```cpp
const int MaxC = 26;
struct Trie {
  int nx[MaxC];
  int failure;
  int leaf;
  int st, fi;

  Trie() {
    memset(nx, 0, sizeof nx);
    failure = 0;
    st = fi = -1;
    leaf = -1;
  }

  int& operator[](int x) {
    return nx[x];
  }
} ;
vector<Trie> trie;
vector<int> children[MaxSz];
string strs[MaxSz];
int pos[MaxN];
char in[MaxN];
int now;
void dfs(int v = 0) {
  trie[v].st = now++;
  for(auto x : children[v])
    dfs(x);
  trie[v].fi = now;
}

void make_aho(int n) {
  trie.emplace_back();
  for(int i = 0; i < n; i++) {
    int cur = 0;
    for(auto x : strs[i]) {
      if(!trie[cur][x - 'a'])
      {
        trie[cur][x - 'a'] = trie.size();
        trie.emplace_back();
      }
      cur = trie[cur][x - 'a'];
    }
    pos[i] = cur;
    trie[cur].leaf = i;
  }

  queue<int> q;

  for(int i = 0; i < MaxC; i++)
    if(trie[0][i]) {
      trie[trie[0][i]].failure = 0;
      q.push(trie[0][i]);
    }

  while(q.size()) {
    int v = q.front(); q.pop();

    for(int i = 0; i < MaxC; i++)
      if(trie[v][i]) {
        trie[trie[v][i]].failure = trie[trie[v].failure][i];
        q.push(trie[v][i]);
      } else trie[v][i] = trie[trie[v].failure][i];
  }

  for(int i = 1; i < (int) trie.size(); ++i)
    children[trie[i].failure].push_back(i);
  dfs();
}
struct SqrtSum {
    long long par_sum[MaxN];
    long long par_sq[Sqrt];

    SqrtSum() {
        memset(par_sum, 0, sizeof par_sum);
        memset(par_sq, 0, sizeof par_sq);
    }

    void add(int x, int val) {
        for(; x % Sqrt && x < MaxN; x++) par_sum[x] += val;
        for(int k = x / Sqrt; k < Sqrt; k += 1) par_sq[k] += val;
    }

    long long get(int x) {
        if(x < 0) return 0;
        return par_sq[x / Sqrt] + par_sum[x];
    }

    long long get(int l, int r) {
        return get(r) - get(l - 1);
    }
} sqrt_sum;
```