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

typedef long long LL;

// suffix array O(log^2(N)) algorithm.
struct entry {
    int nr[2];
    int p;
};

bool cmp (entry a, entry b) {
    if (a.nr[0] == b.nr[0]) return a.nr[1] < b.nr[1];
    else return a.nr[0] < b.nr[0];
}

const int MAXN = 200005;
const int MAXLOG = 20;

char s[MAXN];
entry L[MAXN];
int P[MAXLOG][MAXN];

int stp, cnt;
int N;

int findLCP (int x, int y) {
    int ret = 0;
    if (x == y) return N - x;

    for (int k = stp - 1; k >= 0 && x < N && y < N; k--)
        if (P[k][x] == P[k][y]) {
            x += (1 << k);
            y += (1 << k);
            ret += (1 << k);
        }

    return ret;
}

void suffixArray() {
      for (int i = 0; i < N; i++)
            P[0][i] = (int) (s[i] - 'a');

      for (stp = 1, cnt = 1; cnt >> 1 < N; stp ++, cnt *= 2) {
            // compute L
            for (int i = 0; i < N; i++) {
                L[i].nr[0] = P[stp - 1][i];
                L[i].nr[1] = i + cnt < N ? P[stp - 1][i + cnt] : -1;
                L[i].p = i;
            }

            sort (L, L + N, cmp);

            for (int i = 0; i < N; i++) {
                if (i > 0 && L[i].nr[0] == L[i - 1].nr[0] && L[i].nr[1] == L[i - 1].nr[1])
                        P[stp][L[i].p] = P[stp][L[i - 1].p];
                else P[stp][L[i].p] = i;
            }
      }
}

struct maxSegmentTree {
      vector <int> data;
      int n;

      maxSegmentTree(int _n) {
            n = _n;
            data.resize(4 * n);
            // initialize with -1.
            build(1, 1, n);
      }

      void build(int k, int lo, int hi) {
            if (lo == hi) data[k] = - 1;
            else {
                  int mid = (lo + hi) / 2;
                  build(2 * k, lo, mid);
                  build(2 * k + 1, mid + 1, hi);
                  data[k] = max(data[2 * k], data[2 * k + 1]);
            }
      }

      void add(int pos, int val) {
            // increase pos to make in the range [1, n]
            update(1, 1, n, pos + 1, val);
      }

      void update(int k, int lo, int hi, int pos, int val) {
            if (lo == hi && lo == pos) {
                  data[k] = max(data[k], val);
            } else {
                  int mid = (lo + hi) / 2;
                  if (pos <= mid) update(2 * k, lo, mid, pos, val);
                  else if (pos > mid) update(2 * k + 1, mid + 1, hi, pos, val);
                  data[k] = max(data[2 * k], data[2 * k + 1]);
            }
      }

      int ask(int pos) {
            if (pos < 0)
                  return -1;
            // increase pos .
            return query(1, 1, n, 1, pos + 1);
      }

      int query(int k, int lo, int hi, int left, int right) {
            if (lo == left && hi == right) {
                  return data[k];
            } else {
                  int mid = (lo + hi) / 2;
                  if (right <= mid) return query(2 * k, lo, mid, left, right);
                  else if (left > mid) return query(2 * k + 1, mid + 1, hi, left, right);
                  else {
                        int ans1 = query(2 * k, lo, mid, left, mid);
                        int ans2 = query(2 * k + 1, mid + 1, hi, mid + 1, right);
                        return max(ans1, ans2);
                  }
            }
      }
};

struct minSegmentTree {
      vector <int> data;
      int n;

      minSegmentTree(int _n) {
            n = _n;
            data.resize(4 * n);
            // initialize with n-1.
            build(1, 1, n);
      }

      void build(int k, int lo, int hi) {
            if (lo == hi) data[k] = n - 1;
            else {
                  int mid = (lo + hi) / 2;
                  build(2 * k, lo, mid);
                  build(2 * k + 1, mid + 1, hi);
                  data[k] = min(data[2 * k], data[2 * k + 1]);
            }
      }

      void add(int pos, int val) {
            // increase pos to make in the range [1, n]
            update(1, 1, n, pos + 1, val);
      }

      void update(int k, int lo, int hi, int pos, int val) {
            if (lo == hi && lo == pos) {
                  data[k] = min (data[k], val);
            } else {
                  int mid = (lo + hi) / 2;
                  if (pos <= mid) update(2 * k, lo, mid, pos, val);
                  else if (pos > mid) update(2 * k + 1, mid + 1, hi, pos, val);
                  data[k] = min(data[2 * k], data[2 * k + 1]);
            }
      }

      int ask(int pos) {
            if (pos < 0)
```

```
                  return n - 1;
            // increase pos .
            return query(1, 1, n, 1, pos + 1);
      }

      int query(int k, int lo, int hi, int left, int right) {
            if (lo == left && hi == right) {
                  return data[k];
            } else {
                  int mid = (lo + hi) / 2;
                  if (right <= mid) return query(2 * k, lo, mid, left, right);
                  else if (left > mid) return query(2 * k + 1, mid + 1, hi, left, right);
                  else {
                        int ans1 = query(2 * k, lo, mid, left, mid);
                        int ans2 = query(2 * k + 1, mid + 1, hi, mid + 1, right);
                        return min(ans1, ans2);
                  }
            }
      }
};

int main() {
      int T;
      scanf ("%d", &T);
      while (T--) {
            scanf ("%s", s);
            N = strlen(s);

            suffixArray();

            vector <int> a;
            for (int i = 0; i + 1 < N; i++)
                  a.push_back(findLCP(L[i].p, L[i + 1].p));


            vector <int> mn;
            maxSegmentTree maxSeg(N);
            for (int i = 0; i < a.size(); i++) {
                  int val = maxSeg.ask(a[i] - 1);
                  mn.push_back(val);
                  maxSeg.add(a[i], i);
            }

            vector <int> mx;
            minSegmentTree minSeg(N);
            for (int i = a.size() - 1; i >= 0; i--) {
                  int val = minSeg.ask(a[i] - 1);
                  mx.push_back(val);
                  minSeg.add(a[i], i);
            }
            reverse(mx.begin(), mx.end());

            vector<vector<int> > indices(N);
            for (int i = 0; i < a.size(); i++) {
                  int id = a[i];
                  indices[id].push_back(i);
            }

            vector<LL> D(N + 1);
            // D[i] denotes number substrings which repeats i times exactly.
            // update the D array by the two pointer method as explained in editorial.
            for (int i = 1; i < N; i++) {
                  int right = 0;
                  for (int j = 0; j < indices[i].size(); j++) {
                        int id = indices[i][j];
                        if (id >= right) {
                              int lo = mn[id], hi = mx[id];
                              int t = hi - lo;
                              int mn = i;
                              if (0 <= hi && hi < a.size()) {
                                    assert (i >= a[hi]);
                                    mn = min(mn, i - a[hi]);
                              }
                              if (lo >= 0 && lo < a.size()) {
                                    assert(i >= a[lo]);
                                    mn = min(mn, i - a[lo]);
                              }
                              assert(mn >= 0);
                              D[t] += (LL)t * (LL)mn;
                              right = hi;
                        }
                  }
            }


            LL tot = accumulate(D.begin() + 2, D.end(), 0LL);
            D[1] = (LL) N * ((LL) N + 1) / 2 - tot;

            LL totalSum = accumulate(D.begin(), D.end(), 0LL);
            vector<LL> sum(N + 1);
            for (int i = 1; i <= N; i++) {
                  sum[i] = sum[i - 1] + D[i];
            }

            // process the queries.
            int queries;
            scanf ("%d", &queries);
            while (queries --) {
                  int freq;
                  scanf ("%d", &freq);
                  LL ans = 0;
                  if (freq > N) {
                        ans = 0;
                  } else {
                        ans = totalSum - sum[freq - 1];
                  }
                  assert(ans >= 0);
                  printf ("%lld\n", ans);
            }
      }

      return 0;
}
```