;!p.empty() && !ToLeftTest(p.front()-l[i].p0,l[i].v);q.pop_front(),p.
11                pop_front());
12            if (sgn(Cross(l[i].v,q.back().v))==0)
13                if (ToLeftTest(l[i].p0-q.back().p0),q.back().v){
14                    q.pop_back();
15                    if (!p.empty()) p.pop_back();
16                }
17            if (!q.empty()) p.push_back(GetIntersection(q.back(),l[i]));
18            q.push_back(l[i]);
19        }
20        for (;!p.empty() && !ToLeftTest(p.back()-q.front().p0,q.front().v);q.pop_back(),
21            p.pop_back());
22        p.push_back(GetIntersection(q.back(),q.front()));
23        double area=0.5*Cross(p.back(),p.front()); Point last=p.front();
24        for (p.pop_front();!p.empty();last=p.front(),p.pop_front()) area+=0.5*Cross(last
25            ,p.front());
26        printf("%.1f",Abs(area));
27    }
28 }

```

### 5.4 Min Circle Cover

```

1 namespace CG{
2     Point GetCircleCenter(const Point a,const Point b,const Point c){
3         Point p=(a+b)/2.0,q=(a+c)/2.0;
4         Vector v=Rotate(b-a,pi/2.0),w=Rotate(c-a,pi/2.0);
5         if (sgn(Norm(Cross(v,w)))==0){
6             if (sgn(Norm(a-b)+Norm(b-c)-Norm(a-c))==0) return (a+c)/2;
7             if (sgn(Norm(b-a)+Norm(a-c)-Norm(b-c))==0) return (b+c)/2;

```

```

8         if (sgn(Norm(a-c)+Norm(c-b)-Norm(a-b))==0) return (a+c)/2;
9     }
10    return GetIntersection(Line(p,v),Line(q,w));
11 }
12 void MinCircleCover(Point p[],int n){
13     random_shuffle(p+1,p+1+n);
14     Point c=p[1];
15     double r=0;
16     for (int i=2;i<=n;i++){
17         if (sgn(Norm(c-p[i])-r)>0){
18             c=p[i],r=0;
19             for (int j=1;j<i;j++){
20                 if (sgn(Norm(c-p[j])-r)>0){
21                     c=(p[i]+p[j])/2.0;
22                     r=Norm(c-p[i]);
23                     for (int k=1;k<j;k++){
24                         if (sgn(Norm(c-p[k])-r)>0){
25                             c=GetCircleCenter(p[i],p[j],p[k]);
26                             r=Norm(c-p[i]);
27                         }
28                     }
29                 }
30             }
31             printf("%.10f\n%.10f %.10f",r,c.x,c.y);
32 }

```

## 5.5 Circle Union Area

```

1 //k次覆盖
2 //圆并去重后s[0]
3 typedef pair<double, int> P;
4 const double pi = acos(-1.0);
5 const int MAXN = 10003;
6 P arc[MAXN << 1];
7 int acnt, cnt;
8 double s[1003];
9 bool del[1003];
10 void add(double st, double en) {
11     if(st < -pi) {
12         add(st + 2 * pi, pi);
13         add(-pi, en);
14         return;
15     }
16     if(en > pi) {
17         add(st, pi);
18         add(-pi, en - 2 * pi);
19         return;
20     }
21     arc[++acnt] = P(st, 1);
22     arc[++acnt] = P(en, -1);
23 }
24 double F(double x) {
25     return (x - sin(x)) / 2;
26 }
27 struct Node {
28     int x, y, r;
29     Node(int _x = 0, int _y = 0, int _r = 0):x(_x), y(_y), r(_r) {}
30     bool operator == (const Node& t) {
31         return x == t.x && y == t.y && r == t.r;

```

```

32     }
33     inline void read() {
34         scanf("%d%d%d", &x, &y, &r);
35     }
36 }a[1003];
37 int main() {
38     int n;
39     scanf("%d", &n);
40     for(int i = 1; i <= n; i++) a[i].read();
41     /*
42     //去重
43     int nn = 0;
44     for(int i = 1; i <= n; i++) {
45         bool same = 0;
46         for(int j = 1; j < i; j++) {
47             if(a[i] == a[j]) {
48                 same = 1; break;
49             }
50         }
51         if(!same) a[++nn] = a[i];
52     }
53     n = nn;
54     //去包含
55     for(int i = 1; i <= n; i++) {
56         for(int j = 1; j <= n; j++) if(i != j) {
57             if(hypot(a[i].x - a[j].x, a[i].y - a[j].y) < (double)(a[i].r - a[j].r)) del[
                    j] = 1;
58         }
59     }
60     nn = 0;
61     for(int i = 1; i <= n; i++) if(!del[i]) {
62         a[++nn] = a[i];
63     }
64     n = nn;
65     */
66     for(int i = 1; i <= n; i++) {
67         acnt = 0;
68         for(int j = 1; j <= n; j++) if(i != j) {
69             int dis = (a[i].x - a[j].x) * (a[i].x - a[j].x) + (a[i].y - a[j].y) * (a[i].
                    y - a[j].y);
70             if(a[j].r > a[i].r && dis <= (a[j].r - a[i].r) * (a[j].r - a[i].r)) add(-pi,
                    pi);
71             else if(dis > (a[i].r - a[j].r) * (a[i].r - a[j].r) && dis < (a[i].r + a[j].
                    r) * (a[i].r + a[j].r)){
72                 double c = sqrt(dis);
73                 double angle = acos((a[i].r * a[i].r + c * c - a[j].r * a[j].r) / (2 * a
                    [i].r * c));
74                 double k = atan2(a[j].y - a[i].y, a[j].x - a[i].x);
75                 add(k - angle, k + angle);
76             }
77         }
78         arc[++acnt] = P(pi, -1);
79         sort(arc + 1, arc + acnt + 1);
80         cnt = 0;
81         double last = -pi;
82         for(int j = 1; j <= acnt; j++) {
83             s[cnt] += F(arc[j].first - last) * a[i].r * a[i].r; //扇形 - 三角形
84             double xa = a[i].x + a[i].r * cos(last);
85             double ya = a[i].y + a[i].r * sin(last);
86             last = arc[j].first;
87             double xb = a[i].x + a[i].r * cos(last);

```

```

88         double yb = a[i].y + a[i].r * sin(last);
89         s[cnt] += (xa * yb - xb * ya) / 2; //到圆心的三角形面积
90         cnt += arc[j].second;
91     }
92 }
93 //printf("%.3f\n", s[0]);
94 for (int i = 0; i < n; i++) {
95     printf("[%d] = %.3f\n", i + 1, s[i] - s[i + 1]);
96 }
97 return 0;
98 }

```

## 5.6 Simpson Integrate

```

1 double Simpson(double l, double r){
2     return (r-l)*(F(l)+4*F((l+r)/2)+F(r))/6;
3 }
4 double Integrate(double l, double r, double S){
5     double mid=(l+r)/2;
6     double A=Simpson(l,mid);
7     double B=Simpson(mid,r);
8     if(A+B-S<eps)return S;
9     return Integrate(l,mid,A)+Integrate(mid,r,B);
10 }

```

## 6 Others

### 6.1 Sample

#### 6.1.1 vimrc

```

1  set nocompatible
2  source $VIMRUNTIME/vimrc_example.vim
3  source $VIMRUNTIME/mswin.vim
4  nunmap <c-v>
5  set cindent
6  set number
7  set mouse=a
8  set tabstop=4
9  set shiftwidth=4
10 set cursorline
11 set guifont=Consolas:h14
12 inoremap kj <esc>
13 inoremap jk <esc>
14 inoremap { {}<left>
15 syntax enable
16 func! Compile()
17     exec "w"
18     exec "! g++ % -o %< -Wall -Wextra -Wshadow -Wconversion --std=c++14 -O2"
19     exec "! ./%<"
20 endfunc
21 func! Debug()
22     exec "w"
23     exec "! g++ - % -o %< -g -Wall --std=c++14 && gdb %<"
24 endfunc
25 func! AddTitle()
26     call append(0,"// Cease to struggle and you cease to live")
27     call append(1,"#include <bits/stdc++.h>")
28     call append(2,"using namespace std;")
29     call append(4,"int main() {"
30     call append(5,"    ios::sync_with_stdio(0); cin.tie(0); cout.precision(6); cout <<
        fixed;")
31     call append(7,"    return 0;")
32     call append(8,"}")
33 endfunc
34 map <F9> :call Compile()<CR>
35 map <F5> :call Debug()<CR>
36 map <F8> :call AddTitle()<CR>

```

#### 6.1.2 FastIO

```

1  namespace IO {
2      const int MB = 1048576;
3      const int RMAX = 16 * MB;
4      const int WMAX = 16 * MB;
5      #define getchar() *(rp++)
6      #define putchar(x) (*(wp++) = (x))
7      char rb[RMAX], *rp = rb, wb[WMAX], *wp = wb;
8      inline void init() {
9          fread(rb, sizeof(char), RMAX, stdin);
10     }
11     template <class _T> inline void read(_T &_a) {
12         _a = 0; register bool _f = 0; register int _c = getchar();

```

```

13     while (_c < '0' || _c > '9') _f != _c == '-', _c = getchar();
14     while (_c >= '0' && _c <= '9') _a = _a * 10 + (_c ^ '0'), _c = getchar();
15     _a = _f ? -_a : _a;
16 }
17 template <class _T> inline void write(_T _a) {
18     static char buf[20], *top = buf;
19     if (_a) {
20         while (_a) {
21             register _T tm = _a / 10;
22             *(++top) = char(_a - tm * 10) | '0';
23             _a = tm;
24         }
25         while (top != buf) putchar(*(top--));
26     }
27     else putchar('0');
28 }
29 void output() {
30     fwrite(wb, sizeof(char), wp - wb, stdout);
31 }
32 }

```

### 6.1.3 Java BigInt

```

1 import java.math.*;
2 import java.util.*;
3 public class Main{
4     public static void main(String []args){
5         Scanner in = new Scanner(System.in);
6         while(in.hasNext()){} //EOF
7         BigInteger zero = BigInteger.valueOf(0);
8         BigInteger a = in.nextBigInteger();
9         BigInteger b = in.nextBigInteger();
10        BigInteger c = in.nextBigInteger();
11        int d = in.nextInt();
12        a.add(b);
13        a.subtract(b);
14        a.multiply(b);
15        a.divide(b);
16        a.mod(b);
17        a.compareTo(b);
18        a.negate();
19        a.modInverse(b); //a-1
20        a.modPow(b,c); //ab%c
21        a.pow(d);
22    }
23 }

```

## 6.2 Offline Algorithm

### 6.2.1 CDQ Divide and Conquer

```

1 struct Node {
2     int x, y, z, ans;
3     Node() {}
4     Node(int _x, int _y, int _z):x(_x), y(_y), z(_z) {}
5     bool operator < (const Node &b) const {
6         if(y == b.y) {
7             if(z == b.z) return x < b.x;

```

```

8         return z < b.z;
9     }
10    return y < b.y;
11 }
12 }A[MAXN], B[MAXN], C[MAXN];
13 int bit[MAXN];
14 void add(int k, int v) {
15     for(; k <= m; k += k & -k) bit[k] = max(bit[k], v);
16 }
17 void clear(int k) {
18     for(; k <= m; k += k & -k) bit[k] = 0;
19 }
20 int sum(int k) {
21     int res = 0;
22     for(; k; k -= k & -k) res = max(res, bit[k]);
23     return res;
24 }
25 void solve(int l, int r) {
26     if(l == r) {
27         B[l] = A[l];
28         return;
29     }
30     int mid = (l + r) >> 1;
31     solve(l, mid);
32     for(int i = mid + 1; i <= r; i++) B[i] = A[i];
33     //sort(B + l, B + mid + 1);
34     sort(B + mid + 1, B + r + 1);
35     int L = l;
36     for(int R = mid + 1; R <= r; R++) {
37         while(L <= mid && B[L].y < B[R].y) add(B[L].z, B[L].ans), L++;
38         A[B[R].x].ans = max(A[B[R].x].ans, sum(B[R].z - 1) + 1);
39         B[R].ans = A[B[R].x].ans;
40     }
41     for(int i = l; i <= L; i++) clear(B[i].z);
42     solve(mid + 1, r);
43     L = l;
44     int p = l, q = mid + 1;
45     while(p <= mid || q <= r) {
46         if(q > r || (p <= mid && B[p].y <= B[q].y)) C[L++] = B[p++];
47         else C[L++] = B[q++];
48     }
49     for(int i = l; i <= r; i++) B[i] = C[i];
50 }

```

### 6.2.2 Mo' s Algorithm

```

1 struct Node{
2     int l, r, t, id;
3     bool operator < (const Node& a) const {
4         if(l / sz == a.l / sz) {
5             if(r == a.r) return t < a.t;
6             return r < a.r;
7         }
8         return l / sz < a.l / sz;
9     }
10 }q[MAXN];
11 void solve() {
12     while (t < q[i].t) addTime(t++, 1);
13     while (t > q[i].t) addTime(--t, -1);
14     while(L < q[i].l) add(L++, -1);

```

```

15 while(L > q[i].l) add(--L, 1);
16 while(R < q[i].r) add(++R, 1);
17 while(R > q[i].r) add(R--, -1);
18 }

```

### 6.2.3 Mo's Algorithm On Tree

```

1 struct Edge {
2     int to, nxt;
3 }e[MAXN << 1];
4 int head[MAXN], ecnt;
5 int stack[MAXN], top, belong[MAXN], cnt, sz;
6 struct Node {
7     int l, r, id, ti;
8     bool operator < (const Node &x) const {
9         return belong[l] < belong[x.l] || (belong[l] == belong[x.l] && belong[r] <
10             belong[x.r]) || (belong[l] == belong[x.l] && belong[r] == belong[x.r] && ti
11                 < x.ti);
12     }
13 }q[MAXN];
14 struct Node2 {
15     int l, r, ti;
16 }qq[MAXN];
17 int n, m, Q, Q0, Q1;
18 int V[MAXN], W[MAXN], C[MAXN];
19 int fa[MAXN][S + 3], dep[MAXN];
20 long long ans[MAXN], tans;
21 int vis[MAXN], cur[MAXN];
22 long long sum[MAXN];
23 int l, r, tm;
24 inline int read() {
25     int x = 0; char ch = getchar(); bool fg = 0;
26     while(ch < '0' || ch > '9') { if(ch == '-') fg = 1; ch = getchar(); }
27     while(ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
28     return fg ? -x : x;
29 }
30 inline void add_edge(int u, int v) {
31     e[++ecnt] = (Edge) {v, head[u]}; head[u] = ecnt;
32     e[++ecnt] = (Edge) {u, head[v]}; head[v] = ecnt;
33 }
34 void dfs(int u, int f) {
35     fa[u][0] = f;
36     dep[u] = dep[f] + 1;
37     int bot = top;
38     for(int i = head[u]; i; i = e[i].nxt) {
39         int v = e[i].to;
40         if(v == f) continue;
41         dfs(v, u);
42         if(top - bot >= sz) {
43             cnt++;
44             while(top != bot) belong[stack[top--]] = cnt;
45         }
46     }
47     stack[++top] = u;
48 }
49 void G(int &u, int step) {
50     for(int i = 0; i < S; i++) if((1 << i) & step) u = fa[u][i];
51 }
52 int lca(int u, int v) {
53     if(dep[u] > dep[v]) swap(u, v);

```



```

52     G(v, dep[v] - dep[u]);
53     if(u == v) return u;
54     for(int i = S; i >= 0; i--) if(fa[u][i] != fa[v][i]) {
55         u = fa[u][i]; v = fa[v][i];
56     }
57     return fa[u][0];
58 }
59 inline void modify(int u) {
60     tans -= V[C[u]] * sum[cur[C[u]]];
61     cur[C[u]] += vis[u];
62     vis[u] = -vis[u];
63     tans += V[C[u]] * sum[cur[C[u]]];
64 }
65 inline void update(int u, int v) {
66     if(u == v) return;
67     if(dep[u] > dep[v]) swap(u, v);
68     while(dep[v] > dep[u]) {
69         modify(v);
70         v = fa[v][0];
71     }
72     while(u != v) {
73         modify(u); modify(v);
74         u = fa[u][0]; v = fa[v][0];
75     }
76 }
77 inline void upd(int t) {
78     if(vis[qq[t].l] == -1) {
79         modify(qq[t].l);
80         swap(C[qq[t].l], qq[t].r);
81         modify(qq[t].l);
82     }
83     else swap(C[qq[t].l], qq[t].r);
84 }
85 inline void moveto(int u, int v) {
86     update(l, u); update(r, v);
87     l = u; r = v;
88 }
89 int main() {
90     n = read(); m = read(); Q = read();
91     sz = (int)pow(n, 2.0 / 3.0);
92     for(int i = 1; i <= m; i++) V[i] = read();
93     for(int i = 1; i <= n; i++) W[i] = read();
94     for(int i = 1, u, v; i < n; i++) {
95         u = read(); v = read();
96         add_edge(u, v);
97     }
98     for(int i = 1; i <= n; i++) {
99         C[i] = read();
100         vis[i] = 1;
101         sum[i] = sum[i - 1] + W[i];
102     }
103     for(int i = 1, tp; i <= Q; i++) {
104         tp = read();
105         if(tp) {
106             ++Q1;
107             q[Q1].l = read(); q[Q1].r = read();
108             q[Q1].id = Q1;
109             q[Q1].ti = i;
110         }
111         else {
112             ++Q0;

```

```

113         qq[Q0].l = read(); qq[Q0].r = read();
114         qq[Q0].ti = i;
115     }
116 }
117 dfs(1, 0);
118 while(top) belong[stack[top--]] = cnt;
119 sort(q + 1, q + Q1 + 1);
120 for(int k = 1; k <= S; k++) {
121     for(int i = 1; i <= n; i++) {
122         fa[i][k] = fa[fa[i][k - 1]][k - 1];
123     }
124 }
125 for(int i = 1; i <= Q1; i++) {
126     if(belong[q[i].l] > belong[q[i].r]) swap(q[i].l, q[i].r);
127     moveto(q[i].l, q[i].r);
128     int lc = lca(l, r);
129     modify(lc);
130     while(qq[tm + 1].ti < q[i].ti && tm < Q0) upd(++tm);
131     while(qq[tm].ti > q[i].ti) upd(tm--);
132     ans[q[i].id] = tans;
133     modify(lc);
134 }
135 for(int i = 1; i <= Q1; i++) printf("%lld\n", ans[i]);
136 return 0;
137 }

```

## 6.3 Randomized Algorithm

### 6.3.1 Simulated Annealing

```

1 void solve() {
2     while(T > eps) {
3         double alpha = ((rand() % 30001) / 15000.0) * pi;
4         double theta = ((rand() % 10001) / 10000.0) * pi;
5         tmp.x = cur.x + T * sin(theta) * cos(alpha);
6         tmp.y = cur.y + T * sin(theta) * sin(alpha);
7         tmp.z = cur.z + T * cos(theta);
8         tmp.dis = cal(tmp);
9         if(tmp.dis < cur.dis || (tmp.dis * 0.999 < cur.dis && (rand() & 7) == 7)) cur = tmp;
10        //if(exp((cur.d - tmp.d) / T) > ((double)rand() / RAND_MAX)) cur = tmp;
11
12        T *= 0.999;
13    }
14 }

```

## 6.4 Other Method

### 6.4.1 Enumerate Subset

```

1 for(int i = 0; i < (1 << k); i++) {
2     for(int j = i; ; --j &= i) {
3         // work();
4         if(j == 0) break;
5     }
6 }

```

### 6.4.2 Enumerate $\lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$

```

1 int cal(int n, int m) {
2     if(n > m) swap(n, m);
3     int res = 0, last;
4     for(int i = 1; i <= n; i = last + 1) {
5         last = min(n / (n / i), m / (m / i));
6         res += (n / i) * (m / i) * (sum(last) - sum(i - 1));
7     }
8     return res;
9 }

```

## 6.5 Formula

### 6.5.1 Euler's theorem

$$a^x \equiv \begin{cases} a^{b \% \phi(p)} & \gcd(a, p) = 1 \\ a^b & \gcd(a, p) \neq 1, b < \phi(p) \\ a^{b \% \phi(p) + \phi(p)} & \gcd(a, p) \neq 1, b \geq \phi(p) \end{cases} \pmod{p}$$

### 6.5.2 Dirichlet Convolution

$$(f \times g)(N) = \sum_{d|N} f(d) * g\left(\frac{N}{d}\right)$$

$$\sum_{d|N} \phi(d) * \frac{N}{d} = \phi \times id$$

$$n = \sum_{d|n} \phi(d)$$

$$e(n) = \sum_{d|n} \mu(d)$$

$$id = \phi \times 1$$

$$e = \mu \times 1$$

### 6.5.3 Möbius Inversion Formula

$$id = \phi \times 1$$

$$\phi = id \times \mu$$

$$f = g \times 1$$

$$g = \mu \times f$$