Hash

Pre Code //

int mul(int a, int b) {

int add(int a, int b) {

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

}();

a = ((a % M) + M) % M:

b = ((b % M) + M) % M;

a = ((a % M) + M) % M;

b = ((b % M) + M) % M:

if (!power) return 1;

ret = mul(ret, ret);

for (int i = 1: i < N: i++) {

return ret:

int pre = []() {

return 0:

return (a + b) % M;

return (a * 1LL * b) % M;

```
const int N = 1e5 + 5, P1 = 31, P2 = 37, M = 1e9 + 7:
int pw1[N], pw2[N], inv1[N], inv2[N];
int fastPower(int base, int power) {
    int ret = fastPower(base, power >> 1);
    if (power % 2) ret = mul(ret, base);
    pw1[0] = inv1[0] = pw2[0] = inv2[0] = 1;
    int mulInv1 = fastPower(P1, M - 2);
    int mulInv2 = fastPower(P2, M - 2);
        pw1[i] = mul(pw1[i - 1], P1);
        pw2[i] = mul(pw2[i - 1], P2);
        inv1[i] = mul(inv1[i - 1], mulInv1);
        inv2[i] = mul(inv2[i - 1], mulInv2);
```

Base Code //

```
1 struct Hash {
        vector<pair<int, int>>> prefixHash;
 3
        Hash(string s) {
             prefixHash = vector<pair<int, int>> (s.size(), {0, 0});
 5
             for (int i = 0; i < s.size(); i++) {</pre>
 6
                prefixHash[i].F = mul(s[i] - 'a' + 1, pw1[i]);
                prefixHash[i].S = mul(s[i] - 'a' + 1, pw2[i]);
                if (i) prefixHash[i] = {
                            add(prefixHash[i].F, prefixHash[i - 1].F),
                            add(prefixHash[i].S, prefixHash[i - 1].S)
11
                    };
12
13
14
        pair<int, int> getHashVal() {
15
            return prefixHash.back();
16
17
        pair<int, int> getRangeHashVal(int l, int r) { // 0-based
18
            if(r < 1) return {0, 0};
19
20
            return {
21
                    mul(add(prefixHash[r].F, -(l ? prefixHash[l - 1].F : 0)), inv1[l]),
22
                    mul(add(prefixHash[r].S, -(l ? prefixHash[l - 1].S : 0)), inv2[l])
23
            };
24
25
        pair<int, int> getHashValWithoutRange(int l, int r){ // 0-based
            if(r < l) return getHashVal():</pre>
26
27
28
            auto rem = getRangeHashVal(l, r);
29
            auto hash = getHashVal();
30
31
            return{
32
                add(hash.F, -mul(rem.F, fastPower(P1, l))),
33
                add(hash.S, -mul(rem.S, fastPower(P2, 1)))
34
            };
35
36
        pair<int, int> addRangeFromMeToAnotherHash(pair<int, int> secondHash, int l, int r){ // 0-based
37
            if(r < l) return secondHash;</pre>
38
39
            auto over = getRangeHashVal(l, r);
40
41
42
                add(secondHash.F, mul(over.F, fastPower(P1, l))),
43
                add(secondHash.S, mul(over.S, fastPower(P2, l)))
44
            };
45
46 };
```

Suffix Array

**** Start //

```
1 class SuffixArray {
2  // code
3 };
```

**** Base Code(private) //

```
private:
 2
        string s;
 3
        int n;
        vector<int> p; // Suffix array (sorted suffixes)
 4
 5
        vector<int> c; // Equivalence classes
 6
        vector<int> lcp; // Longest Common Prefix array
 7
 8
        // Counting sort for suffix array construction
 9
        void countSort(vector<int> &p, vector<int> &c) {
             vector<int> cnt(n, 0);
10
             for (auto x : c) cnt[x]++;
11
12
13
             vector<int> p_new(n);
             vector<int> pos(n);
14
15
             pos[0] = 0;
16
             for (int i = 1; i < n; i++)
                 pos[i] = pos[i-1] + cnt[i-1];
17
18
             for (auto x : p) {
19
20
                 int i = c[x]:
21
                 p_new[pos[i]] = x;
22
                 pos[i]++;
23
24
             p = p_new;
25
        }
26
27
        // Kasai's algorithm for LCP array
        void buildLCP() {
28
29
            lcp.resize(n);
30
             int k = 0;
31
             for (int i = 0; i < n-1; i++) {
32
                 int pi = c[i];
33
                 int j = p[pi-1];
34
35
                 while (s[i+k] = s[j+k]) k++;
36
                 lcp[pi] = k:
37
                 k = \max < ll > (k-1, 0);
38
             }
39
```

40

**** Base Code(public) //

```
1 public:
 2
         explicit SuffixArray(string &str, char terminal = '$') : s(str) {
 3
            s += terminal:
            n = s.size();
 5
            p.resize(n);
 6
            c.resize(n);
 7
 8
            // Initial sorting (k=1)
 9
10
                 vector<pair<char, int>> a(n);
11
                 for (int i = 0; i < n; i++)
12
                     a[i] = {s[i], i};
13
14
                 sort(a.begin(), a.end());
15
                 for (int i = 0; i < n; i++)</pre>
16
                     p[i] = a[i].second:
17
18
                 c[p[0]] = 0;
19
                 for (int i = 1; i < n; i++)
                     c[p[i]] = (a[i].first = a[i-1].first) ? c[p[i-1]] : c[p[i-1]]+1;
20
21
22
23
            // Iterative doubling (k=2,4,8,...)
24
            int k = 0;
25
            while ((1 << k) < n) {
26
                 // Shift positions
27
                 for (int i = 0; i < n; i++)</pre>
28
                     p[i] = (p[i] - (1 << k) + n) % n;
29
30
                 countSort(p, c);
31
32
                 vector<int> c_new(n);
33
                 c_new[p[0]] = 0;
34
                 for (int i = 1; i < n; i \leftrightarrow) {
35
                     pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 \ll k)) % n]};
36
                     pair<int, int> curr = \{c[p[i]], c[(p[i] + (1 << k)) \% n]\};
37
                     c new[p[i]] = (curr = prev)? c new[p[i-1]]: c new[p[i-1]]+1;
38
39
                 c = c_new;
40
                 k++;
41
42
43
            buildLCP();
```

Accessors 1 // Accessors 2 const vector<int>& getSuffixArray() const { return p; } 3 const vector<int>& getLCP() const { return lcp; }

**** Utility function //

Print(Suffixes, LCP, Order, Strings, C)

void printSuffixes() {

for (int i = 0; i < n; i++)

2

```
cout \ll p[i] \ll ": " \ll s.substr(p[i]) \ll endl;
 3
 5
    void printLCP() {
        for (int i = 1; i < n; i++)
 7
             cout << lcp[i] << " ";
 8
 9
         cout << endl;
10
    void printOrder() {
        for (int i = 0: i < n: ++i) {
12
             cout \ll p[i] \ll " \ \ "[i = n - 1];
13
14
15
16
    void printStrings() {
17
        for (int i = 0; i < n; ++i) {
18
             cout \ll p[i] \ll ' ' \ll s.substr(p[i]) \ll ' n';
19
20
21
22
23
    void printC(){
24
         for(auto \delta x: this\rightarrow c){
             cout << x << ' ';
25
26
         cout << endl;
27
28
```

**** Applications //

Count Distinct Substrings, Longest Common Substring

```
1
   // Applications
2
        int countDistinctSubstrings() {
 3
            int total = 0;
            for (int i = 1; i < n; i++)
                total += (n - p[i] - 1) - lcp[i];
            return total:
        }
 8
9
        string longestCommonSubstring(string &s1, string &s2) {
10
            string combined = s1 + "$" + s2;
11
            SuffixArray sa(combined, '#');
12
            const vector<int>& sa vec = sa.getSuffixArray();
13
            const vector<int>& lcp_vec = sa.getLCP();
14
15
            int max_len = 0, pos = -1;
16
            int s1_len = s1.size();
17
18
            for (int i = 1; i < sa_vec.size(); i++) {</pre>
                if ((sa_vec[i-1] < s1_len & sa_vec[i] > s1_len) |
19
20
                    (sa_vec[i] < s1_len & sa_vec[i-1] > s1_len)) {
                    if (lcp vec[i] > max len) {
21
                        max_len = lcp_vec[i];
22
23
                        pos = min(sa_vec[i-1], sa_vec[i]);
                    }
24
25
26
27
28
            return (max_len > 0) ? combined.substr(pos, max_len) : "";
29
```

Z Algorithm

```
1  vector<int> z_algo(string s) {
2    vector<int> z(s.size());
3    for(int i = 1, l = 0, r = 0; i < s.size(); i++) {
4        if(i < r) z[i] = min(r - i, z[i - l]);
5        while(i + z[i] < s.size() && s[z[i]] = s[z[i] + i]) z[i]++;
6        if(i + z[i] > r) r = i + z[i], l = i;
7    }
8    return z;
9 }
```

Suffix Automaton

**** Base code //

1 struct suffix automaton {

struct state : map<int, int> {

```
int fail = -1, len{}, cnt = 1;
            bool ed = false:
        };
        int lst, n{};
        vector<state> tr:
        explicit suffix_automaton(const string& s = ""s) : tr(1), lst(0) {
 9
             tr.reserve(s.size() * 2);
10
             for(auto i : s) add(i);
11
12
        void add(int c) {
13
             int x = (int)(tr.size());
14
             tr.emplace_back(), n++;
15
             tr[x].len = tr[lst].len + 1;
16
            int p = lst, q;
17
            while(\sim p \ \delta \theta \ (q = tr[p].emplace(c, x).first \rightarrow second) = x) p = tr[p].fail;
18
            if(p = -1) tr[x].fail = 0;
19
            else {
20
                 if(tr[p].len + 1 = tr[q].len) tr[x].fail = q;
21
22
                     int y = (int)(tr.size()); tr.emplace back(tr[q]);
23
                     tr[y].cnt = 0, tr[y].len = tr[p].len + 1;
                     while(\sim p \ \delta c \ tr[p][c] = q) \ tr[p][c] = y, p = tr[p].fail;
24
                     tr[x].fail = tr[q].fail = y;
25
26
27
28
            lst = x;
29
30
        void init() { // to build cnt, end
             for(int p = lst; ~p; tr[p].ed = true, p = tr[p].fail);
31
32
            vector b(n + 1, vector(0, 0));
             for(int i = 0; i < tr.size(); ++i) b[tr[i].len].push back(i);</pre>
33
34
             for(int l = n; l \ge 1; --l)
35
                 for(int u : b[l])
36
                     tr[tr[u].fail].cnt += tr[u].cnt;
37
38
39
        long long distinct_substrings() {
40
            long long res = 0;
41
             for(int i = 1; i < tr.size(); ++i)</pre>
42
                 res += tr[i].len - tr[tr[i].fail].len;
43
            return res:
44
45 };
```

A Applications //

Longest Common Substring

```
string longest_common_substring(const string& s, const string& t) {
        suffix_automaton sa(s);
 3
        int max_len = 0, pos = 0;
        int current_len = 0, current_state = 0;
         for (int i = 0; i < t.size(); ++i) {</pre>
 5
             while (current_state \neq 0 86 sa.tr[current_state].find(t[i]) = sa.tr[current_state].end())
                current_state = sa.tr[current_state].fail;
 8
                current_len = sa.tr[current_state].len;
 9
10
            if (sa.tr[current_state].find(t[i]) \neq sa.tr[current_state].end()) {
11
                 current_state = sa.tr[current_state][t[i]];
12
                 current len++;
13
                if (current_len > max_len) {
14
                    max len = current len;
                    pos = i - max_len + 1;
15
16
17
18
        return t.substr(pos, max_len);
19
20
```

Longest Palindromic Substring

```
string longest_palindromic_substring(const string& s) {
    string t = s + '#' + string(s.rbegin(), s.rend());
    suffix_automaton sa(t);
    return longest_common_substring(s, string(s.rbegin(), s.rend()));
}
```

Find Occurrences

```
vector<int> find occurrences(const string& s, const string& p) {
        suffix automaton sa(s);
 2
 3
        int state = 0;
        for (char c : p) {
 5
            if (sa.tr[state].find(c) = sa.tr[state].end())
                return {}:
 7
            state = sa.tr[state][c];
 8
 9
        vector<int> occurrences;
        queue<int> q;
10
        q.push(state);
11
12
        while (!q.empty()) {
13
            int u = q.front(); q.pop();
14
            if (sa.tr[u].ed) occurrences.push_back(sa.tr[u].len - p.size());
15
            for (auto [c, v] : sa.tr[u]) q.push(v);
16
17
        return occurrences;
18 }
```

Lex Smallest Substring

```
string lex_smallest_substring(const string& s, int k) {
 2
        suffix automaton sa(s);
 3
        string res;
        int state = 0;
        for (int i = 0; i < k; ++i) {
            for (auto [c, v] : sa.tr[state]) {
                 res += c;
                state = v;
 9
                 break;
10
11
12
        return res;
13
```

Longest Repeated Substring

```
string longest repeated substring(const string& s) {
         suffix automaton sa(s);
 2
        sa.init();
        int max len = 0, state = 0;
         for (int i = 1; i < sa.tr.size(); ++i) {</pre>
             if (sa.tr[i].cnt > 1 & sa.tr[i].len > max_len) {
                 max_len = sa.tr[i].len;
                 state = i:
 9
10
11
         string res;
        while (state \neq 0) {
12
13
            for (auto [c, v] : sa.tr[state]) {
14
                 res += c;
15
                 state = sa.tr[state].fail;
16
                 break;
17
18
         reverse(res.begin(), res.end());
19
20
         return res;
```

21 }

Largest Lex Substring

```
1  vector<int> z_algo(string s) {
2    vector<int> z(s.size());
3    for(int i = 1, l = 0, r = 0; i < s.size(); i++) {
4        if(i < r) z[i] = min(r - i, z[i - l]);
5        while(i + z[i] < s.size() & s[z[i]] = s[z[i] + i]) z[i]++;
6        if(i + z[i] > r) r = i + z[i], l = i;
7    }
8    return z;
9 }
```

Manacher

```
auto manacher(const string &t) {
 2
         string s = "%#";
 3
         s.reserve(t.size() * 2 + 3);
         for(char c : t) s += c + "#"s;
         s += '$';
         // t = aabaacaabaa \rightarrow s = %#a#a#b#a#a#c#a#a#b#a#a#$
         vector<int> res(s.size());
         for(int i = 1, l = 1, r = 1; i < s.size(); i \leftrightarrow ) {
10
             res[i] = max(0, min(r - i, res[l + r - i]));
11
             while(s[i + res[i]] = s[i - res[i]]) res[i]++;
12
             if(i + res[i] > r) {
13
                 l = i - res[i];
14
                 r = i + res[i];
15
16
17
         for(auto &i : res) i--;
18
         return vector(res.begin() + 2, res.end() - 2); // a#a#b#a#a#c#a#a#b#a#a
19
         // get max odd len = res[2 * i]; aba \rightarrow i = b
20
         // get max even len = res[2 * i + 1]; abba \rightarrow i = first b
21 }
```

Aho Corasick

10

14

16

17 18 19

20 21

25

27 28 29

38 39

40 41 42

43

49 50

5.1 52

53

55

62 63 64

65 66

72

73 74

75 76

77

78 79

80

81

82

87

```
struct corasick
          struct node
               array<int, 26> nxt{}, go{};
               vector<int> idx; // all string's indexes have any suffix
               int p, link;
               char ch;
               int cnt = 0;
               explicit node(int p = -1, char ch = '?') : p(p), ch(ch), link(-1) {
                    nxt.fill(-1);
                     go.fill(-1);
           vector<node> tr;
          explicit corasick(vector<string> &v) : tr(1) {
               for(int i = 0; i < v.size(); i++) {
  int x = 0;</pre>
                     for(char c : v[i]) {
    if(tr[x].nxt[c - 'a'] = -1) {
        tr[x].nxt[c - 'a'] = (int)(tr.size());
}
                               tr.emplace_back(x, c);
                          x = tr[x].nxt[c - 'a'];
                     tr[x].idx.push_back(i);
               for(int i = 0; i < tr.size(); i++) {</pre>
                    mxSuffix(i);
          int plus(int x, char c) {
    if(tr[x].go[c - 'a'] = -1) {
        if(tr[x].nxt[c - 'a'] ≠ -1)
        tr[x].go[c - 'a'] = tr[x].nxt[c - 'a'];
}
                     else
                         tr[x].go[c - 'a'] = x = 0? 0: plus(mxSuffix(x), c);
               return tr[x].go[c - 'a'];
          int mxSuffix(int x) {
               if(tr[x].link = -1) {
    if(!x || !tr[x].p)
        tr[x].link = 0;
                     else
                         tr[x].link = plus(mxSuffix(tr[x].p), tr[x].ch);
                     mxSuffix(tr[x].link);
                     tr[x].idx.reserve(tr[x].idx.size() + tr[tr[x].link].idx.size());
                     for(int y : tr[tr[x].link].idx)
    tr[x].idx.push_back(y);
               return tr[x].link:
          vector<int> match(const string &text, int n) {
               vector<int> pattern_count(n);
               int state = 0;
               for(char c : text) {
   state = plus(state, c);
                     tr[state].cnt++;
               vector<int> order(tr.size());
               iota(order.begin(), order.end(), 0);
sort(order.begin(), order.end(), [8](int a, int b) {
    return depth(a) > depth(b);
               3):
               for(int i : order) {
   if(tr[i].link ≠ -1)
                          tr[tr[i].link].cnt += tr[i].cnt;
               for(int i = 0; i < tr.size(); i++) {
    for(int pat : tr[i].idx) {</pre>
                         pattern_count[pat] += tr[i].cnt;
               return pattern_count;
          7
          int depth(int x) {
               int d = 0;
               while(x \neq 0) {
                    x = tr[x].p;
                     ++d;
               return d;
          }
88 };
```