XCPC Template Library

Wu Honglin / real01bit

Li Yuanzhuo / SnowFlavour
Liu Haocheng / lotus_grass

最后一次修改：2025 年 8 月 25 日

目录

第 1 部分　图论 Graph

1.1　连通性相关

1.1.1　双连通分量

1.1.1.1　边双连通分量

在 $O(n)$ 的时间内求出边双连通分量。

```
#include <cstdio>
#include <iostream>
#include <vector>
using namespace std;
const int N = 5e5 + 10, M = 2e6 + 10;
int n, m, head[N], tot = 1;
struct edge {
   int v, nxt;
   bool is_bridge;
} e[M << 1];
void add(int u, int v) { e[++tot].v = v, e[tot].nxt = head[u], head[u] =
    tot; }
int dfn[N], low[N], Time;
void tarjan(int u, int edge) {
   dfn[u] = low[u] = ++Time;
   for (int i = head[u]; i; i = e[i].nxt) {
      int v = e[i].v;
      if (!dfn[v]) {
         tarjan(v, i), low[u] = min(low[u], low[v]);
      } else if (i != (edge ^ 1)) low[u] = min(low[u], dfn[v]);
      if (dfn[u] < low[v]) e[i].is_bridge = e[i ^ 1].is_bridge = true;
   }
}
int dc;
bool in_dcc[N];
vector<int> DCC[N];
void dfs(int u, int dc) {
   in_dcc[u] = true;
   DCC[dc].push_back(u);
   for (int i = head[u]; i; i = e[i].nxt) {
      int v = e[i].v;
      if (in_dcc[v] || e[i].is_bridge) continue;
```

```
32        dfs(v, dc);
33     }
34  }
35  int main() {
36     ios::sync_with_stdio(false);
37     cin.tie(0), cout.tie(0);
38     cin >> n >> m;
39     for (int i = 1; i ⩽ m; i++) {
40        int u, v;
41        cin >> u >> v;
42        add(u, v);
43        add(v, u);
44     }
45     for (int i = 1; i ⩽ n; i++)
46        if (!dfn[i]) tarjan(i, 0);
47     for (int i = 1; i ⩽ n; i++)
48        if (!in_dcc[i]) dfs(i, ++dc);
49     cout << dc << "\n";
50     for (int i = 1; i ⩽ dc; i++) {
51        cout << DCC[i].size() << "␣";
52        for (auto v : DCC[i]) cout << v << "␣";
53        cout << "\n";
54     }
55     return 0;
56  }
```

1.1.1.2 点双连通分量

在 $O(n)$ 的时间内求出点双连通分量。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5  const int N = 500005;
6  const int M = 2000005;
7  int n, m, head[N], tot;
8  struct edge {
9     int v, nxt;
10 } e[M << 1];
11 void add(int u, int v) {
12    e[++tot].v = v;
13    e[tot].nxt = head[u];
14    head[u] = tot;
15 }
16 int dfn[N], low[N], Time;
17 vector<int> dcc[N];
18 int cnt, stk[N], top;
19 void tarjan(int u, int rt) {
```

```
20      dfn[u] = low[u] = ++Time, stk[++top] = u;
21      if (u == rt && head[u] == 0) {
22          dcc[++cnt].push_back(u);
23          return;
24      }
25      for (int i = head[u]; i; i = e[i].nxt) {
26          int v = e[i].v;
27          if (!dfn[v]) {
28              tarjan(v, rt);
29              low[u] = min(low[u], low[v]);
30              if (dfn[u] <= low[v]) {
31                  ++cnt;
32                  do { dcc[cnt].push_back(stk[top--]); } while (stk[top + 1] != v
                        );
33                  dcc[cnt].push_back(u);
34              }
35          } else low[u] = min(low[u], dfn[v]);
36      }
37  }
38  int main() {
39      cin.tie(0)->sync_with_stdio(false);
40      cout.tie(0);
41      cin >> n >> m;
42      for (int i = 1; i <= m; i++) {
43          int u, v;
44          cin >> u >> v;
45          if (u == v) continue;
46          add(u, v);
47          add(v, u);
48      }
49      for (int i = 1; i <= n; i++)
50          if (!dfn[i]) tarjan(i, i);
51      cout << cnt << "\n";
52      for (int i = 1; i <= cnt; i++) {
53          cout << dcc[i].size() << " ";
54          for (auto x : dcc[i]) cout << x << " ";
55          cout << "\n";
56      }
57      return 0;
58  }
```

1.1.2 强连通分量

1.1.2.1 Kosaraju

利用反图求强连通分量。

```
1   #include <bits/stdc++.h>
2   using namespace std;
```

```
const int N = 1e6 + 5;
vector<vector<int>> g, g2; // g 是原图，g2 是反图
vector<int> s;
bool vis[N];
int sccCnt, n, color[N];
void dfs1(int u) {
   vis[u] = true;
   for (int v : g[u])
      if (!vis[v]) dfs1(v);
   s.push_back(u);
}
void dfs2(int u) {
   color[u] = sccCnt;
   for (int v : g2[u])
      if (!color[v]) dfs2(v);
}
void kosaraju() {
   sccCnt = 0;
   for (int i = 1; i ⩽ n; ++i)
      if (!vis[i]) dfs1(i);
   for (int i = n; i ⩾ 1; --i)
      if (!color[s[i]]) {
         ++sccCnt;
         dfs2(s[i]);
      }
}
```

1.1.2.2 Tarjan

利用 Tarjan 算法求强连通分量。

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;
const int N = 10005, M = 100005;
int n, m, head[M], tot;
struct edge {
   int v, nxt;
} e[M];
void add(int u, int v) {
   e[++tot].v = v;
   e[tot].nxt = head[u];
   head[u] = tot;
}
int dfn[N], low[N];
int stk[N], top, Time;
```

```
19  bool ins[N], vis[N];
20  int fa[N], cnt;
21  vector<vector<int>> SCC;
22  void tarjan(int u) {
23     dfn[u] = ++Time, low[u] = dfn[u], stk[++top] = u, ins[u] = true;
24     for (int i = head[u]; i; i = e[i].nxt) {
25        int v = e[i].v;
26        if (!dfn[v]) {
27           tarjan(v);
28           low[u] = min(low[u], low[v]);
29        } else if (ins[v]) low[u] = min(low[u], dfn[v]);
30     }
31     if (dfn[u] == low[u]) {
32        ++cnt;
33        SCC.push_back(vector<int>());
34        while (stk[top] != u) {
35           SCC[cnt].push_back(stk[top]);
36           fa[stk[top]] = cnt, ins[stk[top]] = false, top--;
37        }
38        SCC[cnt].push_back(stk[top]);
39        fa[stk[top]] = cnt, ins[stk[top]] = false, top--;
40     }
41  }
42  int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(0);
45     cout.tie(0);
46     cin >> n >> m;
47     for (int i = 1; i <= m; i++) {
48        int u, v;
49        cin >> u >> v;
50        add(u, v);
51     }
52     SCC.push_back(vector<int>());
53     for (int i = 1; i <= n; i++)
54        if (!dfn[i]) tarjan(i);
55     cout << cnt << "\n";
56     for (int i = 1; i <= cnt; i++) sort(SCC[i].begin(), SCC[i].end());
57     for (int i = 1; i <= n; i++) {
58        if (vis[fa[i]]) continue;
59        else {
60           for (auto x : SCC[fa[i]]) cout << x << "␣";
61           cout << "\n";
62           vis[fa[i]] = true;
63        }
64     }
65     return 0;
66  }
```

1.2 最小生成树

1.2.1 Kruskal

利用 Kruskal 算法求最小生成树。时间复杂度为排序和并查集。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
struct E {
   int u, v, w;
   bool operator<(const E &b) const { return w < b.w; }
} Edge[N];
int f[N];
int getf(int x) { return f[x] == x ? x : f[x] = getf(f[x]); }
vector<int> tr[N];
int main() {
   int n, m;
   cin >> n >> m;
   for (int i = 1; i <= n; i++) f[i] = i;
   for (int i = 1; i <= m; i++) cin >> Edge[i].u >> Edge[i].v >> Edge[i].w;
   sort(Edge + 1, Edge + 1 + m);
   for (int i = 1; i <= m; i++) {
      int fu = getf(Edge[i].u), fv = getf(Edge[i].v);
      if (fu == fv) continue;
      tr[Edge[i].u].push_back(Edge[i].v);
      tr[Edge[i].v].push_back(Edge[i].u);
   }
   return 0;
}
```

1.2.2 Prim

利用 Prim 算法求最小生成树。时间复杂度为堆。

```cpp
#include <cstring>
#include <iostream>
#include <queue>
using namespace std;
constexpr int N = 5050, M = 2e5 + 10;
struct E {
   int v, w, x;
} e[M * 2];
int n, m, h[N], cnte;
void adde(int u, int v, int w) { e[++cnte] = E{v, w, h[u]}, h[u] = cnte; }
struct S {
   int u, d;
};
bool operator<(const S &x, const S &y) { return x.d > y.d; }
```

```
priority_queue<S> q;
int dis[N];
bool vis[N];
int res = 0, cnt = 0;
void Prim() {
   memset(dis, 0x3f, sizeof(dis));
   dis[1] = 0;
   q.push({1, 0});
   while (!q.empty()) {
      if (cnt ⩾ n) break;
      int u = q.top().u, d = q.top().d;
      q.pop();
      if (vis[u]) continue;
      vis[u] = true;
      ++cnt;
      res += d;
      for (int i = h[u]; i; i = e[i].x) {
         int v = e[i].v, w = e[i].w;
         if (w < dis[v]) { dis[v] = w, q.push({v, w}); }
      }
   }
}
int main() {
   cin >> n >> m;
   for (int i = 1, u, v, w; i ⩽ m; ++i) { cin >> u >> v >> w, adde(u, v, w
      ), adde(v, u, w); }
   Prim();
   if (cnt == n) cout << res;
   else cout << "No␣MST.";
   return 0;
}
```

1.3 最短路

1.3.1 Dijkstra

利用 Dijkstra 求最短路。时间复杂度为稳定的 $O(m \log n)$，不能有负权。

```
struct edge {
   int v, w;
};
struct node {
   int dis, u;
   bool operator>(const node &a) const { return dis > a.dis; }
};
vector<edge> e[MAXN];
int dis[MAXN], vis[MAXN];
priority_queue<node, vector<node>, greater<node>> q;
```

```
void dijkstra(int n, int s) {
   memset(dis, 0x3f, (n + 1) * sizeof(int));
   memset(vis, 0, (n + 1) * sizeof(int));
   dis[s] = 0;
   q.push({0, s});
   while (!q.empty()) {
      int u = q.top().u;
      q.pop();
      if (vis[u]) continue;
      vis[u] = 1;
      for (auto ed : e[u]) {
         int v = ed.v, w = ed.w;
         if (dis[v] > dis[u] + w) {
            dis[v] = dis[u] + w;
            q.push({dis[v], v});
         }
      }
   }
}
```

1.3.2 SPFA

利用 SPFA 求最短路。时间复杂度不稳定（意思就是复杂度按照 $O(n^2)$ 算就对了），支持负权，可以在 $O(nm)$ 的时间内找到负环并且判断。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int n, m;
vector<pair<int, int>> g[N];
inline void add(int u, int v, int w) { g[u].push_back({v, w}); }
queue<int> q;
int dis[N];
bool inq[N];
inline void spfa(int s) {
   memset(dis, 0x3f, sizeof(dis));
   memset(inq, false, sizeof(inq));
   inq[s] = true, dis[s] = 0;
   q.push(s);
   while (q.size()) {
      int u = q.front();
      q.pop(), inq[u] = false;
      for (auto [v, w] : g[u])
         if (dis[v] > dis[u] + w) {
            dis[v] = dis[u] + w;
            if (!inq[v]) q.push(v);
         }
   }
}
```

```
int main() {
   cin >> n >> m;
   for (int i = 1, u, v, w; i ⩽ m; i++) cin >> u >> v >> w, add(u, v, w),
       add(v, u, w);
   spfa(1);
   return 0;
}
```

1.4 树相关

1.4.1 求树的重心

求出一棵树所有的重心。

```
int size[MAXN], weight[MAXN], centroid[2];
void GetCentroid(int cur, int fa) {
   size[cur] = 1, weight[cur] = 0;
   for (int i = head[cur]; i ≠ -1; i = e[i].nxt) {
      if (e[i].to ≠ fa) {
         GetCentroid(e[i].to, cur);
         size[cur] += size[e[i].to];
         weight[cur] = max(weight[cur], size[e[i].to]);
      }
   }
   weight[cur] = max(weight[cur], n - size[cur]);
   if (weight[cur] ⩽ n / 2) centroid[centroid[0] ≠ 0] = cur;
}
```

1.4.2 树的直径

求出一棵树的直径。这是贪心的方法，不能处理负权。

```
#include <bits/stdc++.h>
constexpr int N = 10000 + 10;
int n, c, d[N];
std::vector<int> E[N];
void dfs(int u, int fa) {
   for (int v : E[u]) {
      if (v == fa) continue;
      d[v] = d[u] + 1;
      if (d[v] > d[c]) c = v;
      dfs(v, u);
   }
}
int main() {
   scanf("%d", &n);
   for (int i = 1; i < n; i++) {
      int u, v;
```

```
17        scanf("%d␣%d", &u, &v);
18        E[u].push_back(v), E[v].push_back(u);
19     }
20     dfs(1, 0);
21     d[c] = 0, dfs(c, 0);
22     printf("%d\n", d[c]);
23     return 0;
24  }
```

1.4.3 重链剖分

对树进行重链剖分，可以配合线段树进行链上修改。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const int N = 1e5 + 5;
4   int n;
5   vector<int> g[N];
6   int fa[N], dep[N], siz[N], hson[N], dfn[N], idx, top[N];
7   void hld1(int u, int fath) {
8      fa[u] = fath, dep[u] = dep[fath] + 1, siz[u] = 1;
9      for (int v : g[u]) {
10        if (v == fath) continue;
11        hld1(v, u), siz[u] += siz[v];
12        if (siz[v] > siz[hson[u]]) hson[u] = v;
13     }
14  }
15  void hld2(int u, int nowtp) {
16     top[u] = nowtp, dfn[u] = ++idx;
17     if (hson[u]) hld2(hson[u], nowtp);
18     for (int v : g[u])
19        if (v != fa[u] && v != hson[u]) hld2(v, v);
20  }
21  inline int LCA(int x, int y) {
22     while (top[x] != top[y]) return dep[top[x]] < dep[top[y]] ? y = fa[top[y
          ]] : x = fa[top[x]];
23     return dep[x] < dep[y] ? x : y;
24  }
25  int main() { return 0; }
```

1.4.4 虚树

二次排序法建虚树。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const int N = 1e5 + 5;
4   int n, q, m, h[N], s[N], cnt;
5   vector<int> g[N], vt[N];
```

```
int fa[N], dep[N], siz[N], hson[N], dfn[N], top[N], idx;
void hld1(int u, int fath) { ... }
void hld2(int u, int fath) { ... }
int lca(int u, int v) { ... } // 重链剖分相关内容见 oi-algorithms/graph/tree/
    hld.cpp
bool cmp(int a, int b) { return dfn[a] < dfn[b]; }
inline void buildVtree() {
   sort(h + 1, h + 1 + m, cmp);
   s[cnt = 1] = 1;
   for (int i = 1; i ⩽ m; i++) s[++cnt] = h[i], s[++cnt] = lca(h[i], h[i +
       1]);
   s[++cnt] = h[m];
   sort(s + 1, s + 1 + cnt, cmp);
   cnt = unique(s + 1, s + 1 + cnt) - s - 1;
   for (int i = 1; i ⩽ cnt; i++) vt[lca(s[i], s[i + 1])].push_back(s[i +
      1]);
}
int main() { return 0; }
```

1.5 差分约束

（需要补充）

```
#include <cstring>
#include <iostream>
#include <queue>
using namespace std;
struct edge {
   int v, w, next;
} e[40005];
int head[10005], vis[10005], tot[10005], cnt;
long long ans, dist[10005];
queue<int> q;
void addedge(int u, int v, int w) { // 加边
   e[++cnt].v = v;
   e[cnt].w = w;
   e[cnt].next = head[u];
   head[u] = cnt;
}
int main() {
   cin.tie(nullptr)->sync_with_stdio(false);
   int n, m;
   cin >> n >> m;
   for (int i = 1; i ⩽ m; i++) {
      int op, x, y, z;
      cin >> op;
      if (op == 1) {
         cin >> x >> y >> z;
```

```
26            addedge(y, x, z);
27         } else if (op == 2) {
28            cin >> x >> y >> z;
29            addedge(x, y, -z);
30         } else {
31            cin >> x >> y;
32            addedge(x, y, 0);
33            addedge(y, x, 0);
34         }
35      }
36      for (int i = 1; i <= n; i++) addedge(0, i, 0);
37      memset(dist, -0x3f, sizeof(dist));
38      dist[0] = 0;
39      vis[0] = 1;
40      q.push(0);
41      while (!q.empty()) { // 判负环，看上面的
42         int cur = q.front();
43         q.pop();
44         vis[cur] = 0;
45         for (int i = head[cur]; i; i = e[i].next)
46            if (dist[cur] + e[i].w > dist[e[i].v]) {
47               dist[e[i].v] = dist[cur] + e[i].w;
48               if (!vis[e[i].v]) {
49                  vis[e[i].v] = 1;
50                  q.push(e[i].v);
51                  tot[e[i].v]++;
52                  if (tot[e[i].v] >= n) {
53                     cout << "No\n";
54                     return 0;
55                  }
56               }
57            }
58      }
59      cout << "Yes\n";
60      return 0;
61   }
```

第 2 部分　字符串 String

2.1　字符串哈希

2.1.1　双模哈希

利用两个模数同时计算哈希，防止哈希碰撞。

```
#include <cstring>
#include <string>
using std::string;
using ull = unsigned long long;
ull base = 131, mod1 = 212370440130137957, mod2 = 1e9 + 7;
ull get_hash1(string s) {
    int len = s.size();
    ull ans = 0;
    for (int i = 0; i < len; i++) ans = (ans * base + (ull)s[i]) % mod1;
    return ans;
}
ull get_hash2(string s) {
    int len = s.size();
    ull ans = 0;
    for (int i = 0; i < len; i++) ans = (ans * base + (ull)s[i]) % mod2;
    return ans;
}
bool cmp(const string s, const string t) {
    bool f1 = get_hash1(s) != get_hash1(t);
    bool f2 = get_hash2(s) != get_hash2(t);
    return f1 || f2;
}
int main() { return 0; }
```

2.1.2　（需要补充）

（需要补充）

```
#include <cstring>
#include <string>
using std::string;
constexpr int M = 1e9 + 7;
constexpr int B = 233;
using ll = long long;
```

```
int get_hash(const string &s) {
   int res = 0;
   for (int i = 0; i < s.size(); ++i) { res = ((ll)res * B + s[i]) % M; }
   return res;
}
bool cmp(const string &s, const string &t) { return get_hash(s) == get_hash
    (t); }
int main() { return 0; }
```

2.2 后缀数组

2.2.1 （需要补充）

（需要补充）

```
#include <cstring>
#include <iostream>
#define f(x) (x / 3) + (x % 3 == 1 ? 0 : cd) // f(x) 表示原串中后缀 x 在新串中
    的位置
#define g(x) (x >= cd ? (x - cd) * 3 + 2 : x * 3 + 1) // g(x) 表示新串后缀 x
    在原串中的位置
using namespace std;
const int N = 1e6 + 5;
string s;
int n, sa[N * 3 + 100], rk[N * 3 + 100], buc[N], x[N], y[N], height[N];
void sort(int *rk, int *a, int *b, int n, int Sigma) // rk 表示待排序串, a 表
    示指标集, b 表示排序后的指标顺序
{
   for (int i = 0; i <= Sigma; i++) buc[i] = 0;
   for (int i = 0; i < n; i++) buc[rk[a[i]]]++;
   for (int i = 1; i <= Sigma; i++) buc[i] += buc[i - 1];
   for (int i = n - 1; i >= 0; i--) b[--buc[rk[a[i]]]] = a[i];
}
bool cmp3(int *r, int x, int y) { return r[x] == r[y] && r[x + 1] == r[y +
    1] && r[x + 2] == r[y + 2]; }
bool cmps(int *r, int x, int y) {
   if (r[x] != r[y]) return r[x] < r[y];
   if (x % 3 == 1) return buc[x + 1] < buc[y + 1];
   return !cmps(r, y + 1, x + 1);
}
void DC3(int *rk, int *sa, int n, int Sigma) {
   bool h = (n % 3 == 1);
   if (h) rk[n++] = 0;
   int *nrk = rk + n + 2, *nsa = sa + n, cb = 0, p; // nrk, nsa 分别表示新的
       rk 和 sa, cb 表示 B 类后缀的数量
   for (int i = 0; i < n; i++)
      if (i % 3) x[cb++] = i;
   rk[n] = rk[n + 1] = 0;
```

```
29      sort(rk + 2, x, y, cb, Sigma), sort(rk + 1, y, x, cb, Sigma), sort(rk, x
            , y, cb, Sigma); // 指标顺序存在了 y 中
30      int ca = 0, cd = (n + 1) / 3; // ca 表示 A 类后缀的数量, cd 表示 i % 3 = 1
            的后缀的数量
31      nrk[f(y[0])] = p = 1;
32      for (int i = 1; i < cb; i++) {
33         if (!cmp3(rk, y[i], y[i - 1])) p++;
34         nrk[f(y[i])] = p;
35      }
36      if (p < cb) DC3(nrk, nsa, cb, p); // 递归求解 B 类后缀
37      else
38         for (int i = 0; i < cb; i++)
39            if (nrk[i]) nsa[nrk[i] - 1] = i;
40      for (int i = 0; i < cb; i++)
41         if (nsa[i] < cd) y[ca++] = 3 * nsa[i];
42      sort(rk, y, x, ca, Sigma);
43      for (int i = 0; i < cb; i++) buc[y[i] = g(nsa[i])] = i; // 利用 buc 存 B
            类后缀的排名
44      buc[n] = -1, p = 0;
45      int i = 0, j = h;
46      while (i < ca && j < cb) {
47         if (cmps(rk, y[j], x[i])) sa[p++] = y[j++];
48         else sa[p++] = x[i++];
49      }
50      while (i < ca) sa[p++] = x[i++];
51      while (j < cb) sa[p++] = y[j++];
52   }
53   int main() {
54      cin.tie(nullptr)->sync_with_stdio(false);
55      cout.tie(nullptr);
56      cin >> s;
57      n = s.length();
58      for (int i = 0; i < n; i++) rk[i] = s[i] - '0' + 1;
59      DC3(rk, sa, n, 75);
60      for (int i = 0; i < n; i++) cout << sa[i] + 1 << '␣';
61      cout << '\n';
62      for (int i = 0; i < n; i++) rk[sa[i]] = i;
63      for (int i = 0, k = 0; i < n; i++) {
64         if (!rk[i]) continue;
65         if (k) --k;
66         while (s[i + k] == s[sa[rk[i] - 1] + k]) ++k;
67         height[rk[i]] = k;
68      }
69      for (int i = 1; i < n; i++) cout << height[i] << '␣';
70      return cout << '\n', 0;
71   }
```

2.2.2 （需要补充）

（需要补充）

```
#include <algorithm>
#include <cstdio>
using namespace std;
const int N = 1e6 + 10;
typedef long long ll;
template <typename T> inline void gm(T *&bas, int siz, T *&op) { op = bas, bas += siz; }
#define pus(x) (sa[cur[a[x]]--] = x)
#define pul(x) (sa[cur[a[x]]++] = x)
#define inds(lms)                           \
   for (int i = 1; i <= n; i++) sa[i] = -1; \
   for (int i = 1; i <= n; i++) sum[i] = 0; \
   for (int i = 1; i <= n; i++) sum[a[i]]++; \
   for (int i = 1; i <= n; i++) sum[i] += sum[i - 1]; \
   for (int i = 1; i <= n; i++) cur[i] = sum[i]; \
   for (int i = m; i >= 1; i--) pus(lms[i]); \
   for (int i = 1; i <= n; i++) cur[i] = sum[i - 1] + 1; \
   for (int i = 1; i <= n; i++)            \
      if (sa[i] > 1 && !tp[sa[i] - 1]) pul(sa[i] - 1); \
   for (int i = 1; i <= n; i++) cur[i] = sum[i]; \
   for (int i = n; i >= 1; i--)            \
      if (sa[i] > 1 && tp[sa[i] - 1]) pus(sa[i] - 1);
int sa[N], sum[N], cur[N], rk[N], A_bas[N << 4], *A_t;
inline void sais(int n, int *a) {
   int *tp, *p;
   gm(A_t, n + 1, tp), gm(A_t, n + 2, p);
   tp[n] = 1;
   for (int i = n - 1; i >= 1; i--) tp[i] = (a[i] == a[i + 1]) ? tp[i + 1] : (a[i] < a[i + 1]);
   int m = 0;
   for (int i = 1; i <= n; i++) rk[i] = (tp[i] && !tp[i - 1]) ? (p[++m] = i, m) : -1;
   inds(p);
   int tot = 0, *a1;
   gm(A_t, m + 1, a1);
   p[m + 1] = n;
   for (int i = 1, x, y; i <= n; i++)
      if ((x = rk[sa[i]]) != -1) {
         if (tot == 0 || p[x + 1] - p[x] != p[y + 1] - p[y]) tot++;
         else
            for (int p1 = p[x], p2 = p[y]; p2 <= p[y + 1]; p1++, p2++)
               if ((a[p1] << 1 | tp[p1]) != (a[p2] << 1 | tp[p2])) {
                  tot++;
                  break;
               }
```

```
43            a1[y = x] = tot;
44         }
45      if (tot == m)
46         for (int i = 1; i <= m; i++) sa[a1[i]] = i;
47      else sais(m, a1);
48      for (int i = 1; i <= m; i++) a1[i] = p[sa[i]];
49      inds(a1);
50   }
51   char mde[N];
52   int n;
53   int a[N];
54   int tr[300];
55   char buf[20];
56   int cnt;
57   int main() {
58      A_t = A_bas;
59      scanf("%s", mde + 1);
60      while (mde[n + 1] != '\0') n++;
61      for (int i = 1; i <= n; i++) tr[mde[i]] = 1;
62      for (int i = 1; i < 300; i++) tr[i] += tr[i - 1];
63      for (int i = 1; i <= n; i++) a[i] = tr[mde[i]] + 1;
64      a[++n] = 1;
65      sais(n, a);
66      for (int i = 2; i <= n; i++) {
67         int tmp = sa[i];
68         while (tmp) buf[++cnt] = tmp % 10 + 48, tmp /= 10;
69         while (cnt) putchar(buf[cnt--]);
70         putchar('␣');
71      }
72      return 0;
73   }
```

2.3 字典树

2.3.1 可持久化字典树

支持版本回滚的字典树。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const int N = 6e5 + 5, H = 28;
4   int n, m, a[N], s[N], rt[N], ch[N * 33][2], cnt[N * 33], tot;
5   void Insert(int u, int pre, int x) {
6      for (int i = H; i >= 0; i--) {
7         cnt[u] = cnt[pre] + 1;
8         int c = ((x & (1 << i)) ? 1 : 0);
9         if (!ch[u][c]) ch[u][c] = ++tot;
10        ch[u][c ^ 1] = ch[pre][c ^ 1];
```

```
11        u = ch[u][c], pre = ch[pre][c];
12     }
13     cnt[u] = cnt[pre] + 1;
14  }
15  int query(int u, int v, int x) {
16     int res = 0;
17     for (int i = H; i >= 0; i--) {
18        int c = ((x & (1 << i)) ? 1 : 0);
19        if (cnt[ch[u][!c]] - cnt[ch[v][!c]]) u = ch[u][!c], v = ch[v][!c],
              res += (1 << i);
20        else u = ch[u][c], v = ch[v][c];
21     }
22     return res;
23  }
24  int main() {
25     cin.tie(nullptr)->sync_with_stdio(false);
26     cout.tie(nullptr);
27     cin >> n >> m;
28     for (int i = 1, x; i <= n; i++) cin >> a[i], s[i] = s[i - 1] ^ a[i];
29     for (int i = 1; i <= n; i++) rt[i] = ++tot, Insert(rt[i], rt[i - 1], s[i
           ]);
30     char op;
31     for (int l, r, val; m; m--) {
32        cin >> op;
33        if (op == 'A') {
34           n++, cin >> a[n];
35           s[n] = s[n - 1] ^ a[n], rt[n] = ++tot;
36           Insert(rt[n], rt[n - 1], s[n]);
37        }
38        if (op == 'Q') {
39           cin >> l >> r >> val;
40           l--, r--;
41           if (!l) cout << max(s[n] ^ val, query(rt[r], rt[0], s[n] ^ val))
                 << '\n';
42           else cout << query(rt[r], rt[l - 1], s[n] ^ val) << '\n';
43        }
44     }
45     return 0;
46  }
```

2.3.2 字典树

空间复杂度 $O(n|S|)$，时间复杂度 $O(n|S|)$ 地解决一些字符串问题。

```
1  #include <cstring>
2  using namespace std;
3  struct trie {
4     int son[100000][26], cnt;
5     bool exist[100000];
```

```
6     void insert(char *s, int l) {
7        int p = 0;
8        for (int i = 0; i < l; i++) {
9           int c = s[i] - 'a';
10          if (!son[p][c]) son[p][c] = ++cnt;
11          p = son[p][c];
12       }
13       exist[p] = true;
14    }
15    bool find(char *s, int l) {
16       int p = 0;
17       for (int i = 0; i < l; i++) {
18          int c = s[i] - 'a';
19          if (!son[p][c]) return 0;
20          p = son[p][c];
21       }
22       return exist[p];
23    }
24 };
25 int main() { return 0; }
```

2.4 AC 自动机

（需要补充）

```
1  #include <cstring>
2  #include <queue>
3  #include <vector>
4  using namespace std;
5  namespace acam {
6  const int SIZ = 2000005;
7  struct node {
8     int ch[26], fail, ans, id;
9     node() { memset(ch, 0, sizeof(ch)), ans = id = 0; }
10    inline int &operator[](const int x) { return x < 26 ? ch[x] : ch[x - 'a'
          ]; }
11    inline int operator[](const int x) const { return x < 26 ? ch[x] : ch[x
          - 'a']; }
12 } tr[SIZ];
13 int tot, ans[SIZ], pcnt;
14 vector<int> fail[SIZ];
15 inline void insert(char *s, int &id) {
16    int u = 0;
17    for (int i = 1; s[i]; i++) {
18       if (!tr[u][i]) tr[u][i] = ++tot;
19       u = tr[u][i];
20    }
21    if (!tr[u].id) tr[u].id = ++pcnt;
```

```
22      id = tr[u].id;
23   }
24   inline void build() {
25      queue<int> q;
26      for (int i = 0; i < 26; i++)
27         if (tr[0][i]) q.push(tr[0][i]), fail[0].push_back(tr[0][i]);
28      while (!q.empty()) {
29         int u = q.front();
30         q.pop();
31         for (int i = 0; i < 26; i++) {
32            if (tr[u][i]) {
33               tr[tr[u][i]].fail = tr[tr[u].fail][i];
34               fail[tr[tr[u].fail][i]].push_back(tr[u][i]);
35               q.push(tr[u][i]);
36            } else tr[u][i] = tr[tr[u].fail][i];
37         }
38      }
39   }
40   void query(char *t) {
41      int u = 0;
42      for (int i = 1; t[i]; ++i) u = tr[u][t[i]], tr[u].ans++;
43   }
44   void dfs(int u) {
45      for (int v : fail[u]) {
46         dfs(v);
47         tr[u].ans += tr[v].ans;
48      }
49      ans[tr[u].id] = tr[u].ans;
50   }
51   }; // namespace acam
52   int main() { return 0; }
```

2.5 KMP

单模式串进行 $O(1)$ 的匹配。

```
1   #include <cstring>
2   #include <iostream>
3   #include <vector>
4   using namespace std;
5   vector<int> pf(string s) {
6      int n = (int)s.length();
7      vector<int> pi(n);
8      for (int i = 1; i < n; i++) {
9         int j = pi[i - 1];
10        while (j && s[j] != s[i]) j = pi[j - 1];
11        if (s[i] == s[j]) ++j;
12        pi[i] = j;
```

```
13      }
14      return pi;
15  }
16  int main() { return 0; }
```

2.6 Manacher

求出字符串中的最长回文字串。

```
1   #include <cstring>
2   #include <iostream>
3   using namespace std;
4   const int N = 2.2e7 + 10;
5   inline bool chk(char c) { return c ⩾ 'a' && c ⩽ 'z'; }
6   int n, f[N];
7   char s[N], c;
8   int main() {
9       s[0] = '~';
10      while (chk(c = getchar())) s[++n] = '|', s[++n] = c;
11      s[++n] = '|';
12      for (int i = 1, maxr = 0, mid = 0; i ⩽ n; i++) {
13          if (i < maxr) f[i] = min(f[2 * mid - i], maxr - i);
14          for (int j = i + f[i] + 1; j ⩽ n; j++) {
15              if (s[j] == s[2 * i - j]) ++f[i];
16              else break;
17          }
18          if (i + f[i] > maxr) maxr = i + f[i], mid = i;
19      }
20      int ans = 0;
21      for (int i = 1; i ⩽ n; i++)
22          if (f[i] > ans) ans = f[i];
23      cout << ans;
24      return 0;
25  }
```

2.7 Z 函数

（需要补充）

```
1   #include <cstring>
2   #include <iostream>
3   #include <vector>
4   using namespace std;
5   vector<int> z_function(string s) {
6       int n = (int)s.length();
7       vector<int> z(n);
8       for (int i = 1, l = 0, r = 0; i < n; i++) {
9           if (i ⩽ r && z[i - l] < r - i + 1) z[i] = z[i - l];
```

```
10        else {
11           z[i] = r - i + 1;
12           while (i + z[i] < n && s[i + z[i]] == s[z[i]]) ++z[i];
13        }
14        if (i + z[i] - 1 > r) r = i + z[i] - 1, l = i;
15     }
16     return z;
17  }
18  int main() { return 0; }
```

第 3 部分　数学 Math

3.1　线性代数

3.1.1　矩阵运算

矩阵的基本运算。

```
#include <bits/stdc++.h>
using namespace std;
const int MOD = 1e9 + 7;
template<int sz>struct mat {
   using LL = long long;
   LL a[sz][sz];
   mat() { memset(a, 0, sizeof a); }
   mat operator-(const mat &T) const {
      mat res;
      for (int i = 0; i < sz; ++i)
         for (int j = 0; j < sz; ++j) res.a[i][j] = (a[i][j] - T.a[i][j]) %
             MOD;
      return res;
   }
   mat operator+(const mat &T) const {
      mat res;
      for (int i = 0; i < sz; ++i)
         for (int j = 0; j < sz; ++j) res.a[i][j] = (a[i][j] + T.a[i][j]) %
             MOD;
      return res;
   }
   mat operator*(const mat &T) const {
      mat res;
      int r;
      for (int i = 0; i < sz; ++i)
         for (int k = 0; k < sz; ++k) {
            r = a[i][k];
            for (int j = 0; j < sz; ++j) res.a[i][j] += T.a[k][j] * r, res.
                a[i][j] %= MOD;
         }
      return res;
   }

   mat operator^(LL x) const {
```

```
32        mat res, bas;
33        for (int i = 0; i < sz; ++i) res.a[i][i] = 1;
34        for (int i = 0; i < sz; ++i)
35           for (int j = 0; j < sz; ++j) bas.a[i][j] = a[i][j] % MOD;
36        while (x) {
37           if (x & 1) res = res * bas;
38           bas = bas * bas;
39           x >>= 1;
40        }
41        return res;
42     }
43  };
44  int main() { return 0; }
```

3.2 （需要补充）

3.2.1 （需要补充）

（需要补充）

```
1   #include <cmath>
2   #include <cstring>
3   #include <iostream>
4   using namespace std;
5   constexpr int M = 10005, N = 1005, INF = 1e9;
6   int n, m;
7   double a[M][N], b[M], c[N], v;
8   void pivot(int l, int e) { // 转轴操作函数
9      b[l] /= a[l][e];
10     for (int j = 1; j <= n; j++)
11        if (j != e) a[l][j] /= a[l][e];
12     a[l][e] = 1 / a[l][e];
13
14     for (int i = 1; i <= m; i++)
15        if (i != l && fabs(a[i][e]) > 0) {
16           b[i] -= a[i][e] * b[l];
17           for (int j = 1; j <= n; j++)
18              if (j != e) a[i][j] -= a[i][e] * a[l][j];
19           a[i][e] = -a[i][e] * a[l][e];
20        }
21     v += c[e] * b[l];
22     for (int j = 1; j <= n; j++)
23        if (j != e) c[j] -= c[e] * a[l][j];
24     c[e] = -c[e] * a[l][e];
25  }
26  double simplex() {
27     while (true) {
28        int e = 0, l = 0;
```

```
29          for (e = 1; e ⩽ n; e++)
30             if (c[e] > (double)0) break;
31          if (e == n + 1) return v; // 此时v即为最优解
32          double mn = INF;
33          for (int i = 1; i ⩽ m; i++) {
34             if (a[i][e] > (double)0 && mn > b[i] / a[i][e]) {
35                mn = b[i] / a[i][e]; // 找对这个e限制最紧的l
36                l = i;
37             }
38          }
39          if (mn == INF) return INF; // unbounded
40          pivot(l, e);        // 转动l,e
41       }
42    }
43    int main() {
44       cin.tie(nullptr)->sync_with_stdio(false);
45       cin >> n >> m;
46       for (int i = 1; i ⩽ n; i++) cin >> c[i];
47       for (int i = 1; i ⩽ m; i++) {
48          int s, t;
49          cin >> s >> t;
50          for (int j = s; j ⩽ t; j++) a[i][j] = 1; // 表示第i种志愿者在j时间可以服
               务
51          cin >> b[i];
52       }
53       cout << (int)(simplex() + 0.5);
54    }
```

3.3 数论

3.3.1 欧拉函数

3.3.1.1 求欧拉函数

在 $O(\log n)$ 的时间复杂度内求出一个数的欧拉函数。

```
1  #include <cmath>
2  int euler_phi(int n) {
3     int ans = n;
4     for (int i = 2; i * i ⩽ n; i++)
5        if (n % i == 0) {
6           ans = ans / i * (i - 1);
7           while (n % i == 0) n /= i;
8        }
9     if (n > 1) ans = ans / n * (n - 1);
10    return ans;
11 }
12 int main() { return 0; }
```

3.3.1.2 欧拉筛法

在 $O(n)$ 的时间内求出 $[1,n]$ 内的所有数的欧拉函数值。

```
#include <bitset>
using std::bitset;
const int N = 1e7 + 5;
bitset<N> not_prime;
int prime[N], pcnt, phi[N];
inline void sieve(int n) {
   phi[1] = 1;
   for (int i = 2; i ⩽ n; i++) {
      if (!not_prime[i]) prime[++pcnt] = i, phi[i] = i - 1;
      for (int j = 1; j ⩽ pcnt && i * prime[j] ⩽ n; j++) {
         not_prime[i * prime[j]] = true;
         if (i % prime[j] == 0) {
            phi[i * prime[j]] = phi[i] * prime[j];
            break;
         }
         phi[i * prime[j]] = phi[i] * phi[prime[j]];
      }
   }
}
int main() { return 0; }
```

3.3.2 卢卡斯定理

3.3.2.1 扩展卢卡斯定理

（需要补充）

```
#include <iostream>
#include <vector>

// Extended Euclid.
void ex_gcd(int a, int b, int &x, int &y) {
   if (!b) x = 1, y = 0;
   else ex_gcd(b, a % b, y, x), y -= a / b * x;
}

// Inverse of a mod m.
int inverse(int a, int m) {
   int x, y;
   ex_gcd(a, m, x, y);
   return (x % m + m) % m;
}

// Coefficient in CRT.
int crt_coeff(int m_i, int m) {
   long long mm = m / m_i;
```

```
20     mm *= inverse(mm, m_i);
21     return mm % m;
22   }
23
24   // Binominal Coefficient Calculator Modulo Prime Power.
25   class BinomModPrimePower {
26     int p, a, pa;
27     std::vector<int> f;
28
29     // Obtain multiplicity of p in n!.
30     long long nu(long long n) {
31       long long count = 0;
32       do {
33         n /= p;
34         count += n;
35       } while (n);
36       return count;
37     }
38
39     // Calculate (n!)_p mod pa.
40     long long fact_mod(long long n) {
41       bool neg = p != 2 || pa <= 4;
42       long long res = 1;
43       while (n > 1) {
44         if ((n / pa) & neg) res = pa - res;
45         res = res * f[n % pa] % pa;
46         n /= p;
47       }
48       return res;
49     }
50
51    public:
52     BinomModPrimePower(int p, int a, int pa) : p(p), a(a), pa(pa), f(pa) {
53       // Pretreatment.
54       f[0] = 1;
55       for (int i = 1; i < pa; ++i) { f[i] = i % p ? (long long)f[i - 1] * i
             % pa : f[i - 1]; }
56     }
57
58     // Calculate Binom(n, k) mod pa.
59     int binomial(long long n, long long k) {
60       long long v = nu(n) - nu(n - k) - nu(k);
61       if (v >= a) return 0;
62       auto res = fact_mod(n - k) * fact_mod(k) % pa;
63       res = fact_mod(n) * inverse(res, pa) % pa;
64       for (; v; --v) res *= p;
65       return res % pa;
66     }
67   };
```

```
68
69  // Binominal Coefficient Calculator.
70  class BinomMod {
71     int m;
72     std::vector<BinomModPrimePower> bp;
73     std::vector<long long> crt_m;
74
75   public:
76     BinomMod(int n) : m(n) {
77        // Factorize.
78        for (int p = 2; p * p ⩽ n; ++p) {
79           if (n % p == 0) {
80              int a = 0, pa = 1;
81              for (; n % p == 0; n /= p, ++a, pa *= p);
82              bp.emplace_back(p, a, pa);
83              crt_m.emplace_back(crt_coeff(pa, m));
84           }
85        }
86        if (n > 1) {
87           bp.emplace_back(n, 1, n);
88           crt_m.emplace_back(crt_coeff(n, m));
89        }
90     }
91
92     // Calculate Binom(n, k) mod m.
93     int binomial(long long n, long long k) {
94        long long res = 0;
95        for (size_t i = 0; i ≠ bp.size(); ++i) { res = (bp[i].binomial(n, k)
              * crt_m[i] + res) % m; }
96        return res;
97     }
98  };
99
100 int main() {
101    int t, m;
102    std::cin >> t >> m;
103    BinomMod bm(m);
104    for (; t; --t) {
105       long long n, k;
106       std::cin >> n >> k;
107       std::cout << bm.binomial(n, k) << '\n';
108    }
109    return 0;
110 }
```

3.3.2.2 卢卡斯定理

利用预处理，在模数较小的情况下计算组合值。

```cpp
long long Lucas(long long n, long long k, long long p) {
   if (k == 0) return 1;
   return (C(n % p, k % p, p) * Lucas(n / p, k / p, p)) % p;
}
```

3.3.3 莫比乌斯函数

3.3.3.1 莫比乌斯函数的欧拉筛法

计算在 $[1,n]$ 内所有数的莫比乌斯函数值。

```cpp
#include <bitset>
#include <vector>
using std::bitset;
using std::vector;
using ll = long long;
const int N = 1e7 + 5;
bitset<N> not_prime;
vector<int> pr;
int mu[N];
inline void sieve(int n) {
   mu[1] = 1;
   for (int i = 2; i ⩽ n; i++) {
      if (!not_prime[i]) mu[i] = -1, pr.push_back(i);
      for (int x : pr) {
         if (i * x > n) break;
         not_prime[i * x] = true;
         if (i % x == 0) break;
         mu[i * x] = -mu[i];
      }
   }
}
int main() { return 0; }
```

3.3.4 素数相关

3.3.4.1 Miller-Rabin 素性测试

计算一个数是素数的概率。要想保证在 2^64 内不出错，需要七个数，$2, 325, 9375, 28178, 450775, 9780504, 1795265022$

```cpp
#include <bits/stdc++.h>
using namespace std;
bool millerRabin(int n) {
   if (n < 3 || n % 2 == 0) return n == 2;
   if (n % 3 == 0) return n == 3;
   int u = n - 1, t = 0;
   while (u % 2 == 0) u /= 2, ++t;
   // test_time 为测试次数，建议设为不小于 8
   // 的整数以保证正确率，但也不宜过大，否则会影响效率
   for (int i = 0; i < test_time; ++i) {
```

```
        // 0, 1, n-1 可以直接通过测试, a 取值范围 [2, n-2]
        int a = rand() % (n - 3) + 2, v = quickPow(a, u, n);
        if (v == 1) continue;
        int s;
        for (s = 0; s < t; ++s) {
            if (v == n - 1) break; // 得到平凡平方根 n-1, 通过此轮测试
            v = (long long)v * v % n;
        }
        // 如果找到了非平凡平方根, 则会由于无法提前 break; 而运行到 s == t
        // 如果 Fermat 素性测试无法通过, 则一直运行到 s == t 前 v 都不会等于 -1
        if (s == t) return 0;
    }
    return 1;
}
int main() { return 0; }
```

3.3.4.2 素数的线性筛法

筛所有素数。

```
#include <bitset>
#include <iostream>
#include <vector>
using std::bitset;
using std::vector;
const int N = 1e7 + 5;
bitset<N> not_prime;
vector<int> sieve(int n) {
    vector<int> prime;
    for (int i = 2; i <= n; i++) {
        if (!not_prime[i]) prime.emplace_back(i);
        for (int x : prime) {
            if (x * i > n) break;
            not_prime[x * i] = true;
            if (i % x == 0) break;
        }
    }
    return prime;
}
int main() { return 0; }
```

3.3.5 中国剩余定理

求解线性同余方程组。

```
#include <iostream>
using LL = long long;
LL ex_gcd(LL a, LL b, LL &x, LL &y) {
    if (!b) return x = 1, y = 0, a;
```

```
   LL g = ex_gcd(b, a % b, x, y), t = x;
   x = y, y = t - (a / b) * y;
   return g;
}
LL CRT(int k, LL *a, LL *r) {
   LL n = 1, ans = 0;
   for (int i = 1; i ⩽ k; i++) n = n * r[i];
   for (int i = 1; i ⩽ k; i++) {
      LL m = n / r[i], b, y;
      ex_gcd(m, r[i], b, y); // b * m mod r[i] = 1
      ans = (ans + a[i] * m * b % n) % n;
   }
   return (ans % n + n) % n;
}
int main() { return 0; }
```

3.3.6 杜教筛

在 $O(n^{\frac{2}{3}})$ 的复杂度内求积性函数前缀和。

```
#include <cstdio>
#include <iostream>
#include <map>
#define int long long
using namespace std;
const int N = 2000005;
int p[N], mu[N], sum[N], cnt;
bool flg[N];
map<int, int> mp;
void init(int n) {
   mu[1] = 1ll;
   for (int i = 2; i ⩽ n; i++) {
      if (!flg[i]) p[++cnt] = i, mu[i] = -1;
      for (int j = 1; j ⩽ cnt && i * p[j] ⩽ n; j++) {
         flg[i * p[j]] = true;
         if (i % p[j] == 0) {
            mu[i * p[j]] = 0;
            break;
         }
         mu[i * p[j]] = -mu[i];
      }
   }
   for (int i = 1; i ⩽ n; i++) sum[i] = sum[i - 1] + mu[i];
}
int sum_mu(int n) {
   if (n < N) return sum[n];
   if (mp.find(n) ≠ mp.end()) return mp[n];
   int res = 1ll;
   for (int i = 2, j; i ⩽ n; i = j + 1) {
```

```
30        j = n / (n / i);
31        res -= sum_mu(n / i) * (j - i + 1);
32     }
33     return mp[n] = res;
34  }
35  int sum_phi(int n) {
36     int res = 0ll;
37     for (int i = 1, j; i ⩽ n; i = j + 1) {
38        j = n / (n / i);
39        res += (sum_mu(j) - sum_mu(i - 1)) * (n / i) * (n / i);
40     }
41     return (res - 1) / 2 + 1;
42  }
43  signed main() {
44     cin.tie(0)->sync_with_stdio(false);
45     cout.tie(0);
46     init(N - 1);
47     int T, n;
48     cin >> T;
49     while (T--) {
50        cin >> n;
51        cout << sum_phi(n) << "␣" << sum_mu(n) << "\n";
52     }
53     return 0;
54  }
```

3.3.7 扩展欧几里得算法

求解一元二次不定方程。

```
1  int gcd(int a, int b) { return !b ? a : gcd(b, a % b); }
2  int ex_gcd(int a, int b, int &x, int &y) {
3     if (!b) return x = 1, y = 0, a;
4     int g = ex_gcd(b, a % b, x, y), t = x;
5     x = y, y = t - (a / b) * y;
6     return g;
7  }
8  int main() { return 0; }
```

3.3.8 Pollard-Rho

在相对于暴力下较快地分解质因数的方法。

```
1  #include <algorithm>
2  #include <cstdlib>
3  #include <ctime>
4  #include <iostream>
5  using namespace std;
6  using ll = long long;
```

```
using ull = unsigned long long;
int t;
ll max_factor, n;
ll gcd(ll a, ll b) {
   if (b == 0) return a;
   return gcd(b, a % b);
}
ll bmul(ll a, ll b, ll m) { // 快速乘
   ull c = (ull)a * (ull)b - (ull)((long double)a / m * b + 0.5L) * (ull)m;
   if (c < (ull)m) return c;
   return c + m;
}
ll qpow(ll x, ll p, ll mod) { // 快速幂
   ll ans = 1;
   while (p) {
      if (p & 1) ans = bmul(ans, x, mod);
      x = bmul(x, x, mod);
      p >>= 1;
   }
   return ans;
}
bool Miller_Rabin(ll p) { // 判断素数
   if (p < 2) return false;
   if (p == 2) return true;
   if (p == 3) return true;
   ll d = p - 1, r = 0;
   while (!(d & 1)) ++r, d >>= 1; // 将d处理为奇数
   for (ll k = 0; k < 10; ++k) {
      ll a = rand() % (p - 2) + 2;
      ll x = qpow(a, d, p);
      if (x == 1 || x == p - 1) continue;
      for (int i = 0; i < r - 1; ++i) {
         x = bmul(x, x, p);
         if (x == p - 1) break;
      }
      if (x != p - 1) return false;
   }
   return true;
}
ll Pollard_Rho(ll x) {
   ll s = 0, t = 0;
   ll c = (ll)rand() % (x - 1) + 1;
   int step = 0, goal = 1;
   ll val = 1;
   for (goal = 1;; goal *= 2, s = t, val = 1) { // 倍增优化
      for (step = 1; step <= goal; ++step) {
         t = (bmul(t, t, x) + c) % x;
         val = bmul(val, abs(t - s), x);
         if ((step % 127) == 0) {
```

```
            ll d = gcd(val, x);
            if (d > 1) return d;
         }
      }
      ll d = gcd(val, x);
      if (d > 1) return d;
   }
}
void fac(ll x) {
   if (x ⩽ max_factor || x < 2) return;
   if (Miller_Rabin(x)) {     // 如果x为质数
      max_factor = max(max_factor, x); // 更新答案
      return;
   }
   ll p = x;
   while (p ⩾ x) p = Pollard_Rho(x); // 使用该算法
   while ((x % p) == 0) x /= p;
   fac(x), fac(p); // 继续向下分解x和p
}
int main() {
   cin >> t;
   while (t--) {
      srand((unsigned)time(NULL));
      max_factor = 0;
      cin >> n;
      fac(n);
      if (max_factor == n) cout << "Prime\n";
      else cout << max_factor << '\n';
   }
   return 0;
}
```

3.3.9 数论分块

求 $\sum\lfloor\frac{n}{i}\rfloor$。

```
using ll = long long;
ll get_sum(int l, int r) { return r - l + 1; }
ll calc(int n) {
   ll res = 0;
   for (int l = 1, r = 0; l ⩽ n; l = r + 1) {
      r = n / (n / l);
      res += get_sum(l, r);
   }
   return res;
}
int main() { return 0; }
```

3.4 多项式

3.4.1 快速傅里叶变换

3.4.1.1 快速傅里叶变换

（需要补充）

```
#include <cmath>
#include <cstdio>
#define MAXN 4000005
using namespace std;
int n, m;
const double PI = acos(-1.0);
struct CP {
   CP(double xx = 0, double yy = 0) { x = xx, y = yy; }
   double x, y;
   CP operator+(const CP &B) const { return CP(x + B.x, y + B.y); }
   CP operator-(const CP &B) const { return CP(x - B.x, y - B.y); }
   CP operator*(const CP &B) const { return CP(x * B.x - y * B.y, x * B.y +
       y * B.x); }
} f[MAXN], g[MAXN], sav[MAXN];
void FFT(CP *f, int limit, int type) {
   if (limit == 1) return;
   CP *fl = f, *fr = f + limit / 2;
   for (int i = 0; i < limit; i++) sav[i] = f[i];
   for (int i = 0; i < limit / 2; i++) fl[i] = sav[i << 1], fr[i] = sav[i
      << 1 | 1];
   FFT(fl, limit / 2, type);
   FFT(fr, limit / 2, type);
   CP Omega(cos(2.0 * PI / limit), type * sin(2.0 * PI / limit)), w(1, 0);
   for (int i = 0; i < limit / 2; i++, w = w * Omega)
      sav[i] = fl[i] + fr[i] * w, sav[i + limit / 2] = fl[i] - fr[i] * w;
   for (int i = 0; i < limit; i++) f[i] = sav[i];
}
int main() {
   scanf("%d%d", &n, &m);
   for (int i = 0; i <= n; i++) scanf("%lf", &f[i].x);
   for (int i = 0; i <= m; i++) scanf("%lf", &g[i].x);
   m += n, n = 1;
   while (n <= m) n <<= 1;
   FFT(f, n, 1), FFT(g, n, 1);
   for (int i = 0; i < n; i++) f[i] = f[i] * g[i];
   FFT(f, n, -1);
   for (int i = 0; i <= m; i++) printf("%d␣", (int)(f[i].x / n + 0.5));
   return 0;
}
```

3.4.1.2 快速傅里叶变换

（需要补充）

```
#include <cmath>
#include <cstdio>
#include <iostream>
using namespace std;
#define db double
int read() {
   int x = 0, f = 1;
   char c = getchar();
   while (c < '0' || c > '9') f ^= (c == '-'), c = getchar();
   while (c >= '0' && c <= '9') x = (x << 1) + (x << 3) + (c ^ 48), c =
       getchar();
   return f ? x : -x;
}
void write(int x) {
   if (x < 0) putchar('-'), x = -x;
   if (x > 9) write(x / 10);
   putchar(x % 10 + '0');
}
const db PI = acos(-1.0);
struct CP {
   db re, im;
   CP() { re = im = 0; }
   CP(db x, db y) { re = x, im = y; }
   CP operator+(const CP &b) const { return CP(re + b.re, im + b.im); }
   CP operator-(const CP &b) const { return CP(re - b.re, im - b.im); }
   CP operator*(const CP &b) const { return CP(re * b.re - im * b.im, re *
       b.im + im * b.re); }
};
#define N 3000005
int n, m, rev[N];
CP a[N], b[N];
inline void init(int n) {
   int b = log2(n);
   for (int i = 1; i < n; i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (b
       - 1));
}
inline void FFT(CP *a, int n, int op) {
   for (int i = 0; i < n; i++)
      if (i < rev[i]) swap(a[i], a[rev[i]]);
   for (int len = 1; len <= (n >> 1); len <<= 1) {
      CP wn = CP(cos(PI / len), sin(PI / len) * op);
      for (int i = 0; i < n; i += (len << 1)) {
         CP w = CP(1.0, 0.0);
         for (int j = 0; j < len; j++, w = w * wn) {
            CP a0 = a[i + j], a1 = w * a[i + j + len];
```

```
            a[i + j] = a0 + a1, a[i + j + len] = a0 - a1;
         }
      }
   }
}
inline void IFFT(CP *a, int n) {
   FFT(a, n, -1);
   for (int i = 0; i < n; i++) a[i].re = a[i].re / n + 0.5;
}
int main() {
   n = read(), m = read();
   for (int i = 0; i ⩽ n; i++) a[i].re = read();
   for (int i = 0; i ⩽ m; i++) b[i].re = read();
   int len = 1;
   while (len ⩽ n + m) len <⩽ 1;
   init(len);
   FFT(a, len, 1), FFT(b, len, 1);
   for (int i = 0; i < len; i++) a[i] = a[i] * b[i];
   IFFT(a, len);
   for (int i = 0; i ⩽ n + m; i++) write(a[i].re), putchar('␣');
   return 0;
}
```

3.4.2 快速沃尔什变换

（需要补充）

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/trie_policy.hpp>
#define fi first
#define se second
#define mkp make_pair
#define pb push_back
#define eb emplace_back
#define inf 0x3f3f3f3f
#define INF 0x3f3f3f3f3f3f3f3f
#define gi int, vector<int>, greater<int>
#define IOS cin.tie(0)->sync_with_stdio(false), cout.tie(0)
#define fo(i, begin, end)                                                    \
   for (__typeof(end) i = (begin) - ((begin) > (end)); i ≠ (end) - ((begin
       ) > (end)); i += 1 - 2 * ((begin) > (end)))
using namespace std;
using namespace __gnu_pbds;
using db = double;
```

```
21  using ll = long long;
22  using lint = __int128;
23  using pdd = pair<db, db>;
24  using pll = pair<ll, ll>;
25  using pdl = pair<db, ll>;
26  using pdi = pair<db, int>;
27  using pil = pair<int, ll>;
28  using pii = pair<int, int>;
29  using ull = unsigned ll;
30  using uint = unsigned int;
31  inline lint read() {
32     lint x = 0;
33     bool f = 0;
34     char c = getchar();
35     while (c < '0' || c > '9') f = (c == '-'), c = getchar();
36     while (c ≥ '0' && c ≤ '9') x = (x << 1) + (x << 3) + (c ^ 48), c =
          getchar();
37     return f ? -x : x;
38  }
39  void write(lint x) {
40     if (x < 0) { putchar('-'), x = -x; }
41     if (x > 9) write(x / 10);
42     putchar(x % 10 + '0');
43  }
44  const db eps = 1e-10, PI = acos(-1.0);
45  const ll mod = 998244353, MOD = 1e9 + 7, inv2 = 499122177;
46  const int N = (1 << 17) + 5;
47  const ll Cor[2][2] = {{1, 0}, {1, 1}};
48  const ll ICor[2][2] = {{1, 0}, {mod - 1, 1}};
49  const ll Cand[2][2] = {{1, 1}, {0, 1}};
50  const ll ICand[2][2] = {{1, mod - 1}, {0, 1}};
51  const ll Cxor[2][2] = {{1, 1}, {1, mod - 1}};
52  const ll ICxor[2][2] = {{inv2, inv2}, {inv2, mod - inv2}};
53  inline void FWT(ll *f, const ll c[2][2], int n) {
54     for (int len = 1; len < n; len <≤ 1)
55        for (int p = 0; p < n; p += len + len)
56           for (int i = p; i < p + len; ++i) {
57              int tmp = f[i];
58              f[i] = (c[0][0] * f[i] + c[0][1] * f[i + len]) % mod;
59              f[i + len] = (c[1][0] * tmp + c[1][1] * f[i + len]) % mod;
60           }
61  }
62  inline void mul(ll *f, ll *g, const ll c[2][2], const ll ic[2][2], int n) {
63     FWT(f, c, n), FWT(g, c, n);
64     fo(i, 0, n) f[i] = (f[i] * g[i]) % mod;
65     FWT(f, ic, n);
66  }
67  int n;
68  ll a[N], b[N], f[N], g[N];
```

```
69  inline void work(const ll c[2][2], const ll ic[2][2]) {
70     fo(i, 0, n) f[i] = a[i];
71     fo(i, 0, n) g[i] = b[i];
72     mul(f, g, c, ic, n);
73     fo(i, 0, n) cout << f[i] << '␣';
74     cout << '\n';
75  }
76  int main() {
77     IOS;
78     cin >> n, n = (1 << n);
79     fo(i, 0, n) cin >> a[i];
80     fo(i, 0, n) cin >> b[i];
81     work(Cor, ICor), work(Cand, ICand), work(Cxor, ICxor);
82     return 0;
83  }
```

3.4.3 （需要补充）

（需要补充）

```
1  #include <algorithm>
2  #include <cmath>
3  #include <cstdio>
4  #include <cstdlib>
5  #include <cstring>
6  #include <ctime>
7  using namespace std;
8  int read() {
9     int x = 0, f = 1;
10    char ch = getchar();
11    while (ch < '0' || ch > '9') {
12       if (ch == '-') f = -1;
13       ch = getchar();
14    }
15    while (ch ⩽ '9' && ch ⩾ '0') {
16       x = 10 * x + ch - '0';
17       ch = getchar();
18    }
19    return x * f;
20 }
21 void print(int x) {
22    if (x < 0) putchar('-'), x = -x;
23    if (x ⩾ 10) print(x / 10);
24    putchar(x % 10 + '0');
25 }
26 constexpr int N = 300100, P = 998244353;
27 int qpow(int x, int y) {
28    int res(1);
29    while (y) {
```

```
30        if (y & 1) res = 1ll * res * x % P;
31        x = 1ll * x * x % P;
32        y >>= 1;
33     }
34     return res;
35  }
36  int r[N];
37  void ntt(int *x, int lim, int opt) {
38     int i, j, k, m, gn, g, tmp;
39     for (i = 0; i < lim; ++i)
40        if (r[i] < i) swap(x[i], x[r[i]]);
41     for (m = 2; m <= lim; m <<= 1) {
42        k = m >> 1;
43        gn = qpow(3, (P - 1) / m);
44        for (i = 0; i < lim; i += m) {
45           g = 1;
46           for (j = 0; j < k; ++j, g = 1ll * g * gn % P) {
47              tmp = 1ll * x[i + j + k] * g % P;
48              x[i + j + k] = (x[i + j] - tmp + P) % P;
49              x[i + j] = (x[i + j] + tmp) % P;
50           }
51        }
52     }
53     if (opt == -1) {
54        reverse(x + 1, x + lim);
55        int inv = qpow(lim, P - 2);
56        for (i = 0; i < lim; ++i) x[i] = 1ll * x[i] * inv % P;
57     }
58  }
59  int A[N], B[N], C[N];
60  char a[N], b[N];
61  int main() {
62     int i, lim(1), n;
63     scanf("%s", a);
64     n = strlen(a);
65     for (i = 0; i < n; ++i) A[i] = a[n - i - 1] - '0';
66     while (lim < (n << 1)) lim <<= 1;
67     scanf("%s", b);
68     n = strlen(b);
69     for (i = 0; i < n; ++i) B[i] = b[n - i - 1] - '0';
70     while (lim < (n << 1)) lim <<= 1;
71     for (i = 0; i < lim; ++i) r[i] = (i & 1) * (lim >> 1) + (r[i >> 1] >> 1)
          ;
72     ntt(A, lim, 1);
73     ntt(B, lim, 1);
74     for (i = 0; i < lim; ++i) C[i] = 1ll * A[i] * B[i] % P;
75     ntt(C, lim, -1);
76     int len(0);
77     for (i = 0; i < lim; ++i) {
```

```
78        if (C[i] ⩾ 10) len = i + 1, C[i + 1] += C[i] / 10, C[i] %= 10;
79        if (C[i]) len = max(len, i);
80     }
81     while (C[len] ⩾ 10) C[len + 1] += C[len] / 10, C[len] %= 10, len++;
82     for (i = len; ~i; --i) putchar(C[i] + '0');
83     puts("");
84     return 0;
85  }
```

3.5 高精度计算

对大整数进行计算。

```
1   #include <cstdio>
2   #include <cstring>
3   constexpr int LEN = 1004;
4   int a[LEN], b[LEN], c[LEN], d[LEN];
5   void clear(int a[]) {
6      for (int i = 0; i < LEN; ++i) a[i] = 0;
7   }
8   void read(int a[]) {
9      static char s[LEN + 1];
10     scanf("%s", s);
11     clear(a);
12     int len = strlen(s);
13     for (int i = 0; i < len; ++i) a[len - i - 1] = s[i] - '0';
14  }
15  void print(int a[]) {
16     int i;
17     for (i = LEN - 1; i ⩾ 1; --i)
18        if (a[i] ≠ 0) break;
19     for (; i ⩾ 0; --i) putchar(a[i] + '0');
20     putchar('\n');
21  }
22  void add(int a[], int b[], int c[]) {
23     clear(c);
24     for (int i = 0; i < LEN - 1; ++i) {
25        c[i] += a[i] + b[i];
26        if (c[i] ⩾ 10) {
27           c[i + 1] += 1;
28           c[i] -= 10;
29        }
30     }
31  }
32  void sub(int a[], int b[], int c[]) {
33     clear(c);
34     for (int i = 0; i < LEN - 1; ++i) {
35        c[i] += a[i] - b[i];
```

```
36          if (c[i] < 0) {
37             c[i + 1] -= 1;
38             c[i] += 10;
39          }
40       }
41    }
42    void mul(int a[], int b[], int c[]) {
43       clear(c);
44       for (int i = 0; i < LEN - 1; ++i) {
45          for (int j = 0; j ⩽ i; ++j) c[i] += a[j] * b[i - j];
46          if (c[i] ⩾ 10) {
47             c[i + 1] += c[i] / 10;
48             c[i] %= 10;
49          }
50       }
51    }
52    bool greater_eq(int a[], int b[], int last_dg, int len) {
53       if (a[last_dg + len] ≠ 0) return true;
54       for (int i = len - 1; i ⩾ 0; --i) {
55          if (a[last_dg + i] > b[i]) return true;
56          if (a[last_dg + i] < b[i]) return false;
57       }
58       return true;
59    }
60    void div(int a[], int b[], int c[], int d[]) {
61       clear(c), clear(d);
62       int la, lb;
63       for (la = LEN - 1; la > 0; --la)
64          if (a[la - 1] ≠ 0) break;
65       for (lb = LEN - 1; lb > 0; --lb)
66          if (b[lb - 1] ≠ 0) break;
67       if (lb == 0) {
68          puts(">␣<");
69          return;
70       }
71       for (int i = 0; i < la; ++i) d[i] = a[i];
72       for (int i = la - lb; i ⩾ 0; --i) {
73          while (greater_eq(d, b, i, lb)) {
74             for (int j = 0; j < lb; ++j) {
75                d[i + j] -= b[j];
76                if (d[i + j] < 0) {
77                   d[i + j + 1] -= 1;
78                   d[i + j] += 10;
79                }
80             }
81             c[i] += 1;
82          }
83       }
84    }
```

```
int main() {
   read(a);
   char op[4];
   scanf("%s", op);
   read(b);
   switch (op[0]) {
   case '+': add(a, b, c), print(c); break;
   case '-': sub(a, b, c), print(c); break;
   case '*': mul(a, b, c), print(c); break;
   case '/': div(a, b, c, d), print(c), print(d); break;
   default: puts(">␣<");
   }
   return 0;
}
```

3.6 快速幂

在 $O(\log n)$ 的时间内求出 $a^n \bmod p$ 的值。

```
using ll = long long;
inline ll qpow(ll a, ll b, ll mod) {
   ll res = 1;
   a %= mod;
   if (b) {
      if (b & 1) res = res * a % mod;
      a = a * a % mod, b >≥ 1;
   }
   return res % mod;
}
int main() { return 0; }
```

3.7 高斯消元

求线性方程组。

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <iostream>
using namespace std;
double map[111][111];
double ans[111];
double eps = 1e-7;
int main() {
   int n;
   cin >> n;
   for (int i = 1; i ≤ n; i++)
      for (int j = 1; j ≤ n + 1; j++) scanf("%lf", &map[i][j]);
```

```
14      for (int i = 1; i ⩽ n; i++) {
15          int r = i;
16          for (int j = i + 1; j ⩽ n; j++)
17              if (fabs(map[r][i]) < fabs(map[j][i])) r = j; //
                    find_the_biggest_number_of_the_first_column (at present)
18          if (fabs(map[r][i]) < eps) {
19              printf("No Solution");
20              return 0;
21          }
22          if (i ≠ r)
23              swap(map[i], map[r]); // 对换一行或一列,属于找最大当前系数的其中一步。(这
                    样就可以只处理当前行的系数啦!)
24          double div = map[i][i];
25          for (int j = i; j ⩽ n + 1; j++) map[i][j] /= div;
26          for (int j = i + 1; j ⩽ n; j++) {
27              div = map[j][i];
28              for (int k = i; k ⩽ n + 1; k++) map[j][k] -= map[i][k] * div;
29          }
30      }
31      ans[n] = map[n][n + 1];
32      for (int i = n - 1; i ⩾ 1; i--) {
33          ans[i] = map[i][n + 1];
34          for (int j = i + 1; j ⩽ n; j++) ans[i] -= (map[i][j] * ans[j]);
35      } // 回带操作
36      for (int i = 1; i ⩽ n; i++) printf("%.2lf\n", ans[i]);
37      return 0;
38  }
```

第 4 部分　数据结构 Data Structure

4.1　平衡树

4.1.1　FHQ-Treap

一种无需旋转的平衡树。

```
#include <bits/stdc++.h>
using namespace std;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
const int N = 1e5 + 10;
int n, rt, L, R, p, cnt;
struct node {
   int ls, rs, val, pri, siz;
   node() {}
   node(int _l, int _r, int _v, int _p, int _s) { ls = _l, rs = _r, val =
       _v, pri = _p, siz = _s; }
} t[N];
void pushup(int u) { t[u].siz = t[t[u].ls].siz + t[t[u].rs].siz + 1; }
void newnode(int x) { t[++cnt] = node(0, 0, x, rng(), 1); }
void split(int u, int x, int &L, int &R) {
   if (!u) return L = R = 0, void();
   if (t[u].val <= x) L = u, split(t[u].rs, x, t[u].rs, R);
   else R = u, split(t[u].ls, x, L, t[u].ls);
   pushup(u);
}
int merge(int L, int R) {
   if (!L || !R) return L | R;
   if (t[L].pri <= t[R].pri) return t[L].rs = merge(t[L].rs, R), pushup(L),
       L;
   else return t[R].ls = merge(L, t[R].ls), pushup(R), R;
}
inline void Insert(int x) { newnode(x), split(rt, x, L, R), rt = merge(
    merge(L, cnt), R); }
inline void Delete(int x) {
   split(rt, x - 1, L, R), split(R, x, p, R);
   p = merge(t[p].ls, t[p].rs), rt = merge(merge(L, p), R);
}
inline int get_rank(int x) {
   split(rt, x - 1, L, R);
   int y = t[L].siz + 1;
```

```
32      rt = merge(L, R);
33      return y;
34  }
35  inline int get_val(int u, int k) {
36      int now = u;
37      while (now) {
38          int x = t[t[now].ls].siz + 1;
39          if (x == k) return t[now].val;
40          if (x < k) now = t[now].rs, k -= x;
41          else now = t[now].ls;
42      }
43  }
44  inline int get_pre(int x) {
45      split(rt, x - 1, L, R);
46      int y = get_val(L, t[L].siz);
47      rt = merge(L, R);
48      return y;
49  }
50  inline int get_nxt(int x) {
51      split(rt, x, L, R);
52      int y = get_val(R, 1);
53      rt = merge(L, R);
54      return y;
55  }
56  signed main() { return 0; }
```

4.1.2 Splay

带旋平衡树。主要用于 LCT。

```
1   const int N = 1e5 + 5;
2   int rt, siz[N], ch[N][2], val[N], tot, fa[N], cnt[N];
3   void pushup(int p) { siz[p] = siz[ch[p][0]] + siz[ch[p][1]] + cnt[p]; }
4   bool get(int p) { return p == ch[fa[p]][1]; }
5   void clear(int p) { siz[p] = ch[p][0] = ch[p][1] = val[p] = fa[p] = cnt[p]
        = 0; }
6   void rotate(int p) {
7       int f = fa[p], g = fa[f], chk = get(p);
8       ch[f][chk] = ch[p][chk ^ 1];
9       if (ch[p][chk ^ 1]) fa[ch[p][chk ^ 1]] = f;
10      ch[p][chk ^ 1] = f, fa[f] = p, fa[p] = g;
11      if (g) ch[g][f == ch[g][1]] = p;
12      pushup(f), pushup(p);
13  }
14  void splay(int p) {
15      for (int f = fa[p]; f = fa[p], f; rotate(p))
16          if (fa[f]) rotate(get(p) == get(f) ? f : p);
17      rt = p;
18  }
```

```
19 void nw(int k, int fath, int op) { val[++tot] = k, siz[tot] = cnt[tot] = 1,
       fa[tot] = fath, ch[fath][op] = tot; }
20 void ins(int k) {
21    if (!rt) {
22       val[++tot] = k, cnt[tot]++, rt = tot;
23       return pushup(rt);
24    }
25    int cur = rt, f = 0;
26    while (1) {
27       if (val[cur] == k) {
28          cnt[cur]++;
29          pushup(cur), pushup(f), splay(cur);
30          break;
31       }
32       f = cur, cur = ch[cur][val[cur] < k];
33       if (!cur) {
34          val[++tot] = k;
35          cnt[tot]++, fa[tot] = f, ch[f][val[f] < k] = tot;
36          pushup(tot), pushup(f), splay(tot);
37          break;
38       }
39    }
40 }
41 int rk(int k) {
42    int p = rt, res = 0;
43    while (true) {
44       if (k < val[p]) p = ch[p][0];
45       else {
46          res += siz[ch[p][0]];
47          if (!p) return res + 1;
48          if (val[p] == k) return splay(p), res + 1;
49          res += cnt[p], p = ch[p][1];
50       }
51    }
52 }
53 int prev() {
54    int p = ch[rt][0];
55    if (!p) return p;
56    while (ch[p][1]) p = ch[p][1];
57    return splay(p), p;
58 }
59 int next() {
60    int p = ch[rt][1];
61    if (!p) return p;
62    while (ch[p][0]) p = ch[p][0];
63    return splay(p), p;
64 }
65 void del(int k) {
66    rk(k);
```

```
67     if (cnt[rt] > 1) return --cnt[rt], pushup(rt), void();
68     if (!ch[rt][0] && !ch[rt][1]) return clear(rt), rt = 0, void();
69     if (!ch[rt][0]) {
70        int cur = rt;
71        rt = ch[rt][1], fa[rt] = 0;
72        clear(cur);
73        return;
74     }
75     if (!ch[rt][1]) {
76        int cur = rt;
77        rt = ch[rt][0], fa[rt] = 0;
78        clear(cur);
79        return;
80     }
81     int cur = rt, x = prev();
82     fa[ch[cur][1]] = x, ch[x][1] = ch[cur][1];
83     clear(cur), pushup(rt);
84  }
85  int kth(int k) {
86     int p = rt;
87     while (1) {
88        if (ch[p][0] && k ⩽ siz[ch[p][0]]) p = ch[p][0];
89        else {
90           k -= cnt[p] + siz[ch[p][0]];
91           if (k ⩽ 0) return splay(p), val[p];
92           p = ch[p][1];
93        }
94     }
95  }
```

4.2 动态树

4.2.1 Link-Cut Tree

（需要补充）

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 3e5 + 5;
4  int n, m, fa[N], ch[N][2], val[N], tag[N], sum[N];
5  inline int get(int p) { return ch[fa[p]][1] == p; }
6  inline bool is_root(int p) { return ch[fa[p]][0] ≠ p && ch[fa[p]][1] ≠ p;
       }
7  inline void pushup(int p) { sum[p] = sum[ch[p][0]] ^ val[p] ^ sum[ch[p
       ][1]]; }
8  inline void reverse(int p) { swap(ch[p][0], ch[p][1]), tag[p] ^= 1; }
9  inline void pushdown(int p) {
10    if (!tag[p]) return;
```

```
11     reverse(ch[p][0]), reverse(ch[p][1]), tag[p] = 0;
12  }
13  inline void push(int p) {
14     if (!is_root(p)) push(fa[p]);
15     pushdown(p);
16  }
17  inline void rotate(int p) {
18     int f = fa[p], g = fa[f], k = get(p);
19     if (!is_root(f)) ch[g][get(f)] = p;
20     ch[f][k] = ch[p][k ^ 1], fa[ch[p][k ^ 1]] = f;
21     ch[p][k ^ 1] = f, fa[f] = p, fa[p] = g;
22     pushup(f), pushup(p);
23  }
24  void splay(int p) {
25     push(p);
26     for (int f; f = fa[p], !is_root(p); rotate(p))
27        if (!is_root(f)) rotate(get(p) == get(f) ? f : p);
28  }
29  inline void access(int p) {
30     for (int child = 0; p; child = p, p = fa[p]) splay(p), ch[p][1] = child,
           pushup(p);
31  }
32  inline void makeroot(int p) { access(p), splay(p), reverse(p); }
33  inline void split(int u, int v) { makeroot(u), access(v), splay(v); }
34  inline int findroot(int p) {
35     access(p), splay(p), pushdown(p);
36     while (ch[p][0]) p = ch[p][0], pushdown(p);
37     return splay(p), p;
38  }
39  inline void link(int u, int v) { makeroot(u), fa[u] = v; }
40  inline void cut(int u, int v) {
41     makeroot(u);
42     if (findroot(v) == u && fa[v] == u && !ch[v][0]) fa[v] = ch[u][1] = 0,
          pushup(u);
43  }
44  int main() {
45     cin.tie(0)->sync_with_stdio(false);
46     cout.tie(0);
47     cin >> n >> m;
48     for (int i = 1; i ⩽ n; i++) cin >> val[i];
49     for (int op, x, y; m--;) {
50        cin >> op >> x >> y;
51        switch (op) {
52        case 0: split(x, y), cout << sum[y] << '\n'; break;
53        case 1:
54           if (findroot(x) ≠ findroot(y)) link(x, y);
55           break;
56        case 2: cut(x, y); break;
57        case 3: splay(x), val[x] = y; break;
```

```
58        }
59     }
60     return 0;
61  }
```

4.3 树状数组

4.3.1 树状数组 1

单点修改，区间查询的树状数组。

```
1  #include <iostream>
2  using namespace std;
3  const int N = 5e5 + 10;
4  int n, m, a[N];
5  struct BIT {
6     int tr[N];
7     inline int lowbit(int x) { return x & -x; }
8     inline void upd(int x, int c) {
9        while (x ⩽ n) tr[x] += c, x += lowbit(x);
10    }
11    inline int que(int x) {
12       int res = 0;
13       while (x) res += tr[x], x -= lowbit(x);
14       return res;
15    }
16 } tree;
17 int main() {
18    ios::sync_with_stdio(false);
19    cin.tie(0);
20    cout.tie(0);
21    cin >> n >> m;
22    for (int i = 1; i ⩽ n; i++) cin >> a[i], tree.upd(i, a[i]);
23    while (m--) {
24       int op, x, y;
25       cin >> op >> x >> y;
26       if (op == 1) tree.upd(x, y);
27       else cout << tree.que(y) - tree.que(x - 1) << "\n";
28    }
29    return 0;
30 }
```

4.3.2 树状数组 2

区间修改，单点查询的树状数组。

```
1  #include <iostream>
2  using namespace std;
```

```
const int N = 500005;
int n, m, a[N], s[N], t[N];
inline int lowbit(int s) { return s & -s; }
inline void add(int x, int k) {
   for (; x ⩽ n; x += lowbit(x)) t[x] += k;
}
inline int query(int x) {
   int res = 0;
   for (; x; x -= lowbit(x)) res += t[x];
   return res;
}
int main() {
   ios::sync_with_stdio(false);
   cin.tie(0);
   cout.tie(0);
   cin >> n >> m;
   for (int i = 1; i ⩽ n; i++) cin >> a[i];
   while (m--) {
      int op, x, y, k;
      cin >> op >> x;
      if (op == 1) {
         cin >> y >> k;
         add(x, k);
         add(y + 1, -k);
      } else cout << a[x] + query(x) << "\n";
   }
   return 0;
}
```

4.4 线段树

4.4.1 可持久化线段树

4.4.1.1 可持久化线段树 1

支持单点修改，区间查询，版本回滚的可持久化线段树。

```
#include <cstdio>
using namespace std;
const int N = 1e6 + 5;
int n, m, a[N], tot;
int root[N], cnt;
struct node {
   int ls, rs;
   int val;
} tree[N * 30];
int build(int pl, int pr) {
   int rt = tot++;
   if (pl == pr) {
```

```
13          tree[rt].val = a[pl];
14          return rt;
15       }
16       int mid = (pl + pr) >> 1;
17       tree[rt].ls = build(pl, mid);
18       tree[rt].rs = build(mid + 1, pr);
19       return rt;
20    }
21    int update(int pre, int pl, int pr, int loc, int val) {
22       int rt = tot++;
23       if (pl == pr) {
24          tree[rt].val = val;
25          return rt;
26       }
27       int mid = (pl + pr) >> 1;
28       tree[rt].ls = tree[pre].ls;
29       tree[rt].rs = tree[pre].rs;
30       if (loc <= mid) tree[rt].ls = update(tree[pre].ls, pl, mid, loc, val);
31       else tree[rt].rs = update(tree[rt].rs, mid + 1, pr, loc, val);
32       return rt;
33    }
34    int query(int p, int pl, int pr, int loc) {
35       if (pl == pr) return tree[p].val;
36       int mid = (pl + pr) >> 1;
37       if (loc <= mid) return query(tree[p].ls, pl, mid, loc);
38       else return query(tree[p].rs, mid + 1, pr, loc);
39    }
40    int main() {
41       scanf("%d%d", &n, &m);
42       for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
43       root[cnt++] = build(1, n);
44       while (m--) {
45          int v, op, loc, val;
46          scanf("%d%d%d", &v, &op, &loc);
47          if (op == 1) {
48             scanf("%d", &val);
49             root[cnt++] = update(root[v], 1, n, loc, val);
50          } else {
51             root[cnt++] = root[v];
52             int res = query(root[v], 1, n, loc);
53             printf("%d\n", res);
54          }
55       }
56       return 0;
57    }
```

4.4.1.2 可持久化线段树 2

区间第 k 小问题。

```cpp
#include <algorithm>
#include <cstdio>
#include <iostream>
#define endl '\n'
using std::cin;
using std::cout;
using std::lower_bound;
using std::sort;
using std::unique;
const int N = 2e5 + 5;
int n, m, a[N], b[N], size, tot;
int root[N], vcnt;
struct node {
   int l, r;
   int sum;
} tree[N * 30];
int build(int pl, int pr) {
   int rt = ++tot;
   tree[rt].sum = 0;
   int mid = (pl + pr) >> 1;
   if (pl < pr) {
      tree[rt].l = build(pl, mid);
      tree[rt].r = build(mid + 1, pr);
   }
   return rt;
}
int update(int pre, int pl, int pr, int x) {
   int rt = ++tot;
   tree[rt].l = tree[pre].l;
   tree[rt].r = tree[pre].r;
   tree[rt].sum = tree[pre].sum + 1;
   int mid = (pl + pr) >> 1;
   if (pl < pr) {
      if (x ⩽ mid) tree[rt].l = update(tree[pre].l, pl, mid, x);
      else tree[rt].r = update(tree[pre].r, mid + 1, pr, x);
   }
   return rt;
}
int query(int u, int v, int pl, int pr, int k) {
   if (pl == pr) return pl;
   int x = tree[tree[v].l].sum - tree[tree[u].l].sum;
   int mid = (pl + pr) >> 1;
   if (x ⩾ k) return query(tree[u].l, tree[v].l, pl, mid, k);
   else return query(tree[u].r, tree[v].r, mid + 1, pr, k - x);
}
int main() {
   cin >> n >> m;
   for (int i = 1; i ⩽ n; i++) {
```

```
49        cin >> a[i];
50        b[i] = a[i];
51     }
52     sort(b + 1, b + 1 + n);
53     size = unique(b + 1, b + 1 + n) - (b + 1);
54     root[0] = build(1, size);
55     for (int i = 1; i ⩽ n; i++) {
56        int x = lower_bound(b + 1, b + 1 + size, a[i]) - b;
57        root[i] = update(root[i - 1], 1, size, x);
58     }
59     while (m--) {
60        int l, r, k;
61        cin >> l >> r >> k;
62        int res = query(root[l - 1], root[r], 1, size, k);
63        cout << b[res] << endl;
64     }
65     return 0;
66  }
```

4.4.2 李超线段树

在一个平面上添加和修改线段，查询某个坐标位置的最高/最低线段。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   using ll = long long;
4   using pdi = pair<double, int>;
5   constexpr int N = 4e4 + 5, M1 = 39989, M2 = 1e9;
6   constexpr double eps = 1e-9;
7   int comp(double a, double b) {
8      if (a - b > eps) return 1;
9      if (b - a > eps) return -1;
10     return 0;
11  }
12  struct LiChaoT {
13     int cnt, t[N << 2];
14     struct line {
15        double k, b;
16        double operator()(int x) { return k * x + b; }
17     } seg[N << 2];
18     void add(int x1, int y1, int x2, int y2) {
19        double k, b;
20        if (x2 == x1) k = 0, b = max(y1, y2);
21        else k = 1.0 * (y2 - y1) / (x2 - x1), b = y1 - x1 * k;
22        seg[++cnt] = {k, b};
23     }
24     void update(int rt, int L, int R, int x) {
25        int &v = t[rt];
26        int mid = (L + R) / 2;
```

```
27          if (comp(seg[v](mid), seg[x](mid)) < 0) swap(v, x);
28          if (comp(seg[v](L), seg[x](L)) < 0) update(rt << 1, L, mid, x);
29          if (comp(seg[v](R), seg[x](R)) < 0) update(rt << 1 | 1, mid + 1, R, x
                );
30       }
31       void insert(int rt, int L, int R, int l, int r, int x) {
32          if (l <= L && R <= r) return update(rt, L, R, x);
33          int mid = (L + R) / 2;
34          if (l <= mid) insert(rt << 1, L, mid, l, r, x);
35          if (r > mid) insert(rt << 1 | 1, mid + 1, R, l, r, x);
36       }
37       pdi mx(pdi a, pdi b) {
38          int c = comp(a.first, b.first);
39          if (c > 0) return a;
40          if (c < 0) return b;
41          return a.second < b.second ? a : b;
42       }
43       pdi query(int rt, int L, int R, int x) {
44          if (x < L || x > R) return {0, 0};
45          pdi ret = {seg[t[rt]](x), t[rt]};
46          if (L == R) return ret;
47          int mid = (L + R) / 2;
48          return mx(ret, mx(query(rt << 1, L, mid, x), query(rt << 1 | 1, mid +
                1, R, x)));
49       }
50    } t;
51    int main() {
52       cin.tie(nullptr), cout.tie(nullptr);
53       ios::sync_with_stdio(0);
54       int n, last = 0;
55       cin >> n;
56       for (int i = 1; i <= n; i++) {
57          int op, k, x, y, xx, yy;
58          cin >> op;
59          if (op == 0) {
60             cin >> k;
61             k = (k + last - 1) % M1 + 1;
62             cout << (last = t.query(1, 0, 40000, k).second) << '\n';
63          } else {
64             cin >> x >> y >> xx >> yy;
65             x = (x + last - 1) % M1 + 1, xx = (xx + last - 1) % M1 + 1;
66             y = (y + last - 1) % M2 + 1, yy = (yy + last - 1) % M2 + 1;
67             t.add(x, y, xx, yy), t.insert(1, 0, 40000, min(x, xx), max(x, xx),
                    t.cnt);
68          }
69       }
70       return 0;
71    }
```

4.4.3 线段树 1

支持区间修改，区间查询。

```
#include <iostream>
using namespace std;
using ll = long long;
namespace segment_tree {
#define ls (p << 1)
#define rs (p << 1 | 1)
#define mid (pl + pr >> 1)
const int N = 1e6 + 5;
int sum[N], tag[N];
inline void up(int p) { sum[p] = sum[ls] + sum[rs]; }
inline void spread(int p, int pl, int pr) {
   if (!tag[p]) return;
   tag[ls] += tag[p], tag[rs] += tag[p];
   sum[ls] += tag[p] * (mid - pl + 1), sum[rs] += tag[p] * (pr - mid);
   tag[p] = 0;
}
void build(int p, int pl, int pr, int *a) {
   if (pl == pr) return sum[p] = a[pl], void();
   build(ls, pl, mid, a), build(rs, mid + 1, pr, a), up(p);
}
void update(int p, int pl, int pr, int L, int R, int v) {
   if (L <= pl && pr <= R) return sum[p] += v, void();
   spread(p, pl, pr);
   if (L <= mid) update(ls, pl, mid, L, R, v);
   if (R > mid) update(rs, mid + 1, pr, L, R, v);
   up(p);
}
int query(int p, int pl, int pr, int L, int R) {
   if (L <= pl && pr <= R) return sum[p];
   spread(p, pl, pr);
   int res(0);
   if (L <= mid) res += query(ls, pl, mid, L, R);
   if (R > mid) res += query(rs, mid + 1, pr, L, R);
   return res;
}
}; // namespace segment_tree
int main() { return 0; }
```

4.4.4 线段树 1

支持区间修改，区间查询。支持区间乘。

```
#include <cstdio>
#include <iostream>
#define int long long
```

```
4  using namespace std;
5  int n, m, mod, a[101000];
6  struct node {
7     int l, r, sum;
8     int add, mul;
9  } t[401000];
10 void build(int p, int l, int r) {
11    t[p].l = l, t[p].r = r;
12    t[p].mul = 1, t[p].add = 0;
13    if (l == r) {
14       t[p].sum = a[l];
15       return;
16    }
17    int mid = (l + r) / 2;
18    build(p * 2, l, mid);
19    build(p * 2 + 1, mid + 1, r);
20    t[p].sum = t[p * 2].sum + t[p * 2 + 1].sum;
21 }
22 void push_down(int p) {
23    if (t[p].mul != 1) {
24       t[p * 2].mul *= t[p].mul, t[p * 2].mul %= mod;
25       t[p * 2 + 1].mul *= t[p].mul, t[p * 2 + 1].mul %= mod;
26       t[p * 2].sum *= t[p].mul, t[p * 2].sum %= mod;
27       t[p * 2 + 1].sum *= t[p].mul, t[p * 2 + 1].sum %= mod;
28       t[p * 2].add *= t[p].mul, t[p * 2].add %= mod;
29       t[p * 2 + 1].add *= t[p].mul, t[p * 2 + 1].add %= mod;
30       t[p].mul = 1;
31    }
32    if (t[p].add != 0) {
33       t[p * 2].add += t[p].add, t[p * 2].add %= mod;
34       t[p * 2 + 1].add += t[p].add, t[p * 2 + 1].add %= mod;
35       t[p * 2].sum += t[p].add * (t[p * 2].r - t[p * 2].l + 1), t[p * 2].
             sum %= mod;
36       t[p * 2 + 1].sum += t[p].add * (t[p * 2 + 1].r - t[p * 2 + 1].l + 1),
              t[p * 2 + 1].sum %= mod;
37       t[p].add = 0;
38    }
39 }
40 void mulnum(int p, int dat, int l, int r) {
41    if (l <= t[p].l && r >= t[p].r) {
42       t[p].mul *= dat, t[p].mul %= mod;
43       t[p].sum *= dat, t[p].sum %= mod;
44       t[p].add *= dat, t[p].add %= mod;
45       return;
46    }
47    push_down(p);
48    int mid = (t[p].l + t[p].r) / 2;
49    if (l <= mid) mulnum(p * 2, dat, l, r);
50    if (r > mid) mulnum(p * 2 + 1, dat, l, r);
```

```
51    t[p].sum = t[p * 2].sum + t[p * 2 + 1].sum;
52 }
53 void addnum(int p, int dat, int l, int r) {
54    if (l ⩽ t[p].l && r ⩾ t[p].r) {
55       t[p].add += dat, t[p].add %= mod;
56       t[p].sum += dat * (t[p].r - t[p].l + 1), t[p].sum %= mod;
57       return;
58    }
59    push_down(p);
60    int mid = (t[p].l + t[p].r) / 2;
61    if (l ⩽ mid) addnum(p * 2, dat, l, r);
62    if (r > mid) addnum(p * 2 + 1, dat, l, r);
63    t[p].sum = t[p * 2].sum + t[p * 2 + 1].sum;
64 }
65 int query(int p, int l, int r) {
66    if (l ⩽ t[p].l && r ⩾ t[p].r) return t[p].sum % mod;
67    push_down(p);
68    int mid = (t[p].l + t[p].r) / 2, val = 0;
69    if (l ⩽ mid) val += query(p * 2, l, r);
70    if (r > mid) val += query(p * 2 + 1, l, r);
71    return val % mod;
72 }
73 signed main() {
74    cin >> n >> m >> mod;
75    for (int i = 1; i ⩽ n; i++) cin >> a[i];
76    build(1, 1, n);
77    while (m--) {
78       int op, x, y, k;
79       cin >> op >> x >> y;
80       if (op == 1) {
81          cin >> k;
82          mulnum(1, k, x, y);
83       } else if (op == 2) {
84          cin >> k;
85          addnum(1, k, x, y);
86       } else cout << query(1, x, y) << endl;
87    }
88    return 0;
89 }
```

4.4.5 线段树合并

线段树合并可以做什么呢？它可以把两棵线段树合并起来。

```
1 #include <cstdio>
2 #include <iostream>
3 using namespace std;
4 const int N = 100005, M = 100000;
5 int n, m, head[N], tot, ans[N];
```

```
6  struct edge {
7     int v, nxt;
8  } e[N << 1];
9  inline void add(int u, int v) {
10    e[++tot].v = v;
11    e[tot].nxt = head[u];
12    head[u] = tot;
13 }
14 int fa[N], dep[N], siz[N], hson[N], dfn[N], top[N], Time;
15 void dfs1(int u, int fath) {
16    fa[u] = fath, dep[u] = dep[fath] + 1, siz[u] = 1;
17    for (int i = head[u]; i; i = e[i].nxt) {
18       int v = e[i].v;
19       if (v == fath) continue;
20       dfs1(v, u);
21       siz[u] += siz[v];
22       if (!hson[u] || siz[v] > siz[hson[u]]) hson[u] = v;
23    }
24 }
25 void dfs2(int u, int tp) {
26    top[u] = tp, dfn[u] = ++Time;
27    if (hson[u]) dfs2(hson[u], tp);
28    for (int i = head[u]; i; i = e[i].nxt)
29       if (e[i].v != fa[u] && e[i].v != hson[u]) dfs2(e[i].v, e[i].v);
30 }
31 inline int LCA(int x, int y) {
32    while (top[x] != top[y]) {
33       if (dep[top[x]] < dep[top[y]]) swap(x, y);
34       x = fa[top[x]];
35    }
36    return dep[x] <= dep[y] ? x : y;
37 }
38 int cnt = 0, rt[N];
39 struct node {
40    int mx, id;
41    int ls, rs;
42 } t[N * 100];
43 void pushup(int p) {
44    if (t[t[p].ls].mx >= t[t[p].rs].mx) t[p].mx = t[t[p].ls].mx, t[p].id = t
       [t[p].ls].id;
45    else t[p].mx = t[t[p].rs].mx, t[p].id = t[t[p].rs].id;
46 }
47 void update(int &p, int pl, int pr, int k, int val) {
48    if (!p) p = ++cnt;
49    if (pl == pr) {
50       t[p].mx += val, t[p].id = k;
51       return;
52    }
53    int mid = (pl + pr) >> 1;
```

```
54      if (k ⩽ mid) update(t[p].ls, pl, mid, k, val);
55      else update(t[p].rs, mid + 1, pr, k, val);
56      pushup(p);
57   }
58   int merge(int &L, int &R, int pl, int pr) {
59      if (!L) return R;
60      if (!R) return L;
61      if (pl == pr) {
62         t[L].mx += t[R].mx;
63         return L;
64      }
65      int mid = (pl + pr) >> 1;
66      t[L].ls = merge(t[L].ls, t[R].ls, pl, mid);
67      t[L].rs = merge(t[L].rs, t[R].rs, mid + 1, pr);
68      pushup(L);
69      return L;
70   }
71   void dfs(int u) {
72      for (int i = head[u]; i; i = e[i].nxt) {
73         int v = e[i].v;
74         if (v == fa[u]) continue;
75         dfs(v);
76         rt[u] = merge(rt[u], rt[v], 1, M);
77      }
78      ans[u] = t[rt[u]].id;
79      if (!t[rt[u]].mx) ans[u] = 0;
80   }
81   int main() {
82      ios::sync_with_stdio(false);
83      cin.tie(0);
84      cout.tie(0);
85      cin >> n >> m;
86      for (int i = 1; i < n; i++) {
87         int u, v;
88         cin >> u >> v;
89         add(u, v);
90         add(v, u);
91      }
92      dfs1(1, 0);
93      dfs2(1, 1);
94      while (m--) {
95         int l, r, z;
96         cin >> l >> r >> z;
97         update(rt[l], 1, M, z, 1);
98         update(rt[r], 1, M, z, 1);
99         int lca = LCA(l, r);
100        update(rt[lca], 1, M, z, -1);
101        update(rt[fa[lca]], 1, M, z, -1);
102     }
```

```
103     dfs(1);
104     for (int i = 1; i ⩽ n; i++) cout << ans[i] << "\n";
105     return 0;
106 }
```

4.4.6 线段树分裂

（需要补充）

```
1   #include <cstdio>
2   #include <iostream>
3   #define int long long
4   using namespace std;
5   const int N = 2e5 + 5;
6   int n, m, root[N << 2], rcnt = 1, rub[N << 5], cnt, nodecnt;
7   struct treenode {
8       int ls, rs, sum;
9   } tr[N << 5];
10  inline int newnode() { return cnt ? rub[cnt--] : ++nodecnt; }
11  inline void del(int &p) { tr[p].ls = tr[p].rs = tr[p].sum = 0, rub[++cnt] =
        p, p = 0; }
12  inline void push_up(int p) { tr[p].sum = tr[tr[p].ls].sum + tr[tr[p].rs].
        sum; }
13  inline void build(int &p, int pl, int pr) {
14      if (!p) p = newnode();
15      if (pl == pr) return cin >> tr[p].sum, void();
16      int mid = (pl + pr) >> 1;
17      build(tr[p].ls, pl, mid), build(tr[p].rs, mid + 1, pr);
18      push_up(p);
19  }
20  inline void update(int &p, int pl, int pr, int pos, int k) {
21      if (!p) p = newnode();
22      if (pl == pr) return tr[p].sum += k, void();
23      int mid = (pl + pr) >> 1;
24      if (pos ⩽ mid) update(tr[p].ls, pl, mid, pos, k);
25      else update(tr[p].rs, mid + 1, pr, pos, k);
26      push_up(p);
27  }
28  inline int merge(int p, int q, int pl, int pr) {
29      if (!p || !q) return p + q;
30      if (pl == pr) return tr[p].sum += tr[q].sum, del(q), p;
31      int mid = (pl + pr) >> 1;
32      tr[p].ls = merge(tr[p].ls, tr[q].ls, pl, mid);
33      tr[p].rs = merge(tr[p].rs, tr[q].rs, mid + 1, pr);
34      return push_up(p), del(q), p;
35  }
36  inline void split(int &p, int &q, int pl, int pr, int L, int R) {
37      if (R < pl || L > pr) return;
38      if (!p) return;
```

```
39      if (L ⩽ pl && pr ⩽ R) return q = p, p = 0, void();
40      int mid = (pl + pr) >> 1;
41      if (!q) q = newnode();
42      if (L ⩽ mid) split(tr[p].ls, tr[q].ls, pl, mid, L, R);
43      if (R > mid) split(tr[p].rs, tr[q].rs, mid + 1, pr, L, R);
44      push_up(p), push_up(q);
45  }
46  inline int query(int p, int pl, int pr, int L, int R) {
47      if (!p) return 0;
48      if (L ⩽ pl && pr ⩽ R) return tr[p].sum;
49      int mid = (pl + pr) >> 1, res = 0;
50      if (L ⩽ mid) res += query(tr[p].ls, pl, mid, L, R);
51      if (R > mid) res += query(tr[p].rs, mid + 1, pr, L, R);
52      return res;
53  }
54  inline int kth(int p, int pl, int pr, int k) {
55      if (pl == pr) return pl;
56      int mid = (pl + pr) >> 1, left = tr[tr[p].ls].sum;
57      if (k ⩽ left) return kth(tr[p].ls, pl, mid, k);
58      else return kth(tr[p].rs, mid + 1, pr, k - left);
59  }
60  signed main() {
61      cin.tie(0)->sync_with_stdio(false);
62      cout.tie(0);
63      cin >> n >> m;
64      build(root[1], 1, n);
65      for (int i = 1, op, p, x, y; i ⩽ m; i++) {
66          cin >> op >> p >> x;
67          if (op == 0 || op == 2 || op == 3) cin >> y;
68          if (op == 0) split(root[p], root[++rcnt], 1, n, x, y);
69          else if (op == 1) root[p] = merge(root[p], root[x], 1, n);
70          else if (op == 2) update(root[p], 1, n, y, x);
71          else if (op == 3) cout << query(root[p], 1, n, x, y) << "\n";
72          else if (op == 4) {
73              if (tr[root[p]].sum < x) cout << "-1\n";
74              else cout << kth(root[p], 1, n, x) << "\n";
75          }
76      }
77      return 0;
78  }
```

4.5 并查集

（需要补充，我不理解下面两个函数）

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   struct dsu {
```

```
4     vector<size_t> pa, size;
5     explicit dsu(size_t size_) : pa(size_), size(size_, 1) { iota(pa.begin()
          , pa.end(), 0); }
6     inline size_t find(int x) { return pa[x] == x ? x : pa[x] = find(pa[x]);
           }
7     void unite(size_t x, size_t y) {
8        x = find(x), y = find(y);
9        if (x == y) return;
10       if (size[x] < size[y]) swap(x, y);
11       pa[y] = x;
12       size[x] += size[y];
13    }
14    void erase(size_t x) { --size[find(x)], pa[x] = x; }
15    void move(size_t x, size_t y) {
16       auto fx = find(x), fy = find(y);
17       if (fx == fy) return;
18       pa[x] = fy;
19       --size[fx], ++size[fy];
20    }
21 };
```

4.6 ST 表

在 $O(n \log n)$ 的时间复杂度预处理，并且在 $O(1)$ 的时间内查询。

```
1  #include <cmath>
2  #include <cstdio>
3  #include <iostream>
4  using namespace std;
5  int n, m, a[1010000], st[1010000][200];
6  int main() {
7     cin >> n >> m;
8     for (int i = 1; i <= n; i++) scanf("%d", a + i), st[i][i] = a[i];
9     for (int j = 1; 1 + (1 << j) <= n; j++)
10       for (int i = 1; i + (1 << j) - 1 <= n; i++) st[i][j] = max(st[i][j -
             1], st[i + (1 << j - 1)][j - 1]);
11    while (m--) {
12       int l, r, ans;
13       scanf("%d%d", &l, &r);
14       int t = log2(r - l + 1);
15       ans = max(st[l][t], st[r - (1 << t) + 1][t]);
16       printf("%d\n", ans);
17    }
18    return 0;
19 }
```

第 5 部分　杂项 Misc

5.1　离线算法

5.1.1　莫队

5.1.1.1　莫队

静态查询且离线下在 $O(n\sqrt{n})$ 的复杂度内求解一类区间问题。

```
void move(int pos, int sign) { ... }
int unit;
struct node {
   int l, r, id;
   bool operator<(const node &x) const {
      return l / unit == x.l / unit ? (r == x.r ? 0 : ((l / unit) & 1) ^ (r
          < x.r)) : l < x.l;
   }
} querys[N];
void solve() {
   unit = int(ceil(pow(n, 0.5)));
   sort(querys, querys + m);
   for (int i = 0; i < m; ++i) {
      const node &q = querys[i];
      while (l > q.l) move(--l, 1);
      while (r < q.r) move(++r, 1);
      while (l < q.l) move(l++, -1);
      while (r > q.r) move(r--, -1);
      ans[q.id] = nowAns;
   }
}
```

5.1.1.2　（需要补充）

（需要补充）

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
inline int rd() {
   int x = 0, f = 1;
   char c = getchar();
```

```
7     while (c < '0' || c > '9') f ^= (c == '-'), c = getchar();
8     while (c >= '0' && c <= '9') x = (x << 1) + (x << 3) + (c ^ 48), c =
          getchar();
9     return f ? x : -x;
10 }
11 inline void wr(ll x) {
12    if (x < 0) putchar('-'), x = -x;
13    if (x > 9) wr(x / 10);
14    putchar(x % 10 + '0');
15 }
16 const int N = 1e5 + 5;
17 int n, m, a[N], b[N], V, tot;
18 int st[N], ed[N], pos[N], B;
19 struct que {
20    int l, r, id;
21    inline bool operator<(const que &b) const { return (l / tot == b.l / tot
          ) ? r < b.r : l < b.l; }
22 } q[N];
23 ll ans[N];
24 struct ask {
25    int id, op, l, r;
26    ask() { id = op = l = r = 0; }
27    ask(int _id, int _op, int _l, int _r) { id = _id, op = _op, l = _l, r =
          _r; }
28 };
29 ll tr[N];
30 inline void add(int x, int v) {
31    for (; x <= n; x += x & -x) tr[x] += (ll)v;
32 }
33 inline ll qry(int x) {
34    ll res = 0;
35    for (; x; x -= x & -x) res += tr[x];
36    return res;
37 }
38 int le[N], ri[N];
39 ll sl[N], sr[N];
40 vector<ask> v1[N], v2[N];
41 inline void addl(int pos, int op, int l, int r, int id) { v1[pos].push_back
       (ask(id, op, l, r)); }
42 inline void addr(int pos, int op, int l, int r, int id) { v2[pos].push_back
       (ask(id, op, l, r)); }
43 ll val[N], tag[N];
44 inline void updl(int x) {
45    if (tag[pos[x]])
46       for (int i = st[pos[x]]; i <= ed[pos[x]]; i++) val[i] += tag[pos[x]];
47    tag[pos[x]] = 0;
48    for (int i = st[pos[x]]; i <= x; i++) val[i]++;
49    for (int i = 1; i < pos[x]; i++) tag[i]++;
50 }
```

```
51  inline void updr(int x) {
52     if (tag[pos[x]])
53        for (int i = st[pos[x]]; i ⩽ ed[pos[x]]; i++) val[i] += tag[pos[x]];
54     tag[pos[x]] = 0;
55     for (int i = x; i ⩽ ed[pos[x]]; i++) val[i]++;
56     for (int i = pos[x] + 1; i ⩽ B; i++) tag[i]++;
57  }
58  signed main() {
59     n = rd(), m = rd(), tot = 355, sl[0] = sr[n + 1] = 0;
60     for (int i = 1; i ⩽ n; i++) b[i] = a[i] = rd();
61     sort(b + 1, b + 1 + n);
62     V = unique(b + 1, b + 1 + n) - (b + 1);
63     for (int i = 1; i ⩽ n; i++) a[i] = lower_bound(b + 1, b + 1 + V, a[i])
           - b;
64     for (int i = 1; i ⩽ n; i++) le[i] += (i - 1 - qry(a[i])), add(a[i], 1),
           sl[i] = sl[i - 1] + le[i];
65     for (int i = 0; i ⩽ n; i++) tr[i] = 0;
66     for (int i = n; i ⩾ 1; i--) ri[i] += qry(a[i] - 1), add(a[i], 1), sr[i]
           = sr[i + 1] + ri[i];
67     for (int i = 1; i ⩽ m; i++) q[i].l = rd(), q[i].r = rd(), q[i].id = i;
68     sort(q + 1, q + 1 + m);
69     int l = 1, r = 0;
70     for (int i = 1; i ⩽ m; i++) {
71        if (r < q[i].r) ans[q[i].id] += sl[q[i].r] - sl[r], addl(l, -1, r +
              1, q[i].r, q[i].id), r = q[i].r;
72        if (r > q[i].r) ans[q[i].id] -= sl[r] - sl[q[i].r], addl(l, 1, q[i].r
               + 1, r, q[i].id), r = q[i].r;
73        if (l < q[i].l) ans[q[i].id] -= sr[l] - sr[q[i].l], addr(r, 1, l, q[i
              ].l - 1, q[i].id), l = q[i].l;
74        if (l > q[i].l) ans[q[i].id] += sr[q[i].l] - sr[l], addr(r, -1, q[i].
              l, l - 1, q[i].id), l = q[i].l;
75     }
76     B = sqrt(V);
77     for (int i = 1; i ⩽ B; i++) st[i] = (i - 1) * B + 1, ed[i] = i * B;
78     if (ed[B] < V) ed[B] = V;
79     for (int i = 1; i ⩽ B; i++)
80        for (int j = st[i]; j ⩽ ed[i]; j++) pos[j] = i;
81     for (int i = 1; i ⩽ n; i++) {
82        for (ask now : v1[i])
83           for (int j = now.l; j ⩽ now.r; j++) ans[now.id] += (ll)now.op * (
                 tag[pos[a[j] + 1]] + val[a[j] + 1]);
84        updl(a[i]);
85     }
86     for (int i = 0; i ⩽ n; i++) tag[i] = val[i] = 0;
87     for (int i = n; i ⩾ 1; i--) {
88        for (ask now : v2[i])
89           for (int j = now.l; j ⩽ now.r; j++) ans[now.id] += (ll)now.op * (
                 tag[pos[a[j] - 1]] + val[a[j] - 1]);
90        updr(a[i]);
```

```
91      }
92      for (int i = 1; i ⩽ m; i++) ans[q[i].id] += ans[q[i - 1].id];
93      for (int i = 1; i ⩽ m; i++) wr(ans[i]), puts("");
94      return 0;
95  }
```

5.2 珂朵莉树

一个题只要数据随机，那它就一定不是正经题。——aaa12321

```
1   #include <algorithm>
2   #include <cstdio>
3   #include <iostream>
4   #include <set>
5   #include <vector>
6   #define int long long
7   using namespace std;
8   const int N = 1e5 + 5, mod = 1e9 + 7;
9   int n, m, seed, vmax, a[N];
10  int rnd() {
11      int ret = seed;
12      seed = (seed * 7 + 13) % mod;
13      return ret;
14  }
15  int qpow(int a, int b, int Mod) {
16      long long res = 1;
17      a %= Mod;
18      while (b) {
19          if (b & 1) res = res * a % Mod;
20          a = a * a % Mod, b >⩾ 1;
21      }
22      return res;
23  }
24  struct node {
25      int l, r;
26      mutable int v;
27      node(const int &ll, const int &rr, const int &vv) : l(ll), r(rr), v(vv)
            {}
28      bool operator<(const node &b) const { return l < b.l; }
29  };
30  set<node> st;
31  typedef set<node>::iterator it;
32  it split(int x) {
33      if (x > n) return st.end();
34      it I = st.lower_bound((node){x, 0, 0});
35      if (I ≠ st.end() && I->l == x) return I;
36      I--;
37      if (I->r < x) return st.end();
```

```
38      int l = I->l, r = I->r, v = I->v;
39      st.erase(I);
40      st.insert(node(l, x - 1, v));
41      return st.insert(node(x, r, v)).first;
42   }
43   void assign(int l, int r, int v) {
44      it itr = split(r + 1), itl = split(l);
45      st.erase(itl, itr);
46      st.insert(node(l, r, v));
47   }
48   vector<pair<int, int>> tmp;
49   signed main() {
50      cin.tie(0)->sync_with_stdio(false);
51      cout.tie(0);
52      cin >> n >> m >> seed >> vmax;
53      for (int i = 1; i ⩽ n; i++) {
54         a[i] = rnd() % vmax + 1;
55         st.insert(node(i, i, a[i]));
56      }
57      for (int at = 1, op, l, r, x, y, ans; at ⩽ m; at++) {
58         op = (rnd() % 4) + 1;
59         l = (rnd() % n) + 1;
60         r = (rnd() % n) + 1;
61         if (l > r) swap(l, r);
62         if (op == 1) {
63            x = rnd() % vmax + 1;
64            it itr = split(r + 1), itl = split(l);
65            for (; itl ≠ itr; ++itl) itl->v += x;
66         } else if (op == 2) {
67            x = rnd() % vmax + 1;
68            assign(l, r, x);
69         } else if (op == 3) {
70            x = rnd() % (r - l + 1) + 1;
71            tmp.clear();
72            it itr = split(r + 1), itl = split(l);
73            for (it I = itl; I ≠ itr; ++I) tmp.push_back({I->v, I->r - I->l +
                  1});
74            sort(tmp.begin(), tmp.end());
75            ans = -1;
76            for (int i = 0; i < tmp.size(); i++) {
77               x -= tmp[i].second;
78               if (x ⩽ 0) {
79                  ans = tmp[i].first;
80                  break;
81               }
82            }
83            cout << ans << '\n';
84         } else {
85            ans = 0;
```

```
86            x = rnd() % vmax + 1;
87            y = rnd() % vmax + 1;
88            it itr = split(r + 1), itl = split(l);
89            for (it I = itl; I =/= itr; ++I) ans = (ans + qpow(I->v, x, y) * (I
                  ->r - I->l + 1) % y) % y;
90            cout << ans << '\n';
91        }
92    }
93    return 0;
94 }
```