ICPC-ACM Template

Ruoyisius

2023 年 10 月 19 日

目录

一、 图论

1.1 增广路

```cpp
struct augment_path {
    vector<vector<int> > g;
    vector<int> pa; // 匹配
    vector<int> pb;
    vector<int> vis; // 访问
    int n, m;        // 两个点集中的顶点数量
    int dfn;         // 时间戳记
    int res;         // 匹配数

    augment_path(int _n, int _m) : n(_n), m(_m) {
        assert(0 <= n && 0 <= m);
        pa = vector<int>(n, -1);
        pb = vector<int>(m, -1);
        vis = vector<int>(n);
        g.resize(n);
        res = 0;
        dfn = 0;
    }

    void add(int from, int to) {
        assert(0 <= from && from < n && 0 <= to && to < m);
        g[from].push_back(to);
    }

    bool dfs(int v) {
        vis[v] = dfn;
        for (int u : g[v]) {
            if (pb[u] == -1) {
                pb[u] = v;
                pa[v] = u;
                return true;
            }
        }
        for (int u : g[v]) {
            if (vis[pb[u]] != dfn && dfs(pb[u])) {
                pa[v] = u;
                pb[u] = v;
                return true;
            }
        }
        return false;
    }
    int solve() {
        while (true) {
            dfn++;
            int cnt = 0;
            for (int i = 0; i < n; i++) {
```

```
48            if (pa[i] == -1 && dfs(i)) {
49                cnt++;
50            }
51        }
52        if (cnt == 0){
53            break;
54        }
55        res += cnt;
56    }
57    return res;
58    }
59 };
```

1.2 SPFA

```
1  /*
2   * Args:
3   *   g[]: graph, (u, v, w) = (u, g[u][i].first, g[u][i].second)
4   *   st: source vertex
5   * Return:
6   *   dis[]: distance from source vertex to each other vertex
7   */
8  vector<pair<int, int> > g[N];
9  int dis[N], vis[N];
10 void spfa(int st)
11 {
12   memset(dis, -1, sizeof(dis));
13   memset(vis, 0, sizeof(vis));
14   queue<int> q;
15   q.push(st);
16   dis[st] = 0;
17   vis[st] = true;
18   while (!q.empty()) {
19     int u = q.front();
20     q.pop();
21     vis[u] = false;
22     for (auto x : g[u]) {
23       int v = x.first, w = x.second;
24       if (dis[v] == -1 || dis[u] + w < dis[v]) {
25         dis[v] = dis[u] + w;
26         if (!vis[v]) {
27           vis[v] = true;
28           q.push(v);
29         }
30       }
31     }
32   }
33 }
```

1.3 Tarjan

```cpp
vector<int> dfn(n, -1), low(n, -1), be(n, -1);
int tot = 0, cnt = 0;
vector<int> st;
function<void(int)> tarjan = [&](int cur) {
    dfn[cur] = low[cur] = tot++;
    st.push_back(cur);
    for (auto &nex : g[cur]) {
        if (dfn[nex] == -1) {
            tarjan(nex);
            low[cur] = min(low[cur], low[nex]);
        } else if (be[nex] == -1) {
            low[cur] = min(low[cur], dfn[nex]);
        }
    }
    if (dfn[cur] == low[cur]) {
        int v;
        do {
            v = st.back();
            st.pop_back();
            be[v] = cnt;
        } while (v != cur);
        cnt++;
    }
};

for (int i = 0; i < n; i++) {
    if (dfn[i] == -1) {
        tarjan(i);
    }
}
```

1.4 基环树

```cpp
vector<vector<int>> E(n);
vector<int> deg(n);
for (int j = 0; j < n; j++) {
    deg[j] = E[j].size();
}
queue<int> Q;
for (int j = 0; j < n; j++) {
    if (deg[j] == 1) {
        Q.push(j);
    }
}
vector<bool> used(n, false);
while (!Q.empty()) {
```

```
14      int v = Q.front();
15      Q.pop();
16      used[v] = true;
17      for (int w : E[v]) {
18          deg[w]--;
19          if (deg[w] == 1) {
20              Q.push(w);
21          }
22      }
23  }
24  vector<int> p(n, -1);
25  vector<int> d(n, -1);
26  for (int j = 0; j < n; j++) {
27      if (!used[j]) {
28          d[j] = 0;
29          Q.push(j);
30      }
31  }
32  while (!Q.empty()) {
33      int v = Q.front();
34      Q.pop();
35      for (int w : E[v]) {
36          if (d[w] == -1) {
37              d[w] = d[v] + 1;
38              p[w] = v;
39              Q.push(w);
40          }
41      }
42  }
```

1.5   2-SAT

```
1   void solve() {
2       cin >> n >> m;
3       vector<vector<int>> G(n * 2 + 1, vector<int>(0));
4       vector<int> dfn(n * 2 + 1), vis(n * 2 + 1), st(n * 2 + 1),
5                       low(n * 2 + 1), scc(n * 2 + 1);
6       int tim = 0, top = 0, scc_num = 0;
7   function<void(int)> tarjan = [&](int u) {
8       dfn[u] = low[u] = ++tim;
9       st[++top] = u;
10      vis[u] = 1;
11      for (auto v : G[u]) {
12          if (!dfn[v]) {
13              tarjan(v);
14              low[u] = min(low[u], low[v]);
15          } else if (vis[v]) {
16              low[u] = min(low[u], dfn[v]);
17          }
18      }
```

```cpp
        if (low[u] == dfn[u]) {
            scc_num++;
            while (1) {
                int x = st[top--];
                vis[x] = 0;
                scc[x] = scc_num;
                if (x == u) break;
            }
        }
    };
    for (int i = 1; i <= m; i++) {
            int a, b;
            cin >> x >> a >> y >> b;
            if(a == 0 && b == 0) {
                    G[x+n].pb(y);
                    G[y+n].pb(x);
            } else if (a == 1 && b == 0) {
                    G[x].pb(y);
                    G[y+n].pb(x+n);
            } else if (a == 0 && b == 1) {
                    G[x+n].pb(y+n);
                    G[y].pb(x);
            } else {
                    G[x].pb(y+n);
                    G[y].pb(x+n);
            }
        }
    for(int i = 1; i <= 2 * n; i++) {
            if(!dfn[i]) tarjan(i);
    }
    for(int i = 1; i <= n; i++) {
            if(scc[i] == scc[i + n]) {
                    cout << "IMPOSSIBLE" << endl;
                    return;
            }
    }
    cout << "POSSIBLE" << endl;
    for(int i = 1; i <= n; i++) {
            if(scc[i] > scc[i+n]) cout << "1 ";
            else cout << "0 ";
    }
}
```

二、 网络流

2.1 Dinic

```cpp
struct Edge {
    int from, to, cap, flow;
};
struct Dinic {
    int s, t;
    bool vis[N];
    int d[N];
    int cur[N];

    vector<Edge> edges;
    vector<int> G[N];
    void AddEdge(int from, int to, int cap) {
        edges.push_back({from, to, cap, 0});
        edges.push_back({to, from, 0, 0});
        //edges.push_back({to, from, cap, 0}); 如果是无向图.
        G[from].push_back(edges.size() - 2);
        G[to].push_back(edges.size() - 1);
    }

    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<int> Q;
        Q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int i = 0; i < G[u].size(); i++) {
                Edge& e = edges[G[u][i]];
                int v = e.to;
                if (!vis[v] && e.cap > e.flow) {
                    vis[v] = 1;
                    d[v] = d[u] + 1;
                    Q.push(v);
                }
            }
        }
        return vis[t];
    }

    int DFS(int u, int a) {
        if (u == t || a == 0) return a;
        int flow = 0, f;
        for (int& i = cur[u]; i < G[u].size(); i++) { //这里取引用，使得u的当前弧
                被i改变，再次访问到u时，将跳过u已经访问过的支路
            Edge& e = edges[G[u][i]], ee = edges[G[u][i] ^ 1];
            int v = e.to;
```

```
47          if (d[v] == d[u] + 1 && (f = DFS(v, min(a, e.cap - e.flow))) > 0) {
48              e.flow += f;
49              ee.flow -= f;
50              flow += f;
51              a -= f;
52              if (a == 0) break;
53          }
54      }
55      return flow;
56  }
57
58  int Maxflow(int s, int t) {
59      this->s = s; this->t = t;
60      int flow = 0;
61      while (BFS()) {
62          memset(cur, 0, sizeof(cur));
63          flow += DFS(s, INF);
64      }
65      return flow;
66  }
67  };
68
69  #include<bits/stdc++.h>
70  using namespace std;
71  typedef long long ll;
72  #define int long long
73  const int N = 1e5;
74  const int INF = 1e17;
75  struct MF {
76      struct edge {
77          int v, nxt, cap, flow;
78      } e[N];
79
80      int fir[N], cnt = 0;
81
82      int n, S, T;
83      ll maxflow = 0;
84      int dep[N], cur[N];
85
86      void init() {
87          memset(fir, -1, sizeof fir);
88          cnt = 0;
89      }
90
91      void addedge(int u, int v, int w) {
92          e[cnt] = { v, fir[u], w, 0 };
93          fir[u] = cnt++;
94          e[cnt] = { u, fir[v], 0, 0 };
95          fir[v] = cnt++;
96      }
97
98      bool bfs() {
```

```
 99          queue<int> q;
100          memset(dep, 0, sizeof(int) * (n + 1));
101
102          dep[S] = 1;
103          q.push(S);
104          while (q.size()) {
105              int u = q.front();
106              q.pop();
107              for (int i = fir[u]; ~i; i = e[i].nxt) {
108                  int v = e[i].v;
109                  if ((!dep[v]) && (e[i].cap > e[i].flow)) {
110                      dep[v] = dep[u] + 1;
111                      q.push(v);
112                  }
113              }
114          }
115          return dep[T];
116      }
117
118      int dfs(int u, int flow) {
119          if ((u == T) || (!flow)) return flow;
120
121          int ret = 0;
122          for (int& i = cur[u]; ~i; i = e[i].nxt) {
123              int v = e[i].v, d;
124              if ((dep[v] == dep[u] + 1) &&
125                  (d = dfs(v, min(flow - ret, e[i].cap - e[i].flow)))) {
126                  ret += d;
127                  e[i].flow += d;
128                  e[i ^ 1].flow -= d;
129                  if (ret == flow) return ret;
130              }
131          }
132          return ret;
133      }
134
135      void dinic() {
136          while (bfs()) {
137              memcpy(cur, fir, sizeof(int) * (n + 1));
138              maxflow += dfs(S, INF);
139          }
140      }
141  } mf;
```

三、 数据结构

3.1 并查集

```cpp
struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

3.2 树状数组

```cpp
template <typename T>
```

```cpp
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n = 0) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }

    T sum(int x) {
        auto ans = T();
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }
};
```

3.3 线段树

```cpp
#include<bits/stdc++.h>
using i64=long long;
template<class Info>
struct SegmentTree {
```

```cpp
     int n;
     std::vector<Info> info;
     SegmentTree() : n(0) {}
     SegmentTree(int n_, Info v_ = Info()) {
         init(n_, v_);
     }
     template<class T>
     SegmentTree(std::vector<T> init_) {
         init(init_);
     }
     void init(int n_, Info v_ = Info()) {
         init(std::vector(n_, v_));
     }
     template<class T>
     void init(std::vector<T> init_) {
         n = init_.size();
         info.assign(4 << std::__lg(n), Info());
         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
             if (r - l == 1) {
                 info[p] = init_[l];
                 return;
             }
             int m = (l + r) / 2;
             build(2 * p, l, m);
             build(2 * p + 1, m, r);
             pull(p);
         };
         build(1, 0, n);
     }
     void pull(int p) {
         info[p] = info[2 * p] + info[2 * p + 1];
     }
     void modify(int p, int l, int r, int x, const Info &v) {
         if (r - l == 1) {
             info[p] = v;
             return;
         }
         int m = (l + r) / 2;
         if (x < m) {
             modify(2 * p, l, m, x, v);
         } else {
             modify(2 * p + 1, m, r, x, v);
         }
         pull(p);
     }
     void modify(int p, const Info &v) {
         modify(1, 0, n, p, v);
     }
     Info rangeQuery(int p, int l, int r, int x, int y) {
         if (l >= y || r <= x) {
             return Info();
         }
```

```cpp
            if (l >= x && r <= y) {
                return info[p];
            }
            int m = (l + r) / 2;
            return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y
                );
        }
        Info rangeQuery(int l, int r) {
            return rangeQuery(1, 0, n, l, r);
        }
        template<class F>
        int findFirst(int p, int l, int r, int x, int y, F pred) {
            if (l >= y || r <= x || !pred(info[p])) {
                return -1;
            }
            if (r - l == 1) {
                return l;
            }
            int m = (l + r) / 2;
            int res = findFirst(2 * p, l, m, x, y, pred);
            if (res == -1) {
                res = findFirst(2 * p + 1, m, r, x, y, pred);
            }
            return res;
        }
        template<class F>
        int findFirst(int l, int r, F pred) {
            return findFirst(1, 0, n, l, r, pred);
        }
        template<class F>
        int findLast(int p, int l, int r, int x, int y, F pred) {
            if (l >= y || r <= x || !pred(info[p])) {
                return -1;
            }
            if (r - l == 1) {
                return l;
            }
            int m = (l + r) / 2;
            int res = findLast(2 * p + 1, m, r, x, y, pred);
            if (res == -1) {
                res = findLast(2 * p, l, m, x, y, pred);
            }
            return res;
        }
        template<class F>
        int findLast(int l, int r, F pred) {
            return findLast(1, 0, n, l, r, pred);
        }
};

constexpr i64 inf = 1E18;
```

```
108  struct Info {
109      i64 cnt = 0;
110      i64 sum = 0;
111      i64 min = inf;
112  };
113
114  Info operator+(Info a, Info b) {
115      Info c;
116      c.cnt = a.cnt + b.cnt;
117      c.sum = a.sum + b.sum;
118      c.min = std::min(a.min, b.min);
119      return c;
120  }
```

3.4 线段树-数组实现

```
1   struct segmentTree{
2       int mx[N<<2], num[N<<2], sum[N<<2];
3       void update(int id) {
4           int ls = id << 1;
5           int rs = (id << 1) | 1;
6           num[id] = num[ls] + num[rs];
7           sum[id] = sum[ls] + sum[rs];
8           mx[id] = max(mx[rs], mx[ls]);
9           return;
10      }
11      void build(int l, int r, int id) {
12          if (l == r) {
13              mx[id] = a[l];
14              num[id] = a[l] < 100;
15              sum[id] = a[l];
16              return;
17          }
18          int mid = (l + r) >> 1;
19          build(l, mid, id << 1);
20          build(mid + 1, r, (id << 1) | 1);
21          update(id);
22          return;
23      }
24      void change(int x, int y, int l, int r,int id) {
25          if (mx[id] < 10) return;
26          if (l == r) {
27              a[l] = a[l] * 2 / 3;
28              mx[id] = a[l];
29              sum[id] = a[l];
30              num[id] = a[l] < 100;
31              return;
32          }
33          int m = l + r >> 1;
```

```
34          if (x <= m)
35              change(x, y, l, m, id << 1);
36          if (y > m)
37              change(x, y, m + 1, r, id << 1 | 1);
38          update(id);
39      }
40      int querynum(int x, int y, int l, int r, int id) {
41          if(x <= l && r <= y)
42              return num[id];
43          int m = l + r >> 1;
44          if (x <= m && y > m)
45              return querynum(x, y, l, m, id << 1) + querynum(x, y, m +1, r, id <<
                    1 | 1);
46          else if (x <= m)
47              return querynum(x, y, l, m, id << 1);
48          else
49              return querynum(x, y, m + 1, r, id << 1 | 1);
50      }
51      int querysum(int x, int y, int l, int r, int id) {
52          if (x <= l && r <= y)
53              return sum[id];
54          int m = l + r >> 1;
55          if (x <= m && y > m)
56              return querysum(x, y, l, m, id << 1) + querysum(x, y, m + 1, r, id
                    << 1 | 1);
57          else if (x <= m)
58              return querysum(x, y, l, m, id << 1);
59          else
60              return querysum(x, y, m + 1, r, id << 1 | 1);
61      }
62  }T;
63  signed main(void)
64  {
65      std::ios::sync_with_stdio(false);
66      std::cin.tie(nullptr);
67      cin >> n >> m;
68      for (int i = 1; i <= n; i++)
69          cin >> a[i];
70
71      T.build(1, n, 1);
72      for (int i = 1; i <= m; i++) {
73          int op, l, r; cin >> op >> l >> r;
74          if (op == 1) {
75              T.change(l, r, 1, n, 1);
76          } else if (op == 2) {
77              cout << T.querynum(l, r, 1, n, 1) << endl;
78          } else {
79              cout << T.querysum(l, r, 1, n, 1) << endl;
80          }
81      }
82      return 0;
83  }
```

```
84
85
86
87   struct SegmentTree
88   {
89       struct node
90       {
91           ll sum,lz;
92       }st[N<<2];
93       void pushup(int id)
94       {
95           st[id].sum=(st[id<<1].sum+st[id<<1|1].sum);
96       }
97       void pushdown(int id,int lsonlen,int rsonlen)
98       {
99           if (!st[id].lz)
100              return;
101          st[id<<1].lz+=st[id].lz;
102          st[id<<1|1].lz+=st[id].lz;
103          st[id<<1].sum+=st[id].lz*lsonlen;
104          st[id<<1|1].sum+=st[id].lz*rsonlen;
105          st[id].lz=0;
106      }
107      void build(int id,int l,int r)
108      {
109          if (l ==r)
110          {
111              st[id]={a[l],0};
112              return;
113          }
114          int mid=(l + r)/2;
115          build(id << 1,l,mid);
116          build(id<<1|1,mid + 1,r);
117          pushup(id);
118      }
119      void update(int id,int segl,int segr,int l,int r,ll val)
120      {
121          if (l <=segl && segr <=r)
122          {
123              st[id].sum+=val*(segr-segl+1);
124              st[id].lz+=val;
125              return;
126          }
127          int mid=(segl + segr)/2;
128          pushdown(id,mid-segl+1,segr-mid);
129          if (l <=mid)
130              update(id << 1,segl,mid,l,r,val);
131          if (r > mid)
132              update(id<<1|1,mid+1,segr,l,r,val);
133          pushup(id);
134      }
135      ll query(int id,int segl,int segr,int l,int r)
```

```
136        {
137            if (l <=segl && segr <=r)
138                return st[id].sum;
139            int mid=(segl + segr)/2;
140            pushdown(id,mid-segl+1,segr-mid);
141            ll res=0;
142            if (l <=mid)
143                res+=query(id << 1,segl,mid,l,r);
144            if (r > mid)
145                res+=query(id<<1|1,mid+1,segr,l,r);
146            return res;
147        }
148    }st;
149    st.build(1,1,n);
150    st.update(1,1,n,l,r,val);
151    st.query(1,1,n,l,r);
```

3.5 懒标记线段树

```
1    const int N = 1e5 + 5;
2    int n, m, a[N], t[N<<2], x, y, w, lazy[N<<2];
3    void build(int rt, int l, int r) {
4        if (l == r) { t[rt] = a[l]; return; }
5        int mid = (l + r) >> 1;
6        build(rt << 1, l, mid);
7        build(rt << 1 | 1, mid + 1, r);
8        t[rt] = t[rt<<1] + t[rt<<1|1];
9    }
10   void updata(int rt, int l, int r, int w) {
11       t[rt] += (r - l + 1) * w;
12       lazy[rt] += w;
13   }
14   void pushdown(int rt, int l, int r) {
15       int mid = (l + r) >> 1;
16       updata(rt << 1, l, mid, lazy[rt]);
17       updata(rt << 1 | 1, mid + 1, r, lazy[rt]);
18       lazy[rt] = 0;
19   }
20   void add(int rt, int l, int r) {
21       if (x <= l && r <= y) {
22           updata(rt, l, r, w);
23           return;
24       }
25       pushdown(rt, l, r);
26       int mid = (l + r) >> 1;
27       if (x <= mid) add(rt << 1, l, mid);
28       if (y > mid) add(rt << 1 | 1, mid + 1, r);
29       t[rt] = t[rt<<1] + t[rt<<1|1];
30   }
```

```
31   int sum(int rt, int l, int r) {
32       if (x <= l && r <= y) return t[rt];
33       int mid = (l + r) >> 1, ans = 0;
34       pushdown(rt, l, r);
35       if (x <= mid) ans += sum(rt << 1, l, mid);
36       if (y > mid) ans += sum(rt << 1 | 1, mid + 1, r);
37       return ans;
38   }
```

3.6 倍增 LCA-最近公共祖先

```
1    // bfs version
2    void bfs()
3    {
4        dep[1]=1;
5        que[tail++]=1;
6        while(head<tail)
7        {
8            int p=que[head++];
9            for(int x=last[p];x!=0;x=pre[x])
10           {
11               if(f[p][0]!=son[x])
12               {
13                   dep[son[x]]=dep[p]+1;
14                   f[son[x]][0]=p;
15                   que[tail++]=son[x];
16                   for(int j=1;j<=20;j++)
17                       f[son[x]][j]=f[f[son[x]][j-1]][j-1];
18               }
19           }
20       }
21   }
22   int lca(int a,int b)
23   {
24       if(dep[a]>dep[b]) swap(a,b);
25       for(int i=20;i>=0;i--)
26       {
27           if(dep[f[b][i]]>=dep[a]) b=f[b][i];
28           if(a==b) return a;
29       }
30       for(int i=20;i>=0;i--)
31           if(f[a][i]!=f[b][i]) a=f[a][i],b=f[b][i];
32       return f[a][0];
33   }
34   // dfs version
35   void dfs(int sn, int fa, int d) {
36       pa[sn][0] = fa;
37       for (int i = 1; i < MAXP; i++) pa[sn][i] = pa[pa[sn][i - 1]][i - 1];
38       dep[sn] = d;
```

```
39        vec[d].push_back(sn);
40        for (int fn : e[sn]) if (fn != fa) dfs(fn, sn, d + 1);
41    }
42
43    int lca(int x, int y) {
44        if (dep[x] < dep[y]) swap(x, y);
45        for (int i = MAXP - 1; i >= 0; i--) if (dep[pa[x][i]] >= dep[y]) x = pa[x][
              i];
46        if (x == y) return x;
47        for (int i = MAXP - 1; i >= 0; i--) if (pa[x][i] != pa[y][i]) x = pa[x][i],
              y = pa[y][i];
48        return pa[x][0];
49    }
```

3.7 珂朵莉树

```
1     struct node {
2         int l, r;
3         mutable int v;
4         node(int a = 0, int b = 0, int c = 0) :l(a), r(b), v(c) { };
5         friend bool operator<(const node& a, const node& b) {
6             return a.l < b.l;
7         }
8     };
9
10    class ODT :public set<node> {
11    public:
12        ODT() :st(*this) { };
13        ODT(int l, int r, int v) :set<node>({ node(l,r,v) }), st(*this) { };
14        set<node>& st;
15
16        set<node>::iterator split(int pos) {
17            auto it = --st.upper_bound(node{ pos,0,0 });
18            if (it->l == pos) return it;
19            int l = it->l, r = it->r, v = it->v;
20            st.erase(it);
21            st.insert(node(l, pos - 1, v));
22            return st.insert(node(pos, r, v)).first;
23        }
24
25        void assign(int l, int r, int v) {
26            auto itr = split(r + 1), itl = split(l);
27            st.erase(itl, itr);
28            st.insert(node(l, r, v));
29        }
30    };
```

3.8 ST

```
template <typename T>
class SparseTable {
  using VT = vector<T>;
  using VVT = vector<VT>;
  using func_type = function<T(const T &, const T &)>;

  VVT ST;
  VT A;

  static T default_func(const T &t1, const T &t2) { return max(t1, t2); }

  func_type op;

 public:
  SparseTable(const vector<T> &v, func_type _func = default_func) {
    op = _func;
    A = v;
    int len = v.size(), l1 = ceil(log2(len)) + 1;
    ST.assign(len, VT(l1, 0));
    for (int i = 0; i < len; ++i) {
      ST[i][0] = v[i];
    }
    for (int j = 1; j < l1; ++j) {
      int pj = (1 << (j - 1));
      for (int i = 0; i + pj < len; ++i) {
        ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
      }
    }
  }

  T query(int l, int r) {
    if(l == r) return A[l];
    int lt = r - l + 1;
    int q = ceil(log2(lt)) - 1;
    return op(ST[l][q], ST[r - (1 << q) + 1][q]);
  }
};
```

3.9 Trie

```
struct Trie{
    int ch[N][26], tot;
    int dfn[N], cid[N], out[N], cnt;
    int ins(string s){
        int p = 1, len = s.size();
        for(int i=0;i<len;i++){
            int c = s[i] - 'a';
```

```
 8          if(!ch[p][c]) ch[p][c] = ++tot;
 9          p = ch[p][c];
10      }
11      return p;
12  }
13  void dfs(int x){
14      dfn[x] = ++cnt, cid[cnt] = x;
15      for(int i=0;i<26;i++){
16          if(ch[x][i]) dfs(ch[x][i]);
17      }
18      out[x] = cnt;
19  }
20  int find(string s, int k){
21      int p = 1;
22      for(int i=0;i<k;i++){
23          int c = s[i] - 'a';
24          if(!ch[p][c]) return -1;
25          p = ch[p][c];
26      }
27      return p;
28  }
29 }trie;
```

3.10 支配树

```
 1 struct DominatorTree {
 2     int n, cs;
 3     std::vector<std::vector<int>> E, RE, rdom;
 4     std::vector<int> S, RS, par, val, sdom, rp, dom;
 5
 6     DominatorTree(int n) {
 7         this->cs = 0;
 8         this->n = n;
 9         E.resize(n + 1);
10         RE.resize(n + 1);
11         rdom.resize(n + 1);
12         S.resize(n + 1);
13         RS.resize(n + 1);
14         par.resize(n + 1);
15         val.resize(n + 1);
16         sdom.resize(n + 1);
17         rp.resize(n + 1);
18         dom.resize(n + 1);
19         for (int i = 0; i <= n; i++)
20         {
21             par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0;
22             E[i].clear();
23             RE[i].clear();
24             rdom[i].clear();
```

```
25              }
26          }
27
28      void add_edge(int x, int y) {
29          E[x].push_back(y);
30      }
31
32      void Union(int x, int y) {
33          par[x] = y;
34      }
35
36      int Find(int x, int c = 0) {
37          if (par[x] == x)
38              return c ? -1 : x;
39          int p = Find(par[x], 1);
40          if (p == -1)
41              return c ? par[x] : val[x];
42          if (sdom[val[x]] > sdom[val[par[x]]])
43              val[x] = val[par[x]];
44          par[x] = p;
45          return c ? p : val[x];
46      }
47
48      void dfs(int x) {
49          RS[S[x] = ++cs] = x;
50          par[cs] = sdom[cs] = val[cs] = cs;
51          for (int e : E[x]) {
52              if (S[e] == 0)
53                  dfs(e), rp[S[e]] = S[x];
54              RE[S[e]].push_back(S[x]);
55          }
56      }
57
58      int solve(int s, std::vector<int>& up) {
59          dfs(s);
60          for (int i = cs; i; i--) {
61              for (int e : RE[i])
62                  sdom[i] = std::min(sdom[i], sdom[Find(e)]);
63              if (i > 1)
64                  rdom[sdom[i]].push_back(i);
65              for (int e : rdom[i]) {
66                  int p = Find(e);
67                  if (sdom[p] == i)
68                      dom[e] = i;
69                  else
70                      dom[e] = p;
71              }
72              if (i > 1)
73                  Union(i, rp[i]);
74          }
75          for (int i = 2; i <= cs; i++)
76              if (sdom[i] != dom[i])
```

```
77              dom[i] = dom[dom[i]];
78          for (int i = 2; i <= cs; i++)
79              up[RS[i]] = RS[dom[i]];
80          return cs;
81      }
82  };
```

四、 计算几何

4.1 点

```cpp
using real = double;
using segment_type = tuple<real, real, real, real>;
using point_type = tuple <real, real>;
const auto p2p_distance{[&](const point_type& a, const point_type& b) {
    const auto & [x, y] = a;
    const auto & [u, v] = b;
    return hypot(x - u, y - v);
}};
const auto p2s_distance{[&](const point_type& p, const segment_type& s) {
    const auto [x, y]{p};
    const auto [a, b, c, d]{s};
    if ((x - a) * (c - a) + (y - b) * (d - b) < 0)
        return p2p_distance({a, b}, {x, y});
    if ((x - c) * (a - c) + (y - d) * (b - d) < 0)
        return p2p_distance({c, d}, {x, y});
    real cross = (a - c) * (b - y) - (a - x) * (b - d);
    return abs(cross) / p2p_distance({a, b}, {c, d});
}};
// 三维点
const double INF = 1e18;
const double eps = 1e-8;
#define zero(x) (((x) > 0 ? (x) : -(x)) < eps)
struct point3
{
    double x, y, z;
    point3 operator+(const point3 &o) const
    {
        return {x + o.x, y + o.y, z + o.z};
    }
    point3 operator-(const point3 &o) const
    {
        return {x - o.x, y - o.y, z - o.z};
    }
    point3 operator*(const double &o) const
    {
        return {x*o , y *o, z *o};
    }
    point3 operator/(const double &o) const
    {
        return {x/o , y /o, z /o};
    }
    bool operator<(const point3 &o) const
    {
        if (!zero(x - o.x))
            return x < o.x;
        if (!zero(y - o.y))
            return y < o.y;
```

```
48          return z < o.z;
49      }
50      bool operator!=(const point3 &o) const
51      {
52          return (!zero(x - o.x) || !zero(y - o.y) || !zero(z - o.z));
53      }
54  }a[N];
55  vector<point3> line;
56  double vlen(point3 p)
57  {
58      return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
59  }
60  point3 xmult(point3 u, point3 v)
61  {
62      point3 ret;
63      ret.x = u.y * v.z - v.y * u.z;
64      ret.y = u.z * v.x - u.x * v.z;
65      ret.z = u.x * v.y - u.y * v.x;
66      return ret;
67  }
68  double dmult(point3 u, point3 v)
69  {
70      return u.x * v.x + u.y * v.y + u.z * v.z;
71  }
72  point3 projection(point3 p, point3 u)
73  {
74      return u*dmult(p,u);
75  }
```

五、 字符串

5.1 Hash

```
struct Shash{
    const ll base[2]={29,31};
    const ll hashmod[2]={(ll)1e9,998244353};

    array<vector<ll>,2>hsh,pwMod;
    void init(string &s){
        int n=s.size();s='␣'+s;
        hsh[0].resize(n+1),hsh[1].resize(n+1);
        pwMod[0].resize(n+1),pwMod[1].resize(n+1);
        for(int i=0;i<2;i++){
            pwMod[i][0]=1;
            for(int j=1;j<=n;j++){
                pwMod[i][j]=pwMod[i][j-1]*base[i]%hashmod[i];
                hsh[i][j]=(hsh[i][j-1]*base[i]+s[j])%hashmod[i];
            }
        }
    }
    pair<ll,ll>get(int l,int r){
        pair<ll,ll>ans;
        ans.fi=(hsh[0][r]-hsh[0][l-1]*pwMod[0][r-l+1])%hashmod[0];
        ans.se=(hsh[1][r]-hsh[1][l-1]*pwMod[1][r-l+1])%hashmod[1];
        ans.fi=(ans.fi+hashmod[0])%hashmod[0];
        ans.se=(ans.se+hashmod[1])%hashmod[1];
        return ans;
    }
    bool same(int la,int ra,int lb,int rb){
        return get(la,ra)==get(lb,rb);
    }
};
```

5.2 AC 自动机

```
struct AC {
  int n, tot, alp;
  std::vector<int> fail;
  std::vector<std::vector<int>> tr;
  AC() {}
  AC(int n, int m = 26) {
    alp = m;
    fail.resize(n);
    tr.resize(n);
    fail.assign(n, 0);
    for (int i = 0; i < n; i++) {
      tr[i].assign(m, 0);
```

```
13        }
14      init();
15    }
16    void init() {
17      tot = -1, new_node();
18    }
19    int new_node() { return ++tot, fail[tot] = 0, tr[tot].assign(tr[tot].size(),
        0), tot; }
20    void insert(const std::string& s) {
21      for (int i = 0, u = 0; i < s.size(); i++) {
22        int c = s[i];
23        if(!tr[u][c]) tr[u][c] = new_node();
24        u = tr[u][c];
25      }
26    }
27    void build() {
28      std::queue<int> q;
29      int ql = 1, qr = 0;
30      for (int i = 0; i < alp; i++) {
31        if (tr[0][i]) {
32          q.push(tr[0][i]);
33        }
34      }
35      while(!q.empty()) {
36        int u = q.front();
37        q.pop();
38        for (int c = 0; c < alp; c++) {
39          if (tr[u][c]) fail[tr[u][c]] = tr[fail[u]][c], q.push(tr[u][c]);
40          else tr[u][c] = tr[fail[u]][c];
41        }
42      }
43    }
44  };
```

5.3   KMP& EXKMP

```
1   struct KMP {
2       std::vector<int> fail;
3       std::string pattern;
4       KMP() {}
5
6       KMP(const std::string& p) {
7           init(p);
8       }
9
10      void init(const std::string& p) {
11          pattern = p;
12          fail.resize(pattern.size() + 1);
13          fail[0] = -1; // 失配数组第一个元素为-1
14          int j = -1; // j表示失配数组的值
```

```cpp
        for (int i = 0; i < pattern.size(); i++) { // 遍历模式串
            while (j >= 0 && pattern[i] != pattern[j]) { // 如果失配，则回溯
                j = fail[j]; // 回溯到失配位置的失配数组值
            }
            j++; // 失配数组值加1
            fail[i + 1] = j; // 更新失配数组
        }
    }
    // 匹配函数，返回所有匹配位置
    std::vector<int> match(const std::string& s) {
        std::vector<int> res;
        int j = 0;
        for (int i = 0; i < s.size(); i++) { // 遍历文本串
            while (j >= 0 && s[i] != pattern[j]) { // 如果失配，则回溯
                j = fail[j]; // 回溯到失配位置的失配数组值
            }
            // dbg(i, j, s[i], pattern[j], pattern.size());
            j++; // 失配数组值加1
            if (j == pattern.size()) { // 如果匹配成功

                res.push_back(i - j + 1); // 存储匹配位置
                j = fail[j]; // 回溯到失配位置的失配数组值
            }
        }
        return res; // 返回所有匹配位置
    }
};
struct EXKMP {
    string pattern;
    vector<int> z;
    EXKMP() {}

    EXKMP(const std::string& p) {
        init(p);
    }

    void init(const std::string& p) {
        pattern = p;
        int n = p.size();
        z.resize(n);
        z.assign(n, 0);
        z[0] = p.size();
        for (int i = 1, l, r = -1; i < n; i++) {
            if (i <= r) z[i] = min(z[i - l], r - i + 1);
            while(i + z[i] < n && p[z[i]] == p[i + z[i]]) z[i]++;
            if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
        }
    }
    // 匹配函数，返回所有位置的最长前缀
    std::vector<int> match(const std::string& s) {
        vector<int> pre(s.size(), 0);
        int m = s.size(), n = pattern.size();
```

```cpp
67          for (int i = 0, l, r = -1; i < m; i++) {
68              if (i <= r) pre[i] = min(z[i - l], r - i + 1);
69              while(pre[i] < n && i + pre[i] < m && pattern[pre[i]] == s[i + pre[i
                    ]]) pre[i]++;
70              if (i + pre[i] - 1 > r) l = i, r = i + pre[i] - 1;
71          }
72          return pre;
73      }
74  };
```

5.4 get_next

```cpp
1           s = "␣" + s;
2           vector<int> nxt(s.size(), 0);
3           vector<int> dep(s.size(), 0);
4           vector<int> len(s.size(), 0);
5           dep[1] = 1; // 出错点 dep[0] = 0 而不是 1
6           auto get_next = [&] () {
7                   int i, j;
8                   for (i = 2, j = 0; i < nxt.size(); i++) {
9                           while(j && s[j + 1] != s[i]) j = nxt[j];
10                          if (s[j + 1] == s[i]) j++;
11                          nxt[i] = j;
12                          dep[i] = dep[j] + 1;
13                          len[i] = len[j] + j;
14                  }
15      cout << __PRETTY_FUNCTION__ << endl;
16          };
17          get_next();
```

5.5 Manacher

```cpp
1   struct Manacher{
2       string s, t; // s 为原串 t 为补充后的串
3       int n;       // t 的长度
4       vector<int> d; // 回文半径
5       ma(string s) : s(s), n(s.size() * 2 + 3) {
6           init(t);
7           build(d);
8       }
9       void init(string &t) {
10          t = "$#";
11          for (int i = 0; i < s.size(); i++) {
12              t += s[i];
13              t += "#";
14          }
15          t += '@';
```

```
16        }
17        void build(vector<int> & d) {
18            auto equ = [&](char l, char r) {
19                return l == r;
20            };
21            d = vector<int>(n, 0);
22            d[1] = 1;
23            for (int i = 2, l = 1, r = 1; i < n; i++) {
24                if (i <= r) d[i] = min(d[r + l - i], r - i + 1);
25                while (equ(t[i - d[i]], t[i + d[i]])) {
26                    d[i]++;
27                }
28                if (i + d[i] - 1 > r) r = i + d[i] - 1, l = i - d[i] + 1;
29            }
30            return;
31        }
32    };
```

5.6 后缀自动机

```
1    class SAM {
2    public:
3        class state {
4        public:
5            state() = default;
6            state(int len, int link) :len(len), link(link) { };
7            state(int len, int link, map<char, int>& next) :len(len), link(link),
                     next(next) { };
8            int len, link;
9            std::map<char, int> next;
10        };
11        vector<state> st;
12        SAM() {
13            st.push_back(state(0, -1));
14        }
15        SAM(const string& s) :SAM() {
16            for (auto ch : s) {
17                sam_extend(ch);
18            }
19        }
20        void sam_extend(char ch) {
21            int p = st.size() - 1, cur = st.size();
22            st.push_back(state(st.back().len + 1, -1));
23            while (p != -1 && st[p].next.count(ch) == 0) {
24                st[p].next[ch] = cur;
25                p = st[p].link;
26            }
27            if (p == -1) {
28                st[cur].link = 0;
```

```
29              } else {
30                  int q = st[p].next[ch];
31                  if (st[q].len == st[p].len + 1) {
32                      st[cur].link = q;
33                  } else {
34                      st.push_back(state(st[p].len + 1, st[q].link, st[q].next));
35                      int clone = st.size() - 1;
36                      while (p != -1 && st[p].next[ch] == q) {
37                          st[p].next[ch] = clone;
38                          p = st[p].link;
39                      }
40                      st[q].link = st[cur].link = clone;
41                  }
42              }
43          }
44  };
```

```
1   struct SAM {
2       int vcnt, last;
3       std::vector<int> len, link;
4       std::vector<std::vector<int>> tr;
5
6       void init(int size) {
7           vcnt = last = 0;
8           len.resize(size);
9           link.resize(size);
10          tr.resize(size, std::vector<int>(26, 0));
11          link[0] = -1;
12      }
13
14      void clear() {
15          vcnt = last = 0;
16          len.clear();
17          link.clear();
18          tr.clear();
19      }
20
21      void add(int c) {
22          int cur = ++vcnt;
23          len[cur] = len[last] + 1;
24          int p = last;
25          while (p != -1 && !tr[p][c])
26              tr[p][c] = cur, p = link[p];
27          if (p == -1)
28              link[cur] = 0;
29          else {
30              int q = tr[p][c];
31              if (len[q] == len[p] + 1)
32                  link[cur] = q;
33              else {
```

```cpp
            int clone = ++vcnt;
            len[clone] = len[p] + 1;
            link[clone] = link[q];
            tr[clone] = tr[q];
            while (p != -1 && tr[p][c] == q)
                tr[p][c] = clone, p = link[p];
            link[q] = clone;
            link[cur] = clone;
        }
    }
    last = cur;
    }
};

int main() {
    SAM S;
    int size = 2 * 100; // 根据需要的大小设置
    S.init(size);

    // 使用 SAM 对字符串进行处理
    std::string input = "abcabca";
    for (char c : input) {
        int index = c - 'a';
        S.add(index);
    }

    // 输出 SAM 中的一些信息
    std::cout << "Number of states: " << S.vcnt << std::endl;
    std::cout << "Last state: " << S.last << std::endl;

    return 0;
}
```

5.7 序列自动机

```cpp
class SequenceAM :public vector<vector<int>>{
public:
    SequenceAM() = default;
    SequenceAM(const string& s, int sigma = 26)
     :vector<vector<int>>(s.size() + 1, vector<int>(sigma, 0)) {
        auto &nxt=*this;
        for(int i=s.size();i>=1;i--) {
            nxt[i][s[i-1]-'a']=i;
            nxt[i-1]=nxt[i];
        }
    }
};
```

六、 数学

6.1 欧拉函数

单独求欧拉函数 phi(x)

```cpp
int phi(int n) {
        int m = 1;
        for (int i = 2; n > 1; ++i) {
                if (n % i == 0) {
                        m *= i - 1;
                        n /= i;
                        while (n % i == 0) {
                                m *= i;
                                n /= i;
                        }
                }
        }
        return m;
}
```

预处理 phi(x)

```cpp
for (i = 1; i <= maxn; i++) phi[i] = i;
for (i = 2; i <= maxn; i += 2) phi[i] /= 2;
for (i = 3; i <= maxn; i += 2) if(phi[i] == i) {
    for (j = i; j <= maxn; j += i)
        phi[j] = phi[j] / i * (i - 1);
}
```

6.2 扩展 GCD

求 x, y 满足 gcd(a, b) = a * x + b * y

```cpp
int exgcd(int a, int b, int & x, int & y) {
    if(b == 0) {
        x == 1, y == 1;
        return a;
    }
    int ret = exgcd(b, a % b, x, y);
    int tmp = x; x = y; y = tmp - a / b * y;
    return ret;
}
```

6.3 各种筛

```
线性筛素数
保证每个数只会被它的最小质因子给筛掉（不同于埃氏筛中每个数会被它所有质因子筛一遍从
    而使复杂度过高）
int pri[N],tot,zhi[N];//zhi[i]为1的表示不是质数
void sieve()
{
    zhi[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]==0) break;
        }
    }
}
所有线性筛积性函数都必须基于线性筛素数。
线性筛莫比乌斯函数
int mu[N],pri[N],tot,zhi[N];
void sieve()
{
    zhi[1]=mu[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i,mu[i]=-1;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]) mu[i*pri[j]]=-mu[i];
            else {mu[i*pri[j]]=0;break;}
        }
    }
}
线性筛欧拉函数
int phi[N],pri[N],tot,zhi[N];
void sieve()
{
    zhi[1]=phi[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i,phi[i]=i-1;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]) phi[i*pri[j]]=phi[i]*phi[pri[j]];
            else {phi[i*pri[j]]=phi[i]*pri[j];break;}
        }
    }
}
线性筛约数个数
```

```
51  记d(i)
52  d(i)表示i的约数个数,d(i)=k (i=1)(ai+1) d(i)=i=1k(ai+1)
53  维护每一个数的最小值因子出现的次数（即a1）即可
54  int d[N],a[N],pri[N],tot,zhi[N];
55  void sieve()
56  {
57      zhi[1]=d[1]=1;
58      for (int i=2;i<=n;i++)
59      {
60          if (!zhi[i]) pri[++tot]=i,d[i]=2,a[i]=1;
61          for (int j=1;j<=tot&&i*pri[j]<=n;j++)
62          {
63              zhi[i*pri[j]]=1;
64              if (i%pri[j]) d[i*pri[j]]=d[i]*d[pri[j]],a[i*pri[j]]=1;
65              else {d[i*pri[j]]=d[i]/(a[i]+1)*(a[i]+2);a[i*pri[j]]=a[i]+1;break;}
66          }
67      }
68  }
```

6.4 多项式

```
1   using i64 = long long;
2   template<class T>
3   #define constexpr
4   constexpr T power(T a, i64 b) {
5       T res = 1;
6       for (; b; b /= 2, a *= a) {
7           if (b % 2) {
8               res *= a;
9           }
10      }
11      return res;
12  }
13
14  template<int P>
15  struct MInt {
16      int x;
17      constexpr MInt() : x{} {}
18      constexpr MInt(i64 x) : x{norm(x % getMod())} {}
19
20      static int Mod;
21      constexpr static int getMod() {
22          if (P > 0) {
23              return P;
24          } else {
25              return Mod;
26          }
27      }
28      constexpr static void setMod(int Mod_) {
```

```
29            Mod = Mod_;
30        }
31        constexpr int norm(int x) const {
32            if (x < 0) {
33                x += getMod();
34            }
35            if (x >= getMod()) {
36                x -= getMod();
37            }
38            return x;
39        }
40        constexpr int val() const {
41            return x;
42        }
43        explicit constexpr operator int() const {
44            return x;
45        }
46        constexpr MInt operator-() const {
47            MInt res;
48            res.x = norm(getMod() - x);
49            return res;
50        }
51        constexpr MInt inv() const {
52            assert(x != 0);
53            return power(*this, getMod() - 2);
54        }
55        constexpr MInt &operator*=(MInt rhs) & {
56            x = 1LL * x * rhs.x % getMod();
57            return *this;
58        }
59        constexpr MInt &operator+=(MInt rhs) & {
60            x = norm(x + rhs.x);
61            return *this;
62        }
63        constexpr MInt &operator-=(MInt rhs) & {
64            x = norm(x - rhs.x);
65            return *this;
66        }
67        constexpr MInt &operator/=(MInt rhs) & {
68            return *this *= rhs.inv();
69        }
70        friend constexpr MInt operator*(MInt lhs, MInt rhs) {
71            MInt res = lhs;
72            res *= rhs;
73            return res;
74        }
75        friend constexpr MInt operator+(MInt lhs, MInt rhs) {
76            MInt res = lhs;
77            res += rhs;
78            return res;
79        }
80        friend constexpr MInt operator-(MInt lhs, MInt rhs) {
```

```
 81           MInt res = lhs;
 82           res -= rhs;
 83           return res;
 84       }
 85       friend constexpr MInt operator/(MInt lhs, MInt rhs) {
 86           MInt res = lhs;
 87           res /= rhs;
 88           return res;
 89       }
 90       friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
 91           i64 v;
 92           is >> v;
 93           a = MInt(v);
 94           return is;
 95       }
 96       friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a)
                {
 97           return os << a.val();
 98       }
 99       friend constexpr bool operator==(MInt lhs, MInt rhs) {
100           return lhs.val() == rhs.val();
101       }
102       friend constexpr bool operator!=(MInt lhs, MInt rhs) {
103           return lhs.val() != rhs.val();
104       }
105   };
106
107   template<>
108   int MInt<0>::Mod = 1;
109
110   template<int V, int P>
111   constexpr MInt<P> CInv = MInt<P>(V).inv();
112
113   const int P = 998244353;
114   using Z = MInt<P>;
115
116   std::vector<int> rev;
117   template<int P>
118   std::vector<MInt<P>> roots{0, 1};
119
120   template<int P>
121   constexpr MInt<P> findPrimitiveRoot() {
122       MInt<P> i = 2;
123       int k = __builtin_ctz(P - 1);
124       while (true) {
125           if (power(i, (P - 1) / 2) != 1) {
126               break;
127           }
128           i += 1;
129       }
130       return power(i, (P - 1) >> k);
131   }
```

```
132
133   template<int P>
134   constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
135
136   template<>
137   constexpr MInt<998244353> primitiveRoot<998244353> {31};
138
139   template<int P>
140   constexpr void dft(std::vector<MInt<P>> &a) {
141       int n = a.size();
142
143       if (int(rev.size()) != n) {
144           int k = __builtin_ctz(n) - 1;
145           rev.resize(n);
146           for (int i = 0; i < n; i++) {
147               rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
148           }
149       }
150
151       for (int i = 0; i < n; i++) {
152           if (rev[i] < i) {
153               std::swap(a[i], a[rev[i]]);
154           }
155       }
156       if (roots<P>.size() < n) {
157           int k = __builtin_ctz(roots<P>.size());
158           roots<P>.resize(n);
159           while ((1 << k) < n) {
160               auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1)
                      );
161               for (int i = 1 << (k - 1); i < (1 << k); i++) {
162                   roots<P>[2 * i] = roots<P>[i];
163                   roots<P>[2 * i + 1] = roots<P>[i] * e;
164               }
165               k++;
166           }
167       }
168       for (int k = 1; k < n; k *= 2) {
169           for (int i = 0; i < n; i += 2 * k) {
170               for (int j = 0; j < k; j++) {
171                   MInt<P> u = a[i + j];
172                   MInt<P> v = a[i + j + k] * roots<P>[k + j];
173                   a[i + j] = u + v;
174                   a[i + j + k] = u - v;
175               }
176           }
177       }
178   }
179
180   template<int P>
181   constexpr void idft(std::vector<MInt<P>> &a) {
182       int n = a.size();
```

```
183      std::reverse(a.begin() + 1, a.end());
184      dft(a);
185      MInt<P> inv = (1 - P) / n;
186      for (int i = 0; i < n; i++) {
187          a[i] *= inv;
188      }
189  }
190
191  template<int P = 998244353>
192  struct Poly : public std::vector<MInt<P>> {
193      using Value = MInt<P>;
194
195      Poly() : std::vector<Value>() {}
196      explicit constexpr Poly(int n) : std::vector<Value>(n) {}
197
198      explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a
         ) {}
199      constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(
         a) {}
200
201      template<class InputIt, class = std::_RequireInputIter<InputIt>>
202      explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(
         first, last) {}
203
204      template<class F>
205      explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
206          for (int i = 0; i < n; i++) {
207              (*this)[i] = f(i);
208          }
209      }
210
211      constexpr Poly shift(int k) const {
212          if (k >= 0) {
213              auto b = *this;
214              b.insert(b.begin(), k, 0);
215              return b;
216          } else if (this->size() <= -k) {
217              return Poly();
218          } else {
219              return Poly(this->begin() + (-k), this->end());
220          }
221      }
222      constexpr Poly trunc(int k) const {
223          Poly f = *this;
224          f.resize(k);
225          return f;
226      }
227      constexpr friend Poly operator+(const Poly &a, const Poly &b) {
228          Poly res(std::max(a.size(), b.size()));
229          for (int i = 0; i < a.size(); i++) {
230              res[i] += a[i];
231          }
```

```
232        for (int i = 0; i < b.size(); i++) {
233            res[i] += b[i];
234        }
235        return res;
236    }
237    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
238        Poly res(std::max(a.size(), b.size()));
239        for (int i = 0; i < a.size(); i++) {
240            res[i] += a[i];
241        }
242        for (int i = 0; i < b.size(); i++) {
243            res[i] -= b[i];
244        }
245        return res;
246    }
247    constexpr friend Poly operator-(const Poly &a) {
248        std::vector<Value> res(a.size());
249        for (int i = 0; i < int(res.size()); i++) {
250            res[i] = -a[i];
251        }
252        return Poly(res);
253    }
254    constexpr friend Poly operator*(Poly a, Poly b) {
255        if (a.size() == 0 || b.size() == 0) {
256            return Poly();
257        }
258        if (a.size() < b.size()) {
259            std::swap(a, b);
260        }
261        int n = 1, tot = a.size() + b.size() - 1;
262        while (n < tot) {
263            n *= 2;
264        }
265        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
266            Poly c(a.size() + b.size() - 1);
267            for (int i = 0; i < a.size(); i++) {
268                for (int j = 0; j < b.size(); j++) {
269                    c[i + j] += a[i] * b[j];
270                }
271            }
272            return c;
273        }
274        a.resize(n);
275        b.resize(n);
276        dft(a);
277        dft(b);
278        for (int i = 0; i < n; ++i) {
279            a[i] *= b[i];
280        }
281        idft(a);
282        a.resize(tot);
283        return a;
```

```
284          }
285          constexpr friend Poly operator*(Value a, Poly b) {
286              for (int i = 0; i < int(b.size()); i++) {
287                  b[i] *= a;
288              }
289              return b;
290          }
291          constexpr friend Poly operator*(Poly a, Value b) {
292              for (int i = 0; i < int(a.size()); i++) {
293                  a[i] *= b;
294              }
295              return a;
296          }
297          constexpr friend Poly operator/(Poly a, Value b) {
298              for (int i = 0; i < int(a.size()); i++) {
299                  a[i] /= b;
300              }
301              return a;
302          }
303          constexpr Poly &operator+=(Poly b) {
304              return (*this) = (*this) + b;
305          }
306          constexpr Poly &operator-=(Poly b) {
307              return (*this) = (*this) - b;
308          }
309          constexpr Poly &operator*=(Poly b) {
310              return (*this) = (*this) * b;
311          }
312          constexpr Poly &operator*=(Value b) {
313              return (*this) = (*this) * b;
314          }
315          constexpr Poly &operator/=(Value b) {
316              return (*this) = (*this) / b;
317          }
318          constexpr Poly deriv() const {
319              if (this->empty()) {
320                  return Poly();
321              }
322              Poly res(this->size() - 1);
323              for (int i = 0; i < this->size() - 1; ++i) {
324                  res[i] = (i + 1) * (*this)[i + 1];
325              }
326              return res;
327          }
328          constexpr Poly integr() const {
329              Poly res(this->size() + 1);
330              for (int i = 0; i < this->size(); ++i) {
331                  res[i + 1] = (*this)[i] / (i + 1);
332              }
333              return res;
334          }
335          constexpr Poly inv(int m) const {
```

```
336        Poly x{(*this)[0].inv()};
337        int k = 1;
338        while (k < m) {
339            k *= 2;
340            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
341        }
342        return x.trunc(m);
343    }
344    constexpr Poly log(int m) const {
345        return (deriv() * inv(m)).integr().trunc(m);
346    }
347    constexpr Poly exp(int m) const {
348        Poly x{1};
349        int k = 1;
350        while (k < m) {
351            k *= 2;
352            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
353        }
354        return x.trunc(m);
355    }
356    constexpr Poly pow(int k, int m) const {
357        int i = 0;
358        while (i < this->size() && (*this)[i] == 0) {
359            i++;
360        }
361        if (i == this->size() || 1LL * i * k >= m) {
362            return Poly(m);
363        }
364        Value v = (*this)[i];
365        auto f = shift(-i) * v.inv();
366        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k)
               ;
367    }
368    constexpr Poly sqrt(int m) const {
369        Poly x{1};
370        int k = 1;
371        while (k < m) {
372            k *= 2;
373            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
374        }
375        return x.trunc(m);
376    }
377    constexpr Poly mulT(Poly b) const {
378        if (b.size() == 0) {
379            return Poly();
380        }
381        int n = b.size();
382        std::reverse(b.begin(), b.end());
383        return ((*this) * b).shift(-(n - 1));
384    }
385    constexpr std::vector<Value> eval(std::vector<Value> x) const {
386        if (this->size() == 0) {
```

```
387            return std::vector<Value>(x.size(), 0);
388        }
389        const int n = std::max(x.size(), this->size());
390        std::vector<Poly> q(4 * n);
391        std::vector<Value> ans(x.size());
392        x.resize(n);
393        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
394            if (r - l == 1) {
395                q[p] = Poly{1, -x[l]};
396            } else {
397                int m = (l + r) / 2;
398                build(2 * p, l, m);
399                build(2 * p + 1, m, r);
400                q[p] = q[2 * p] * q[2 * p + 1];
401            }
402        };
403        build(1, 0, n);
404        std::function<void(int, int, int, const Poly &)> work = [&](int p, int
                l, int r, const Poly &num) {
405            if (r - l == 1) {
406                if (l < int(ans.size())) {
407                    ans[l] = num[0];
408                }
409            } else {
410                int m = (l + r) / 2;
411                work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
412                work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
413            }
414        };
415        work(1, 0, n, mulT(q[1].inv(n)));
416        return ans;
417    }
418 };
419
420 template<int P = 998244353>
421 Poly<P> berlekampMassey(const Poly<P> &s) {
422     Poly<P> c;
423     Poly<P> oldC;
424     int f = -1;
425     for (int i = 0; i < s.size(); i++) {
426         auto delta = s[i];
427         for (int j = 1; j <= c.size(); j++) {
428             delta -= c[j - 1] * s[i - j];
429         }
430         if (delta == 0) {
431             continue;
432         }
433         if (f == -1) {
434             c.resize(i + 1);
435             f = i;
436         } else {
437             auto d = oldC;
```

```
438         d *= -1;
439         d.insert(d.begin(), 1);
440         MInt<P> df1 = 0;
441         for (int j = 1; j <= d.size(); j++) {
442             df1 += d[j - 1] * s[f + 1 - j];
443         }
444         assert(df1 != 0);
445         auto coef = delta / df1;
446         d *= coef;
447         Poly<P> zeros(i - f - 1);
448         zeros.insert(zeros.end(), d.begin(), d.end());
449         d = zeros;
450         auto temp = c;
451         c += d;
452         if (i - temp.size() > f - oldC.size()) {
453             oldC = temp;
454             f = i;
455         }
456     }
457 }
458 c *= -1;
459 c.insert(c.begin(), 1);
460 return c;
461 }
462
463
464 template<int P = 998244353>
465 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
466     int m = q.size() - 1;
467     while (n > 0) {
468         auto newq = q;
469         for (int i = 1; i <= m; i += 2) {
470             newq[i] *= -1;
471         }
472         auto newp = p * newq;
473         newq = q * newq;
474         for (int i = 0; i < m; i++) {
475             p[i] = newp[i * 2 + n % 2];
476         }
477         for (int i = 0; i <= m; i++) {
478             q[i] = newq[i * 2];
479         }
480         n /= 2;
481     }
482     return p[0] / q[0];
483 }
```

6.5 factorize

```
1   /* method 1 */
```

```cpp
int primes[N], minp[N];
bool st[N];
int cnt;
void get_primes(int n)
{

    for(int i = 2; i <= n; i++)
    {
        if(!st[i]) primes[cnt++] = i,minp[i] = i;
        for(int j = 0; primes[j] <= n/i; j++)
        {
            st[i*primes[j]] = 1;
            minp[primes[j]*i] = primes[j];
            if(i % primes[j] == 0) break;
        }
    }
}
void solve() {
    auto fac = [&](auto self, int n) {
        for (int i = 0; primes[i] * primes[i] <= n; i++) {
            if (n % primes[i] == 0) {
                int cnt = 0;
                while(n % primes[i] == 0) {
                    n /= primes[i];
                    cnt++;
                }
                f.pb({primes[i], cnt});
            }
        }
        if (n != 1) f.pb({n, 1});
    };
    auto iter = [&](auto self, int p, int cur) -> void {
        if (p == f.size()) {
            yin.pb(cur);
            return;
        }
        int mul = 1;
        for (int i = 0; i <= f[p].second; i++) {
            self(self, p + 1, cur * mul);
            mul *= f[p].first;
        };
    };
}
/* method 2 */
for (int i = 1; 1ll * i * i <= c; i++){
    if (c % i == 0)
    {
        factor.push_back(i);
        if (i * i != c)
            factor.push_back(c / i);
    }
}
```

6.6 斯特林数

```cpp
std::vector S(n + 2, std::vector<Z>(n + 2));
for (int i = 0; i <= n + 1; i++) {
    S[i][0] = !i;
    for (int j = 1; j <= i; j++) {
        S[i][j] = S[i - 1][j - 1] + S[i - 1][j] * j;
    }
}
```

6.7 高斯消元

```cpp
void gauss(int n, double g[maxn][maxn]) { // input: N * (N + 1) Matrix
    for (int i = 1; i <= n; ++i) {
        double temp = 0;
        int pos = -1;
        for (int j = i; j <= n; ++j) {
            if (fabs(g[j][i]) > temp) temp = fabs(g[j][i]), pos = j;
        }
        if (pos == -1) continue;
        for (int k = 1; k <= n + 1; ++k) swap(g[pos][k], g[i][k]);
        temp = g[i][i];
        for (int k = 1; k <= n + 1; ++k) g[i][k] /= temp;
        for (int j = i + 1; j <= n; ++j) {
            temp = g[j][i];
            for (int k = 1; k <= n + 1; ++k) g[j][k] -= temp * g[i][k
                ];
        }
    }
    for (int i = n; i >= 1; --i) {
        for (int j = 1; j < i; ++j) {
            g[j][n + 1] -= g[i][n + 1] * g[j][i];
            g[j][i] = 0;
        }
    }
}
```

七、 其他

7.1 Frame

```
1   #include<bits/stdc++.h>
2
3   using namespace std;
4   // #pragma GCC optimize("Ofast,no-stack-protector,unroll-loops,fast-math")
5   // #pragma GCC target("sse,sse2,sse3,ssse3,sse4.1,sse4.2,avx,avx2,popcnt,tune=
        native")
6   // #pragma GCC optimize(2)
7   #define int long long
8   #define pb push_back
9   #define pii pair<int,int>
10  typedef long long ll;
11  const int mod = 1e9 +7;
12  const int INF = 1e15;
13  #define dbg(x...) do { cerr << #x << "  -> "; err(x); } while (0);
14  void err() {cerr << endl; }
15  template<class T, class... Ts> void err(const T& arg, const Ts&... args) {cerr
        << arg << " "; err(args...); }
16  int qp(int a, int b) {
17      int res = 1;
18      for (; b ; b >>= 1, a = 1ll * a * a % mod) {
19          if(b & 1) {
20              res = 1ll * res * a % mod;
21          }
22      }
23      return res;
24  }
25
26  unsigned int generateSeedFromTimestamp() {
27      auto now = std::chrono::system_clock::now(); // 获取当前时间点
28      auto timestamp = std::chrono::duration_cast<std::chrono::milliseconds>(now.
            time_since_epoch()); // 转换为毫秒级的时间戳
29
30      return static_cast<unsigned int>(timestamp.count()); // 将时间戳转换为整数种
            子值
31  }
32
33  void solve() {
34
35  }
36
37  signed main () {
38      std::ios::sync_with_stdio(false);
39      std::cin.tie(0);
40      std::cout.tie(0);
41      // int testcase; cin >> testcase;
42      // while (testcase--)
43          solve();
```

```
44        return 0;
45    }
```

7.2 快读快写

```
1    template <typename T> inline void read(T& t) {
2        int f = 0, c = getchar(); t = 0;
3        while (!isdigit(c)) f |= c == '-', c = getchar();
4        while (isdigit(c)) t = t * 10 + c - 48, c = getchar();
5        if (f) t = -t;
6    }
7
8    template <typename T> void print(T x) {
9        if (x < 0) x = -x, putchar('-');
10       if (x > 9) print(x / 10);
11       putchar(x % 10 + 48);
12   }
```