```
        }
        cur = cur->next[x];
        cur->cnt += 1;
      }
      cur->completedWord = true;
    }

    bool trieSearch(const string &s) {
      node *cur = root;
      for (char ch : s) {
        int x = ch - 'a'; // for lowercase letter
        if (cur->next[x] == nullptr) {
          return false;
        }
        cur = cur->next[x];
      }
      return cur->completedWord;
    }

    int prefixCount(const string &s) {
      node *cur = root;
      for (char ch : s) {
        int x = ch - 'a'; // for lowercase letter
        if (cur->next[x] == nullptr) {
          return 0;
        }
        cur = cur->next[x];
      }
      return cur->cnt;
    }

    void reset(node* cur) {
      for(int i = 0; i < rangeSize; i++)
        if(cur->next[i])
          reset(cur->next[i]);
      delete cur;
    }

    void clear() {
      reset(root); // Delete all nodes
      root = new node(); // Re-initialize root node
for reuse
    }

    ~Trie() { // Destructor
      reset(root);
    }
} trie;
```

Sparse Table:

```
// 0-based indexing, query finds in range [first,
last]
#define lg(x) (31 - __builtin_clz(x))
const int N = 1e5 + 7;
const int K = lg(N);

struct sparse_table {
  ll tr[N][K + 1];

  ll f(ll p1, ll p2) { // Change this function
according to the problem.
    return p1 + p2; // <===
  }

  void build(int n, const vector<ll> &a) { // O(N *
logN)
    for (int i = 0; i < n; i++) {
      tr[i][0] = a[i];
    }
    for (int j = 1; j <= K; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        tr[i][j] = f(tr[i][j - 1], tr[i + (1 << (j - 1))][j -
1]);
      }
    }
  }

  ll query1(int l, int r) { // find Sum, LCM =>
O(LogN)
    ll val = 0; // for sum => val = 0 and lcm => val
= 1
    for (int j = K; j >= 0; j--) {
      if ((1 << j) <= r - l + 1) {
        val = f(val, tr[l][j]);
        l += 1 << j;
      }
    }
    return val;
  }

  ll query2(int l, int r) { // find Min, Max, GCD,
AND, OR, XOR => O(1)
    int d = lg(r - l + 1);
    return f(tr[l][d], tr[r - (1 << d) + 1][d]);
  }
} spt;
```

Arthiculation Point:

```
const int N = 3e5 + 9;
int T, low[N], dis[N], art[N];
vector<int> g[N];
int n, m;

void dfs(int u, int pre = 0) {
  low[u] = dis[u] = ++T;
  int child = 0;
  for (auto &v : g[u]) {
    if (!dis[v]) {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
```