Pontificia Universidad Católica del Perú - FCI

XieXieLucas Notebook - Froz/Phibrain/Ands

November 9, 2017

Contents

1 Aho Corasick + Compression

```
// Aho Corasick automaton. O(n) in size of Trie.
// Allows searching for a dictionary of patterns in a string.
// Consider using DP[u, pos], for instance.

const int MAXN = 500000; // Sum of words*length
const int SZA = 26;           // Alphabet size

map<int,int> adj[MAXN]; // Trie
int isEnd[MAXN];      // Example: How many words end at node u
int gid;                              // Id of last node set
int f[MAXN];          // Aho Corasick failure function

void init(int id){
        isEnd[id] = 0;
        adj[id].clear();
}

void add(string s){
        int u = 0;     // Current node
        REP(p,0,sz(s)){
                int id = s[p] - 'a';
                if (!adj[u].count(id)){
                        adj[u][id] = ++gid;   // Lazy initialization
                        init(gid);
                }
                u = adj[u][id];
        }
        isEnd[u]++;
}

void build(){
   // BFS-DP Aho Corasick construction
        queue<int> q;
        f[0] = 0;
        REPIT(it, adj[0]){
                int u = it->snd;
                q.push(u);
                f[u] = 0;
        }
        while (!q.empty()){
                int e = q.front();
                q.pop();
                REPIT(it, adj[e]){
                        int i = it->fst;
                        int u = it->snd;
                        q.push(u);
```

```
                        int v = f[e];
                        while (v && !adj[v].count(i)) v = f[v];
                        f[u] = (adj[v].count(i) ? adj[v][i] : 0);
                        // Aggregate necessary information here
                        // In general, S[u] += S[f[u]]
                        isEnd[u] += isEnd[f[u]];
                }
        }
}

// Search string s for all strings in trie
ll search(string s){
        ll ans = 0;
        int u = 0;
        REP(p,0,sz(s)){
                int id = s[p] - 'a';
                while (u && !adj[u].count(id)) u = f[u];
                if (adj[u].count(id)) u = adj[u][id];
                ans += isEnd[u];
        }
        return ans;
}

int main(){
        gid = 0;
        init(0);
        // Ready for add(s), build(), search(t)
        return 0;
}
```

2 Aho Corasick

```
// ----------------------------aho corasick-------------------------
// cantidad de repeticiones de cada string sobre un text en O(M+N)

#define N 100000 // ñtamao del text
#define M 1005 //ñtamao de cada string a buscar

ll n;
char text[N];// string donde buscar
char buf[N]; // string a buscar
ll cnt[M]; // cnt[i]: cantidad de ocurrencias del string i

ll root, nodes;
// nodes: cantidad de nodos en el trie,
//root: que nodo del trie estoy
struct trieNode{
    bool seen;
    ll matchFail,fail;
    vi matches;
    map< char, ll > next;
    trieNode(){}
    trieNode(bool seen, ll &matchFail, ll &fail, vi & matches, map<char,
        ll> & next):
    seen(seen), matchFail(matchFail),fail(fail),matches(matches), next(
        next){}
} trie[N];
// antes de insertar, notar que root=0 y nodes=1
inline void insert(char * s, ll wordId){ //
    //wordId: id del string
    ll x = root, ta=strlen(s);
    REP(i,0,ta){
        ll &nxt = trie[x].next[ s[i] ];
        if (!nxt) nxt = ++nodes;
        x = nxt;
    }
    trie[x].matches.push_back(wordId);
}

inline ll find(ll x, char ch){
    while (x && !trie[x].next.count(ch)) x = trie[x].fail;
    return x ? trie[x].next[ch] : root;
}

inline void bfs(){
    trie[root].fail = 0;
    queue< ll > q;
    q.push(root);
    while(q.empty()){
        ll u = q.front(),v; q.pop();
        char ch;
        for (auto &it: trie[u].next){
            ch = it.fst, v = it.snd;
            ll f = find(trie[u].fail, ch);
            trie[v].fail = f;
            trie[v].matchFail = trie[f].matches.empty() ? trie[f].matchFail
                 : f;
```

```
            q.push(v);
        }
    }
}
inline void search(){
    ll x = root;
    ll ta=strlen(text);
    REP(i,0,ta){
        x = find(x, text[i]);
        for (ll t = x; t && !trie[t].seen; t = trie[t].matchFail){
            trie[t].seen = true;
            REP(j,0, sz(trie[t].matches)) cnt[trie[t].matches[j]] ++;
        }
    }
}

int main(){
    root = ++nodes;//inicializacion
    scanf( "%s", &text );
    scanf( "%d", &n );
    REP(i,0, n){
        scanf( "%s", &buf );
        insert(buf, i);
    }
    bfs(); search();
    REP(i,0,n) printf( "%s\n", cnt[i]>0 ? "Y" : "N" );

    return 0;
}
```

3 Knuth Morris Pratt

```
// KMP algorithm for finding a pattern in a string in O(n+m).

const int MAX = 1000000;

int b[MAX]; // Fail function
char p[MAX]; // Pattern string
char t[MAX]; // Text string

int n; // Text string length
int m; // Pattern string length

void kmpPreprocess(){
    int i=0, j=-1;
    b[i]=j;
    while (i<m){
        while (j>=0 && p[i]!=p[j]) j=b[j];
        i++; j++;
        b[i]=j;
    }
}

void report(int x){
        cout << "Found on: " << x << endl;
}

void kmpSearch(){
    int i=0, j=0;
    while (i<n){
        while (j>=0 && t[i]!=p[j]) j=b[j];
        i++; j++;
        if (j==m){
            report(i-j);
            j=b[j];
        }
    }
}
```

4 Manacher Algorithm

```
// Manacher's algorithm for finding all palindromes
// in a string in O(n).

int n;
char s[200200];
char aux[100100];
int p[200200];

int main(){
        scanf("%s%n", aux, &n);
        s[0] = '^';
        s[1] = '#';
        REP(i,0,n){
```

```
		s[2*i+2] = aux[i];
		s[2*i+3] = '#';
	}
	s[2*n+2] = '\0';
	int c = 0, r = 0;
	REP(i,0,2*n+2){
		if (i > r) p[i] = 0;
		else p[i] = min(r-i, p[2*c-i]);
		while (s[i+p[i]+1] == s[i-p[i]-1]) p[i]++;
		if (i + p[i] > r){
			c = i;
			r = i + p[i];
		}
	}

	printf("%s\n", s);
	REP(i,0,2*n+2) {
		printf("%d", p[i]);
	}
	printf("\n");
	return 0;
}
```

5 Palindromic Tree

```
// adamant's palindromic tree online O(n*log(|E|)) construction
// Tutorial: http://adilet.org/blog/25-09-14/
// Add/Delete operation can be supported in O(logn) by doing
// check(link[v]), v = slink[v] in get_link
// (periodicity -> same initial char)
const int maxn = 5e5, sigma = 26, INF = 1e9;
int s[maxn], len[maxn], link[maxn], to[maxn][sigma];
int n, last, sz;
// All these optional (palindromic factoring)
int d[maxn], slink[maxn], dpe[maxn], dpo[maxn];
int anse[maxn], anso[maxn], prve[maxn], prvo[maxn];

void init(){ // Call with n=0
	s[n++] = -1;
	link[0] = 1;
	len[1] = -1;
	sz = 2;
	anse[0] = 0;
	anso[0] = INF;
}

int get_link(int v){
	while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
	return v;
}

ii getmin(int v, int* ans, int* dp, int* prv){
	dp[v] = ans[n - (len[slink[v]] + d[v]) - 1];
	int best = n - (len[slink[v]] + d[v]) - 1;
	if (d[v] == d[link[v]]){
		if (dp[v] > dp[link[v]]){
			dp[v] = dp[link[v]];
			best = prv[n-1-d[v]];
		}
	}
	return mp(dp[v] + 1, best);
}

void add_letter(int c){
	s[n++] = c;
	last = get_link(last);
	if(!to[last][c]) {
		len [sz] = len[last] + 2;
		link[sz] = to[get_link(link[last])][c];
		d[sz] = len[sz] - len[link[sz]];
		if (d[sz] == d[link[sz]]) slink[sz] = slink[link[sz]];
		else slink[sz] = link[sz];
		to[last][c] = sz++;
	}
	last = to[last][c];

	anse[n-1] = INF;
	for (int v = last; len[v] > 0; v = slink[v]){
		ii acte = getmin(v, anso, dpe, prve);
		if (act.fst < anse[n-1]){
			anse[n-1] = act.fst;
			prve[n-1] = act.snd;
		}
	}

	anso[n-1] = INF;
	for (int v = last; len[v] > 0; v = slink[v]){
```

```
            ii act = getmin(v, anse, dpo, prvo);
            if (act.fst <= anso[n-1]){
                anso[n-1] = act.fst;
                prvo[n-1] = act.snd;
            }
        }
}
```

6 Suffix Array

```
// ---------------------------Suffix array--------------------------
// construccion en nlog^2(n)
//usa lcp(x,y)=mi[lcp(x,x+1),lcp(x+1,x+2)....lcp(y-1,y)]
//construye el lcp(x,y) con sparce table, notar que los indices son 0 base
//s=ababa
//s1[0]=ababa,s1[1]=baba,s1[2]=aba, s1[3]=ba,s1[4]=a, s1[5]='$'
//s2={$,a,aba,ababa,ba,baba}={5,4,2,0,3,1}=r
//r[i] lista de los sufijos ordenados en 0 base
//indice de s1={ababa,baba,aba,ba,a,$}={3,5,2,4,1,0}=p
//p[i] posicion del i substring en el suffix array (s1) en 0 base

#define N 100010
#define M 20
inline ll ma(ll a, ll b){ return ((a-b>0)? a:b);}
inline ll mi(ll a, ll b){return ((a-b>0)? b:a);}

struct SA{
  //asignar s:string(char), n ñtamao del string
  ll n,t;
  ll p[N],r[N],h[N];
  char s[N];
  ll rmq[M][N];
  ll flog2[N];
  inline void fix_index(ll b, ll e){
    ll lastpk, pk, d;
    lastpk = p[r[b]+t];
    d = b;
    REP(i,b,e){
      if (((pk = p[r[i]+t]) != lastpk) && (b > lastpk || pk >= e)){
        lastpk = pk;
        d = i;
      }
      p[r[i]]= d;
    }
  }
  //calculo de r y p
  inline void suff_arr(){
    s[n++] = '$';
    ll bc[256];
    REP(i,0,256) bc[i]=0;
    REP(i,0,n) bc[(ll)s[i]]++;
    REP(i,1,256) bc[i] += bc[i-1];
    RREP(i,n-1,0) r[--bc[(ll)s[i]]] = i;
    RREP(i,n-1,0) p[i] = bc[(ll)s[i]];
    for (t = 1; t < n; t<<=1){
      for (ll i = 0, j = 1; i < n; i = j++){
        while (j < n && p[r[j]] == p[r[i]]) ++j;
        if (j-i > 1){
          sort(r+i, r+j, [&](const ll &i, const ll &j){return p[i+t] < p[j+
              t];});
          fix_index(i, j);
        }
      }
    }
  }
  //calcula h[i] en O(n) usando Kasai algorithm
  inline void initlcp(){
    ll tam = 0, j;
    REP(i,0,n-1){
      j = r[p[i]-1];
      while(s[i+tam] == s[j+tam]) ++tam;
      h[p[i]-1] = tam;
      if (tam > 0) --tam;
    }
  }
  //construccion del RMQ para hallar lcp en un rango
  inline void makelcp(){
    initlcp();
    REP(i,0,n-1) rmq[0][i] = h[i];
    ll lg = 0, pw = 1;
    do{
      REP(i,pw,pw*2) flog2[i] = lg;
      lg++; pw*=2;
      REP(i,0,n-1){
        if (i+pw/2 < n-1) rmq[lg][i] = mi(rmq[lg-1][i], rmq[lg-1][i+pw/2]);
        else rmq[lg][i] = rmq[lg-1][i];
      }
```

```
    } while(pw < n);
  }
  //calcula el lcp en [i,j] de s1(suffix array);
  inline ll lcp(ll i, ll j){
    if (i == j) return n - r[i] - 1;
    ll lg = flog2[j-i], pw = (1<<lg);
    return mi(rmq[lg][i], rmq[lg][j-pw]);
  }
  //limpia y construye
  inline void build(){
    memset(p,0,sizeof(p));
    memset(r,0,sizeof(r));
    memset(h,0,sizeof(h));
    memset(rmq,0,sizeof(rmq));
    memset(flog2,0,sizeof(flog2));
    suff_arr();
    makelcp();
  }
};
int main(){
  //ejemplo, hallar la cantidad de diferentes substrings para t1 strings;
  ll t1; scanf("%lld", &t1);
  REP(ik,0,t1){
    SA sa; scanf("%s", &sa.s);
    ll ta=strlen(sa.s);
    sa.n=ta; sa.build();
    ll ans=0;
    REP(i,1,ta){
        ans+=sa.lcp(i,i+1);
    }
    ll xd=(ta*(ta+1)/2)-ans;
    printf("%lld\n",xd);
  }
  return 0;
}
```

7 Suffix Automaton

```
// O(n) Online suffix automaton construction
// len[u]: Max length of a string accepted by u
// link[u]: Suffix link of u
// Link edges give the suffix tree of reverse(s)
// Terminal nodes can be obtained by
//     traversing last's links

const int MAX = 1000000;
int len[MAX*2];
int link[MAX*2];
map<char,int> adj[MAX*2];
int sz, last;

// To reuse, clear adj[]
void sa_init() {
        sz = last = 0;
        len[0] = 0;
        link[0] = -1;
        sz++;
}

void sa_extend (char c) {
        int cur = sz++;
        len[cur] = len[last] + 1;
        int p;
        for (p=last; p!=-1 && !adj[p].count(c); p = link[p])
                adj[p][c] = cur;
        if (p == -1)
                link[cur] = 0;
        else {
                int q = adj[p][c];
                if (len[p] + 1 == len[q])
                        link[cur] = q;
                else {
                        int clone = sz++;
                        len[clone] = len[p] + 1;
                        adj[clone] = adj[q];
                        link[clone] = link[q];
                        for (; p != -1 && adj[p][c] == q; p = link[p])
                                adj[p][c] = clone;
                        link[q] = link[cur] = clone;
                }
        }
        last = cur;
}
```

8 Z-Algorithm

```
//Zfun(i) devuelve la longitud del maximo prefijo que empieza en i
vi Zfun(string s){
	vi Z(s.sz,0);
	int l = 0, r = 0;
	REP(i,1,sz(s)){
		if ( i<=r ) Z[i] = min(Z[i-l], r-i+1);
		while ( i+Z[i]<s.sz and s[i+Z[i]]==s[Z[i]] ) Z[i]++;
		if ( i+Z[i]-1>r ) l = i, r = i+Z[i]-1;
	}
	return Z;
}
```