

Suffix Automaton

Andrés Valencia Oliveros^{1,2}

Facultad de Ingeniería, Diseño e Innovación
Institución Universitaria Politécnico Grancolombiano
Bogotá, Colombia

Resumen

Keywords: autómata finito, autómata de sufijo, coincidencia de cadenas.

1. Introducción

2. Grafo dirigido

Un grafo $G(V, E)$ es una colección de puntos, llamados vértices o nodos $V = \{v_1, v_2, \dots\}$, y segmentos de línea que conectan esos puntos, llamados aristas o arcos (en inglés *edges*) $E = \{e_1, e_2, \dots\}$; cada arista e tiene dos *puntos finales*, que son vértices.

Un digrafo o grafo dirigido $G(V, E)$ se define de manera similar a un grafo, excepto que el par de *puntos finales* (u, v) de cada arista ahora está ordenado. Se escribe $u \xrightarrow{e} v$, donde u es el vértice inicial de e ; y v es el vértice final de e . Se dice que la arista e está dirigida de u a v [1].

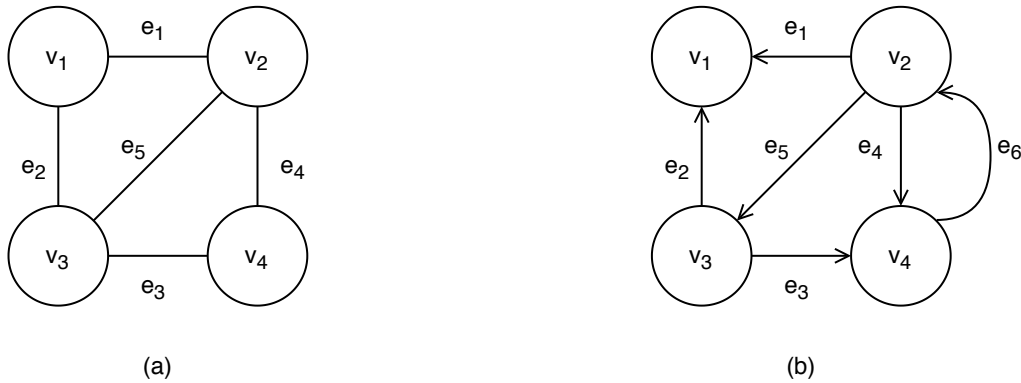


Figura 1. Tipos de grafos. (a) No dirigido. (b) Dirigido o digrafo.

¹ GitHub: [anvalenciao](#)

² Email: anvalenciao@poligran.edu.co

3. Autómata finito determinista

Formalmente, un autómata finito es una 5-tupla $(Q, \Sigma, q_0, \delta, F)$ donde:

- Q , es un conjunto finito de estados;
- Σ , es un conjunto finito de **símbolos** llamado **alfabeto**;
- $q_0 \in Q$ es el estado inicial;
- $\delta: Q \times \Sigma \rightarrow Q$ es una función de transición;
- $F \subseteq Q$ es un conjunto de estados finales o de aceptación.

Un **Autómata Finito Determinista** (AFD), es un autómata/máquina que tiene un número finito de estados y además es un sistema determinista, es decir, para cada **símbolo** de entrada, se puede determinar el estado al que se moverá el autómata [2].

Un AFD está representado por un grafo dirigido llamado diagrama de estado.

- Los estados son representados por vértices o nodos $Q = \{S_1, S_2, S_3, \dots\}$.
- Las aristas o arcos etiquetados con un **alfabeto** Σ , representan las transiciones δ .
- El estado inicial q_0 se denota por una sola arista entrante vacía.
- El o los estados finales F están indicados por círculos dobles.
- Cada transición se escribe $\delta(q_1, \sigma) = q_2$, también se puede denotar como $q_1 \xrightarrow{\sigma} q_2$.

Ejemplo 3.1 El siguiente ejemplo es de un AFD L , con un alfabeto binario, que reconoce el lenguaje regular conformado exclusivamente por las cadenas con un número par de ceros y un número par de unos.

$M = (Q, \Sigma, q_0, \delta, F)$ donde:

- $Q = \{S_1, S_2, S_3, S_4\}$
- $\Sigma = \{0, 1\}$
- $q_0 = S_1$
- $F = \{S_1\}$
- $\delta : \delta(S_1, 0) = S_3, \delta(S_1, 1) = S_2, \delta(S_2, 0) = S_4, \delta(S_2, 1) = S_1, \delta(S_3, 0) = S_1, \delta(S_3, 1) = S_4, \delta(S_4, 0) = S_2, \delta(S_4, 1) = S_3$

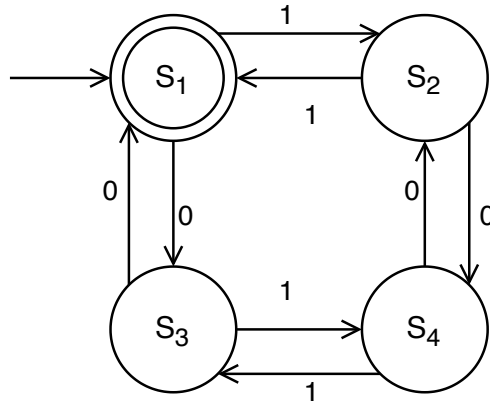


Figura 2. El diagrama de estado de L

El lenguaje reconocido por L es el lenguaje regular dado por la expresión regular [3]:

$$\wedge(00|11|(01|10)(00|11)^*(01|10))^*\$$$

La Figura 2 da un ejemplo de un autómata simple M que acepta la cadena:

1001101011001010010001

4. Autómata de sufijo

Un autómata de sufijo es una estructura de datos eficiente y compacta, también conocido como Directed Acyclic Word Graph (DAWG), es el AFD mínimo, que reconoce el conjunto de sufijos de una **cadena** $S = s_1 s_2 s_3 \dots s_n$ [4], es decir, se puede usar un autómata de sufijo para determinar si una **cadena** S es una **subcadena** en tiempo lineal en su longitud $O(|S|)$ [5].

Teorema 4.1 (Principal) *El tamaño de un autómata sufijo de una **cadena** S es $O(|S|)$. El autómata puede ser implementado en tiempo $O(|S| \times \log \text{card}(A))$ y $O(|S|)$ espacio extra [6].*

4.1. Propiedades

El autómata de sufijo contiene información sobre todas las **subcadenas** de la **cadena** S . Para construir un autómata de sufijo en tiempo lineal, es necesario comprender dos conceptos *End Positions* y *Suffix Links*.

4.1.1. End Positions (endpos)

Los estados del autómata no son **subcadenas**, los estados representan clases de equivalencia. Cada **subcadena** de una **cadena** pertenece a una clase de equivalencia llamada *endpos* [7].

Definición 4.1 *endpos(t): El conjunto de todas las posiciones en la **cadena** S donde termina la **subcadena** t [8].*

Ejemplo 4.1 Sea $t_1 = "bc"$ y $t_2 = "abc"$ **subcadenas** de $S = "abcdbc"$, entonces $\text{endpos}(t_1) = \{3, 6\}$ y $\text{endpos}(t_2) = \{3\}$ [8]. Sea $t = "bra"$ una **subcadena** de $S = "abracadabra"$, entonces $\text{endpos}(t) = \{3, 10\}$. Nótese que $\text{endpos}(t)$ es un conjunto cuyos elementos son posiciones finales de t en todas sus ocurrencias en S [7].

Dos **subcadenas**, t_1 y t_2 , son endpos-equivalentes sí y sólo si $\text{endpos}(t_1) = \text{endpos}(t_2)$.

Ejemplo 4.2 Sea $t_1 = "abra"$, $t_2 = "bra"$, $t_3 = "ra"$, $t_4 = "a"$ **subcadenas** de $S = "abracadabra"$, entonces $\text{endpos}(t_1) = \{3, 10\}$, $\text{endpos}(t_2) = \{3, 10\}$, $\text{endpos}(t_3) = \{3, 10\}$, $\text{endpos}(t_4) = \{0, 3, 5, 7, 10\}$.

| S | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|
| a | b | r | a | c | a | d | a | b | r | a |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Por lo tanto, $\text{endpos}(t_1), \text{endpos}(t_2), \text{endpos}(t_3)$ son endpos-equivalentes, lo que equivale a decir que una de las palabras t_1, t_2, t_3 es un **sufijo** de la otra. Nótese que $\text{endpos}(t_{1 \leq 3})$ y $\text{endpos}(t_4)$ no lo son.

4.1.2. Suffix Links (endpos)

Suffix Links relaciona *endpos* de forma unidireccional. Sea V un estado $v \neq t_0$, se tiene que v es una clase de equivalencia que contiene a los **cadena** con el mismo *endpos*. Se sabe que los primeros sufijos de w pertenecen a v . Sin embargo, en el primer momento en que un sufijo de w tenga un *endpos* diferente, se arma un *suffix link* entre v y el endpos de ese sufijo [7].

Definición 4.2 *link(Q): punta a la clase de equivalencia *endpos* definida por "el sufijo más largo de w que no pertenece a Q " [8].*

5. Algoritmo

El algoritmo es una construcción *on-line*, es decir, se agregarán los caracteres de la cadena uno por uno, y se modificará el autómata en consecuencia en cada paso [9]. En cada etapa de la construcción, justo después de procesar un prefijo $x_1 x_2 \dots x_\ell$ de x , el autómata de sufijo está construido. Los estados terminales se conocen implícitamente por la ruta del sufijo del $\text{last}_{x_1 x_2 \dots x_\ell}$. El estado $\text{last}_{x_1 x_2 \dots x_\ell}$ está explícitamente representado por una variable en la función que construye el autómata.

También se utilizan otros dos elementos: *Length* and *F*. El arreglo *Length* representa la función length_x definida sobre los estados del autómata, se utiliza para determinar los llamados *solid edges* o *transiciones* en la construcción del sufijo autómata.

Los *suffix links* de estados (diferente del estado inicial) son almacenados en un arreglo denotado por F que representa la función f_x [6].

5.0.1. Pseudocódigo

Los algoritmos 5.1 y 5.2 dan el pseudocódigo para construir el autómata de sufijo.

Algoritmo 5.1 *Suffix Automaton - Construcción on-line del autómata de sufijo de la cadena x .*

```

1: function SA( $x$ )
2:   let  $\delta$  be the transition function of  $(Q, i, T, E)$ 
3:    $(Q, E) \leftarrow (\emptyset, \emptyset)$ 
4:    $i \leftarrow \text{State-Creation}$ 
5:    $\text{Length}[i] \leftarrow 0$ 
6:    $F[i] \leftarrow \text{UNDEFINED}$ 
7:    $\text{last} \leftarrow i$ 
8:   for  $\ell$  from 1 up to  $|x|$  do
9:      $\text{sa\_extend}(\ell)$ 
10:  end for
11:   $T \leftarrow \emptyset$ 
12:   $p \leftarrow \text{last}$ 
13:  while  $p \neq \text{UNDEFINED}$  do
14:     $T \leftarrow T + \{p\}$ 
15:     $p \leftarrow F[p]$ 
16:  end while
17:  return  $((Q, i, T, E), \text{Length}, F)$ 
18: end function

```

Algoritmo 5.2 *Suffix Automaton Extend*

```

1: function SA_EXTEND( $\ell$ )
2:    $\text{sa}[i] \leftarrow x_\ell$ 
3:    $\text{newlast} \leftarrow \text{State-Creation}$ 
4:    $\text{Length}[\text{newlast}] \leftarrow \text{Length}[\text{last}] + 1$ 
5:    $p \leftarrow \text{last}$ 
6:   while  $p \neq \text{UNDEFINED}$  and  $\delta(p, a) = \text{UNDEFINED}$  do
7:      $E \leftarrow E + \{(p, a, \text{newlast})\}$ 
8:      $p \leftarrow F[p]$ 
9:   end while
10:  if  $p = \text{UNDEFINED}$  then
11:     $F[\text{newlast}] \leftarrow i$ 
12:  else
13:     $q \leftarrow \delta(p, a)$ 
14:    if  $\text{Length}[q] = \text{Length}[p] + 1$  then
15:       $F[\text{newlast}] \leftarrow q$ 
16:    else
17:       $q' \leftarrow \text{State-Creation}$ 
18:      for each letter  $b$  such that  $\delta(q, b) \neq \text{UNDEFINED}$  do
19:         $E \leftarrow E + \{(q', b, \delta(q, b))\}$ 
20:      end for
21:       $\text{Length}[q'] \leftarrow \text{Length}[p] + 1$ 
22:       $F[\text{newlast}] \leftarrow q'$ 
23:       $F[q'] \leftarrow F[q]$ 
24:       $F[q] \leftarrow q'$ 
25:      while  $p \neq \text{UNDEFINED}$  and  $\delta(p, a) = q$  do
26:         $E \leftarrow E - \{(p, a, q)\} + \{(p, a, q')\}$ 
27:         $p \leftarrow F[p]$ 
28:      end while
29:    end if
30:  end if
31:   $\text{last} \leftarrow \text{newlast}$ 
32: end function

```

6. Aplicación

Un autómata de sufijo se puede utilizar para contar el número de subcadenas distintas que se producen en una cadena determinada, también en el reconocimiento de voz, la compresión de datos, tareas de extracción de

información e identificación musical y emparejamiento en secuencias del genoma [4].

6.1. UVA 719 - Glass Beads

| | | | |
|---------------------|----------------------|-----------------------|-----------------|
| 01 - helloworld | 01 - amandamanda | 01 - dontcallmebfu | 01 - aaabaaa |
| 02 - elloworldh | 02 - mandamandaa | 02 - ontcallmebfud | 02 - aabaaaa |
| 03 - llloworldhe | 03 - andamandaam | 03 - ntcallmebfudo | 03 - abaaaaa |
| 04 - loworldhel | 04 - ndamandaama | 04 - tcallmebfudon | 04 - baaaaaa |
| 05 - oworldhell | 05 - damandaaman | 05 - callmebfudont | 05 - aaaaaab |
| 06 - worldhello | 06 - amandaamand | 06 - allmebfudontc | 06 - aaaaaba |
| 07 - orldhellow | 07 - mandaamanda | 07 - llmebfudontca | 07 - aaaabaa |
| 08 - rldhellowo | 08 - andaamandam | 08 - lmebfudontcal | |
| 09 - ldhellowor | 09 - ndaamandama | 09 - mebfudontcall | |
| 10 - dhelloworl | 10 - daamandaman | 10 - ebfudontcallm | |
| | 11 - aamandamand | 11 - bfudontcallme | |
| | | 12 - fudontcallmeb | |
| | | 13 - udontcallmebf | |
| (a) 10 = dhelloworl | (b) 11 = aamandamand | (c) 6 = allmebfudontc | (d) 5 = aaaaaab |

Figura 3. Problema: UVA 719 - Glass Beads. Casos de entrada que contienen la descripción del collar y el número de la perla que es la primera en la peor separación posible. Cada perla está representada por un carácter en minúscula del alfabeto inglés ($a-z$), donde $a < b < \dots < z$. (a) *helloworld*. (b) *amandamanda*. (c) *dontcallmebfu*. (d) *aaabaaa*.

Implementación 1: Autómata de sufijo para la solución del problema UVA 719 - Glass Beads

```

1  /**
2  * $ g++ -o GlassBeads GlassBeads.cpp
3  * $ ./GlassBeads < Input.in > Output.out
4  */
5  #include<bits/stdc++.h>
6
7  using namespace std;
8
9  struct state {
10     int len, link;
11     map<char, int> next;
12 };
13
14 vector<state> st;
15 int sz, last;
16
17 void sa_init(int size) {
18     st.clear();
19     st.resize(2 * size);
20     st[0].len = 0;
21     st[0].link = -1;
22     sz = last = 0;
23     sz++;
24 }
25
26 void sa_extend(char c) {
27     int cur = sz++;
28     st[cur].len = st[last].len + 1;
29     int p = last;
30     while (p != -1 && !st[p].next.count(c)) {
31         st[p].next[c] = cur;
32         p = st[p].link;
33     }
34     if (p == -1)
35         st[cur].link = 0;
36     else {
37         int q = st[p].next[c];
38         if (st[p].len + 1 == st[q].len)
39             st[cur].link = q;
40         else {
41             int clone = sz++;
42             st[clone].len = st[p].len + 1;
43             st[clone].next = st[q].next;
44             st[clone].link = st[q].link;

```

```

45         while (p != -1 && st[p].next[c] == q) {
46             st[p].next[c] = clone;
47             p = st[p].link;
48         }
49         st[q].link = st[cur].link = clone;
50     }
51 }
52 last = cur;
53 }
54
55 void sa(string x) {
56     int xsize = x.size();
57     sa_init(xsize);
58     for(int i = 0; i < xsize; i++) {
59         sa_extend(x[i]);
60     }
61 }
62
63 int UVa719_GlassBeads(const string S) {
64     int at = 0;
65     int length = 0;
66     int ssize = S.size();
67     while(length != ssize) {
68         for (auto it : st[at].next) {
69             at = it.second;
70             length++;
71             break;
72         }
73     }
74     return (st[at].len - 1) - ssize + 2LL;
75 }
76
77 int main() {
78     ios::sync_with_stdio(false);
79     cin.tie(nullptr);
80     int N;
81     cin >> N;
82     while(N--) {
83         string A;
84         cin >> A;
85         string AA = A + A;
86         sa(AA);
87         cout << UVa719_GlassBeads(A) << endl;
88     }
89     return 0;
90 }

```

Glosario de términos

AFD *Autómata Finito Determinista*. [2](#), [3](#)

alfabeto Conjunto finito de símbolos. Un alfabeto se indica normalmente con Σ , que es el conjunto de letras en un alfabeto. [2](#)

cadena Una cadena finita formada por la concatenación de un número de símbolos. [3](#)

puntos finales Dos vértices conectados por una arista. [1](#)

subcadena Una subcadena (segmento, subpalabra o factor) de una cadena es cualquier secuencia de símbolos consecutivos que aparecen en la cadena. En lenguaje formal, t es una subcadena de S sí y sólo si existe $x, y \in \Sigma^*$ tal que $S = xty$. [3](#)

sufijo Un sufijo es una subcadena que aparece al final de una cadena. Formalmente, t es un sufijo de T sí y sólo hay algún $x \in \Sigma^*$ tal que $T = xt$. [3](#)

símbolo Un dato arbitrario que tiene algún significado o efecto en la máquina. A estos símbolos también se les llama "letras" o "átomos". [2](#)

Referencias

- [1] S. Even, *Graph algorithms*. Cambridge University Press, 2011.
- [2] Wikipedia, “Autómata finito — wikipedia, la enciclopedia libre,” 2020.
- [3] T. Biegeleisen, “regex - Regular expression for even number of 0’s and even number of 1’s - Stack Overflow,” 2015.
- [4] Wikipedia contributors, “Suffix automaton — Wikipedia, the free encyclopedia,” 2020.
- [5] M. Mohri, P. Moreno, and E. Weinstein, “General suffix automaton construction algorithm and space bounds,” *Theor. Comput. Sci.*, vol. 410, p. 3553–3562, Sept. 2009.
- [6] M. Crochemore and C. Hancart, *Automata for Matching Patterns*, pp. 399–462. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997.
- [7] S. A. P. Cali, “Autómatas de sufijos.”
- [8] Akshay Jaggi, “Suffix automata,” 2016.
- [9] CP-Algorithms, “Suffix automaton,” 2020.