
Fundamentos de Análisis y Diseño de Algoritmos

Mini-Proyecto

Integrantes:

KAROL MARIANA MAYORQUIN HERRERA - 1860682 3743
JENIFFER GUARIN ARISTIZABAL - 1860661 3743
EDINSON ORLANDO DORADO DORADO - 1941966 3743
SANTIAGO RAMIREZ OSPINA - 1841391 3743

Profesor:

JESUS ALEXANDER ARANDA BUENO

FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE SISTEMAS & COMPUTACIÓN

Marzo 22, 2022

1. Resumen y estructura del informe

Lo primero que se presenta en el informe es el pseudocódigo que busca entre todas las posibilidades, una solución óptima, este algoritmo es recursivo, y usa fuerza bruta, o una técnica ciega. Este algoritmo será de utilidad al hacer el análisis de los 2 algoritmos siguientes. Un algoritmo que usa una técnica voraz, y que al compararlo con el primer algoritmo vamos a poder ver cuál es la relación de soluciones óptimas que encuentra el algoritmo voraz comparado con el algoritmo que usa fuerza bruta. El último algoritmo que se muestra y analiza usa la técnica de programación dinámica, que en teoría, (en teoría debido a que esto se va a probar en el análisis de dicho algoritmo), encuentra la mejor solución al problema, al igual que el primer algoritmo, pero de una forma mucho más eficiente.

2. Solución usando fuerza bruta

Una forma segura de encontrar la solución óptima para algún problema, es procesar todas las soluciones. La forma que utiliza el algoritmo mostrado abajo es comparar en cada llamado recursivo los dos caminos posibles que esa instancia del problema tiene. Esto lo hace, por supuesto, llamándose a sí mismo con esas dos instancias, ambas, más pequeñas que el problema original. Esto se hace hasta que se llega a un estado trivial, que es cuando el problema queda de un solo elemento en el array a y b (así mismo el array ab y ba quedan vacíos).

Como para solucionar cada instancia del problema, se crean dos más, el número de operaciones que se deben hacer son 2^n aproximadamente. Lo cual tiene una complejidad $O(2^n)$.

En adición, como para este algoritmo se tiene que especificar desde cuál linea (a o b) empezar, se tiene que utilizar dos veces, uno para 'a' y mirar si se empieza por 'a', o si empezando con 'b' se obtiene una mejor solución.

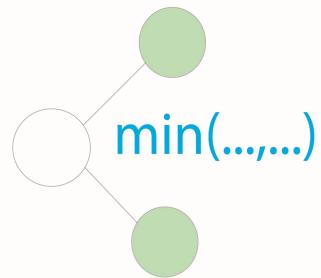
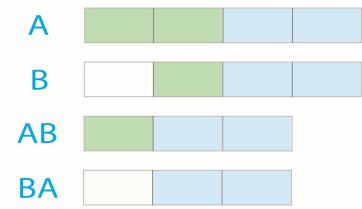
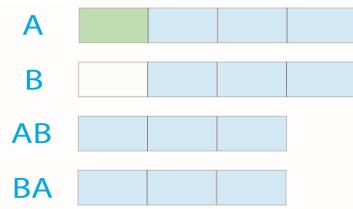
Cabe recordar, que el objetivo con este primer algoritmo es solamente con fines de apoyar el análisis de los dos algoritmos siguientes.

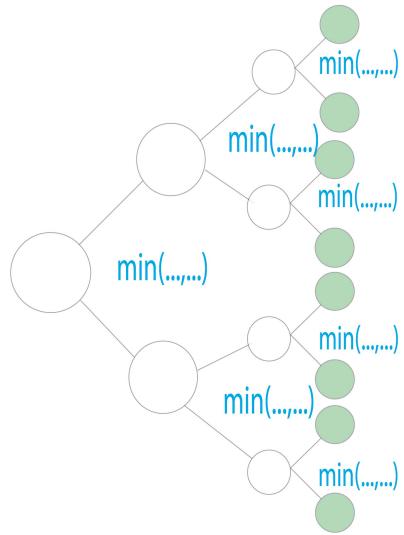
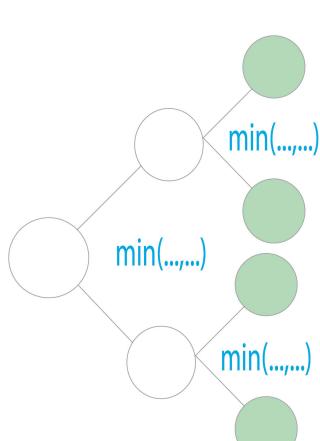
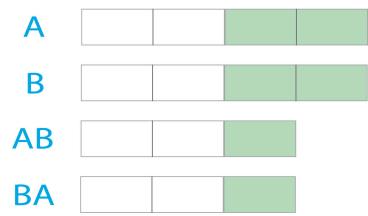
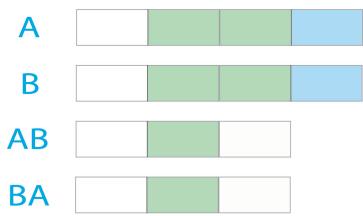
```
1: procedure ALGORITMOCIEGO(n, a, b, ab, ba, counter, lineas, lineaActual)
2:   if (n == 1) then
3:     if (lineaActual =='a') then
4:       return counter + a[len(a) - n], lineas + [a[len(a) - n]]
5:     else
6:       return counter + b[len(b) - n], lineas + [b[len(b) - n]]
7:     end if
8:   else
9:     if (lineaActual =='a') then
10:      return
11:      min(
12:        algoritmoCiego(n-1, a, b, ab, ba, counter + a[len(a) - n], lineas +['a :'+ str(a[len(a) - n]),'a') ,
13:        algoritmoCiego(n-1, a, b, ab, ba, counter + ab[len(ab)-n+1] + a[len(a) - n],lineas+['a :'+str(a[len(a) - n])]+['cab :'+str(ab[len(ab)-n+1])],'b'))
14:      else
15:        return
16:        min(
17:          algoritmoCiego(n-1, a, b, ab, ba, counter + ba[len(ba)-n+1] + b[len(b) - n],lineas+['b :'+str(b[len(b) - n]),'a']),
18:          algoritmoCiego(n-1, a, b, ab, ba, counter + b[len(b) - n], lineas + ['b :'+str(b[len(b) - n]])],'b'))
19:      end if
20:    end if
21:  end procedure
22:
```

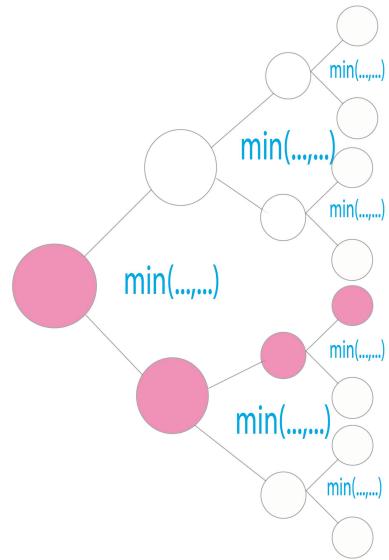
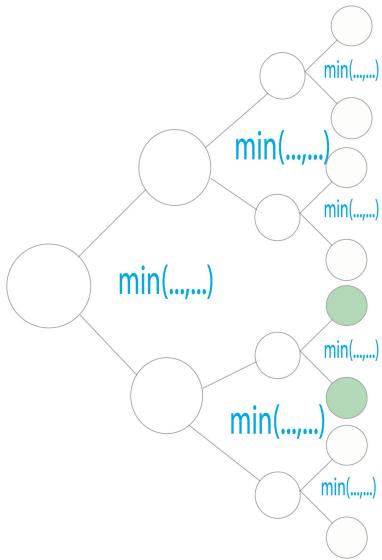
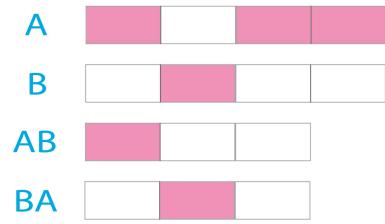
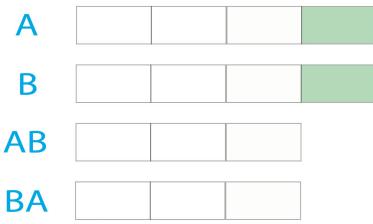
```

17: procedure PROCESAR SALIDA ALGORITMO CIEGO( $n, a, b, ab, ba$ )
18:   llamadoAlgoritmoSiDebeIniciarEnA = None
19:   llamadoAlgoritmoSiDebeIniciarEnB = None
20:   if ( $n > 0$ ) then
21:     llamadoAlgoritmoSiDebeIniciarEnA = algoritmoCiego( $n, a, b, ab, ba, 0, [], 'a'$ )
22:     llamadoAlgoritmoSiDebeIniciarEnB = algoritmoCiego( $n, a, b, ab, ba, 0, [], 'b'$ )
23:   else
24:     raise 'error , n no puede ser 0'
25:   end if
26:   if (llamadoAlgoritmoSiDebeIniciarEnA[0] < llamadoAlgoritmoSiDebeIniciarEnB[0]) then
27:     return [ $n, llamadoAlgoritmoSiDebeIniciarEnA[0], llamadoAlgoritmoSiDebeIniciarEnA[1]$ ]
28:   end if
29:   return [ $n, llamadoAlgoritmoSiDebeIniciarEnB[0], llamadoAlgoritmoSiDebeIniciarEnB[1]$ ]
30: end procedure

```







3. Solución usando Programación Dinámica

3.1. Descripción de la idea

Se hará uso de las tablas hash para indicar cada punto de ensamblaje con su respectiva posición y contenido. La llave estará conformada por: el carril (línea a ó línea b) y la ubicación (posición del punto de ensamblaje). El contenido estará dado por: líneas (camino más óptimo que hay hasta un punto de ensamblaje) y counter (almacena el tiempo total del camino más óptimo que hay hasta un punto de ensamblaje).

Se crea la función principal con los atributos: lineaInicial (indica en que linea se inicia la actividad 1), hash (tabla hash), a (linea a), b (linea b), ab (cambios de "a" a "b"), ba (cambios de "b" a "a"). Además se crea la variable estacion que indicará la posición del punto de ensamblaje que se está analizando, la cual se irá iterando dependiendo de los n puntos de ensamblaje. Se harán dos condiciones principales cuando (estación = 0) y cuando (estación > 0).

En la condición (estación = 0) se determina donde empieza la actividad 1; si (lineaInicial = a), se añade a la tabla hash "b" en la posición 0 con infinito (∞) y si (lineaInicial = b), se añade a la tabla hash "a" en la posición 0 con infinito (∞).

En la segunda condición (estación > 0), se crea la variable estacionDeAtras que será igual a estacion-1 y permitirá "mirar hacia atrás".^{ej}r comparando en cada posición el menor valor que hay entre a y b. Para escoger el camino óptimo y guardarlo en la tabla hash se tiene en cuenta el valor del punto de ensamblaje siguiente en a y b, como también el valor de cada cambio (ab o ba).

En un primer if se analiza los posibles caminos optimos que hay si se toma los puntos de ensamblaje que hay en "a", entonces ubicados en un punto de ensamblaje en a miramos hacia atrás y comparamos el punto que esta justo antes del que se está analizando, si es menor que la suma del punto en "b" de la estacionDeAtras y el cambio ba, si es así, en la tabla hash se almacena la linea "a" y el valor que tiene el punto de ensamblaje que está justo antes del punto que se está analizando en a.

En el segundo if de igual manera se analiza los posibles caminos optimos que hay si se toma los puntos de ensamblaje que hay en "b". Ubicados en un punto de ensamblaje en "b" miramos hacia atrás y comparamos si tal punto es menor que la suma del punto en "a" de la estacionDeAtras y el cambio ab, si es así, en la tabla hash se almacena la linea "b" y el valor que tiene el punto de ensamblaje que está justo antes del punto que se está analizando.

Por último se crea una función que permitirá procesar la información de la función principal, es decir que se encargue de completar y procesar la tabla hash con la respectiva información del camino y tiempo óptimo.

Etapa 1

1	5	9	13
---	---	---	----

15	6	10	14
----	---	----	----

3	7	11
---	---	----

4	8	12
---	---	----

Etapa 2

1	5	9	13
---	---	---	----

15	6	10	14
----	---	----	----

3	7	11
---	---	----

4	8	12
---	---	----

hash = {

$$\begin{array}{|c|c|} \hline a & 0 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline b & 0 \\ \hline \end{array} = \begin{array}{|c|} \hline 15 \\ \hline \end{array}$$

}

hash = {

$$\begin{array}{|c|c|} \hline a & 0 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline b & 0 \\ \hline \end{array} = \begin{array}{|c|} \hline 15 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline a & 1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 5 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline b & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 1 & 3 & 6 \\ \hline \end{array}$$

}

Etapa 3

1	5	9	13
---	---	---	----

15	6	10	14
----	---	----	----

3	7	11
---	---	----

4	8	12
---	---	----

Etapa 4

1	5	9	13
---	---	---	----

15	6	10	14
----	---	----	----

3	7	11
---	---	----

4	8	12
---	---	----

hash = {

a	0
	=
	1

b	0
	=
	15

a	1
	=
	1 5

b	1
	=
	1 3 6

a	2
	=
	1 5 9

b	2
	=
	1 3 6 10

}

hash = {

a	0
	=
	1

b	0
	=
	15

a	1
	=
	1 5

b	1
	=
	1 3 6

a	2
	=
	1 5 9

b	2
	=
	1 3 6 10

a	3
	=
	1 5 9 13

b	3
	=
	1 3 6 10 14

}

3.2. Pseudocódigo

```

1: procedure CREARLLAVE(carril, ubicacion)
2:   return (carril, ubicacion)
3: end procedure
4: procedure CREARCONTENIDO(lineas, counter)
5:   return [lineas, counter]
6: end procedure
7: procedure RECORRERLINEASPRODUCCION(lineaInicial, hash, a, b, ab, ba)
8:   for i in length(a) do
9:     estacion = i
10:    if (estacion == 0) then
11:      if (lineaInicial == 'a') then
12:        hash[crearLlave('a', 0)] = crearContenido(['a'+": -str(a[estacion])], a[estacion])
13:        hash[crearLlave('b', 0)] = crearContenido(['b'+": -str(b[estacion])], math.inf)
14:      end if
15:      if (lineaInicial == 'b') then
16:        hash[crearLlave('b', 0)] = crearContenido(['b'+": -str(b[estacion])], b[estacion])
17:        hash[crearLlave('a', 0)] = crearContenido(['a'+": -str(a[estacion])], math.inf)
18:      end if
19:    else if (estacion > 0) then
20:      estacionDeAtras = estacion - 1
21:      if (hash[('a', estacionDeAtras)][1] < hash[('b', estacionDeAtras)][1]+ba[estacionDeAtras])
22:        then
23:          hash[crearLlave('a', estacion)] = crearContenido(hash[('a', estacionDeAtras)][0]+['a'+": -str(a[estacion])], hash[('a', estacionDeAtras)][1]+a[estacion])
24:        else
25:          hash[crearLlave('a', estacion)] = crearContenido(hash[('b', estacionDeAtras)][0]+['cab'+": -str(ba[estacionDeAtras])]+['a'+": -str(a[estacion])], hash[('b', estacionDeAtras)][1]+ba[estacionDeAtras]+a[estacion])
26:        if (hash[('b', estacionDeAtras)][1] < hash[('a', estacionDeAtras)][1]+ab[estacionDeAtras])
27:          then
28:            hash[crearLlave('b', estacion)] = crearContenido(hash[('b', estacionDeAtras)][0]+['b'+": -str(b[estacion])], hash[('b', estacionDeAtras)][1]+b[estacion])
29:          else
30:            hash[crearLlave('b', estacion)] = crearContenido(hash[('a', estacionDeAtras)][0]+['cab'+": -str(ab[estacionDeAtras])]+['b'+": -str(b[estacion])], hash[('a', estacionDeAtras)][1]+ab[estacionDeAtras]+b[estacion])
31:          end if
32:        end if
33:      end for
34: end procedure

```

```

35: procedure PROCESARLINEASPRODUCCION( $n, a, b, ab, ba$ )
36:   hash = {}
37:   if ( $n > 0$ ) then
38:     if ( $a[0] > b[0]$ ) then
39:       recorrerLineasProduccion('b', hash, a, b, ab, ba)
40:     else
41:       recorrerLineasProduccion('a', hash, a, b, ab, ba)
42:     end if
43:     lineaSiTerminaEnA, counterSiTerminaEnA = hash[('a',n-1)]
44:     lineaSiTerminaEnB, counterSiTerminaEnB = hash[('b',n-1)]
45:     if ( $counterSiTerminaEnA < counterSiTerminaEnB$ ) then
46:       return [ $n, counterSiTerminaEnA, lineaSiTerminaEnA$ ]
47:       return [ $n, counterSiTerminaEnB, lineaSiTerminaEnB$ ]
48:     end if
49:   end if
50: end procedure

```

3.3. Análisis de complejidad

Al analizar el algoritmo anterior, se observa que se hace uso de un for el cual se itera n veces, es decir, que depende de los n puntos de ensamblaje que hay en las líneas de producción y además de eso, en cada iteración se hacen dos operaciones principales, se procesa la enésima estación en la línea de producción "a" y de la misma forma para la línea "b". En estas operaciones principales, en resumen se miran los dos caminos posibles que se tienen para llegar a esa estación: si se llega a esa estación siguiente la línea de producción directamente, o si se llega a esa estación cambiando de línea. La decisión se toma dependiendo de cuál decisión tiene menor costo, y por ende, necesita menos tiempo, minimizando la cantidad de tiempo necesitado para llegar al final de las líneas de producción.

Por tal análisis, teniendo en cuenta que en cada iteración del ciclo for, se hace un número constante de operaciones, se tiene que la complejidad del algoritmo es lineal, siendo exactos de la longitud del array a o del array b.

Cabe destacar que en este algoritmo aunque se logra una complejidad lineal, requiere más espacio en memoria, para guardar lo que se ha procesado en las estaciones anteriores de cada estación, permitiendo ahorrar el número de operaciones que se hacen.

Por último se hace la aceveración de que buscar una llave en el set (o hash), es de complejidad O(1).

Las funciones auxiliares usadas por la función recorrerLineasProducción: CREARLLAVE Y CREARCONTENIDO, se puede asegurar que su complejidad de O(1), debido a que lo que hacen es retornar una tupla con los argumentos y un array con los argumentos respectivamente.

La última función auxiliar tiene una complejidad O(n), debido a que llama la función que recorre y procesa cada estación a y b en cada línea de ensamblaje, y la complejidad de esta es O(1).

Siendo la complejidad total de los 4 algoritmos que dan una solución óptima al problema de líneas de ensamblajes, usando programación dinámica (la complejidad es lineal, no se procesan todos los posibles subproblemas, y se guarda la información de lo procesado anteriormente para ahorrar el número de operaciones a procesar) es de O(n).

3.4. Detalles principales de la implementación

Características del hardware:

Comando utilizado para el Procesador: cat /proc/cpuinfo

Procesador AMD EPYC 7B12

Información del plan de Replit:

Memoria RAM: 500 Mb

Información del plan de Replit:

Disco sólido: 500 Mb

Características del software:

Comando utilizado el Software: cat /etc/os-release

Software Ubuntu

Replit Python

Aspectos propios de las estructuras de datos usadas y conceptos propios de la implementación de los algoritmos:

Una estructura de datos utilizada fue dict (diccionario), la cual nos permitió almacenar como keys la linea de producción y la posición o ubicación de los n puntos de ensamblaje y como values, el contenido del camino óptimo (lineas utilizadas) y el tiempo total, es decir el menor valor que retorna dicho camino.

Es importante recordar la complejidad en el caso promedio que tiene realizar cada operación en la estructura de datos diccionario: -Copy e iteration es $O(n)$

-In, get, set y delete de un objeto es $O(1)$

3.5. Descripción y análisis de las pruebas realizadas

En las pruebas que se realizaron analizamos que el tiempo de ejecución no cambia mucho en entradas con la misma cantidad de puntos de ensamblaje (n) independientemente de los valores que tome hacer cada actividad, se ve diferencia en el tiempo cuando n se hace cada vez más grande.

Tests 1			
Entradas	P. voraz	P. dinamica	Algoritmo Ciego
7 1 2 6 1 5 9 1 3 1 3 2 8 1 9 1 1 1 1 3 5 2 4 2 1 3 2 1	0.000469446 n l[]	0.000061750 18 l['a: 2', 'cab: 1', 'b: 1', 'b: 3', 'cba: 1', 'a: 1', 'a: 5', 'a: 3', 'a: 1']	0.001425027 18 ['a: 1', 'cab: 1', 'b: 1', 'b: 3', 'b: 2', 'b: 8', 'b: 1', 'cba: 1', 'a: 1']
9 5 2 3 4 6 2 7 8 2 1 2 5 4 9 2 4 1 2 3 2 1 1 5 1 2 1 1 1 5 2 4 3 2 1	0.001000165	0.000123500 27 ['b: 1', 'b: 2', 'cba: 1', 'a: 3', 'a: 4', 'a: 6', 'a: 2', 'cab: 1', 'b: 4', 'b: 1', 'b: 2']	0.006434679 27 ['b: 1', 'b: 2', 'cba: 1', 'a: 3', 'a: 4', 'a: 6', 'a: 2', 'cab: 1', 'b: 4', 'b: 1', 'b: 2']
12 2 5 9 6 3 2 5 4 1 2 1 3 1 5 3 6 4 2 1 5 2 2 1 4 1 1 2 1 3 1 4 2 1 3 2 2 3 1 1 4 2 5 1 2 1 3	0.000998735	0.000149965 35 ['b: 1', 'b: 5', 'b: 3', 'b: 6', 'b: 4', 'b: 2', 'b: 1', 'b: 5', 'cba: 1', 'a: 1', 'a: 2', 'a: 1', 'a: 3']	0.041909217 35 ['b: 1', 'b: 5', 'b: 3', 'b: 6', 'b: 4', 'b: 2', 'b: 1', 'b: 5', 'cba: 1', 'a: 1', 'a: 2', 'a: 1', 'a: 3']
15 1 2 5 4 3 6 5 2 1 2 1 2 8 9 3 2 5 8 7 6 3 4 2 5 4 5 8 2 1 5 1 1 1 2 5 4 3 2 6 2 1 2 3 1 2 2 1 3 6 5 1 2 1 1 2 4 7 2	0.000999212	0.000149250 44 ['a: 1', 'a: 2', 'a: 5', 'a: 4', 'a: 3', 'a: 6', 'a: 5', 'a: 2', 'a: 1', 'a: 2', 'a: 1', 'a: 2', 'cab: 2', 'b: 2', 'b: 1', 'b: 5']	0.628857851 44 ['a: 1', 'a: 2', 'a: 5', 'a: 4', 'a: 3', 'a: 6', 'a: 5', 'a: 2', 'a: 1', 'a: 2', 'a: 1', 'a: 2', 'cab: 2', 'b: 2', 'b: 1', 'b: 5']
11 1 2 5 3 4 2 3 1 6 5 2 1 3 2 5 4 7 3 8 2 1 2 1 2 5 4 2 1 2 1 3 2 2 3 1 1 1 2 1 2 3 1	0.000999689	0.000116825 26 ['b: 1', 'b: 3', 'b: 2', 'cba: 1', 'a: 3', 'a: 4', 'a: 2', 'a: 3', 'a: 1', 'cab: 1', 'b: 2', 'b: 1', 'b: 2']	0.072787761 26 ['b: 1', 'b: 3', 'b: 2', 'cba: 1', 'a: 3', 'a: 4', 'a: 2', 'a: 3', 'a: 1', 'cab: 1', 'b: 2', 'b: 1', 'b: 2']
5 1 2 3 4 5 7 6 5 4 3 1 1 1 3 4 2 1 3	0.000146090	0.000058412 14 ['a: 1', 'a: 2', 'a: 3', 'cab: 1', 'b: 4', 'b: 3']	0.000457525 14 ['a: 1', 'a: 2', 'a: 3', 'cab: 1', 'b: 4', 'b: 3']

Tests 2			
Entradas	P. voraz	P. dinamica	Algoritmo Ciego
4 3 6 5 1 2 5 3 4 2 1 3 1 1 2	0.000140726	0.000044107 13 [‘b: 2’, ‘b: 5’, ‘b: 3’, ‘cba: 2’, ‘a: 1’]	0.000248670 13 [‘b: 2’, ‘b: 5’, ‘b: 3’, ‘cba: 2’, ‘a: 1’]
5 3 1 4 8 6 2 3 1 5 7 2 1 1 3 1 2 1 4	0.000059843	0.000153243 18 [‘b: 2’, ‘cba: 1’, ‘a: 1’, ‘cab: 1’, ‘b: 1’, ‘b: 5’, ‘b: 7’]	0.000371217 18 [‘b: 2’, ‘b: 3’, ‘b: 1’, ‘b: 5’, ‘b: 7’]
3 2 1 5 3 2 5 1 1 1 2	0.000041723	0.0000999 8 [‘a: 2’, ‘a: 1’, ‘a: 5’]	0.000148534 8 [‘a: 2’, ‘a: 1’, ‘a: 5’]
7 1 2 6 1 5 2 1 3 1 3 2 8 1 9 1 1 1 3 5 2 4 2 1 3 2 1	0.000090837	0.000233411 8 [‘a: 1’, ‘cab: 1’, ‘b: 1’, ‘b: 3’, ‘cba: 1’, ‘a: 1’, ‘a: 5’, ‘a: 2’, ‘a: 1’]	0.001791954 8 [‘a: 1’, ‘cab: 1’, ‘b: 1’, ‘b: 3’, ‘cba: 1’, ‘a: 1’, ‘a: 5’, ‘a: 2’, ‘a: 1’]

Se puede concluir de la tabla...

3.6. Conclusiones y aspectos para mejorar