

SOUTH CHINA UNIVERSITY OF TECHNOLOGY

SCUT\_GUGUGU

# TEMPLATE



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

Last build at July 19, 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
1.6	Cactus . . . . .	15
1.6.1	Circle-Square Tree . . . . .	15
<b>2</b>	<b>Data Structures</b>	<b>19</b>
2.1	Basic Structures . . . . .	19
2.1.1	RMQ . . . . .	19
2.1.2	Divide Blocks . . . . .	19
2.2	Tree Structures . . . . .	19
2.2.1	Tree Decomposition . . . . .	19
2.2.2	Link-Cut Tree . . . . .	22
2.3	Sequence Structures . . . . .	23
2.3.1	Segment Tree . . . . .	23
2.3.2	Splay Tree . . . . .	24
2.4	Persistent Data Structures . . . . .	26
2.4.1	Chairman Tree . . . . .	26
2.4.2	Persistent Trie . . . . .	27
<b>3</b>	<b>String</b>	<b>28</b>
3.1	Basics . . . . .	28
3.1.1	Hash . . . . .	28
3.1.2	KMP && exKMP . . . . .	28
3.1.3	AC Automaton . . . . .	29

3.1.4	Minimum String	31
3.2	Suffix Related	31
3.2.1	Suffix Array	31
3.2.2	Suffix Automaton	32
3.3	Palindrome Related	34
3.3.1	Manacher	34
3.3.2	Palindromic Automaton	35
<b>4</b>	<b>Math</b>	<b>36</b>
4.1	Algebra	36
4.1.1	FFT	36
4.1.2	NTT	36
4.1.3	FWT	37
4.1.4	Linear Basis	38
4.2	Math Theory	38
4.2.1	Inverse	38
4.2.2	Lucas	39
4.2.3	CRT && exCRT	40
4.2.4	BSGS	41
4.2.5	Miller-Rabin && PollardRho	41
4.2.6	$\Phi(n)$	42
4.2.7	Euler Sieve	42
4.2.8	DuJiao Sieve	43
4.2.9	Möbius Inversion	44
<b>5</b>	<b>Geometry</b>	<b>46</b>
5.1	Commonly Definition and Functions	46
5.1.1	Const and Functions	46
5.1.2	Point Definition	46
5.1.3	Line Definition	46
5.1.4	Get Area	47
5.1.5	Get Circumference	47
5.1.6	Anticlockwise Sort	48
5.2	Convex Hull	48
5.2.1	Get Convex Hull	48
5.2.2	Point in Convex Hull	48
5.3	Minkowski Sum	49
5.4	Rotating Calipers	49
5.4.1	The Diameter of Convex Hull	49
5.4.2	The Min Distance Between two Convex Hull	49
5.5	Half Plane Intersection	50
5.6	Min Circle Cover	50
5.7	Circle Union Area	51
5.8	Simpson Integrate	53

<b>6</b>	<b>Others</b>	<b>54</b>
6.1	Sample . . . . .	54
6.1.1	vimrc . . . . .	54
6.1.2	Check . . . . .	54
6.1.3	FastIO . . . . .	55
6.1.4	Java BigInt . . . . .	55
6.2	Offline Algorithm . . . . .	57
6.2.1	CDQ Divide and Conquer . . . . .	57
6.2.2	Mo's Algorithm . . . . .	58
6.2.3	Mo's Algorithm On Tree . . . . .	58
6.3	Randomized Algorithm . . . . .	61
6.3.1	Simulated Annealing . . . . .	61
6.4	Other Method . . . . .	61
6.4.1	Enumerate Subset . . . . .	61
6.4.2	Enumerate $\lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$ . . . . .	61
6.5	Formula . . . . .	62
6.5.1	Euler's Theorem . . . . .	62
6.5.2	Möbius Inversion Formula . . . . .	62
6.5.3	Math Theory Tips . . . . .	62

# 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      void addEdge(int u, int v) {
6          G[u].push_back(v);
7      }
8      void tarjan(int u) {
9          dfn[u] = low[u] = ++dfs_clock;
10         S.push(u);
11         for(auto v : G[u]) {
12             if(!dfn[v]) {
13                 tarjan(v);
14                 low[u] = min(low[u], low[v]);
15             } else if(!sccno[v]) {
16                 low[u] = min(low[u], dfn[v]);
17             }
18         }
19         if(dfn[u] == low[u]) {
20             scc_cnt++;
21             for(;;) {
22                 int v = S.top(); S.pop();
23                 sccno[v] = scc_cnt;
24                 if(v == u) break;
25             }
26         }
27     }
28     void findSCC(int n) {

```

```

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

```

### 1.5.2 BCC

```

1 namespace BCC{
2     struct Edge {
3         int to, nxt;
4     }e[MAXM << 1];
5     int ecnt, head[MAXN];
6     int dfs_clock, dfn[MAXN], low[MAXN];
7
8     int is_vertex[MAXN], vbcc_cnt, vbccno[MAXN];
9     vector<int> vbcc[MAXN];
10    stack<int> vS;
11
12    int ebcc_cnt, ebccno[MAXN];
13    stack<int> eS;
14
15    inline void addEdge(int u, int v) {
16        e[++ecnt] = (Edge) {v, head[u]}; head[u] = ecnt;
17        e[++ecnt] = (Edge) {u, head[v]}; head[v] = ecnt;
18    }
19    inline void init(int n) {
20        ecnt = 1;
21        dfs_clock = 0;
22        vbcc_cnt = 0;
23        ebcc_cnt = 0;
24        for(int i = 1; i <= n; ++i){
25            head[i] = dfn[i] = low[i] = 0;
26            is_vertex[i] = 0;
27            vbccno[i] = 0;
28            ebccno[i] = 0;
29        }
30        while(!vS.empty()) vS.pop();
31    }
32    //root's edge = -1;
33    void tarjan(int u, int edge) {
34        dfn[u] = low[u] = ++dfs_clock;
35        int ch = 0;
36        vS.push(u);
37        eS.push(u);
38        for(int i = head[u], v; i; i = e[i].nxt) {
39            if(!dfn[v = e[i].to]) {
40                tarjan(v, i ^ 1);
41                low[u] = min(low[u], low[v]);
42                if(low[v] >= dfn[u]) {
43                    ++ch;
44                    if(edge > 0 || ch > 1) is_vertex[u] = 1;
45                    vbcc[++vbcc_cnt].clear();
46                    vbcc[vbcc_cnt].push_back(u);

```

```

47         for(int x;;){
48             x = vS.top();vS.pop();
49             vbcc[vbcc_cnt].push_back(x);
50             vbccno[x] = vbcc_cnt;
51             if(x == v)break;
52         }
53     }
54     if(low[v] > dfn[u]) {
55         // i && i ^ 1 is bridge
56     }
57 }
58 else if(dfn[v] < dfn[u] && i != edge)
59     low[u] = min(low[u], dfn[v]);
60 }
61 if(dfn[u] == low[u]) {
62     ebcc_cnt++;
63     for(int v;;) {
64         v = eS.top(); eS.pop();
65         ebccno[v] = ebcc_cnt;
66         if(v == u) break;
67     }
68 }
69 }
70 void findBCC(int n){
71     for(int i = 1; i <= n; i++)
72         if(!dfn[i]) tarjan(i, -1);
73 }
74 //findBridge
75 for(int u = 1; u <= n; u++) {
76     for(int i = head[u], v; i; i = e[i].nxt)
77         if(ebccno[u] != ebccno[v = e[i].to]) {
78             //is bridge
79         }
80 }
81 }
82 }

```

## 1.6 Cactus

### 1.6.1 Circle-Square Tree

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef pair<int, int> P;
4  const int MAXN = 2e4 + 5;
5  const int S = 15;
6  namespace Tree {
7      struct Edge {
8          int to, nxt, w;
9      }e[MAXN << 1];
10     int ecnt, head[MAXN];
11     int rt, isrt[MAXN], fa[MAXN][S + 3];
12     int sz[MAXN];
13     inline void addEdge(int u, int v, int w) {
14         e[++ecnt] = (Edge) {v, head[u], w}; head[u] = ecnt;
15         fa[v][0] = u;
16     }
17 }
18 int n, m, Q;

```



```

19 namespace BCC {
20     struct Edge {
21         int to, nxt, w;
22     }e[MAXN << 1];
23     int ecnt, head[MAXN];
24     int dfs_clock, dfn[MAXN], low[MAXN];
25     int is_vertex[MAXN], vbcc_cnt, vbccno[MAXN];
26     vector<P> vbcc[MAXN];
27     stack<P> vs;
28     int tag[MAXN];
29     inline void addEdge(int u, int v, int w) {
30         e[++ecnt] = (Edge) {v, head[u], w}; head[u] = ecnt;
31         e[++ecnt] = (Edge) {u, head[v], w}; head[v] = ecnt;
32     }
33     inline void init(int n) {
34         ecnt = 1;
35         dfs_clock = 0;
36         vbcc_cnt = 0;
37         for(int i = 0; i <= 2 * n; i++){
38             head[i] = dfn[i] = low[i] = 0;
39             vbccno[i] = 0;
40             tag[i] = 0;
41         }
42         while(!vs.empty()) vs.pop();
43     }
44     //root's edge = -1;
45     void tarjan(int u, int edge) {
46         dfn[u] = low[u] = ++dfs_clock;
47         vs.push(P(u, e[edge ^ 1].w));
48         for(int i = head[u], v; i; i = e[i].nxt) {
49             if(!dfn[v = e[i].to]) {
50                 tarjan(v, i ^ 1);
51                 low[u] = min(low[u], low[v]);
52                 if(low[v] >= dfn[u]) {
53                     if(vs.top().first == v) {
54                         Tree::addEdge(u, v, vs.top().second);
55                         vs.pop();
56                         continue;
57                     }
58                     vbcc[++vbcc_cnt].clear();
59                     vbcc[vbcc_cnt].push_back(P(u, 0));
60                     Tree::isrt[u] = 1;
61                     int &sz = Tree::sz[n + vbcc_cnt];
62                     tag[vs.top().first] = n + vbcc_cnt;
63                     //Tree::addEdge(u, rt, 0);
64                     for(P x;;) {
65                         x = vs.top(); vs.pop();
66                         sz += x.second;
67                         //Tree::addEdge(rt, x.first, sz);
68                         vbcc[vbcc_cnt].push_back(x);
69                         vbccno[x.first] = vbcc_cnt;
70                         if(x.first == v) break;
71                     }
72                 }
73             }
74             else if(dfn[v] < dfn[u] && i != edge)
75                 low[u] = min(low[u], dfn[v]);
76         }
77         for(int i = head[u], v; i; i = e[i].nxt) {
78             if(tag[v = e[i].to]) {
79                 int r = tag[v]; Tree::sz[r] += e[i].w;

```

```

80         tag[v] = 0;
81     }
82 }
83 }
84 void findBCC(int n) {
85     for(int i = 1; i <= n; i++)
86         if(!dfn[i]) tarjan(i, -1);
87 }
88 }
89 namespace Tree {
90     int dis[MAXN], dep[MAXN], len[MAXN];
91     inline void init(int n) {
92         BCC::init(n);
93         rt = n;
94         ecnt = 1;
95         for(int i = 0; i <= 2 * n; i++) {
96             head[i] = 0;
97             fa[i][0] = isrt[i] = dis[i] = dep[i] = len[i] = 0;
98         }
99     }
100     void dfs(int x) {
101         for(int i = head[x], y; i; i = e[i].nxt) {
102             if(!dep[y = e[i].to]) {
103                 dep[y] = dep[x] + 1;
104                 dis[y] = dis[x] + e[i].w;
105                 dfs(y);
106             }
107         }
108     }
109     void pre() {
110         for(int k = 1; k <= BCC::vbcc_cnt; k++) {
111             rt++;
112             vector<P> &E = BCC::vbcc[k];
113             addEdge(E[0].first, rt, 0);
114             int cnt = 0;
115             for(int i = E.size() - 1; i >= 1; i--) {
116                 cnt += E[i].second;
117                 len[E[i].first] = cnt;
118                 addEdge(rt, E[i].first, min(cnt, sz[rt] - cnt));
119             }
120         }
121         for(int k = 1; k <= S; k++) {
122             for(int i = 1; i <= rt; i++) {
123                 fa[i][k] = fa[fa[i][k - 1]][k - 1];
124             }
125         }
126         dep[1] = 1;
127         dfs(1);
128     }
129     int up(int x, int d) {
130         for(int i = S; i >= 0; i--) {
131             if(dep[fa[x][i]] >= d) x = fa[x][i];
132         }
133         return x;
134     }
135     int lca(int u, int v) {
136         if(dep[u] > dep[v]) swap(u, v);
137         v = up(v, dep[u]);
138         if(u == v) return u;
139         for(int i = S; i >= 0; i--) {
140             if(fa[u][i] != fa[v][i]) {

```

```

141         u = fa[u][i], v = fa[v][i];
142     }
143 }
144 return fa[u][0];
145 }
146 int query(int u, int v) {
147     int l = lca(u, v);
148     if(l <= n) return dis[u] + dis[v] - 2 * dis[l];
149     int x = up(u, dep[l] + 1), y = up(v, dep[l] + 1);
150     int res = dis[u] - dis[x] + dis[v] - dis[y];
151     int tmp = abs(len[x] - len[y]);
152     return res + min(tmp, sz[l] - tmp);
153 }
154 }
155
156 int main() {
157     ios::sync_with_stdio(0); cin.tie(0); cout.precision(6); cout << fixed;
158     using namespace Tree;
159     cin >> n >> m >> Q;
160     init(n);
161     for(int i = 1, u, v, w; i <= m; i++) {
162         cin >> u >> v >> w;
163         BCC::addEdge(u, v, w);
164     }
165     BCC::findBCC(n);
166     pre();
167     int u, v;
168     while(Q--) {
169         cin >> u >> v;
170         cout << query(u, v) << endl;
171     }
172     return 0;
173 }

```

## 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 fa[MAXN];
3      void get_fail(char* t, int tn) {
4          fa[0] = -1;
5          int i = 0, j = -1;
6          while(i < tn) {
7              if (j == -1 || t[i] == t[j]) {
8                  ++i; ++j;
9                  fa[i] = t[i] != t[j] ? j : fa[j];
10             }else{
11                 j = fa[j];
12             }
13         }
14     }
15     void kmp(char* s, int sn, char* t, int tn) {
16         int i = 0, j = 0;
17         while(i < sn) {
18             if (j == -1 || s[i] == t[j]) {
19                 i++; j++;
20                 if(j == tn) {
21                     }
22                 }else j = fa[j];
23             }
24         }
25     }
26 namespace exKMP {
27     int nxt[MAXN], ext[MAXN];
28     void get_nxt(char* t, int tn) {
29         int j = 0, mx = 0;
30         nxt[0] = tn;
31         for(int i = 1; i < tn; i++) {

```

```

32         if(i >= mx || i + nxt[i - j] >= mx) {
33             if(i > mx) mx = i;
34             while(mx < tn && t[mx] == t[mx - i]) mx++;
35             nxt[i] = mx - i;
36             j = i;
37         }else nxt[i] = nxt[i - j];
38     }
39 }
40 void exkmp(char *s, int sn, char *t, int tn) {
41     int j = 0, mx = 0;
42     for(int i = 0; i < sn; i++) {
43         if(i >= mx || i + nxt[i - j] >= mx) {
44             if(i > mx) mx = i;
45             while(mx < sn && mx - i < tn && s[mx] == t[mx - i]) mx++;
46             ext[i] = mx - i;
47             j = i;
48         }else ext[i] = nxt[i - j];
49     }
50 }
51 }

```

### 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 }
55 namespace AC {
56     int root, tcnt;
57     int ch[MAXN][sigma_size], fa[MAXN];
58     inline int newnode() {
59         fa[++tcnt] = 0;
60         for(int i = 0; i < sigma_size; ++i) ch[tcnt][i] = 0;
61         return tcnt;
62     }
63     inline void init() {
64         tcnt = -1;
65         root = newnode();
66     }
67     inline int idx(char c) { return c - 'a'; }
68     void extend(char *s, int sn) {
69         int cur = root;
70         for(int i = 0, c; i < sn; i++) {
71             if(!ch[cur][c = idx(s[i])])
72                 ch[cur][c] = newnode();
73             cur = ch[cur][c];
74         }
75     }
76     int q[MAXN], qh, qt;
77     void get_fail() {
78         qh = 1; qt = 0;
79         fa[root] = 0;
80         for(int c = 0, now; c < sigma_size; c++)
81             if((now = ch[root][c]) != 0)
82                 q[++qt] = now;
83         while(qh <= qt) {
84             int cur = q[qh++];
85             for(int c = 0, now; c < sigma_size; c++)
86                 if((now = ch[cur][c]) != 0) {
87                     fa[now] = ch[fa[cur]][c];
88                     q[++qt] = now;
89                 }else
90                     ch[cur][c] = ch[fa[cur]][c];
91         }
92     }
93     //统计模板串出现次数，每个模板串只计算一次
94     //     int cur = root, ans = 0;
95     //     for(int i = 0; i < sn; ++i) {
96     //         cur = ch[cur][idx(s[i])];
97     //         for(int j = cur; j && cnt[j] != -1; j = fa[j]) {
98     //             ans += cnt[j];

```

```

99 //          cnt[j] = -1;
100 //      }
101 //  }
102
103 }

```

### 3.1.4 Minimum String

```

1 namespace minstring{
2     int getmin(char *s, int sn) {
3         int i = 0, j = 1, k = 0, t;
4         while(i < sn && j < sn && k < sn) {
5             t = s[(i + k) % sn] - s[(j + k) % sn];
6             if(!t) k++;
7             else {
8                 if(t > 0) i += k + 1; else j += k + 1;
9                 if(i == j) j++;
10                k = 0;
11            }
12        }
13        return i < j ? i : j;
14    }
15 }

```

## 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        void buildHeight() {
29            int k = 0;

```



```

30     for(int i = 0; i < n; i++) rank[sa[i]] = i;
31     for(int i = 0; i < n; i++) {
32         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

```

1 namespace SAM{
2     int scnt, root, last;
3     int fa[MAXN<<1], len[MAXN<<1], ch[MAXN<<1][26];
4     int sc[MAXN<<1], tmp1[MAXN<<1], minl[MAXN<<1];
5
6     int newnode(int _len, int q = 0) {
7         fa[++scnt] = fa[q]; len[scnt] = _len;
8         sc[scnt] = 0; tmp1[scnt] = 0; minl[scnt] = INF;
9         for(int i = 0; i < 26; i++) ch[scnt][i] = ch[q][i];
10        return scnt;
11    }
12    void init() {
13        scnt = 0;
14        root = last = newnode(0);
15    }
16    void extend(int c) {
17        int p = last, np = newnode(len[p] + 1);
18        for(; p && ch[p][c] == 0; p = fa[p]) ch[p][c] = np;
19        if(!p) fa[np] = root;
20        else{
21            int q = ch[p][c];
22            if(len[p] + 1 == len[q]) fa[np] = q;
23            else{
24                int nq = newnode(len[p] + 1, q);
25                fa[np] = fa[q] = nq;
26                for(; p && ch[p][c] == q; p = fa[p]) ch[p][c] = nq;
27            }
28        }
29        last = np;
30    }
31    int c[MAXN], rs[MAXN << 1];
32    void radix_sort(int n){
33        for(int i = 0; i <= n; i++) c[i] = 0;
34        for(int i = 1; i <= scnt; i++) c[len[i]]++;
35        for(int i = 1; i <= n; i++) c[i] += c[i-1];
36        for(int i = scnt; i >= 1; i--) rs[c[len[i]]--] = i;
37    }
38    void go(){
39        scanf("%s", s);
40        int n = strlen(s);
41        for(int i = 0; i < n; ++i)
42            extend(s[i] - 'a');
43        radix_sort(n);

```

```

44 //以下sc集合意义不同
45 { //每个节点对应的位置之后有多少个不同子串
46     for(int i = scnt; i >= 1; i--) {
47         int S = 0;
48         for(int j = 0; j < 26; j++)
49             S += sc[ ch[rs[i]][j] ];
50         sc[rs[i]] = S + 1;
51     }
52 }
53 { //right集合大小
54     int cur = root;
55     for(int i = 0; i < n; ++i) {
56         cur = ch[cur][s[i] - 'a'];
57         sc[cur]++;
58     }
59     for(int i = scnt; i >= 1; --i) {
60         sc[ fa[rs[i]] ] += sc[rs[i]];
61     }
62 }
63 //公共子串
64 //tpl, 当前字符串: 在状态cur, 与模板串的最长公共后缀
65 //minl, 多个字符串: 在状态cur, 与模板串的最长公共后缀
66 //注意: 在状态cur匹配成功时, cur的祖先状态与字符串的最长公共后缀
67 for(; ~scanf("%s", s);) {
68     int cur = root, Blen = 0;
69     for(int i = 0; i <= scnt; i++)
70         tpl[i] = 0;
71     n = strlen(s);
72     for(int i = 0, x; i < n; i++) {
73         x = s[i] - 'a';
74         if(ch[cur][x]) {
75             ++Blen;
76             cur = ch[cur][x];
77         } else {
78             for(; cur && ch[cur][x] == 0; cur = fa[cur]);
79             if(cur) {
80                 Blen = len[cur] + 1;
81                 cur = ch[cur][x];
82             } else {
83                 cur = root; Blen = 0;
84             }
85         }
86         tpl[cur] = max(tpl[cur], Blen);
87     }
88     for(int i = scnt; i > 0; --i) {
89         if( tpl[ fa[rs[i]] ] < tpl[ rs[i] ] )
90             tpl[ fa[rs[i]] ] = len[ fa[rs[i]] ];
91         minl[ rs[i] ] = min(minl[ rs[i] ], tpl[ rs[i] ]);
92     }
93 }
94 }
95 }
96 namespace exSAM{
97     int scnt, root;
98     int fa[MAXN<<1], len[MAXN<<1], ch[MAXN<<1][26];
99     int sc[MAXN<<1], tpl[MAXN<<1], minl[MAXN<<1];
100
101     int newnode(int _len, int q = 0) {
102         fa[++scnt] = fa[q]; len[scnt] = _len;
103         sc[scnt] = 0; tpl[scnt] = 0; minl[scnt] = INF;
104         for(int i = 0; i < 26; i++) ch[scnt][i] = ch[q][i];

```

```

105     return scnt;
106 }
107 void init() {
108     scnt = 0;
109     root = newnode(0);
110 }
111 int work(int p, int c){
112     int q = ch[p][c];
113     int nq = newnode(len[p] + 1, q);
114     fa[q] = nq;
115     for(; p && ch[p][c] == q; p = fa[p]) ch[p][c] = nq;
116     return nq;
117 }
118 int extend(int p, int c) {
119     if (ch[p][c]){
120         int q = ch[p][c];
121         if (len[p] + 1 == len[q]) return q;
122         return work(p, c);
123     }
124     int np = newnode(len[p] + 1);
125     for(; p && ch[p][c] == 0; p = fa[p]) ch[p][c] = np;
126     if (!p) fa[np] = root;
127     else{
128         int q = ch[p][c];
129         if (len[p] + 1 == len[q]) fa[np] = q;
130         else fa[np] = work(p, c);
131     }
132     return np;
133 }
134 void solve() {
135     int n; scanf("%d",&n);
136     for(int i = 1; i <= n; i++) {
137         scanf("%s", s);
138         int sn = strlen(s);
139         int last = root;
140         for(int j = 0; j < sn; ++j)
141             last = extend(last, s[j] - 'a');
142     }
143 }
144 }

```

### 3.3 Palindrome Related

#### 3.3.1 Manacher

```

1 namespace Manacher {
2     char S[MAXN << 1];
3     int scnt, ans;
4     int p[MAXN << 1]; //p[i] - 1
5     void init(char *s0, int sn0) {
6         S[0] = '$'; S[1] = '#';
7         for(int i = 0; i < sn0; i++) {
8             S[2 * i + 2] = s0[i];
9             S[2 * i + 3] = '#';
10        }
11        scnt = sn0 * 2 + 2;
12        S[scnt] = '&';
13    }
14    void manachar() {

```

```

15     int id = 0, mx = 0;
16     for(int i = 1; i < scnt; i++) {
17         p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
18         while(S[i + p[i]] == S[i - p[i]]) p[i]++;
19         if(i + p[i] > mx) {
20             mx = i + p[i];
21             id = i;
22         }
23     }
24 }
25 }

```

### 3.3.2 Palindromic Automaton

```

1 namespace PAM {
2     int scnt, S[MAXN];
3     int pcnt, last, len[MAXN], fail[MAXN], ch[MAXN][26];
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 newnode(int _len) {
7         len[pcnt] = _len;
8         cnt[pcnt] = num[pcnt] = 0;
9         for(int i = 0; i < 26; i++) ch[pcnt][i] = 0;
10        return pcnt++;
11    }
12    inline void init() {
13        S[scnt = 0] = -1;
14        pcnt = 0; newnode(0); newnode(-1);
15        fail[0] = 1; last = 0;
16    }
17    int getfail(int x) {
18        while(S[scnt - len[x] - 1] != S[scnt]) x = fail[x];
19        return x;
20    }
21    void extend(int c) {
22        S[++scnt] = c;
23        int cur = getfail(last);
24        if(!ch[cur][c]) {
25            int now = newnode(len[cur] + 2);
26            fail[now] = ch[getfail(fail[cur])][c];
27            ch[cur][c] = now;
28            num[now] = num[fail[now]] + 1;
29        }
30        last = ch[cur][c];
31        cnt[last]++;
32    }
33    void count() {
34        for(int i = pcnt - 1; i >= 0; i--) cnt[fail[i]] += cnt[i];
35    }
36 };

```

## 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 FWT

```

1 void FWT(LL *a, int n) {
2     for(int i = 2; i <= n; i <= 1) {
3         for(int j = 0; j < n; j += i) {
4             for(int d = 0, w = i >> 1; d < w; d++){
5                 LL u = a[j + d], v = a[j + d + w];
6                 //xor: a[j + d] = u + v, a[j + d + w] = u - v;
7                 //and: a[j + d] = u + v;
8                 //or : a[j + d + w] = u + v;
9             }
10        }
11    }
12 }
13 void UFWT(LL *a, int n) {
14     for(int i = 2; i <= n; i <= 1) {
15         for(int j = 0; j < n; j += i) {
16             for(int d = 0, w = i >> 1; d < w; d++) {
17                 LL u = a[j + d], v = a[j + d + w];
18                 //xor: a[j + d] = (u + v) / 2, a[j + d + w] = (u - v) / 2;
19                 //and: a[j + d] = u - v;
20                 //or : a[j + d + w] = v - u;
21             }
22        }
23    }
24 }
25 void solve(int n) {
26     FWT(a, n); FWT(b, n);
27     for(int i = 0; i < n; i++) a[i] = a[i] * b[i];
28     UFWT(a, n);
29 }

```

#### 4.1.4 Linear Basis

```

1  //dynamic
2  const int D = 60;
3  struct Basis {
4      vector<int> ind;
5      vector<LL> base;
6      Basis() {
7          ind.resize(D, -1);
8          base.resize(D);
9      }
10     bool update(LL x, int id) {
11         for(int i = 0; i < D; i++) if(~ind[i] && x >> i & 1) {
12             x ^= base[i];
13         }
14         if(!x) return 1;
15         int pos = __builtin_ctzll(x);
16         ind[pos] = id;
17         base[pos] = x;
18         return 0;
19     }
20 };
21 //array
22 int Gauss(int n, int m) {
23     int num = 1;
24     for(int x = 1; x <= n && x <= m; x++) {
25         int t = 0;
26         for(int j = x; j <= m; j++) if(g[j][x]) { t = j; break; }
27         if(t) {
28             swap(g[x], g[t]);
29             for(int i = x + 1; i <= n; i++) {
30                 if(g[i][x]) {
31                     for(int k = 1; k <= m; k++) g[i][k] ^= g[x][k];
32                 }
33             }
34             num++;
35         }
36     }
37     return --num;
38 }
39 //long long
40 int Gauss() {
41     int num = 1;
42     for(int k = 61; k >= 0; k--) {
43         int t = 0;
44         for(int j = num; j <= cnt; j++) if((A[j] >> k) & 1) { t = j; break; }
45         if(t) {
46             swap(A[t], A[num]);
47             for(int j = num + 1; j <= cnt; j++) if((A[j] >> k) & 1) A[j] ^= A[num];
48             num++;
49         }
50     }
51     return --num;
52 }

```

## 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 }
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]; //  $x_i = a[i] \pmod{m[i]}$ 
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 += 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 &amp; 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 DuJiao Sieve

$$\sum_{i=1}^n \phi(i)$$

```

1  vector<int> prime;
2  int phi[MAXN], P[MAXN];
3  bool isp[MAXN];
4  unordered_map<LL, int> mp;
5  void Euler(int n) {
6      phi[1] = 1;
7      for(int i = 2; i <= n; i++) {
8          if(!isp[i]) {
9              prime.push_back(i);
10             phi[i] = i - 1;
11         }
12         for(auto x : prime) {
13             if(i * x > n) break;
14             isp[i * x] = 1;
15             if(i % x == 0) {
16                 phi[i * x] = phi[i] * x;
17                 break;
18             }
19             phi[i * x] = phi[i] * (x - 1);
20         }
21     }
22     for(int i = 1; i <= n; i++) P[i] = (P[i - 1] + phi[i]) % mod;
23 }
24 LL cal(LL n) {
25     if(n < MAXN) return P[n];
26     if(mp.count(n)) return mp[n];
27     LL res = 0;
28     for(LL i = 2, last; i <= n; i = last + 1) {
29         last = n / (n / i);

```

```

30     res += (last - i + 1) % mod * cal(n / i) % mod;
31     res %= mod;
32 }
33 mp[n] = ((__int128)n * (n + 1) / 2 % mod + mod - res) % mod;
34 return mp[n];
35 }

```

$$\sum_{i=1}^n \mu(i)$$

```

1 LL cal(LL n) {
2     if(n < MAXN) return M[n];
3     if(mp.count(n)) return mp[n];
4     LL res = 0;
5     for(LL i = 2, last; i <= n; i = last + 1) {
6         last = n / (n / i);
7         res += (last - i + 1) * cal(n / i);
8     }
9     mp[n] = 1 - res;
10    return 1 - res;
11 }

```

#### 4.2.9 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) %
mod;
34     ll last;
35     for(ll d = 1; d <= min(n, m); d = last + 1) {
36         last = min(n / (n / d), m / (m / d));
37         ans = (ans + (last - d + 1) * (d + last) / 2 % mod * F(n / d, m / d) % mod) %
mod;
38     }
39     ans = (ans + mod) % mod;
40     cout<<ans<<endl;
41     return 0;
42 }
```

## 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 mult_Cross(const Vector a,const Vector b,const Vector c){return (a.x-c.x)*(b.
y-c.y)-(b.x-c.x)*(a.y-c.y);}
15    double mult_Dot(const Vector a,const Vector b,const Vector c){return (a.x-c.x)*(b.x-
c.x)+(a.y-c.y)*(b.y-c.y);}
16    double Norm(const Vector a){return sqrt(Dot(a,a));}
17    double Angle(const Vector a,const Vector b){return acos(Dot(a,b)/Norm(a)/Norm(b));}
18    Vector Rotate(const Vector a,const double theta){return Vector(a.x*cos(theta)-a.y*
sin(theta),a.x*sin(theta)+a.y*cos(theta));}
19    boolToLeftTest(const Vector a,const Vector b){return Cross(a,b)<0;}
20    double DisPP(const Vector a,const Vector b){return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y
)*(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=
atan2(v.y,v.x);}

```

```

6      // Line(Point _p0=0,Point _v=0,double _t=1):p0(_p0),p1(_v){v=(p1-p0)/t; theta=
      atan2(v.y,v.x);}
7  };
8  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     }
35     return (sgn(c1)*sgn(c2)<0 && sgn(c3)*sgn(c4)<0);
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 }

```



### 5.1.6 Anticlockwise Sort

```

1 namespace CG{
2     \p
3     void clockwise_sort(Point *p,int n){
4         for(int i=0;i<n-2;i++){
5             double tmp = mult_Cross(p[i+1],p[i+2],p[i]);
6             if(tmp>0) return;
7             else if(tmp<0){
8                 reverse(p,p+n);
9                 return;
10            }
11        }
12    }
13 }

```

## 5.2 Convex Hull

### 5.2.1 Get Convex Hull

```

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

```

### 5.2.2 Point in Convex Hull

```

1 namespace CG{
2     bool PointInConvexHull(Point A){
3         int l=1,r=tot-2,mid;
4         while(l<=r){
5             mid=(l+r)>>1;
6             double a1=Cross(p[mid]-p[0],A-p[0]);
7             double a2=Cross(p[mid+1]-p[0],A-p[0]);
8             if(a1>=0 && a2<=0){
9                 if(Cross(p[mid+1]-p[mid],A-p[mid])>=0) return true;
10                return false;
11            }
12            else if(a1<0) r=mid-1;
13            else l=mid+1;
14        }
15        return false;
16    }
17 }

```

### 5.3 Minkowski Sum

```

1 namespace CG{
2     void Minkowski(Point *C1,int n,Point *C2,int m){
3         for(int i=1;i<=n;i++) s1[i]=C1[i]-C1[i-1];
4         for(int i=1;i<=m;i++) s2[i]=C2[i]-C2[i-1];
5         A[tot=1]=C1[1]+C2[1];
6         int p1=1,p2=1;
7         while (p1<=n && p2<=m) ++tot,A[tot]=A[tot-1]+(s1[p1]*s2[p2]>=0?s1[p1++]:s2[p2
++]);
8         while (p1<=n) ++tot,A[tot]=A[tot-1]+s1[p1++];
9         while (p2<=m) ++tot,A[tot]=A[tot-1]+s2[p2++];
10        tot=ConvexHull(A,tot);
11    }
12 }

```

### 5.4 Rotating Calipers

#### 5.4.1 The Diameter of Convex Hull

```

1 namespace CG{
2     double RotatingCalipers(Point *p,int n){
3         double dis=0;
4         for(int i=0,j=2;i<n;++i){
5             while (abs(Cross(p[i+1]-p[i],p[j]-p[i]))<abs(Cross(p[i+1]-p[i],p[j+1]-p[i])))
6             ) j=(j+1)%n;
7             dis=max(dis,max(DisPP(p[j],p[i]),DisPP(p[j],p[i+1])));
8         }
9         return dis;
10    }

```

#### 5.4.2 The Min Distance Between two Convex Hull

```

1 namespace CG{
2     ///点c到线段ab的最短距离
3     double GetDist(Point a,Point b,Point c){
4         if(dis(a,b)<esp) return dis(b,c); ///a,b是同一个点
5         if(mult_Dot(b,c,a)<-esp) return dis(a,c); ///投影
6         if(mult_Dot(a,c,b)<-esp) return dis(b,c);
7         return fabs(mult_Cross(b,c,a)/dis(a,b));
8     }
9
10    ///求一条线段ab的两端点到另外一条线段bc的距离，反过来一样，共4种情况
11    double MinDist(Point a,Point b,Point c,Point d){
12        return min(min(GetDist(a,b,c),GetDist(a,b,d)),min(GetDist(c,d,a),GetDist(c,d,b))
13    );
14    }
15    double RotatingCalipers(Point *p,int n,Point *q,int m){
16        int yminP = 0,ymaxQ=0;
17        for(int i=1;i<n;i++){ ///找到点集p组成的凸包的左下角
18            if(p[i].y<p[yminP].y || (p[i].y==p[yminP].y)&&(p[i].x<p[yminP].x)) yminP = i;
19        }
20        for(int i=1;i<m;i++){ ///找到点集q组成的凸包的右上角
21            if(q[i].y>q[ymaxQ].y || (q[i].y==q[ymaxQ].y)&&(q[i].x>q[ymaxQ].x)) ymaxQ = i;
22        }
23        double ans = DisPP(p[yminP],q[ymaxQ]); ///距离(yminP,ymaxQ)维护为当前最小值。

```

```

23     for(int i=0;i<n;i++){
24         double tmp;
25         while(tmp=(mult_Cross(q[ymaxQ+1],p[yminP],p[yminP+1])-mult_Cross(q[ymaxQ],p[
yminP],p[yminP+1]))>esp)
26             ymaxQ = (ymaxQ+1)%m;
27         if(tmp<=-esp) ans = min(ans,GetDist(p[yminP],p[yminP+1],q[ymaxQ]));
28         else ans=min(ans,MinDist(p[yminP],p[yminP+1],q[ymaxQ],q[ymaxQ+1]));
29         yminP = (yminP+1)%n;
30     }
31     return ans;
32 }
33 }

```

## 5.5 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.
pop_back());
9             for (;!p.empty() && !ToLeftTest(p.front()-l[i].p0,l[i].v);q.pop_front(),p.
pop_front());
10            if (sgn(Cross(l[i].v,q.back().v))==0)
11                if (ToLeftTest(l[i].p0-q.back().p0),q.back().v){
12                    q.pop_back();
13                    if (!p.empty()) p.pop_back();
14                }
15            if (!q.empty()) p.push_back(GetIntersection(q.back(),l[i]));
16            q.push_back(l[i]);
17        }
18        for (;!p.empty() && !ToLeftTest(p.back()-q.front().p0,q.front().v);q.pop_back(),
p.pop_back());
19        p.push_back(GetIntersection(q.back(),q.front()));
20        double area=0.5*Cross(p.back(),p.front()); Point last=p.front();
21        for (p.pop_front();!p.empty();last=p.front(),p.pop_front()) area+=0.5*Cross(last
,p.front());
22        printf("%.1f",Abs(area));
23    }
24 }

```

## 5.6 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     printf("%.10f\n%.10f %.10f",r,c.x,c.y);
31 }
32 }

```

## 5.7 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[
58                 j] = 1;
59         }
60     }
61     nn = 0;
62     for(int i = 1; i <= n; i++) if(!del[i]) {
63         a[++nn] = a[i];
64     }
65     n = nn;
66     /*
67     for(int i = 1; i <= n; i++) {
68         acnt = 0;
69         for(int j = 1; j <= n; j++) if(i != j) {
70             int dis = (a[i].x - a[j].x) * (a[i].x - a[j].x) + (a[i].y - a[j].y) * (a[i].
71             y - a[j].y);
72             if(a[j].r > a[i].r && dis <= (a[j].r - a[i].r) * (a[j].r - a[i].r)) add(-pi,
73             pi);
74             else if(dis > (a[i].r - a[j].r) * (a[i].r - a[j].r) && dis < (a[i].r + a[j].
75             r) * (a[i].r + a[j].r)){
76                 double c = sqrt(dis);
77                 double angle = acos((a[i].r * a[i].r + c * c - a[j].r * a[j].r) / (2 * a
78                 [i].r * c));
79                 double k = atan2(a[j].y - a[i].y, a[j].x - a[i].x);
80                 add(k - angle, k + angle);
81             }
82         }
83         arc[++acnt] = P(pi, -1);
84         sort(arc + 1, arc + acnt + 1);
85         cnt = 0;
86         double last = -pi;
87         for(int j = 1; j <= acnt; j++) {
88             s[cnt] += F(arc[j].first - last) * a[i].r * a[i].r; //扇形 - 三角形
89             double xa = a[i].x + a[i].r * cos(last);
90             double ya = a[i].y + a[i].r * sin(last);
91             last = arc[j].first;
92             double xb = a[i].x + a[i].r * cos(last);
93             double yb = a[i].y + a[i].r * sin(last);
94             s[cnt] += (xa * yb - xb * ya) / 2; //到圆心的三角形面积
95             cnt += arc[j].second;
96         }
97     }
98 }

```

```
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 }
```

## 5.8 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:h12
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(3,"")
30     call append(4,"int main() {")
31     call append(5,"    ios::sync_with_stdio(0); cin.tie(0); cout.precision(6); cout <<
fixed;")
32     call append(6,"")
33     call append(7,"    return 0;")
34     call append(8,"}")
35 endfunc
36 map <F9> :call Compile()<CR>
37 map <F5> :call Debug()<CR>
38 map <F8> :call AddTitle()<CR>

```

#### 6.1.2 Check

```

1 while true; do
2     ./data > in
3     ./tmp < in > out
4     ./std < in > ans
5     diff out ans
6     if [ $? -ne 0 ] ; then exit; fi
7     echo Passed
8 done

```

### 6.1.3 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 }

```

### 6.1.4 Java BigNum

```

1 import java.math.*;
2 import java.util.*;
3 import java.lang.*;
4
5 public class Main{
6     public static void main(String []args){}
7 }
8 //IO
9 Scanner in = new Scanner(System.in);
10 while(in.hasNext()){} //EOF
11 //fast-IO
12 public static void main(String argv[]) throws IOException{}
13 StreamTokenizer cin = new StreamTokenizer(new BufferedReader(new InputStreamReader(
14     System.in)));
15 PrintWriter cout = new PrintWriter(new OutputStreamWriter(System.out));
16 while(cin.nextToken() != StreamTokenizer.TT_EOF) ;//EOF
17 cin.nextToken();int n = (int)cin.nval;String s = cin.sval;
18 cout.println( Type );cout.flush();
19 cin.ordinaryChar( '/' );
20
21 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
22 br.ready()//EOF

```



```

22 while ((valueString=bf.readLine())!=null);
23 br.close();
24 //true fast-IO
25 static class InputReader {
26     public BufferedReader reader;
27     public StringTokenizer tokenizer;
28
29     public InputReader(InputStream stream) {
30         reader = new BufferedReader(new InputStreamReader(stream), 32768);
31         tokenizer = null;
32     }
33
34     public String next() {
35         while (tokenizer == null || !tokenizer.hasMoreTokens()) {
36             try {
37                 tokenizer = new StringTokenizer(reader.readLine());
38             } catch (IOException e) {
39                 throw new RuntimeException(e);
40             }
41         }
42         return tokenizer.nextToken();
43     }
44
45     public int nextInt() {
46         return Integer.parseInt(next());
47     }
48 }
49
50 //类 Number
51 //doubleValue()
52 //intValue()
53 //longValue()
54 //shortValue()
55 //类 BigDecimal
56 //ROUND_CEILING 接近正无穷大的舍入模式。
57 //ROUND_FLOOR 接近负无穷大的舍入模式。
58 //ROUND_DOWN 接近零的舍入模式
59 //ROUND_HALF_UP 四舍五入 >=0.5向上舍入
60 //ROUND_HALF_DOWN 四舍五入 >0.5向上舍入
61 //BigDecimal(BigInteger val)
62 //BigDecimal(BigInteger unscaledVal, int scale)
63 //BigDecimal(char[] in, int offset, int len, MathContext mc)
64 //BigDecimal(double val, MathContext mc)不建议
65 //BigDecimal(int val, MathContext mc)
66 //BigDecimal(long val, MathContext mc)
67 //BigDecimal(String val, MathContext mc)
68 //abs()
69 //add(BigDecimal augend, MathContext mc)
70 //compareTo(BigDecimal val)
71 //divide(BigDecimal divisor, MathContext mc)
72 //divideToIntegralValue(BigDecimal divisor, MathContext mc)
73 //max(BigDecimal val)
74 //min(BigDecimal val)
75 //multiply(BigDecimal multiplicand, MathContext mc)
76 //negate() 其值为 (-this), 其标度为 this.scale()
77 //pow(int n)
78 //remainder(BigDecimal divisor) 返回其值为 (this % divisor) 的 BigDecimal
79 //round(MathContext mc) 返回根据 MathContext 设置进行舍入后的 BigDecimal。
80 //scaleByPowerOfTen(int n) 返回其数值等于 (this * 10^n) 的 BigDecimal。
81 //subtract(BigDecimal subtrahend, MathContext mc)
82 //setScale(int newScale, RoundingMode roundingMode)

```

```

83 //toString()
84 //ulp() 返回此 BigDecimal 的 ulp (最后一位的单位) 的大小
85 //String s = b.stripTrailingZeros().toPlainString(); 让 bigdecimal 不用科学计数法显示
86 //类 BigInteger
87 //parseInt
88 //BigInteger zero = BigInteger.valueOf(0);
89 //BigInteger a = in.nextBigInteger();
90 //abs()
91 //and(BigInteger val) 返回其值为 (this & val)
92 //or(BigInteger val) 返回其值为 (this | val)
93 //andNot(BigInteger val) 返回其值为 (this & ~val)
94 //compareTo(BigInteger val)
95 //add(BigInteger val)
96 //divide(BigInteger val)
97 //BigInteger[] divideAndRemainder(BigInteger val) 返回包含 (this / val) 后跟 (this %
    val) 的两个 BigInteger 的数组。
98 //equals(Object x)
99 //gcd(BigInteger val)
100 //isProbablePrime(int certainty) e.g.: a.isProbablePrime(4)
101 //max(BigInteger val) min(BigInteger val)
102 //mod(BigInteger m)
103 //modInverse(BigInteger m) 返回其值为 (this-1 mod m)
104 //modPow(BigInteger exponent, BigInteger m) 返回其值为 (thisexponent mod m)
105 //multiply(BigInteger val)
106 //not() 返回其值为 (~this)
107 //shiftLeft(int n) 返回其值为 (this << n)
108 //shiftRight(int n) 返回其值为 (this >> n)
109 //toString()
110 //valueOf(long val)
111 //xor(BigInteger val) 返回其值为 (this ^ val)
112 //other
113 //Arrays.sort(array);

```

## 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 < x.ti);
11    }
12    }q[MAXN];
13    struct Node2 {
14        int l, r, ti;
15    }qq[MAXN];
16    int n, m, Q, Q0, Q1;
17    int V[MAXN], W[MAXN], C[MAXN];
18    int fa[MAXN][S + 3], dep[MAXN];
19    long long ans[MAXN], tans;
20    int vis[MAXN], cur[MAXN];
21    long long sum[MAXN];
22    int l, r, tm;
23    inline int read() {
24        int x = 0; char ch = getchar(); bool fg = 0;
25        while(ch < '0' || ch > '9') { if(ch == '-') fg = 1; ch = getchar(); }
26        while(ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
27        return fg ? -x : x;
28    }
29    inline void add_edge(int u, int v) {
30        e[++ecnt] = (Edge) {v, head[u]}; head[u] = ecnt;
31        e[++ecnt] = (Edge) {u, head[v]}; head[v] = ecnt;
32    }
33    void dfs(int u, int f) {
34        fa[u][0] = f;
35        dep[u] = dep[f] + 1;
36        int bot = top;
37        for(int i = head[u]; i; i = e[i].nxt) {
38            int v = e[i].to;
39            if(v == f) continue;
40            dfs(v, u);
41            if(top - bot >= sz) {
42                cnt++;
43                while(top != bot) belong[stack[top--]] = cnt;
44            }
45            stack[++top] = u;
46        }
47    }
48    void G(int &u, int step) {
49        for(int i = 0; i < S; i++) if((1 << i) & step) u = fa[u][i];
50    }
51    int lca(int u, int v) {
52        if(dep[u] > dep[v]) swap(u, v);
53        G(v, dep[v] - dep[u]);
54        if(u == v) return u;
55        for(int i = S; i >= 0; i--) if(fa[u][i] != fa[v][i]) {
56            u = fa[u][i]; v = fa[v][i];
57        }
58        return fa[u][0];
59    }
60    inline void modify(int u) {
61        tans -= V[C[u]] * sum[cur[C[u]]];
62        cur[C[u]] += vis[u];
63        vis[u] = -vis[u];
64        tans += V[C[u]] * sum[cur[C[u]]];
65    }
66    inline void update(int u, int v) {
67        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 Möbius Inversion Formula

Dirichlet Convolution is  $(f \times g)(N) = \sum_{d|N} f(d) * g(\frac{N}{d})$

Theorem:

$$\begin{cases} f = g \times 1 \\ g = f \times \mu \end{cases}$$

### 6.5.3 Math Theory Tips

$$n = \sum_{d|n} \phi(d) \tag{1}$$

$$e(n) = \sum_{d|n} \mu(d) \tag{2}$$

$$\sum_i^n \sum_j^m \gcd(i, j) = \sum_d^{\max(n, m)} \phi(d) * \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor \tag{3}$$

$$\sum_i^n \sum_j^m e(\gcd(i, j)) = \sum_d^{\min(n, m)} \mu(d) * \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor \tag{4}$$

$$\sum_{i=1}^n |\mu(i)| = \sum_{i=1}^{\lfloor \sqrt{n} \rfloor} \mu(i) * \lfloor \frac{n}{i * i} \rfloor \tag{5}$$

$$\begin{cases} sum(x, y) = \sum_i^x \sum_j^y i * j = \frac{x * (x + 1)}{2} * \frac{y * (y + 1)}{2} \\ F(x, y) = \sum_{i=1}^{\min(x, y)} i^2 * \mu(i) * sum(\lfloor \frac{x}{i} \rfloor, \lfloor \frac{y}{i} \rfloor) \\ \sum_i^n \sum_j^m lcm(i, j) = \sum_{i=1}^{\min(n, m)} d * F(\lfloor \frac{n}{i} \rfloor, \lfloor \frac{y}{i} \rfloor) \end{cases} \tag{6}$$