Reiknirit Ahos og Corasicks (1975)

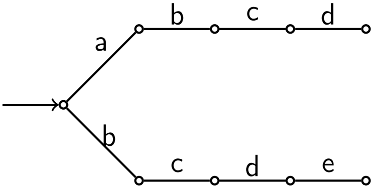
Bergur Snorrason

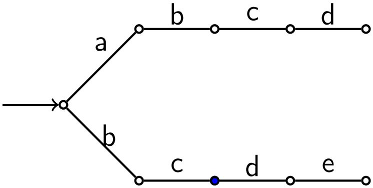
7. apríl 2022

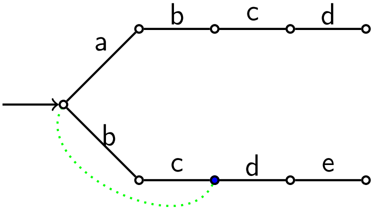
- Gerum ráð fyrir að við séum með stafróð Σ, streng s og lista af n strengjum p, þar sem j-ti strengurinn kallast p_j.
- Látum $|s|$ tákna lengd strengsins s og $|p| = |p_1| + \cdots + |p_n|$.
- Við viljum finna alla hlutstrengi s sem eru í listanum p.
- Við getum notað reiknirit Knuths, Morrisar og Pratts n sinnum.
- Þessi aðferð hefur tímaflækjuna $\mathcal{O}(n \cdot |s| + |p|)$.
- Reiknirit Ahos og Corasicks bætir þetta.

- Byrjum á að setja all strengina í p inn í forstrengstré T.
- Við viljum síðan búa til stöðuvél úr T.
- Hnútarnir í trénu eru stöðurnar, en okkur vantar að finna færslur fyrir hverja stöðu og bókstaf í Σ.
- Gerum ráð fyrir að við séum í hnút v í T og viljum færast fyrir staf c í Σ.
- Ef það er leggur í T úr v merktur með c þá ferðumst við eftir honum.
- Annars þurfum við að fara aftur í hnút w þannig að strengurinn sem svarar til w er bakstrengur strengsins sem svarar til v.
- Við viljum hafa strenginn w sem lengstan (með öðrum orðum viljum við fara sem styst aftur).
- Í hvern hnút bætum við við legg sem svarar til slíkrar færslu.
- Við köllum þessa leggi *bakstrengsleggi* (e. *suffix links*).
- Takið eftir að þeir eru í raun óháðir bókstafnum c.
- Við látum bakstrengslegg rótarinn benda á sjálfa sig.

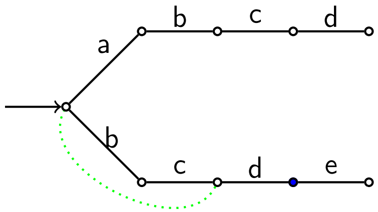
- Hvernig finnum við alla bakstrengsleggina?
- Við notum kvika bestun.
- Látum $f(w, c)$ tákna færslu úr stöðu w með staf c og $g(w)$ tákna bakstrengslegg w.
- Gerum einnig ráð fyrir að foreldri v sé p og $f(p, a) = v$.
- Við sjáum þá að $g(v) = f(g(p), a)$.
- Með öðrum orðum förum við upp í foreldrið, ferðumst eftir bakstrenglegg þaðan og færum okkur í stöðuvélinni eftir a.
- Við höfum þá rakningarformúlu sem við getum notað til að reikna bakstrengshlekkina.

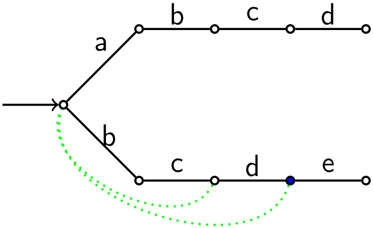

a
b
c
d
b
c
d
e

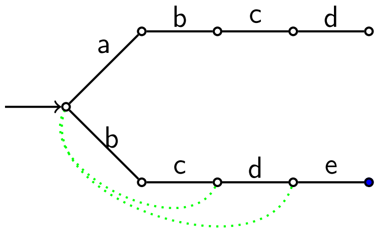

a
b
c
d
b
c
d
e

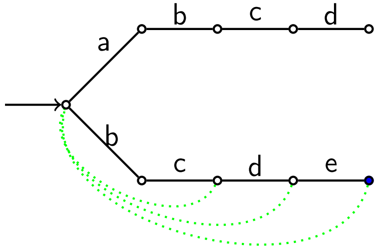


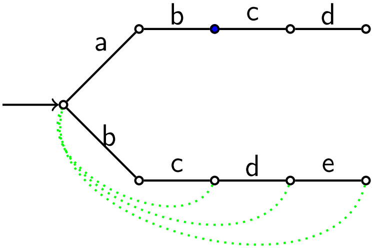
a
b
c
d
b
c
d
e

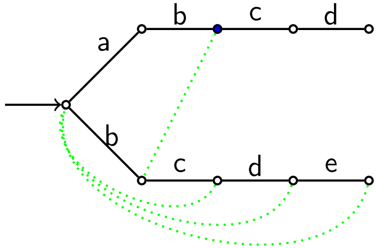

a
b
c
d
b
c
d
e


a
b
c
d
b
c
d
e

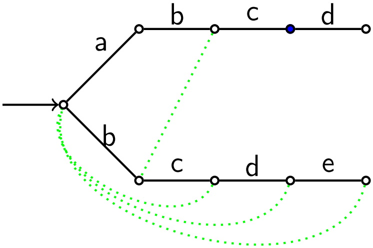

a
b
c
d
b
c
d
e


a
b
c
d
b
c
d
e

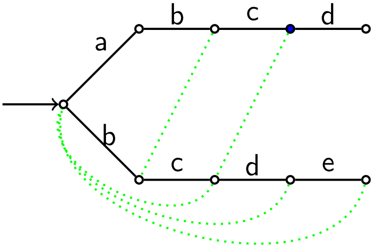

a
b
c
d
b
c
d
e

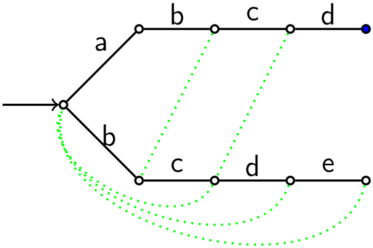


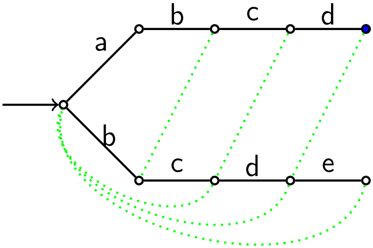
a
b
c
d
b
c
d
e



a
b
c
d
b
c
d
e


a
b
c
d
b
c
d
e

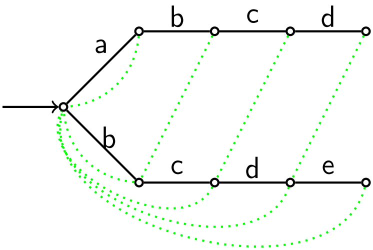

a
b
c
d
b
c
d
e



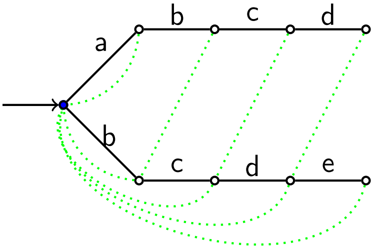
a
b
c
d
b
c
d
e

- Í T merkjum við lokastöður þar sem strengir enda.
- Við ferðumst svo í gegnum strenginn s og hliðrum stöðvélinni miðað við stafina í s.

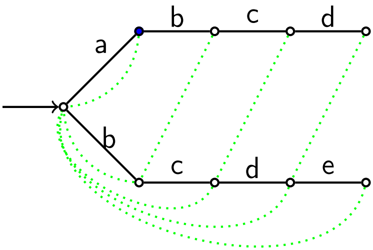
„abcdcdeaaabcdeabcxab"



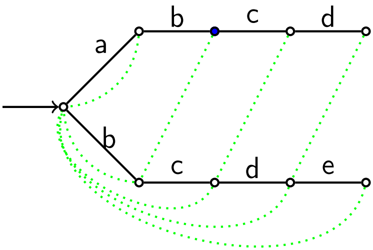
„abcdcdeaaabcdeabcxab"



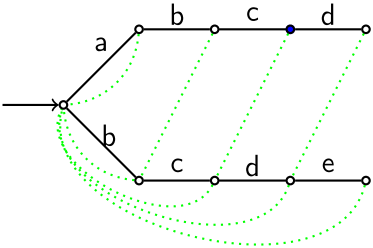
„bcdcdeaaabcdeabcxab"



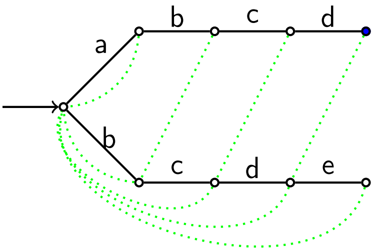
„cdcdeaaabcdeabcxab"



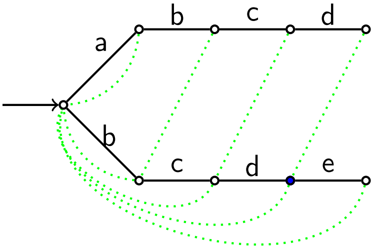
„dcdeaaabcdeabcxab"



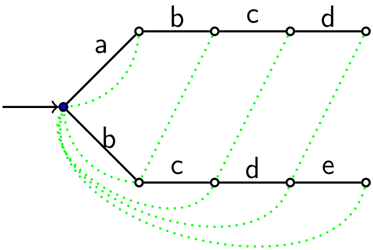
„cdeaaabcdeabcxab"



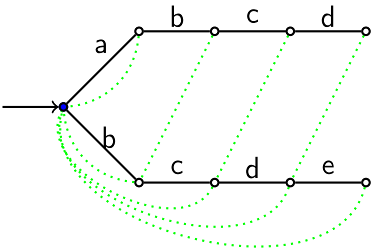
„cdeaaabcdeabcxab"



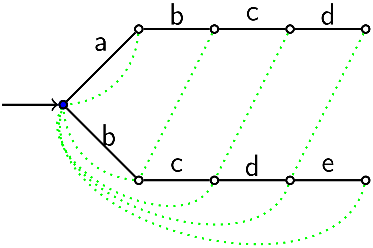
„cdeaaabcdeabcxab"



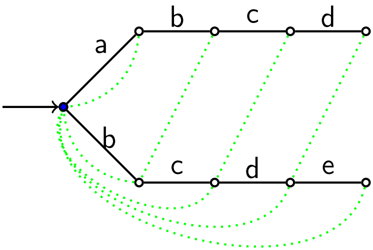
„deaaabcdeabcxab"



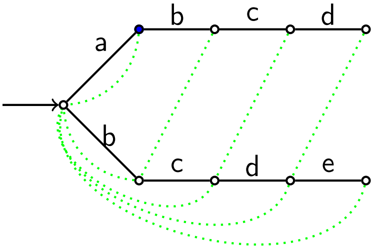
„eaaabcdeabcxab"



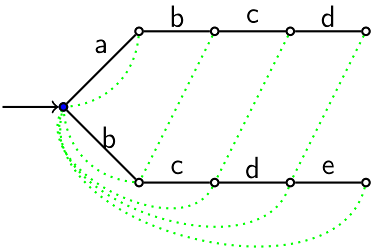
„aaabcdeabcxab"



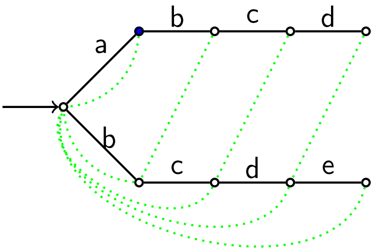
„aabcdeabcxab"



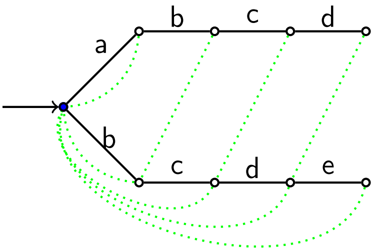
„aabcdeabcxab"



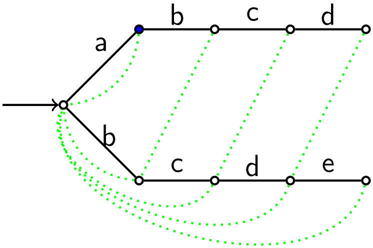
„abcdeabcxab"



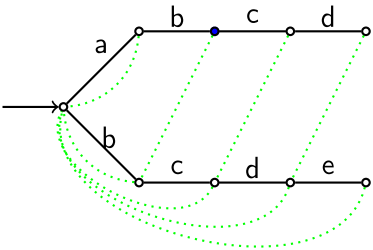
„abcdeabcxab"



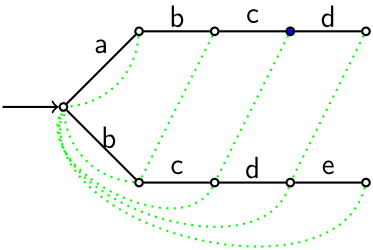
„bcdeabcxab"



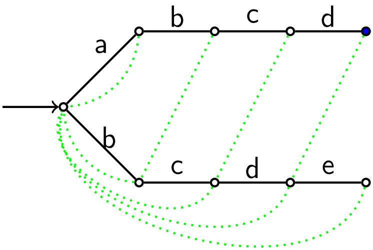
„cdeabcxab"



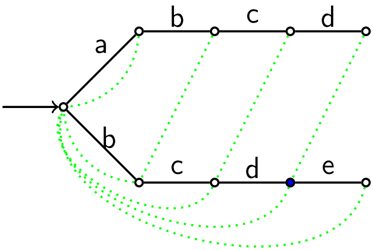
„deabcxab"



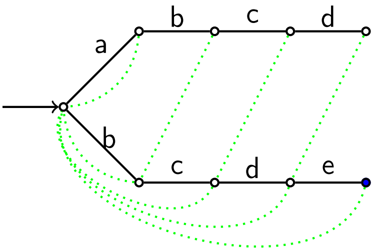
„eabcxab"



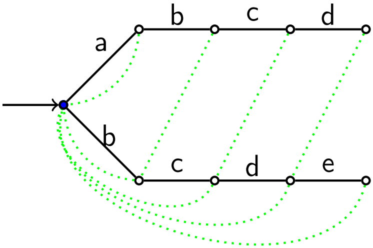
„eabcxab"



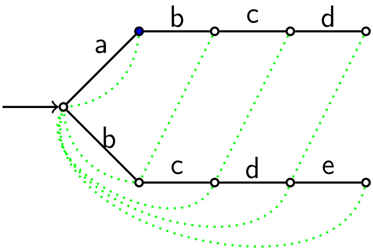
„abcxab"



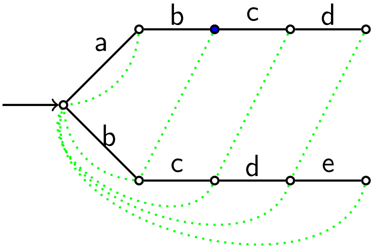
„abcxab"



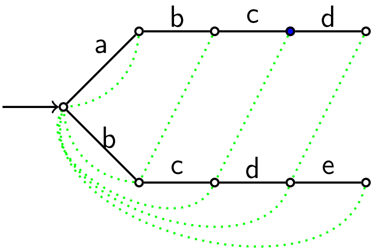
„bcxab"



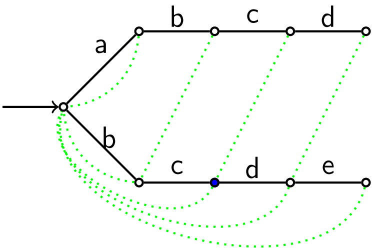
„cxab"



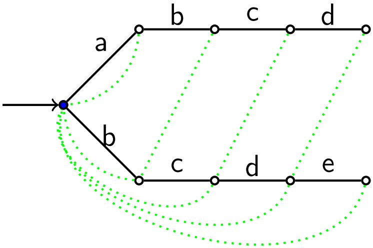
„xab"



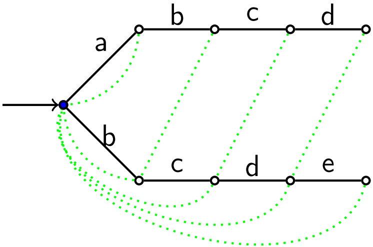
„xab"



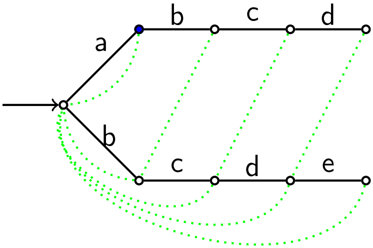
„xab”



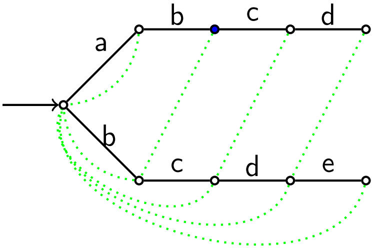
„ab''



„b"

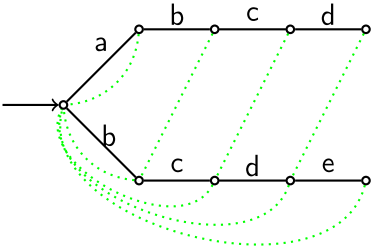


,,''



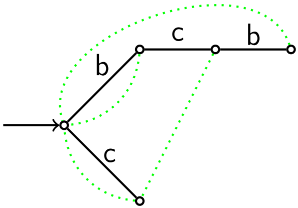
a
b
c
d
b
c
d
e

’’
„


a
b
c
d
b
c
d
e

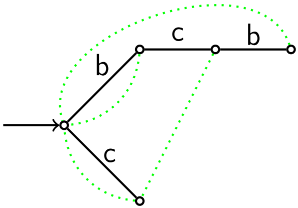
- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?

- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í *s* sem er í *p*.
- En eru þetta einu slíku hlutstrengirnir?



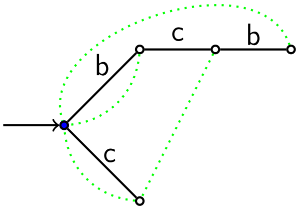
- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?

„bcb"



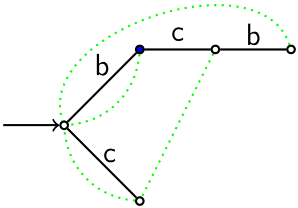
- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?

„bcb"



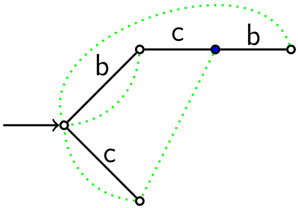
- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?

„cb"

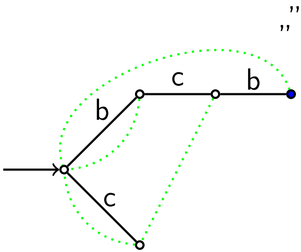


- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?

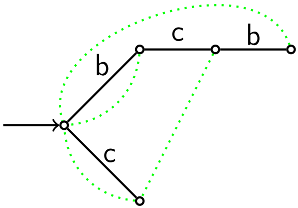
„b"



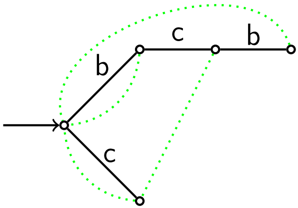
- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?



- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?



- Ljóst er að alltaf þegar við erum í lokastöðu þá erum við með hlutstreng í s sem er í p.
- En eru þetta einu slíku hlutstrengirnir?



- Nei, við þurfum líka, í hverju skrefi, að athuga hvort við getum komist í lokastöðu ef við ferðumst eftir bakstrengsleggjum.
- Til að koma í vega fyrir að tímaflækjan verði of slæm þá notum við aftur kvika bestun.
- Við bætum í raun leggjum inn í tréð, sem við köllum *lokaleggi* (e. *exit link*).

- Í útfærslunni munum við notast við þrjú hjálparföll.
- Fyrsta heitir `trie_step(...)` sem er notað til að ferðast um stöðuvélina.
- Annað heitir `trie_suffix(...)` sem er notað til að finna bakstrengsleggi.
- Þriðja heitir `trie_exit(...)` sem er notað til að finna lokaleggina.
- Öll þessi föll eru endurkvæm og notast við minnun.

```c
#define ALPHABET 128
#define MAXN 1000000
typedef struct { int v, n; } listnode;
typedef struct { int t[ALPHABET], l, e, p, c, d; } trienode;
typedef struct { int s, r, l; trienode m[MAXN + 1]; listnode w[MAXN];} trie;
int val(char c) { return c; }
int list_node(trie *t, int v, int n)
{
    t->w[t->l].v = v, t->w[t->l].n = n;
    return t->l++;
}
int trie_node(trie *t, int p, int c)
{
    int i;
    for (i = 0; i < ALPHABET; i++) t->m[t->s].t[i] = -1;
    t->m[t->s].l = -1, t->m[t->s].e = -1, t->m[t->s].p = p,
        t->m[t->s].c = c, t->m[t->s].d = -1;
    return t->s++;
}
void trie_init(trie *t) { t->s = t->l = 0, t->r = trie_node(t, -1, -1); }

void trie_insert(trie *t, char *s, int x)
{
    int h;
    for (h = t->r; *s; h = t->m[h].t[val(*s++)])
        if (t->m[h].t[val(*s)] == -1)
            t->m[h].t[val(*s)] = trie_node(t, h, val(*s));
    t->m[h].l = list_node(t, x, t->m[h].l);
}
```

```c
int trie_step(trie*, int, int);
int trie_suffix(trie *t, int h)
{
    if (t->m[h].d != -1) return t->m[h].d;
    if (h == t->r || t->m[h].p == t->r) return t->m[h].d = t->r;
    return t->m[h].d = trie_step(t, trie_suffix(t, t->m[h].p), t->m[h].c);
}

int trie_step(trie *t, int h, int c)
{
    if (t->m[h].t[c] != -1) return t->m[h].t[c];
    return t->m[h].t[c] = h == t->r ? t->r :
        trie_step(t, trie_suffix(t, h), c);
}

int trie_exit(trie *t, int h)
{
    if (t->m[h].e != -1) return t->m[h].e;
    if (h == 0 || t->m[h].l != -1) return t->m[h].e = h;
    return t->m[h].e = trie_exit(t, trie_suffix(t, h));
}
```

```c
trie t;
int aho_corasick(char *s, char **p, int m)
{
    trie_init(&t);
    int h, i, j, k, w, l[m];
    for (i = 0; i < m; i++) l[i] = strlen(p[i]);
    for (i = 0; i < m; i++) trie_insert(&t, p[i], i);
    printf("                ");
    for (i = 0; i < strlen(s); i++) printf("%d", i%10); printf("\n");
    printf("searching in %s\n", s);
    for (i = 0, j = 0, h = t.r; ; j++)
    {
        k = trie_exit(&t, h);
        while (t.m[k].l != -1)
        {
            for (w = t.m[k].l; w != -1; w = t.w[w].n)
            {
                printf("  '%s' found at index %d\n",
                       p[t.w[w].v], j - l[t.w[w].v]);
            }
            k = trie_exit(&t, trie_suffix(&t, k));
        }
        h = trie_step(&t, h, val(*s));
        if (*s++ == '\0') break;
    }
    return i;
}
```

- Gerum ráð fyrir að strengirnar í p komi fyrir k sinnum í s.
- Þá er tímaflækjan $\mathcal{O}(|s| + |\Sigma| \cdot |p| + k)$.
- Ef við höfum bara áhuga á að finna töluna k getum við breytt `trie_exit(...)` þannig að það reikni fjölda lokastaða á leiðinni að rót eftir bakstrengsleggjum.
- Þá verður tímaflækjan $\mathcal{O}(|s| + |\Sigma| \cdot |p|)$.
- Takið eftir að ef stafrófið er takmarkað þá er seinni tímaflækjan línuleg.