asifcse10 / **abuasifkhan.me**

Branch: gh-pages ▾ abuasifkhan.me / _posts / 2015-06-13-aho-corasick.md Find file Copy path


mdminhazulhaque 2015-09-23 16:17:34 6c2c856 on Sep 23, 2015

1 contributor

`206 lines (155 sloc) 13.8 KB`

layout	title	date	permalink
post	Aho-Corasick দিয়ে String Matching	2015-06-13 05:00:00 -0700	/aho-corasick.html

Aho-Corasick একটা স্ট্রিং ম্যাচিং/সার্চিং অ্যালগরিদম। এই টেকনিক দিয়ে একটা স্ট্রিং-এ কিছু সাবস্ট্রিং কতবার আছে সেটা খুজে বের করা যায়। এখন তুমি ভাবতে পারো সাব-স্ট্রিং সার্চ বা কাউন্ট করার জন্য তো Knuth-Morris-Prat (KMP) algorithm আছেই, তাইলে এটার কি দরকার। হ্যা, KMP খুবই efficient একটা অ্যালগরিদম (O(n+m)) যদি একটি সাবস্ট্রিং সার্চ করা বা কাউন্ট করা হয়। যদি k সংখ্যক সাবস্ট্রিং সার্চ করতে যাও তাহলে কমপ্লেক্সিটি অনেক বেড়ে যাবে O(k*(n+m)). Aho-Corasick এই দিক দিয়ে অনেক efficient (O(k+m+n)). Aho-Corasick এ একটা Word থাকে, এবং একাধিক Dictionary থাকে। Word এর মধ্যে Dictionary গুলো আছে কিনা, থাকলে কতবার আছে এই সংক্রান্ত তথ্য খুবই সহজেই বের করতে পারবে। LightOj এ একটা প্রবলেম আছে 1427 - Substring Frequency (II). এটা সলভ করতে করতে Aho-Corasick শেখার চেষ্টা করবো।

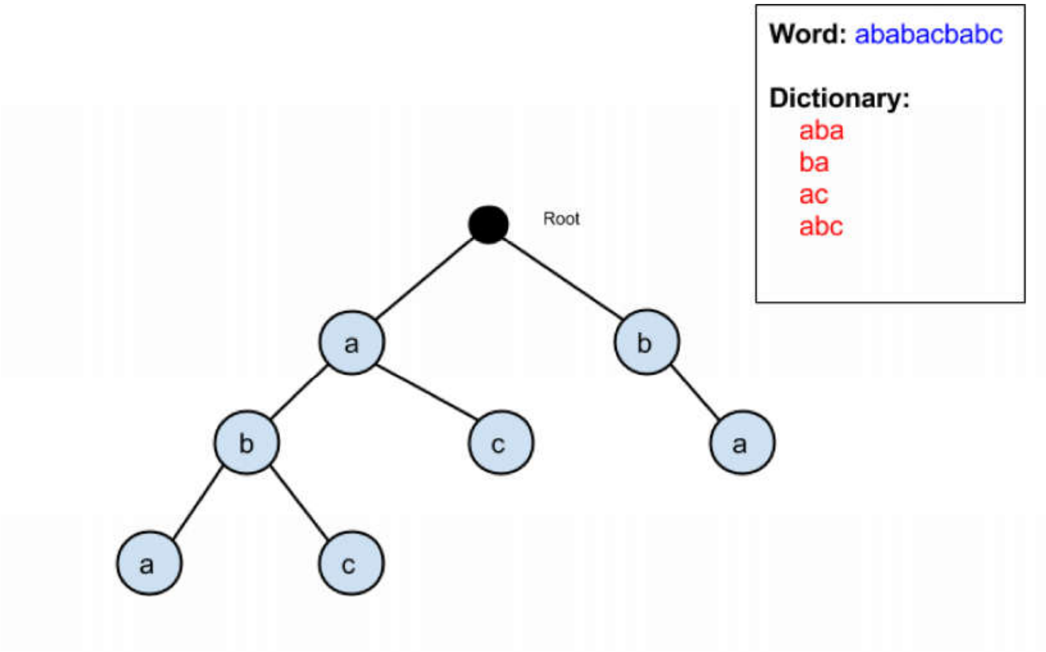
প্রবলেমে T একটা স্ট্রীং, এবং n টা কোয়েরি আছে। প্রতিটা কোয়েরিতে P_i একটা স্ট্রীং। তোমাকে খুজতে হবে T তে P_i কতবার আছে। প্রথমে Naive ভাবে চিন্তা করলে প্রতিটা সাবস্ট্রিং এর জন্য len^2 অর্থাৎ সবগুলো সাবস্ট্রিং এর জন্য $n*len^2$ কমপ্লেক্সিটিতে হয়ে যাওয়ার কথা। কিন্তু সমস্যা তখন হয়ে যাবে যদি $|T|<=10^6$ এবং n<=10 হয় এবং $| P_i |<=500$ হয়।

যারা Knuth-Morris-Pratt (KMP) অ্যালগরিদম জানো তারা হয়তো এই লিমিট দেখে খুশি হয়ে গেছো যে এটা দিয়ে খুব easily $O(n*(len+|p_i|))$ complexity দিয়ে এটা সলভ করে ফেলতে পারবে।

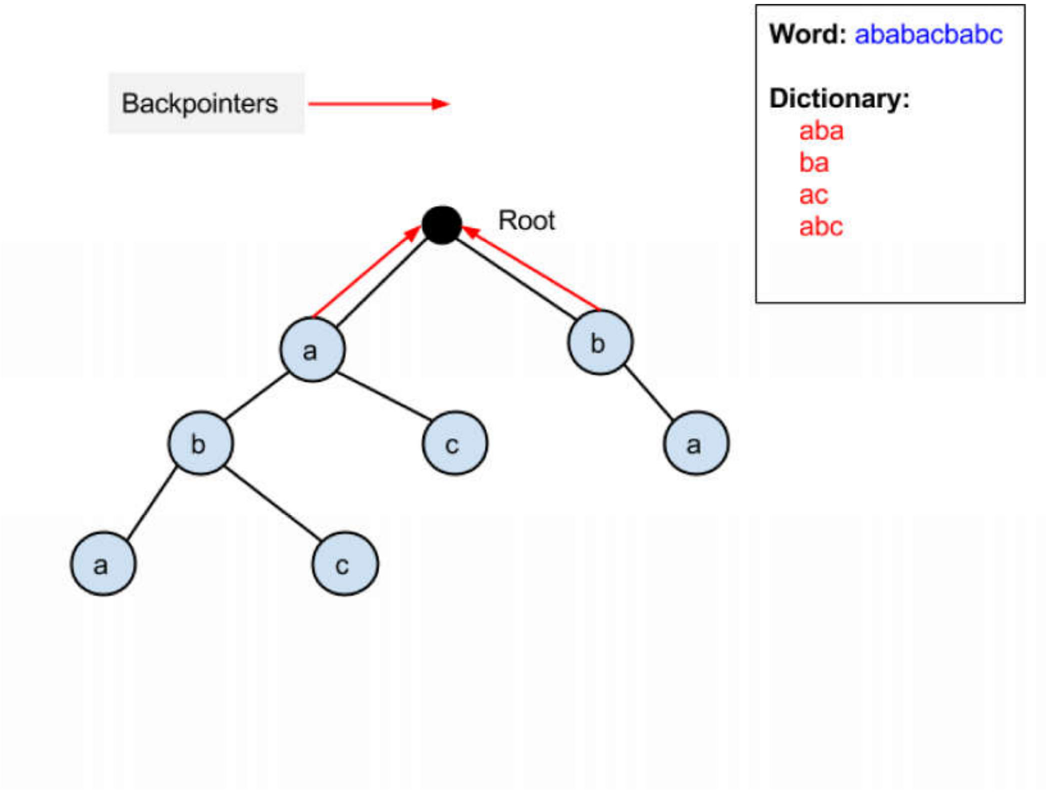
এবার আসল প্রবলেমে আসি যেখানে $|T|<=10^6$ এবং n<=500 হয় এবং $| P_i |<=500$. এক্ষেত্রে আর KMP দিয়ে করা সম্ভব হবে না, TLE খেয়ে যাবে। এই প্রবলেম সলভ করতে গেলে Aho-Corasick টেকনিকটা জানতে হবে। এবার দেখা যাক এই টেকনিকটা কিভাবে কাজ করে।

এটা শিখতে গেলে কিভাবে Trie/Prefix Tree বানাতে হয় সেটা শিখতে হবে তোমাকে। ডাটা স্ট্রাকচার: ট্রাই (প্রিফিক্স ট্রি/রেডিক্স ট্রি) শাফায়েত ভাইয়ের ব্লগ থেকে কিভাবে Trie/Prefix Tree বানাতে হয় সেটা শিখে আসো যদি না জানা থাকে।

ধরো T/Word=”ababacbabc” এবং Queries/ Dictionary: “aba”, “ba”, “ac”, “abc”.



প্রথম ধাপ: Query গুলো নিয়ে একটা প্রিফিক্স ট্রী বানিয়ে ফেলো। যেখানে Root এর immediate adjacent nodes এর backpointer Root হবে। আমি লাল রঙের লাইন দিয়ে backpointer বোঝাতে চেয়েছি। Backpointer কেন লাগবে সেটাতে পরে আসতেছি।

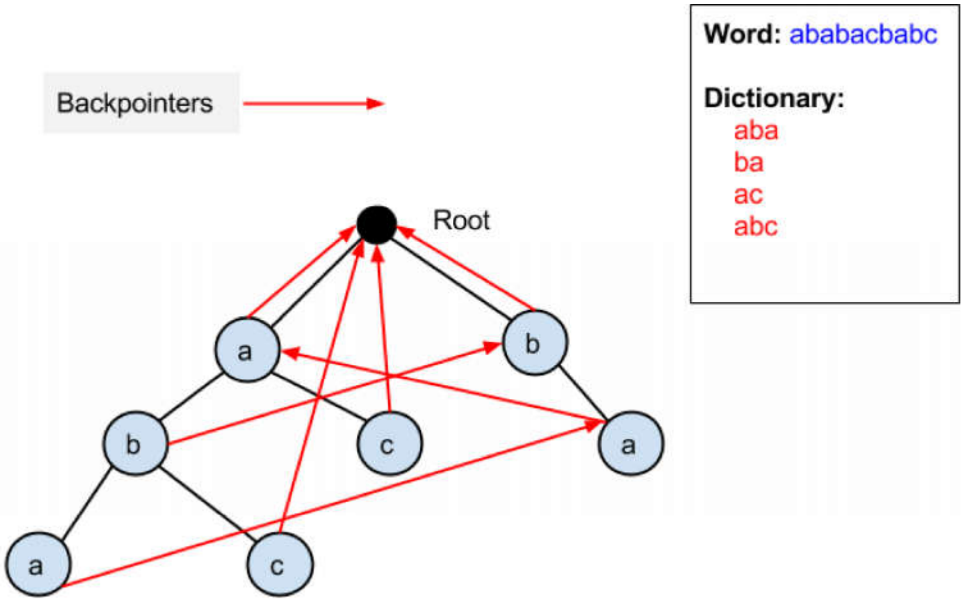


দ্বিতীয় ধাপ: এই নোডগুলো queue তে পুশ করে বাকিসবগুলো নোডের জন্য BFS চালাতে হবে এখন। এখানে কিছু কন্ডিশন মাথায় রাখতে হবে।

1. বর্তমানে নোডের প্যারেন্টের Backpointer এ যদি নোডে যে ক্যারেক্টার আছে সেটাই থেকে থাকে তাহলে বর্তমান নোডের ব্যাকপয়েন্টার প্যারেন্টের ব্যাকপয়েন্টার হবে।

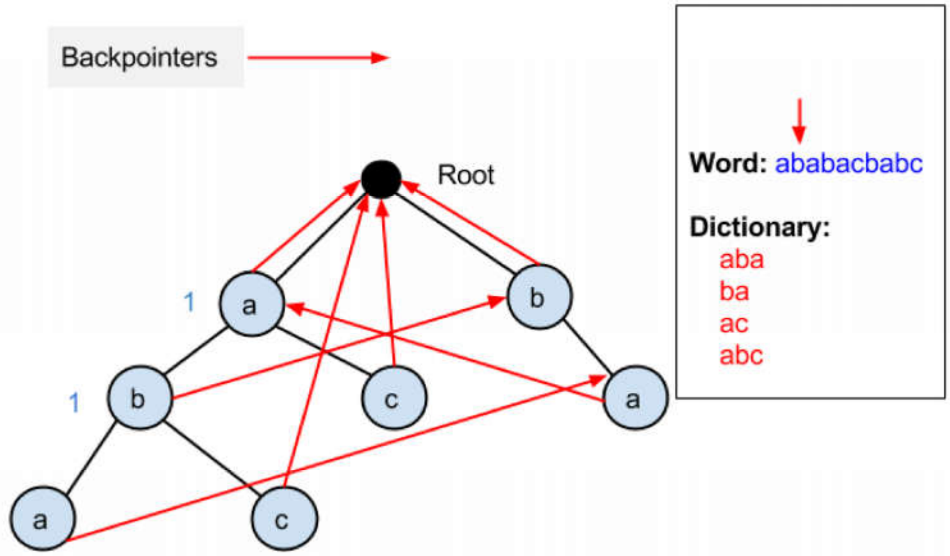
2. যতক্ষন পর্যন্ত প্যারেন্টের Backpointer এর ক্যারেক্টার বর্তমান নোডের ক্যারেক্টারের সমান না হবে ততক্ষন প্যারেন্টটি বর্তমান প্যারেন্টের Backpointer হয়ে root এর কাছাকাছি iterate করে যেতে থাকবে।
3. যদি বর্তমান প্যারেন্ট root হয়ে থাকে এবং নোডটির ক্যারেক্টার root এর কোন child এ পাওয়া যায় তাহলে নোডটির Backpointer ওই child হবে। যদি না পাওয়া যায় তাহলে root ই হবে ওই নোডের Backpointer।

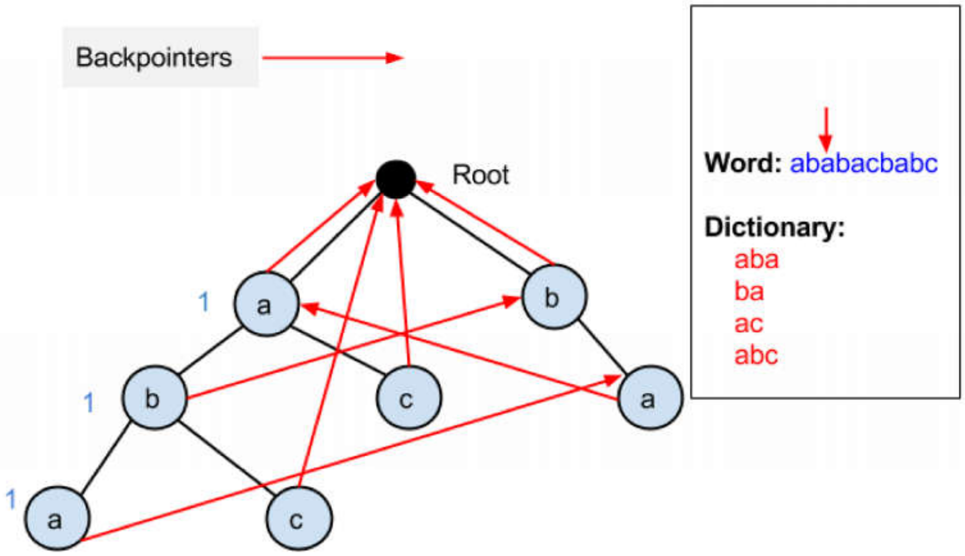
এভাবে Trie টি সাজালে উপরের ছবিটা দেখতে কিছুটা এমন হবে।

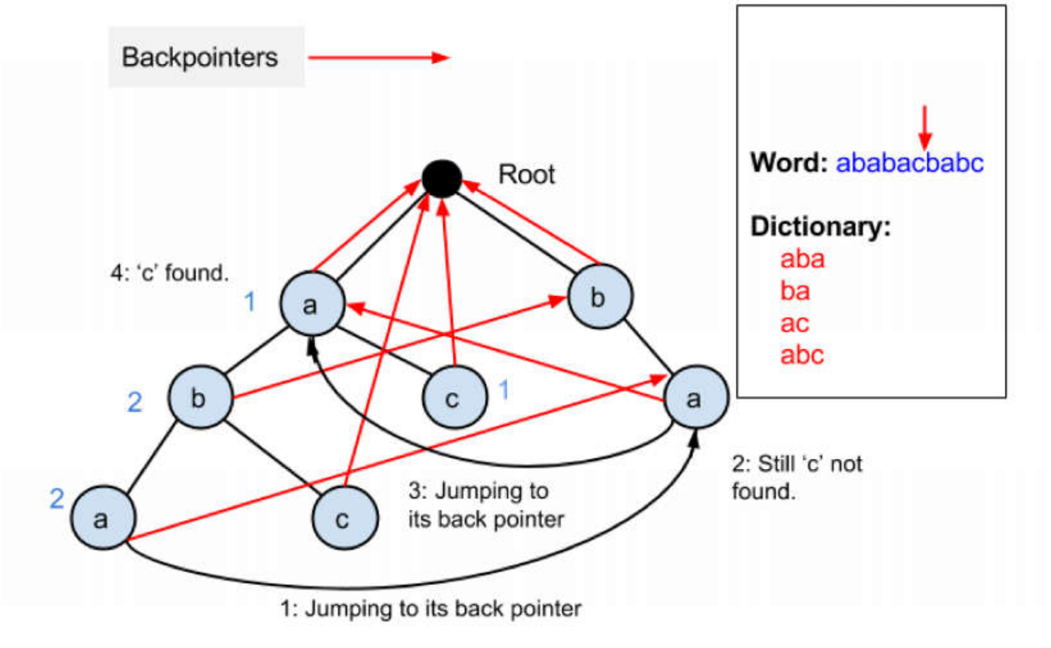


তৃতীয় ধাপ: Word বা T কে linearly scan করতে হবে। এক্ষেত্রেও কিছু কন্ডিশন আছে...

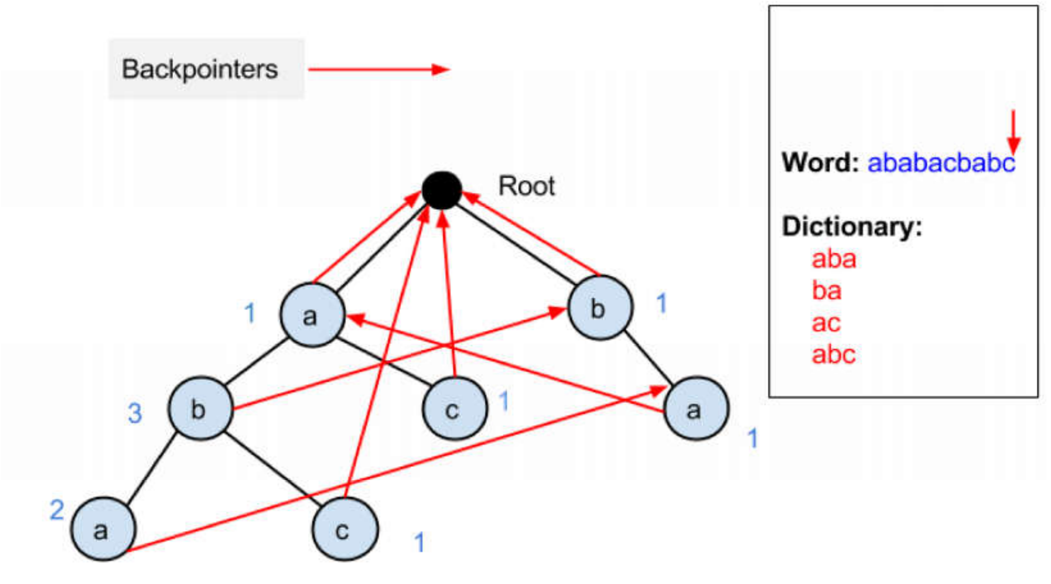
1. যদি ক্যারেক্টারটি root এর কোন child এ পাওয়া যায় তাহলে সেখানে যাও এবং ওই নোডের count এক বাড়িয়ে দাও এবং T এর পরের ক্যারেক্টারে চলে যাও। যতবার এই নোডে আসা লাগবে ততবার count increment হবে। না পাওয়া গেলে root এই থাকো।
2. পরের ক্যারেক্টারটা যদি বর্তমান নোডের কোন child এ পাওয়া যায় তাহলে ওই child এর count increment করে child এ চলে যাও এবং T এর next character এ চলে যাও। যদি না পাওয়া যায় তাহলে বর্তমান নোডটির Backpointer এ যাও এবং দেখো তার কোন child এ এই ক্যারেক্টার আছে কিনা। ক্যারেক্টার পাওয়া গেলেই ওই নোডের count বাড়িয়ে দিবে।


Backpointers
Root
1
a
b
1
b
c
a
a
c
Word: ababacbabc
Dictionary:
aba
ba
ac
abc

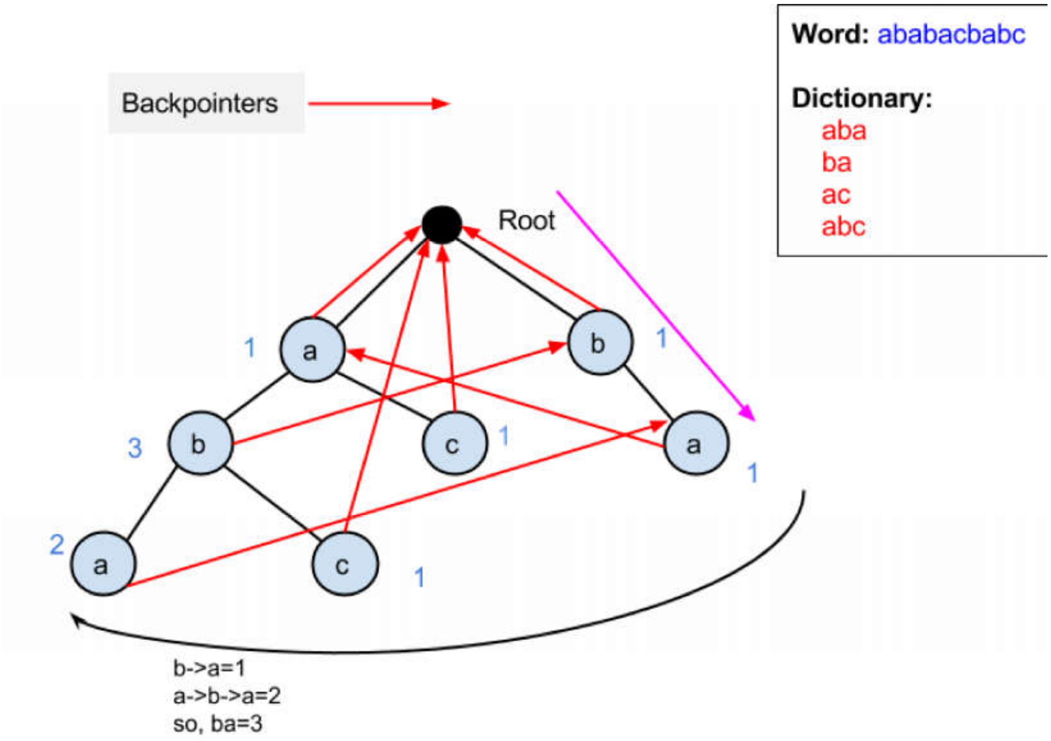

Backpointers
Root
1
a
b
1
b
c
a
1
a
c
Word: ababacbabc
Dictionary:
aba
ba
ac
abc



এভাবে T/word এর সবগুলো ক্যারেক্টার প্রোসেস করা হয়ে গেলে Trie টা কিছুটা এমন দেখাবে।



চতুর্থ ধাপ: Backpointer গুলোর ব্যাপারে কিছু জিনিস লক্ষ্য করো। ধরো তুমি “ba” কতবার আছে সেটা জানতে চাচ্ছো। তাহলে তোমাকে প্রথমে Trie দিয়ে b->a পথে যেতে হবে। b->a পথে a এর count=1. সুতরাং T/word এ একবার ba অবশ্যই আছে। এবার দেখবো যে বর্তমান a তে কোন কোন backpointer point করে আছে। দেখা যাচ্ছে a->b->a পথের a এর backpointer বর্তমান নোডকে পয়েন্ট করছে তাই a->b->a পথের a তে চলে যাও তুমি কারন ba খুজতে গেলে a->b->a পথে অবশ্যই ba থাকবে। এখানে আবার count=2, তারমানে এই পথে ba দুইবার দেখা গিয়েছে। তাই মোট ba এর সংখ্যা হবে 1+2=3. বর্তমান a কে কোন backpointer point করছে না সুতরাং আর কোথাও যাওয়া লাগবে না। এভাবে প্রতিটা query এর জন্য dfs চালিয়ে অনেক সহজেই substring frequency বের করতে পারবে।



সবগুলো কোয়েরির জন্য Complexity: O(n * $|P_i|$).

Code:

```
int m, n, res;
typedef pair<int, int> Point;

struct NODE {
    int cnt;
    bool vis;
    NODE *next[27];
    vector <NODE *> out;
    NODE() {
        for(int i = 0; i < 27; i++) {
            next[i] = NULL;
        }
        out.clear();
        vis = false;
        cnt=0;
    }

    ~NODE() {
        for(int i = 1; i < 27; i++)
            if(next[i] != NULL && next[i] != this)
                delete next[i];
    }
}*root;

void buildtrie(char dictionary[][MX],int n) {    // processing the dictionarytionary

    root = new NODE();

    /*usual trie part*/
    for(int i = 0; i < n; i++) {
        NODE *p = root;
        for(int j = 0; dictionary[i][j] ; j++) {
            char c = dictionary[i][j]- 'a' + 1;
            if(!p->next[c])
                p->next[c] = new NODE();
            p = p->next[c];
```

```
        }
    }

    /* Pushing the nodes adjacent to root into queue */
    queue <NODE *> q;
    for(int i = 0; i < 27; i++) {
        if(!root->next[i])
            root->next[i] = root;
        else {
            q.push(root->next[i]);
            root->next[i]->next[0] = root;  // ->next[0] = back Pointer

        }
    }

    /* Building Aho-Corasick tree */
    while(!q.empty()) {
        NODE *u = q.front();    // parent node
        q.pop();

        for(int i = 1; i < 27; i++) {
            if(u->next[i]) {
                NODE *v = u->next[i];   // child node
                NODE *w = u->next[0];   // back pointer of parent node
                while( !w->next[i] )    // Until the char(i+'a'-1) child is found
                    w = w->next[0];     // go up and up to back pointer.

                v->next[0] = w = w->next[i];  // back pointer of v will be found child above.
                w->out.push_back(v);    // out will be used in dfs step.
                                        // here w is the new found match node.
                q.push(v);              // Push v into queue.
            }
        }
    }
}

void aho_corasick(NODE *p, char *word) {    // Third step, processing the text.
    for(int i = 0; word[i]; i++) {
        char c = word[i]-'a'+1 ;
        while(!p->next[c])
            p = p->next[0];
        p = p->next[c];
        p->cnt++;
    }
}

int dfs( NODE *p ) {    // DFS for counting.
    if(p->vis) return p->cnt;
    for(int i = 0; i < p->out.size(); i++)
        p->cnt += dfs(p->out[i]);
    p->vis = true;
    return p->cnt;
}

char query[1000100];
char dictionary[MX][MX];

int main() {
    int t, tc, y, z;
    int i, j, k, l, h;
    char ch;
    scanf ("%d", &tc);
    for (t = 1; t <= tc; t++) {
        int n;
        scanf("%d",&n);

        scanf("%s",query);

        for (int i=0; i<n; ++i) {
            scanf("%s",dictionary[i]);
        }
```

```
            buildtrie(dictionary, n);

            aho_corasick(root, query);

            printf("Case %d:\n",t);

            for(int i = 0; i < n; i++) {
                NODE *p = root;
                for(int j = 0; dictionary[i][j]; j++) {
                    char c = dictionary[i][j] -'a' +1;
                    p = p->next[c];
                }
                printf("%d\n", dfs(p));
            }
            delete root;
        }

        return 0;
    }
```

Aho-Corasick দিয়ে এই প্রবলেমগুলো সলভ করতে পারো:

I Love Strings!!

Word Puzzles

Emoticons

Substring Problem

Keep Coding ... :)