ANÁLISIS DE ALGORITMOS PARA LA CONSTRUCCIÓN DE ARREGLO DE SUFIJOS EN TIEMPO LINEAL

César Alejandro Arango Palacio*
Ricardo Baeza-Yates**

Recibido: 14/08/2008
Aceptado: 08/10/2010

RESUMEN

El avance en la investigación de los arreglos de sufijos permitió en el año 2003 el planteamiento de tres algoritmos de tiempo lineal para la generación de tales estructuras. Anterior a estos se venía trabajando con algoritmos de orden O(n log n) con probada calidad para la obtención de los arreglos. Desde este punto de partida planteamos la necesidad de conocer experimentalmente el rendimiento de los algoritmos O(n) frente a los de orden O(n log n) y conocer entre los planteados de orden *n* cual es el mejor en términos de tiempo de ejecución y uso de recursos computacionales. Después de un profundo trabajo de investigación, pruebas y análisis en el laboratorio podemos concluir basados en los resultados experimentales y los criterios de tiempo y recursos que los algoritmos O(n log n) en este caso alcanzan un mejor rendimiento que los O(n).

Palabras clave: arreglos de sufijos, orden de magnitud, indexación, tiempo lineal, algoritmo

* Ingeniero de Sistemas de la Universidad de Medellín, Especialista en Sistemas de Información de la Universidad EAFIT. Medellín, Colombia. SAP BI Certified. Jefe de Proyectos de Informática PREBEL S.A. Correo electrónico: cesar.arango@prebel.com.co.

** VP of Research for EMEA &LatAm.Yahoo! Research Barcelona.Ph.D. in Computer Science. Correo electrónico: ricardo.baeza@barcelonamedia.org. Web: research.yahoo.com. Personal site: www.baeza.cl Barcelona Media InnovationCtre.

ALGORITHMS ANALYSIS FOR CONSTRUCTION OF SUFFIX ARRAYS IN LINEAR TIME

ABSTRACT

The advance in the investigation about suffix arrays allowed the publication for three new algorithms about the linear time construction in 2003. Before that, it had been working with algorithms in the order O(n log n) which were considered the best in this topic. At this point, it was decided to work on a project for proving which one of the algorithms was better in terms of time and computational resources. Algorithms O(n) vs were compared. O(n log n) and the O(n) vs. O(n) were also compared. Later, after the investigation at the lab, it was discovered that algorithms in the order O(n log n) are better than the others in O(n) based on time and computational resources

Key words: suffix array, orders of growth, index, linear time, algorithm.

INTRODUCCIÓN

Durante los últimos años la teoría referente a los arreglos de sufijos ha tenido un gran avance, entre el 2003 y 2004 se presentaron algunos muy significativos, tras la publicación de diferentes algoritmos para la creación de arreglos de sufijos en tiempo lineal. Partiendo de esto decidimos realizar un análisis en el laboratorio para determinar cuál presenta un mejor rendimiento, en cuanto a tiempo de proceso y a porcentaje de recursos de memoria principal utilizados.

Durante los meses de mayo y junio de 2004 se realizaron pruebas computacionales con cinco algoritmos para la construcción de arreglos de sufijos. Tres de ellos usan tiempo lineal O(n) para cumplir el proceso y los otros dos necesitan de O(n log n); estos últimos sirven como punto de referencia para realizar comparaciones tanto en alfabeto de enteros como en el caso de caracteres, como se vera más a profundidad en el desarrollo de este trabajo.

A continuación se muestran y analizan los resultados obtenidos mediante este estudio de los distintos algoritmos para la indexación de arreglos de sufijos en tiempo lineal.

1. MATERIALES Y MÉTODOS

Teóricamente los algoritmos se definen con un orden de complejidad o magnitud el cual se denota en una escala de orden, desde O(1) que denota el orden constante, pasando por el O(log n) de orden logarítmico, llegando a los ordenes cuadráticos $O(n^2)$, polinomiales $O(n^a)$ entre otros. En este análisis nos referimos específicamente a los órdenes O(n), el cual se denomina lineal, y el orden O(n log n). En esta escala ascendente los algoritmos de O(n) alcanzan siempre un mejor desempeño en cuanto tiempo frente a la cantidad de datos procesados.

Para realizar el análisis se toman los algoritmos de orden O(n) planteados en los siguientes papers:

- Simple Linear Work Suffix Array Construction [1], (skew), y el código fuente de Peter Sanders. [2]
- A Fast Algorithm for Constructing SuffixArraysforFixed-SizeAlphabets [3], (kim), cuyocódigo fuente fue proporcionado por el señor Dong Kyue Kim.
- Space Efficient Linear Time Construction of Suffix Array [4], (alu), y el códigofuentede Pang Ko. [5]

De ordenO(n log n):

- SuffixArrays: a new methodfor online stringsearches [6], implementado por Doug McIlroy's, (manber), códigofuenteobtenido de la página de Bell-Labs. [7]
- FasterSuffixSorting [8], (larsson), código fuente tomado de la página de Jesper Larsson. [9]

Para las pruebas con alfabetos de enteros se usaron los tres algoritmos de orden lineal comparados con respecto a *Larsson*. En el caso del alfabeto de enteros se usó el algoritmo *alu*O(n), comparado con *manber* O(n log n). Se tuvo en cuenta el algoritmo planteado en Linear-Time Construction of SuffixArrays [10] pero se descartóya que el planteado en [3] es una mejoría de este.

Para realizar las pruebas con alfabetos de enteros, se desarrolló una aplicación llamada *tester_linear* que tiene las siguientes características:

Esta aplicación, desarrollada en C bajo Linux, permite llamar a los diferentes métodos de indexación de arreglos de sufijos, *tester_linear*entrega para cada algoritmo el tiempo de usuario que se consume en la tarea de generar el arreglo. Al momento de ejecutar la aplicación se pasan los siguientes parámetros:

tester [in] [out] [método] MB

tester: nombre del archivo ejecutable.

in: archivo de entrada para ser indexado.

out: archivo de salida donde se guarda el arreglo de sufijos.

método: se define con que algoritmo se desea hacer la indexación así:

- *larsson*
- *alu*
- *skew*
- *kim*

Los cuatro algoritmos adaptados al tester_linear trabajan para indexar alfabetos comparables de enteros y long (4 bytes).

Para realizar las pruebas con alfabetos de caracteres (1 byte), se hizo la ejecución directa de la aplicación generada con los algoritmos manber y larsson. Para la ejecución se tomaron partes del archivo "ohsumed.88-91" desde 10MB hasta 100MB.

Los archivos usados para las pruebas fueron:

- ohsumed.87; 60MB
- ohsumed.88-91; 324MB

Estos conforman la colección de datos TREC-9 obtenida de Text Retrieval Conference(TREC) [11]y se consideran información confiable y aceptada para realizar las pruebas requeridas para lograr el objetivo del estudio sobre los algoritmos de tiempo lineal para la indexación de arreglos de sufijos. La colección de datos OHSUMED es un set de 348.566 referencias del área de la medicina, que recoge información del periodo comprendido entre 1987 y 1991.

Las pruebas se realizaron utilizando una máquina con un procesador Intel Xeon de 3.06GHZ y con 2GB de RAM, al momento de realizar las pruebas se tenían el 90% de recursos disponibles en cuanto a procesador y alrededor del 70% en cuanto a RAM.

2. RESULTADOS

Se realizaron las siguientes pruebas:

Archivo: ohsumed.87

Se utilizó para la creación de arreglos de sufijos con alfabeto de enteros.

Se corrieron 180 pruebas bajos las condiciones definidas para el proyecto, 45 con cada uno de los algoritmos probados, se accedió a nivel de 1, 2, 5, 10, 20... 60 MB con *tester_linear.*

Archivo: ohsumed.88-91

Tomamos éste para realizar pruebas con alfabetos de enteros y de caracteres.

Se corrieron para alfabeto de enteros 200 pruebas y 100 para el de caracteres, en ambos casos continuando con las condiciones definidas para el estudio. En el primer caso se accedió desde 10MB hasta 100MB mediante la utilización de *tester_linear*, para el segundo se crearon archivos en el rango de 10 a 100 MB.

Los resultados se presentan codificados así:

larsson = Larsson
skew = Sanders
alu = Aluru
kim = Kim
manber = Manber

En la tabla 1 y la figura 1 se muestra el tiempo promedio que toma cada uno de los métodos para finalizar el proceso de creación de arreglos de sufijos. En este caso las pruebas han sido realizadas con ohsumed.87.

En la tabla 2 y la figura 2 se aprecian las pruebas realizadas con el archivo ohsumed.88-91 bajo las condiciones mencionadas anteriormente.

Con los datos obtenidos podemos observar características en cuanto al tiempo de usuario necesario para la indexación de X MB de datos; además de esta información, durante las pruebas obtuvimos datos que nos permiten conocer el porcentaje

Tabla 1. Tiempo promedio para creación de arreglos de sufijos con alfabeto de enteros (ohsumed.87).

	1MB	2MB	5MB	10MB	20MB	30MB	40MB	50MB	60MB
Larsson	0,690	1,306	3,684	8,230	18,550	30,316	42,882	57,104	67,684
Sanders	2,590	5,592	14,776	32,414	66,318	107,248	146,526	186,640	220,156
Aluru	2,344	5,078	13,758	29,050	61,178	93,584	126,090	160,660	185,622
Kim	2,134	4,856	13,210	27,916	58,644	91,280	125,848	161,244	187,350

Fuente: elaboración de los autores.

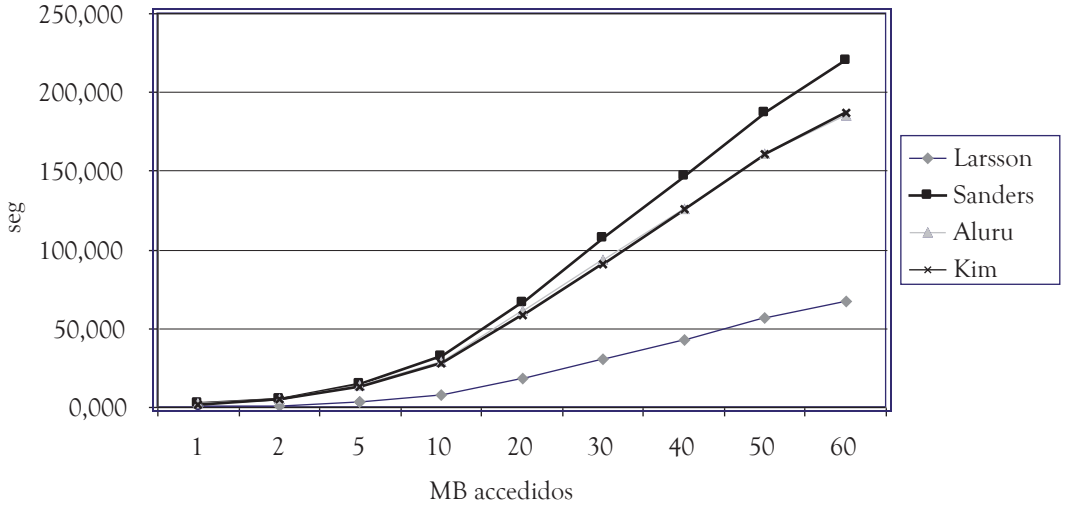

Figura 1. Comparación de algoritmos para indexación de arreglo de sufijos con alfabeto de enteros (ohsumed.87)

Fuente: elaboración de los autores.

Tabla 2. Tiempo promedio para creación de arreglos de sufijos con alfabeto de enteros (ohsumed.88-91)

	10MB	20MB	30MB	40MB	50MB	60MB	70MB	80MB	90MB	100MB
Larsson	8,288	18,550	30,446	43,050	56,372	70,554	85,012	100,388	114,510	134,002
Sanders	27,770	66,720	103,714	146,036	185,304	231,102	275,464	317,758	369,352	N/A
Aluru	29,022	60,916	93,560	126,368	159,384	193,802	228,056	263,544	298,282	336,562
Kim	27,770	59,488	92,174	125,830	161,692	196,664	232,936	270,048	310,23	N/A

Fuente: elaboración de los autores

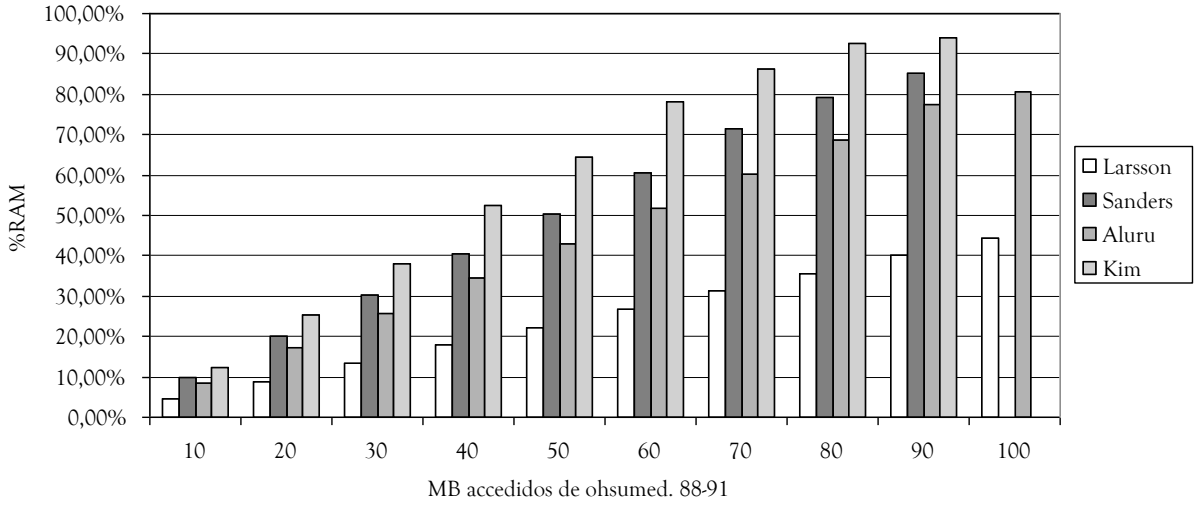

Figura 2. Comparación de algoritmos para indexación de arreglos de sufijos con alfabeto de enteros (ohsumed.88-91)

Fuente: elaboración de los autores.

máximo de memoria principal utilizada durante la ejecución de los procesos. Dichos porcentajes son presentados en la tabla 3 y la figura 3. A partir de éstos, obtuvimos la cantidad máxima de MB de RAM usados durante la ejecución de las pruebas por cada proceso, en este caso se presentan la tabla 4 y la figura 4.

A continuación, presentamos los resultados obtenidos utilizando un alfabeto de caracteres. Cabe resaltar que en estas pruebas sólo se usaron dos algoritmos, éstos reciben directamente texto y luego lo procesan; los demás algoritmos analizados no reciben caracteres (1 byte). Si se quisiera indexar un alfabeto de caracteres con los algoritmos usados en las pruebas descritas anteriormente se haría un pre-procesamiento para convertir los caracteres a enteros, de esta manera los cuatro algoritmos descritos en las pruebas anteriormente expuestas

Tabla 3. Porcentaje máximo de uso de RAM durante ejecución tomando ohsumed.88-91

	10MB	20MB	30MB	40MB	50MB	60MB	70MB	80MB	90MB	100MB
Larsson	4,50%	8,90%	13,40%	17,80%	22,30%	26,70%	31,20%	35,60%	40,10%	44,50%
Sanders	10,00%	19,90%	30,40%	40,40%	50,50%	60,60%	71,50%	79,20%	85,20%	N/A
Aluru	8,60%	17,20%	25,80%	34,40%	43,00%	51,60%	60,20%	68,80%	77,40%	80,60%
Kim	12,40%	25,50%	37,90%	52,30%	64,30%	78,10%	86,40%	92,60%	93,90%	N/A

Fuente: elaboración de los autores.

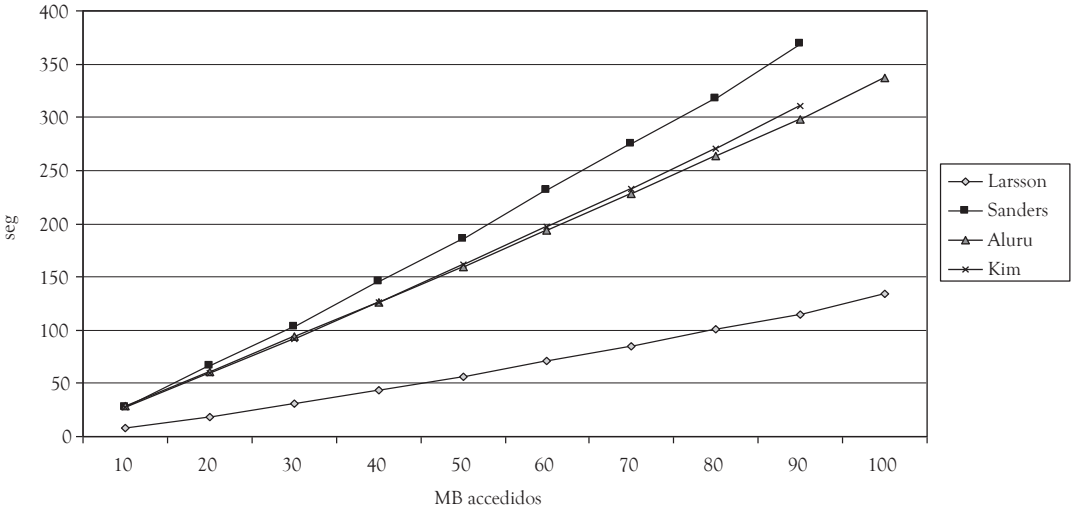

Figura 3. Comparativo de máximo porcentaje de RAM usada para procesar ohsumed.88-91

Fuente: elaboración de los autores

Tabla 4. Cantidad de KB de RAM usados para indexar ohsumed.88-91

	10MB	20MB	30MB	40MB	50MB	60MB	70MB	80MB	90MB	100MB
Larsson	93370	184666	276036	396331	462702	553997	647368	738663	832033	923329
Sanders	207490	412904	630768	838258	1047822	1257387	1483551	1643318	1767811	
Aluru	178441	356882	535323	713764	892205	1070646	1249087	1427528	1605970	1672366
Kim	257287	529098	786386	1085171	1334158	1620494	1792710	1921354	1948327	

Fuente: elaboración de los autores.

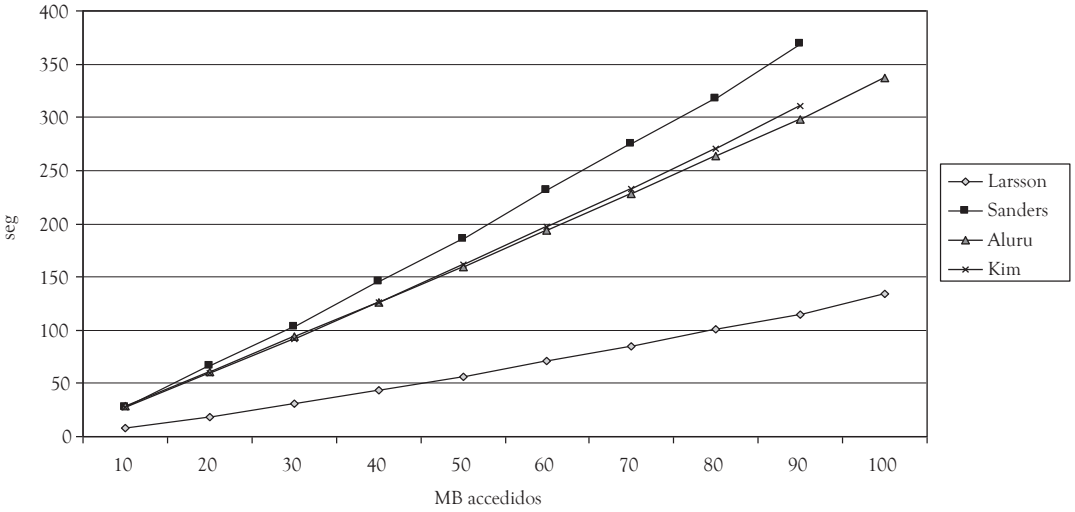

Figura 4. Comparativo de la cantidad de KB usados para indexar ohsumed.88-91

Fuente: elaboración de los autores.

son aplicables a cualquier alfabeto que se pueda convertir a enteros.

Los resultados presentados en la tabla 5 y la figura 5 para los alfabetos de caracteres se obtuvieron de un análisis completo del tiempo de usuario necesario para la indexación de los caracteres en un arreglo de sufijos. Recordemos que para estas pruebas se usaron particiones del archivo ohsumed.88-91.

Cabe anotar que todos los resultados se registraron para emplear un método de comparación lineal buscando un factor de 4 dígitos significativos.

3. DISCUSIÓN

Basados en los resultados expuestos en la sección anterior se abordaron los dos frentes de análisis definidos para los resultados, desde el punto de vista de tiempo y uso de memoria principal; también desde la división realizada para alfabeto de enteros y de caracteres.

Alfabeto de enteros: es posible apreciar que el 100% de los algoritmos probados presentan un comportamiento similar en consumos de memoria y tiempo necesario para indexar entre 1 y 10 MB como se aprecia en las pruebas realizadas sobre ohsumed.87 en la tabla 1 y la figura 1, el algoritmo presentado en [8] a partir de los 20MB presenta grandes diferencias con los demás tanto en tiempo como de cantidad de RAM utilizada; esto es posible apreciarlo en tabla 2 y gráfico 2. Un punto adicional que se analizo, referente al máximo porcentaje de RAM utilizada durante el proceso de generación del arreglo de sufijos también demostró que el algoritmo presentado en [8] presenta un comportamiento superior a los demás algoritmos [1, 3, 4].

Alfabeto de caracteres: analizando los aspectos estudiados para los dos algoritmos disponibles para este tipo de alfabetos la discusión se centra en la escasa diferencia de uso de recursos y tiempo

Tabla 5: Tiempo promedio para creación de arreglos de sufijos con alfabeto de caracteres.

	10MB	20MB	30MB	40MB	50MB	60MB	70MB	80MB	90MB	100MB
Manber	14,342	32,640	52,710	73,672	95,662	118,708	141,820	165,872	189,824	214,990
Aluru	15,372	32,920	50,794	71,336	89,092	108,046	127,492	147,520	168,258	187,464

Fuente: elaboración de los autores.

de ejecución que se tiene entre [4, 6]; tomando al detalle los datos obtenidos durante las pruebas y que se presentan en la tabla 5 y la figura 5 es posible determinar que a partir de 30MB el algoritmo planteado en [4] representa una mejor solución, para el caso estudiado.

4. CONCLUSIONES

Tras las pruebas y análisis realizados y teniendo como base los resultados obtenidos que anteriormente se expusieron podemos concluir, sin lugar a dudas, que el mejor algoritmo en los aspectos estudiados para la creación de arreglos de sufijos es el planteado en [8]; esto se puede afirmar tras confirmar experimentalmente que, en cuanto a tiempo de usuario y memoria principal requerida para la creación de los arreglos, supera a los demás algoritmos estudiados. Aun siendo un algoritmo de orden O(n log n) que teóricamente es inferior, prevalece ampliamente en los aspectos estudiados sobre los algoritmos de orden O(n); esto respecto a lo analizado con los resultados obtenidos para alfabetos de enteros.

En el caso del alfabeto de caracteres las pruebas no son concluyentes pero podemos determinar que el algoritmo planteado en [4] tiene un mejor comportamiento respecto al planteado en [6]; esta característica de mejor uso de recursos se puede encontrara partir de 30MB y se acentúa con el crecimiento de la cantidad de datos a indexar en el arreglo. Así, en el caso de alfabetos de caracteres el algoritmo estudiado de orden O(n) supera al de orden O(n log n).

– Respecto a cuál de los algoritmos de orden O(n) es mejor según los parámetros del estudio, se puede afirmar que el planteado en [4] es el indicado. En los análisis hechos en cuanto a tiempo de usuario tiene un rendimiento similar al planteado en [3] pero a medida que aumenta la cantidad de datos es superior a éste; en el aspecto de memoria, se determina que presenta un mejor uso de recursos a los de su mismo orden.

REFERENCIAS

[1] J. Kärkkäinen y P. Sanders,"Simple Linear Work Suffix Array Construction", En30th *International Colloquium on Automata, Languages and Programming*, LNCS 2719, Springer 2003, pp. 943-955.

[2] S. Peter. [En línea] Disponible en: http://www.mpi-sb.mpg.de/~sanders/programs/suffix/

[3] D.K. Kim *et al.*, "A Fast Algorithm for Constructing Suffix Arrays for Fixed-Size Alphabets", En *Experimental and Efficient Algorithms*, LNCS 3059, Springer 2004, pp. 301-314

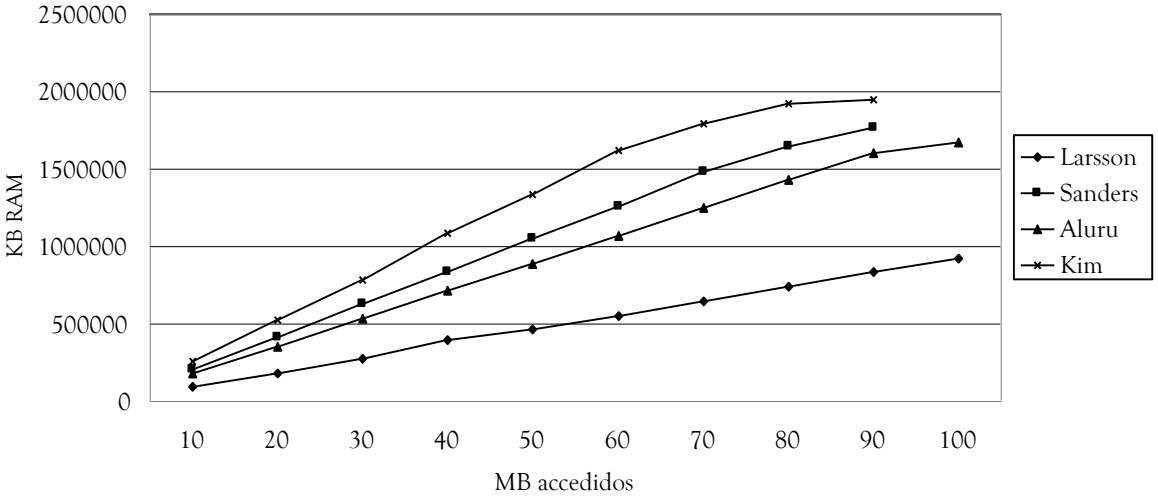

Figura 5. Comparación de algoritmos para indexación de arreglos de sufijos con alfabeto de caracteres.

Fuente: elaboración de los autores.

[4] P. Koy y S. Aluru,"Space Efficient Linear Time Construction of Suffix Array", En*14th Annual Symposium on Combinatorial Pattern Matching*, Springer 2003, pp. 200-210

[5] K. Pang. (2004, May.). [En línea] Disponible http://www.public.iastate.edu/~kopang/

[6] U. Manber y G. Myers, "Suffix Arrays: a new method for online string searches", En *SIAM Journal of Computing 22*, 1993, pp. 935-948.

[7] M. Peter y M. Douglas. (2004, May.) [En línea] Disponible: http://cm.bell-labs.com/cm/cs/who/doug/ssort.c

[8] J. Larsson y K. Sadakane," Faster Suffix Sorting", Tech. Rep. LU-CS-TR: 99-214, LUNFD6/(NFCS-3140)/1-20/(1999), Department of Computer Science, Lund University, Sweden,1999

[9] L. Jesper. (2004, May.) [En línea] Disponible: http://larsson.dogma.net/research.html

[10] D.K. Kim y J.S. Sim, H. Parky K. Park,"Linear-Time Construction of Suffix Arrays", En *Combinatorial Pattern Matching: 14th Annual Symposium, CPM 2003*, LNCS 2676, Springer 2003, pp. 186-199.

[11] Text RetrievalConference (TREC). (2004, May.)[En linea] Disponible: http://trec.nist.gov/data.html