Suffix Automaton

Universidad Nacional de Rosario

Ucundinamarca Workshop

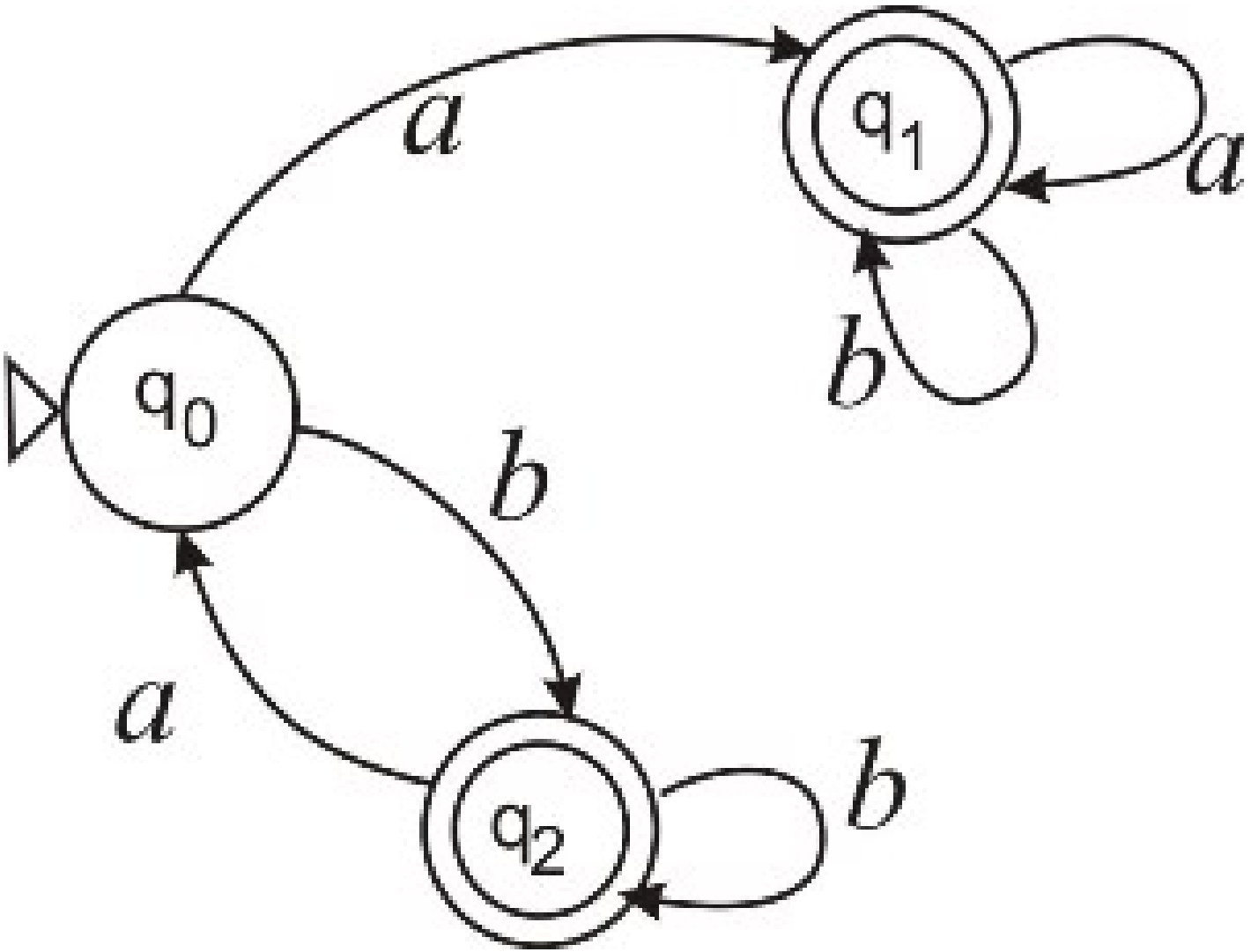
Contenidos

Contenidos

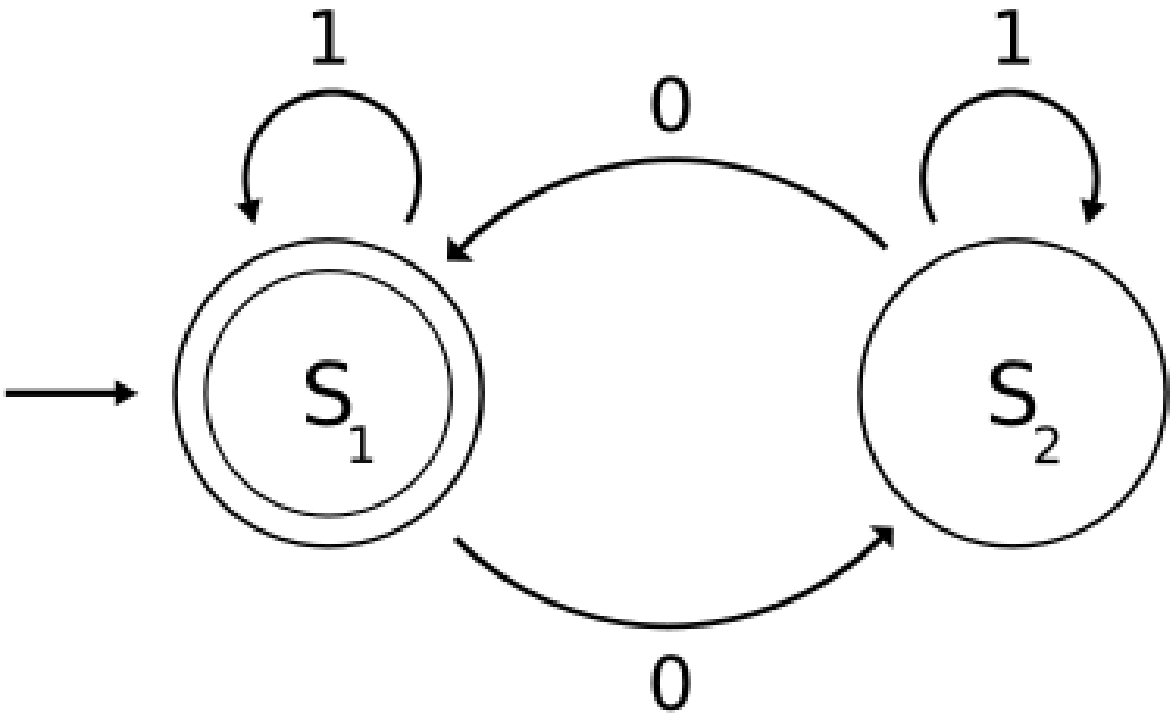
Autómata Estado Finito

- Un Autómata de Estado Finito es un conjunto de estados conectados por "flechas"que poseen como label un caracter. Se tiene un estado inicial S_0 y un conjunto de estados terminales o de aceptación.
- Se denomina las cadenas aceptadas por el autómata a aquellas que se corresponden a los labels de cierto camino finito que empieza en S_0 y termina en algún estado de aceptación.
- El conjunto de labels válidos se llama el alfabeto del autómata (ej: 0,1, el alfabeto inglés, etc). A cada elemento de éste son caracteres del alfabeto.
- Tradicionalmente todos los estados definen para cada caracter UNA transición a otro estado.

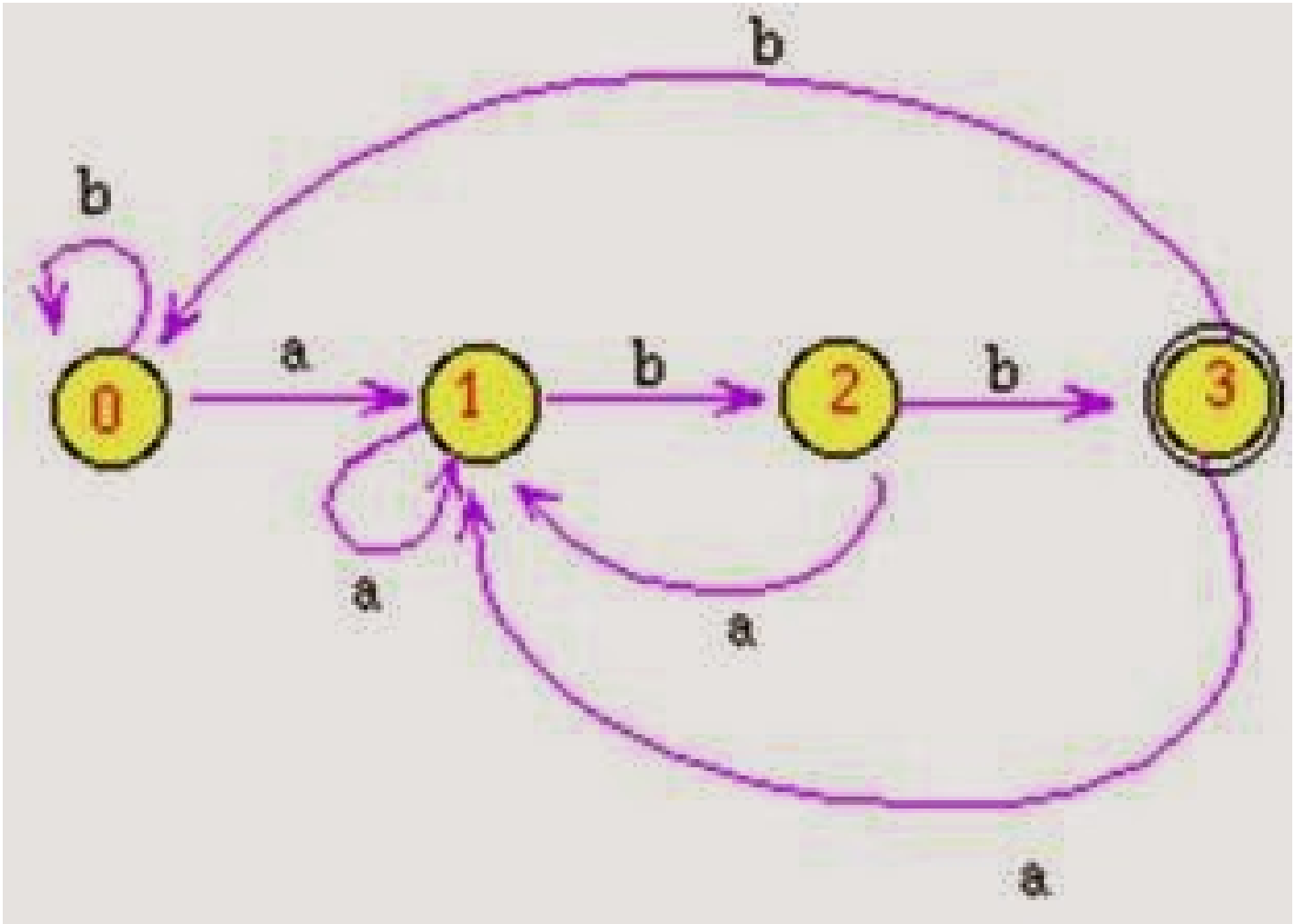
Autómata Estado Finito



Autómata Estado Finito



Autómata Estado Finito



Contenidos

Suffix Automaton

"Un problema de string, traigan la artillería!"

FUENTE: `e-maxx.ru/algo/suffix_automata` (traducir del ruso)
INGLÉS: `cp-algorithms.com/string/suffix-automaton`
(la version inglés está incompleta)

Definición

Definición

Es un Autómata de Estados Finitos que acepta todos los sufijos de una cadena S. Contiene mínima cantidad de estados con esta condición.

Pensar:

- ¿Siempre el DAG tiene 'máximo' y 'mínimo'?

Propiedad

Cada camino de la fuente corresponde a una substring y viceversa

Definición

Definición

Es un Autómata de Estados Finitos que acepta todos los sufijos de una cadena S. Contiene mínima cantidad de estados con esta condición.

Pensar:

- ¿Podemos construir uno con la primera condición?

Propiedad

Cada camino de la fuente corresponde a una substring y viceversa

Definición

Definición

Es un Autómata de Estados Finitos que acepta todos los sufijos de una cadena S. Contiene mínima cantidad de estados con esta condición.

Pensar:

- ¿Podemos construir uno con la primera condición?
- ¿Siempre es un DAG?

Propiedad

Cada camino de la fuente corresponde a una substring y viceversa

Definición

Definición

Es un Autómata de Estados Finitos que acepta todos los sufijos de una cadena S. Contiene mínima cantidad de estados con esta condición.

Pensar:

- ¿Podemos construir uno con la primera condición?
- ¿Siempre es un DAG?
- ¿Siempre el DAG tiene 'máximo' y 'mínimo'?

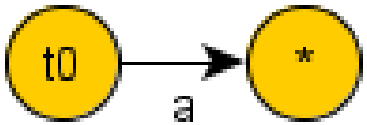
Propiedad

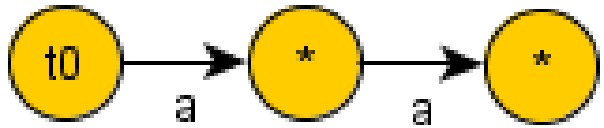
Cada camino de la fuente corresponde a una substring y viceversa

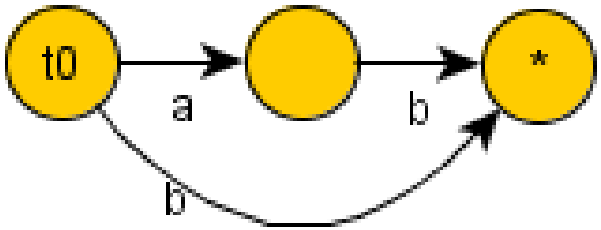
Ejemplos

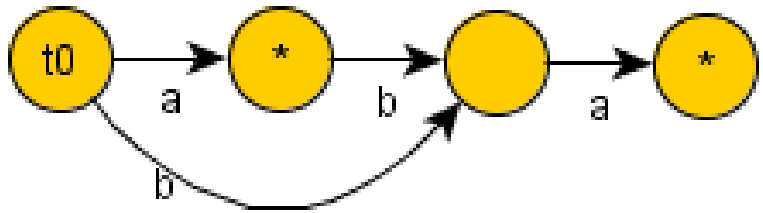
El estado inicial lo notamos con t_0 y los terminales con un $*$

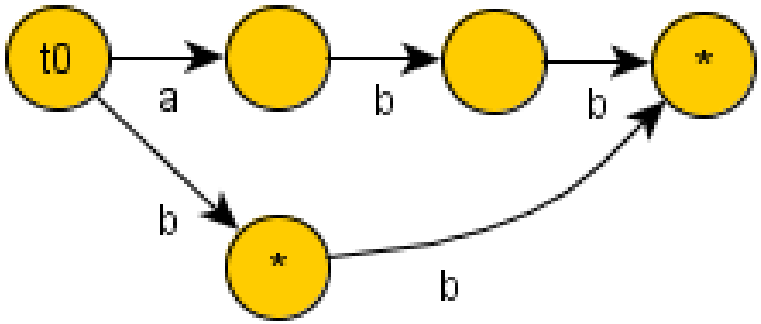
"" 

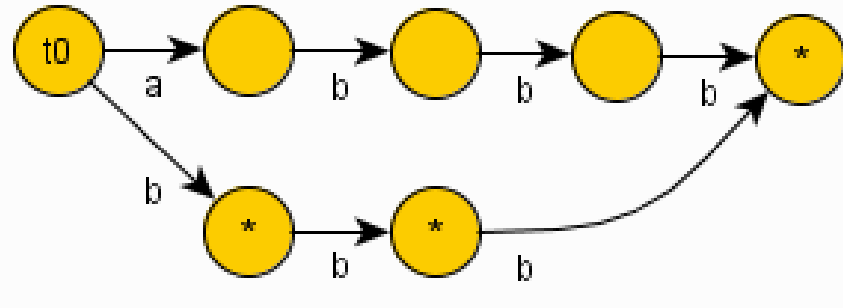
"a" 

"aa" 

"ab" 

"aba" 

"abb" 

"abbb" 

Contenidos

Definición endpos

Convenciones previas:

- S será la cadena analizada. *sstr* el conjunto de todas las subcadenas de S.
- Sea t en *sstr*. $endpos(t)$ será el subconjunto de todas las posiciones donde ocurre t.
- $endpos(\emptyset) = [-1..len - 1]$

Cocientando *sstr*

Podemos pensar $sstr/endpos$.
Podríamos ver que subcadenas de igual endpos corresponden al mismo estado en el autómata. (Y viceversa!).

Lemas

Sean u,$w \in sstr$ no vacías y ($length(u) \leq length(w)$).

Lema 1

u y v *endpos* equivalentes $\Leftrightarrow$ u ocurre en S sólo como sufijo de w.

Lema 2

- $endpos(w) \subseteq enendpos(u)$ si u sufijo w
- $endpos(u) \cap endpos(w) = \emptyset$ si no

(Ver que existe una biyección entre los árboles enraizados y las familias finitas de conjuntos que son de a pares o bien disjuntos o bien uno incluído en otro.)

Lemas cont.

Sea $v \in endpos$. Clases endpos y estados del automata seran lo mismo: si luego de recorrer el automata por la cadena s llego al estado v me hallo en un estado definido por el conjunto de "las posiciones en las que me puedo hallar". Hablaremos indistintamente de la clase $endpos(s)$ o v.

Lema 3.1

No hay dos cadenas distintas de igual longitud en una misma clase.

De lo anterior se deduce que $v = \{u_1, u_2, ...u_k\}$ con $len(u_i) < len(u_{i+1})$

Lema 3.2

Sea $len(u_i) = len(u_{i+1}) + 1$ $i \neq k$.

Si $|v| = k$ esta formada por los k - esimos sufijos mas largos de u_k, la cadena mas larga de v.

Suffix links

- El $(k+1)$ - esimo sufijo pertenecera a una clase mas grande $v' \supset v$.
- Definiremos $link(v) = v'$ la clase del sufijo más largo que no pertenece a v.
- Al estado inicial t_0 le corresponde solo la cadena vacía y por convencion, su *endpos* es $[-1..length(S) - 1]$. $link(t_0)$ no esta definido.

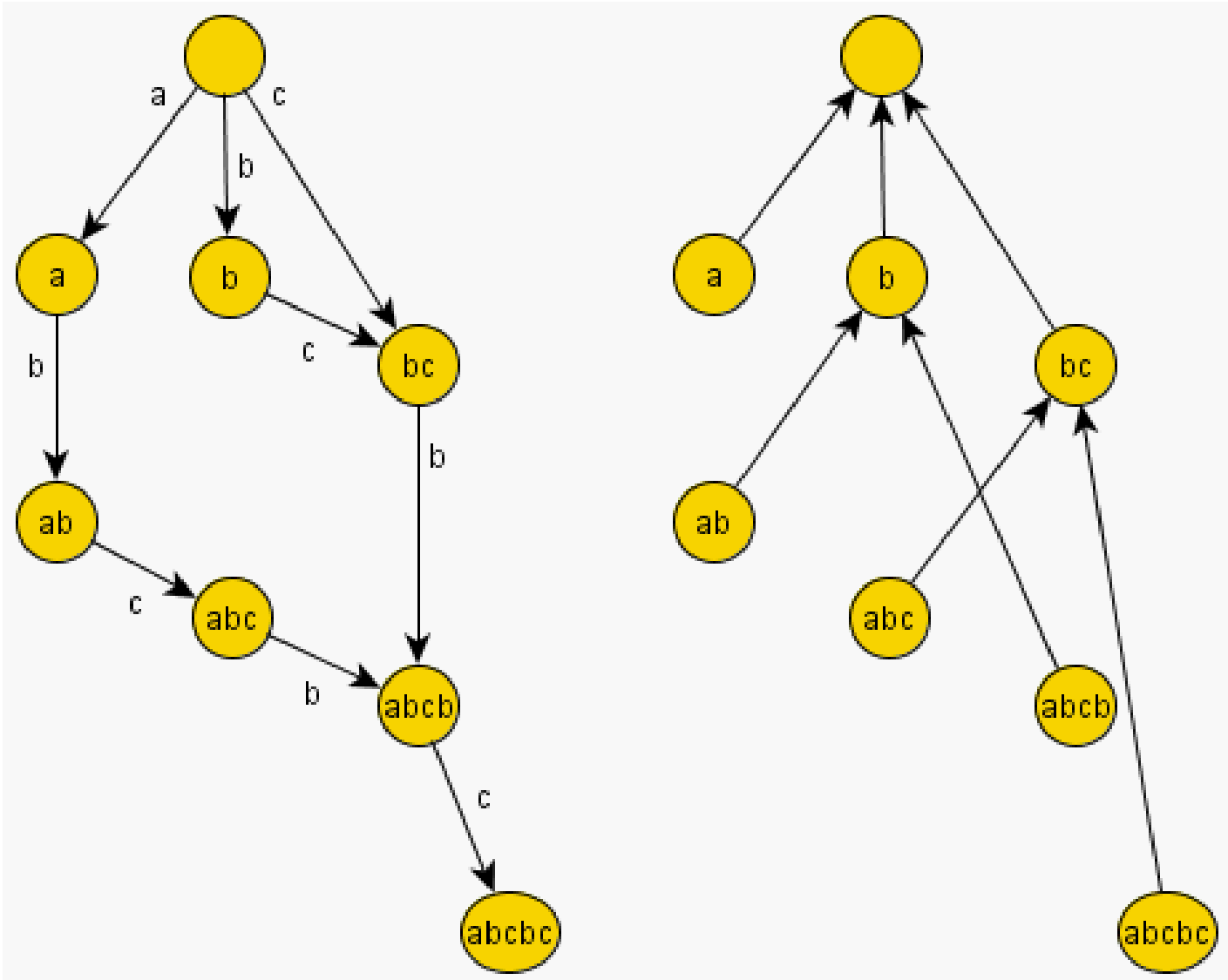
Lema 4

Los suffix links forman un árbol enraizado cuya raíz es el estado inicial.

Lema 5

Si construimos el árbol inducido por los $endpos(sstr)$ será el mismo que el anterior. $endpos(v) \subset endpos(link(v))$

Suffix links



Resumiendo

- Las subcadenas de S pueden ser divididas en clases de equivalencia según el conjunto de posiciones donde terminan en S, *endpos*.
- El suffix automaton contiene un estado inicial t_0 y uno por cada clase de equivalencia *endpos*. Ahora hablaremos indistintamente de estado o clase.
- Cada estado contiene un subconjunto de cadenas. Notaremos $longest(v)$ y $shortest(v)$ las cadenas más larga y más corta respectivamente y a $len(v)$ y $minlen(v)$ sus respectivas longitudes. Las cadenas de v son todos los sufijos de $longest(v)$ con longitud en el rango $[minlen(v)..len(v)]$

Resumiendo cont.

- Para cada estado $v \neq t_0$ definimos $link(v)$ el estado del sufijo de $longest(v)$ con longitud $minlen(v) - 1$.
- $link(v)$ forma un árbol con raíz en t_0 que es el mismo árbol inducido por la inclusión de los *endpos*.
- Para todo $v \neq t_0$ vale $minlen(v) = len(link(v)) + 1$
- Si empezamos de un estado v_0 y seguimos los suffix links eventualmente llegaremosa a t_0. Al mismo tiempo obtendremos una secuencia de segmentos disjuntos $[minlen(v_i), len(v_i)]$. Cuya unión da un "segmento sólido".

Algoritmo

- El algoritmo es **online** agregamos un caracter por vez.
- Empezamos con un solo estado t_0 para la cadena vacia. $len(t_0) = 0$, $link(t_0) = -1$.

Implementacion

Java: `https://sites.google.com/site/indy256/algo/suffix_automaton`

```
1  struct state {
2    int len, link;
3    map<char, int> next;
4    state() { }
5  };
6  const int MAXLEN = 10010;
7  state st[MAXLEN * 2];
8  int sz, last;
9  void sa_init() {
10   forn(i,sz) st[i].next.clear();
11   sz = last = 0;
12   st[0].len = 0;
13   st[0].link = -1;
14   ++sz;
15 }
```

Implementacion

```
1  void sa_extend (char c) {
2    int cur = sz++;
3    st[cur].len = st[last].len + 1;
4    int p;
5    for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
6      st[p].next[c] = cur;
7    if (p == -1)
8      st[cur].link = 0;
9    else {
10     int q = st[p].next[c];
11     if (st[p].len + 1 == st[q].len)
12       st[cur].link = q;
13     else {
14       int clone = sz++;
15       // no le ponemos la posicion actual a clone sino indirectamente por el link de cur
16       st[clone].len = st[p].len + 1;
17       st[clone].next = st[q].next;
18       st[clone].link = st[q].link;
19       for (; p!=-1 && st[p].next.count(c) && st[p].next[c]==q; p=st[p].link)
20         st[p].next[c] = clone;
21       st[q].link = st[cur].link = clone;
22     }
23   }
24   last = cur;
25 }
26
27 string s; cin >> s;
28 sa_init();
29 forn(i, sz(s)) sa_extend(s[i]);
30 return 0;
```

Observaciones del automata

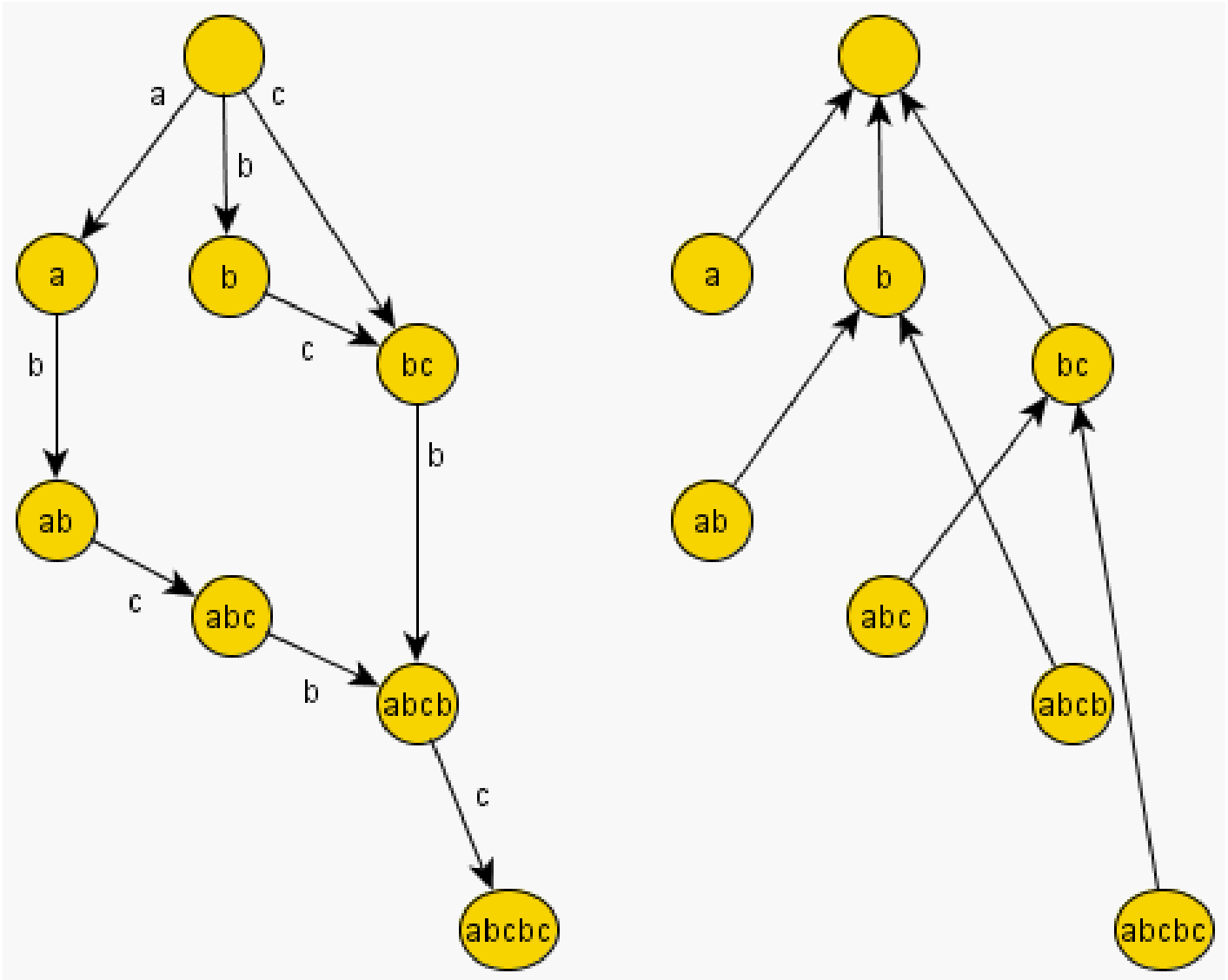
Sea $v \in endpos$ un estado.

- Es un DAG de una sola fuente y una sola hoja.
- $size(v) = |\{s \in v\}|$ cantidad de cadenas que pertenecen a la clase:
  - podemos calcularlo con la cantidad de caminos de t_0 a v (con DP).
  - podemos calcularlo como $len(v) - len(link(v))$ con $v \neq t_0$.
  - $len(v)$ es la longitud del camino mas largo v a t_0.
  - $minlen(v)$ es la longitud del camino mas corto v a t_0.
  - para cado $k \in [minlen(v), len(v)]$ existe un camino de longitud k de v a t_0.
- Recordar que no necesitamos guardar minlen, $minlen(v) = len(link(v)) + 1$

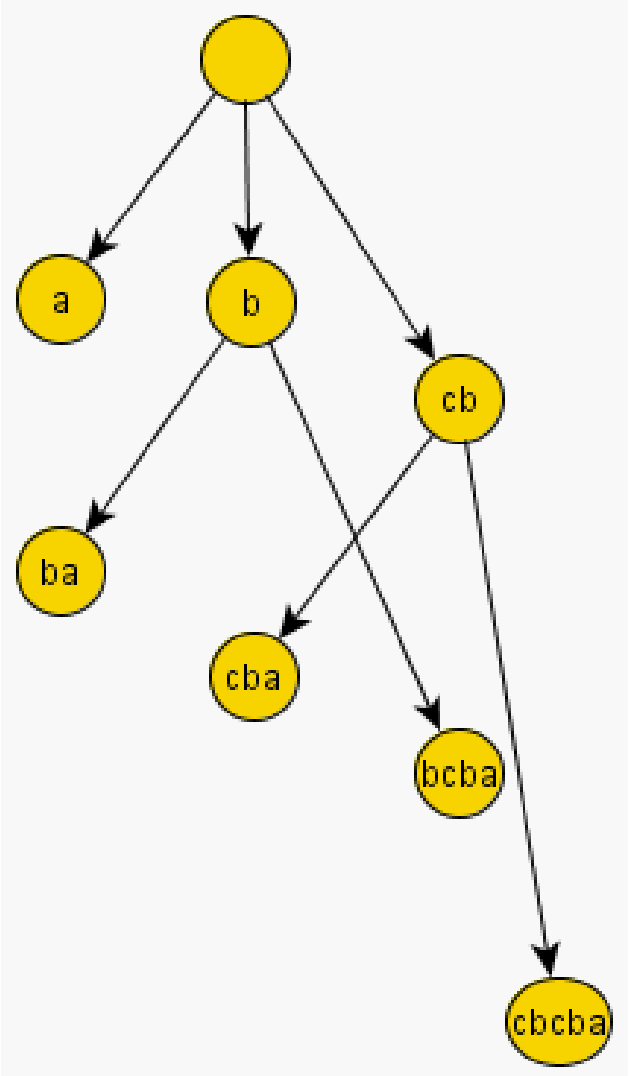
Observaciones del automata (cont.)

- La posicion $p \in [0, |T|]$ puede pertenecer a los *endpos* de muchos estados. Pero estos estados son $T[0, p]$ y sus sufijos.
- O sea el estado de $T[0, p]$ y subiendo por el arbol de sufijos.
- El estado de $T[0, p]$ es que creamos cuando agregamos $T[p]$.
- Cuando creamos un estado podemos guardar un *endpos* del estado.
- Luego podemos hacer dfs en el arbol de los suffix links para determinar el minimo y maximo.
- Podriamos asi calcular la posicion mas a la izquierda de $longest(v)$ o $shortest(v)$.
- Tambien se puede hacer con DP (mas facil, pensar como!).

Relación con Suffix Tree



Relación con Suffix Tree



Problemas (que tiene la idea explicada)

La idea de todos estos problemas esta explicada en `http://e-maxx.ru/algo/suffix_automata`

- Dado un texto T y queries con cadenas P_i,
  - si P_i ocurre en T.
  - cuántas apariciones de P_i en T existen. (`https://www.spoj.com/problems/SUBST1/`)
  - cuál es su primera y última ocurrencia.
  - listar todas las ocurrencias.
- Contar el número de substrings diferentes en una cadena S. Para pensar: qué orden de magnitud es el resultado.
- Dada una cadena S cuál es la suma de las longitudes de sus subcadenas distintas.
- Dada una string S, se dan queries K_i y hay que encontrar la K_i-ésima subcadena de S según el orden lexicográfico.

Problemas (cont.)

- Dada una subcadena encontrar mínima rotación lexicográfica. (`https://www.spoj.com/problems/MINMOVE/`)
- La cadena más corta que no es subcadena.
- La subcadena común más larga de dos cadenas. (`https://www.spoj.com/problems/LCS/`)
- La subcadena común más larga de K cadenas. (`https://www.spoj.com/problems/LCS2/`)
- La cadena mas corta que no aparece en S.

Problemas de juez

- Otro tutorial más con problemas: `codeforces.com/blog/entry/20861`
- Más Problemas:
  - `codeforces.com/problemset/problem/123/D`
  - `www.urionlinejudge.com.br/judge/en/problems/view/1530` (este está resuelto en el drive que aparece abajo)
  - Sacados de Competitiva Programming 3 de Steven Halim: (están todas mis soluciones en el drive `https://drive.google.com/open?id=16xEse2hdPivizhOGhT5OZX7gmTjclPQq`):
    - UVa 00719 Glass Beads
    - UVa 00760 DNA Sequencing
    - UVa 01223 Editor
    - UVa 01254 Top 10
    - UVa 11107 Life Forms
    - UVa 11512 GATTACA

Mas problemas de juez

- https://www.spoj.com/problems/NSUBSTR/
- https://codeforces.com/contest/235/problem/C
- https://codeforces.com/contest/452/problem/E
- Compilado! https://a2oj.com/category?ID=227

Problemas String Regionales Latam

Problemas de string regionales latam de 2007 en adelante (algunos pueden que no sean de suffix automaton!):

- **2007**: ambiguous code `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=294&page=show_problem&problem=1928`
- **2008**: dna sequence `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=330&page=show_problem&problem=2214`
- **2009**: File Recover `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=365&page=show_problem&problem=2478`
- **2010**: Growing strings `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=400&page=show_problem&problem=2812`

Problemas String Regionales Latam(cont.)

- **2011**: File Retrieval `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=531&page=show_problem&problem=3805`
- **2012**: Cellphone typing `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=572&page=show_problem&problem=4144`
- **2013**: Blogger Language `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=615&page=show_problem&problem=4537`