Suffix Tree. Ukkonen's Algorithm

Table of Contents

This article is a stub and doesn't contain any descriptions. For a description of the algorithm, refer to other sources, such as Algorithms on Strings, Trees, and Sequences by Dan Gusfield.

This algorithm builds a suffix tree for a given string s of length n in $O(n\log(k))$ time, where k is the size of the alphabet (if k is considered to be a constant, the asymptotic behavior is linear).

The input to the algorithm are the string s and its length n, which are passed as global variables.

The main function `build_tree` builds a suffix tree. It is stored as an array of structures `node`, where `node[0]` is the root of the tree.

In order to simplify the code, the edges are stored in the same structures: for each vertex its structure `node` stores the information about the edge between it and its parent. Overall each `node` stores the following information:

- `(l, r)` - left and right boundaries of the substring `s[l..r-1]` which correspond to the edge to this node,
- `par` - the parent node,
- `link` - the suffix link,
- `next` - the list of edges going out from this node.

```
string s;
int n;

struct node {
    int l, r, par, link;
    map<char,int> next;

    node (int l=0, int r=0, int par=-1)
        : l(l), r(r), par(par), link(-1) {}
    int len()  {  return r - l;  }
    int &get (char c) {
        if (!next.count(c))  next[c] = -1;
        return next[c];
    }
};
node t[MAXN];
int sz;

struct state {
    int v, pos;
    state (int v, int pos) : v(v), pos(pos)  {
};
state ptr (0, 0);
```

```
state go (state st, int l, int r) {
    while (l < r)
        if (st.pos == t[st.v].len()) {
            st = state (t[st.v].get( s[l] ), 0
            if (st.v == -1)  return st;
        }
        else {
            if (s[ t[st.v].l + st.pos ] != s[l
                return state (-1, -1);
            if (r-l < t[st.v].len() - st.pos)
                return state (st.v, st.pos + r
            l += t[st.v].len() - st.pos;
            st.pos = t[st.v].len();
        }
    return st;
}

int split (state st) {
    if (st.pos == t[st.v].len())
        return st.v;
    if (st.pos == 0)
        return t[st.v].par;
    node v = t[st.v];
    int id = sz++;
    t[id] = node (v.l, v.l+st.pos, v.par);
    t[v.par].get( s[v.l] ) = id;
    t[id].get( s[v.l+st.pos] ) = st.v;
    t[st.v].par = id;
    t[st.v].l += st.pos;
    return id;
}
```

```
int get_link (int v) {
    if (t[v].link != -1)  return t[v].link;
    if (t[v].par == -1)  return 0;
    int to = get_link (t[v].par);
    return t[v].link = split (go (state(to,t[t
}

void tree_extend (int pos) {
    for(;;) {
        state nptr = go (ptr, pos, pos+1);
        if (nptr.v != -1) {
            ptr = nptr;
            return;
        }

        int mid = split (ptr);
        int leaf = sz++;
        t[leaf] = node (pos, n, mid);
        t[mid].get( s[pos] ) = leaf;

        ptr.v = get_link (mid);
        ptr.pos = t[ptr.v].len();
        if (!mid)  break;
    }
}

void build_tree() {
    sz = 1;
    for (int i=0; i<n; ++i)
        tree_extend (i);
}
```

Compressed Implementation

This compressed implementation was proposed by freopen.

```
const int N=1000000,INF=1000000000;
string a;
int t[N][26],l[N],r[N],p[N],s[N],tv,tp,ts,la;

void ukkadd (int c) {
    suff:;
    if (r[tv]<tp) {
        if (t[tv][c]==-1) { t[tv][c]=ts;  l[ts
            p[ts++]=tv;  tv=s[tv];  tp=r[tv]+1
        tv=t[tv][c]; tp=l[tv];
    }
    if (tp==-1 || c==a[tp]-'a') tp++; else {
        l[ts+1]=la;  p[ts+1]=ts;
        l[ts]=l[tv];  r[ts]=tp-1;  p[ts]=p[tv]
        l[tv]=tp;  p[tv]=ts;  t[p[ts]][a[l[ts]
        tv=s[p[ts-2]];  tp=l[ts-2];
        while (tp<=r[ts-2]) {  tv=t[tv][a[tp]-
        if (tp==r[ts-2]+1)  s[ts-2]=tv;  else
        tp=r[tv]-(tp-r[ts-2])+2;  goto suff;
    }
}

void build() {
    ts=2;
    tv=0;
    tp=0;
    fill(r,r+N,(int)a.size()-1);
```

```
    s[0]=1;
    l[0]=-1;
    r[0]=-1;
    l[1]=-1;
    r[1]=-1;
    memset (t, -1, sizeof t);
    fill(t[1],t[1]+26,0);
    for (la=0; la<(int)a.size(); ++la)
        ukkadd (a[la]-'a');
}
```

Same code with comments:

```
const int N=1000000,    // maximum possible nu
    INF=1000000000; // infinity constant
string a;       // input string for which the
int t[N][26],   // array of transitions (state
    l[N],   // left...
    r[N],   // ...and right boundaries of the
    p[N],   // parent of the node
    s[N],   // suffix link
    tv,     // the node of the current suffix
    tp,     // position in the string which co
    ts,     // the number of nodes
    la;     // the current character in the st

void ukkadd(int c) { // add character s to the
    suff:;      // we'll return here after eac
    if (r[tv]<tp) { // check whether we're sti
        // if we're not, find the next edge. I
        if (t[tv][c]==-1) {t[tv][c]=ts;l[ts]=l
        tv=t[tv][c];tp=l[tv];
```

```
    } // otherwise just proceed to the next ed
    if (tp==-1 || c==a[tp]-'a')
        tp++; // if the letter on the edge equ
    else {
        // otherwise split the edge in two wit
        l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[t
        // add leaf ts+1. It corresponds to tr
        t[ts][c]=ts+1;l[ts+1]=la;p[ts+1]=ts;
        // update info for the current node -
        l[tv]=tp;p[tv]=ts;t[p[ts]][a[l[ts]]-'a
        // prepare for descent
        // tp will mark where are we in the cu
        tv=s[p[ts-2]];tp=l[ts-2];
        // while the current suffix is not ove
        while (tp<=r[ts-2]) {tv=t[tv][a[tp]-'a
        // if we're in a node, add a suffix li
        // (we'll create ts on next iteration)
        if (tp==r[ts-2]+1) s[ts-2]=tv; else s[
        // add tp to the new edge and return t
        tp=r[tv]-(tp-r[ts-2])+2;goto suff;
    }
}

void build() {
    ts=2;
    tv=0;
    tp=0;
    fill(r,r+N,(int)a.size()-1);
    // initialize data for the root of the tre
    s[0]=1;
    l[0]=-1;
    r[0]=-1;
    l[1]=-1;
```

```
    r[1]=-1;
    memset (t, -1, sizeof t);
    fill(t[1],t[1]+26,0);
    // add the text to the tree, letter by let
    for (la=0; la<(int)a.size(); ++la)
        ukkadd (a[la]-'a');
}
```

Practice Problems

- UVA 10679 - I Love Strings!!!