Índice

1. Introducción

Este trabajo práctico nos permitió incorporar gráficos para analizar el comportamiento de los algoritmos y poder contrastarlos con los análisis teóricos que veníamos haciendo desde Algoritmos y Estructuras de Datos II. A su vez, nos remarcó la importancia de considerar las dimensiones de los datos que utilizamos y realizar nuestros análisis con modelos en función de los mismos.

Dado que no teníamos conocimientos previos sobre grafos, sus propiedades ni su terminología, cuando hacemos referencia a caminos (simples, extensibles, etc.) hablamos exclusivamente en el marco de nuestras propias definiciones. Todas las herramientas, definiciones y conceptos externos que utilizamos son referenciados e indicados en la sección **Bibliografía**.

1.1. Diseño

1.2. Medición de tiempos

La medición de tiempos de ejecución la realizamos mediante la función `gettimeofday()`, calculando la diferencia de tiempos entre el inicio y el final del ciclo con una precisión de microsegundos. Preferimos medir tiempos frente a la opción de calcular a través del número de operaciones porque nos resultaba una forma novedosa y sencilla de implementación y los valores reflejados nos parecieron más claros. Las mediciones de cada problema fueron corridas en una misma computadora (salvo indicaciones al respecto) para evitar errores causados por el uso de distintos equipos.

1.3. Análisis gráfico

Para el análisis gráfico y estadístico de los tiempos de ejecución recurrimos al paquete sugerido por la materia Qtiplot. El mismo nos permitió graficar las mediciones y realizar ajustes y comparaciones con funciones conocidas. Incluímos el *coeficiente de determinación* de los ajustes (r^2) que fue introducido en clase como medida indicadora de la relación entre nuestros datos y la función utilizada para ajustar (cuanto más cercano es r^2 a 1, más precisa es la comparación.

1.4. Instrucciones de uso

Las soluciones de los ejercicios se pueden compilar desde Unix usando el comando *make*. Desde windows, se puede compilar usando el Dev-CPP, o con el comando *make* si los programas del entorno de programacion *mingw32* estan en el path. Si no, se puede compilar con los siguientes comandos:

- set CXX = ruta completa al compilador g++"
- ruta completa a la utilidad make"

Se proveen también los proyectos para Dev-CPP.

Los ejecutables toman hasta 2 parámetros por línea de comandos en este orden:

- **archivo de salida** *(por defecto: Tp1EjX.out)*
- **archivo de entrada** *(por defecto: Tp1EjX.in)*

(donde X es el número del ejercicio)
El tiempo (en segundos) que tarda en ejecutarse la función junto con el tamaño de la entrada (ej1/2: cantidad de películas - ej3: n) se muestra por consola (stdout).

En los ejercicios 1 y 2, si no se puede abrir el archivo de salida, el resultado se muestra por consola (stderr).
En el ejercicio 3, si no se puede abrir el archivo de salida, o es /dev/null, se descarta.

2. Ejercicio 1

2.1. Introducción

- Decimos que dos películas están *relacionadas* si y sólo si existe al menos un actor o actriz que haya trabajado en ambas.
- Dadas dos películas distintas p y q, un *camino* que va de p a q es una secuencia de películas que empieza en p, termina en q, y tal que todo par de películas consecutivas de la secuencia está compuesto por películas relacionadas.

Nuestro problema es encontrar la longitud del camino más corto entre dos películas dadas.

2.1.1. Pseudocódigo

Algoritmo 1 caminoMasCorto(p, q, tabla)

```
1:  agregar(listaActiva, p)
2:  marcar(p)
3:  #nivel ← 1
4:
5:  while (q ∉ listaActiva) ∧ (listaActiva ≠ φ) do
6:    vaciar(listaPasiva)
7:    for each x in listaActiva do
8:      for each i in relacionadosCon(x) do
9:        if noMarcado?(i) then
10:         agregar(listaPasiva, i)
11:         marcar(i)
12:       end if
13:     end for
14:   end for
15:   #nivel ← #nivel + 1
16:   swap(listaActiva, listaPasiva)
17: end while
18:
19: if noMarcado?(q) then
20:   return -1
21: else
22:   return #nivel
23: end if
```

2.2. Correctitud

Algunas aclaraciones previas:

- Si la película X está relacionada con Y, decimos xRy, o equivalentemente yRx.
- La longitud de un camino es la longitud de la secuencia que lo define. Nótese que el camino más corto que puede existir entre 2 elementos distintos es de longitud 2, mientras que para el camino que va de p a p, la longitud es 1.
- La longitud del camino más corto entre p y q lo llamamos $d(p, q)$.
- Definimos *M(p,i)* como el conjunto de todas las películas x tal que $d(p, x) = i$

Proposición

Luego de la i-ésima iteración del ciclo de la línea 5 vale que: $(\forall h)\ h \in listaActiva \Leftrightarrow h \in M(p, i+1)$

Caso Base: $i = 1$ (primera iteración del ciclo) $(\Rightarrow y \Leftarrow)$

La listaActiva tiene un sólo elemento: p. Entonces la instrucción **for each** de la línea 7 se ejecuta una única vez. El segundo **for each** (línea 8), verificará una por una, si las películas relacionadas con p han sido marcadas, y si no lo fueron, las agregará a la listaPasiva. Como esta es la primera iteración del ciclo, (y en la *tabla* las relaciones de una película no tienen repetidos), podemos afirmar que ninguna de las peliculas estará marcada, por lo que se agregarán a la lista todas

las relacionadas con p. Antes de terminar la iteración se incrementa la variable *#nivel* y se intercambian la listaActiva por la listaPasiva.

Entonces al finalizar la primera iteración vale que $(\forall x)\ x \in listaActiva \Leftrightarrow xRp$ (pues se agregaron a la lista todas las peliculas relacionadas con p). Ahora: si xRp entonces [p,x] es un camino válido entre p y x, y longitud([p,x]) = 2 por lo que es de longitud mínima.

Entonces tenemos que: $listaActiva = M(p, 2) = M(p, 1 + 1)$

Paso Inductivo: $1 < i$ - *Supongo que vale* $P(k)$ *para* $k < i$

$\Leftarrow$) Qvq: Luego de la i-ésima iteración del ciclo, $(\forall h)\ h \in M(p, i + 1)\ \Rightarrow\ h \in listaActiva$

Supongamos que al final de la i-ésima iteración existe una película m tal que $m \notin listaActiva\ \wedge m \in M(p, i + 1)$. Entonces existe un camino mínimo de p a m de longitud $i + 1$, con esta forma: [p , ... , x , m]

Observación: el subcamino [p , ... , x] tiene que ser mínimo, pues sino podríamos construir otro camino de p a x de longitud menor por lo que obtendríamos un camino de longitud menor de p a q, y esto no puede ser porque habíamos supuesto que el que teníamos era mínimo.

Como [p , ... , x] es un camino mínimo de p a x de longitud i, podemos decir que $x \in M(p,\ (i-1)+1)$. Entonces por hipótesis inductiva, al finalizar la iteración $i-1$, $x \in listaActiva$. Es decir que al inicio de la iteración i vale que $x \in listaActiva$.

Vale también que m no está marcado, pues sino debería haber sido agregado a la listaActiva en alguna iteración anterior, y esto (por hipótesis inductiva) implicaría que existe un camino de longitud menor a $i + 1$ entre p y m, lo que es imposible porque supusimos $m \in M(p, i + 1)$.
Pero entonces en el **for each** de la línea 7 deberíamos haber agregado m a la listaPasiva (pues xRm y m no está marcado). Tendríamos entonces, al finalizar la iteración i-ésima, que $m \in listaActiva$.

Absurdo, pues habíamos supuesto que $m \notin listaActiva$. Entonces debe ser $m \in listaActiva$.

$\Rightarrow$) Qvq: Luego de la i-ésima iteración del ciclo, $(\forall h)\ h \in listaActiva\ \Rightarrow\ h \in M(p, i + 1)$

Supongamos que al final de la i-ésima iteración existe una película m tal que $m \in listaActiva\ \wedge m \notin M(p, i + 1)$.
Pero si está en la listaActiva, quiere decir que fue agregada en esta iteración a la listaPasiva. Es decir que m no estaba marcado, y que al iniciar la iteración i-ésima existía una película x en la listaActiva, tal que xRm.

Pero si m estaba en la listaActiva antes de iniciar la iteración i-ésima, quiere decir que estaba también al finalizar la iteración (i-1)-ésima. Por hipótesis inductiva, vale que $x \in M(p, (i - 1) + 1)$.

Entonces existe un camino de p a x de longitud mínima i, de la forma [p , ... , x]. Como mRx, podemos armar un camino de p a m de la forma [p , ... , x , m] con longitud $i + 1$.

Como m no estaba marcado, significa que en ninguna de las iteraciones anteriores se agregó m a la listaActiva. Entonces por hipótesis inductiva, para $j < i$ vale que $m \notin M(p, j + 1)$. Esto equivale a que el camino mínimo de p a m no tiene longitud menor a $i + 1$, es decir que $d(p, m) \geq i + 1$.

Entonces el camino [p , ... , x , m] de longitud $i+1$ es camino mínimo. Es decir que $d(p, m) = i+1$, lo que implica que $m \in M(p, i + 1)$

2.3. Complejidad

Utilizaremos el modelo uniforme para medir la complejidad temporal del algoritmo, pues suponemos que los datos con los que trabajamos tienen una longitud acotada y entran en una única celda de memoria.

2.3.1. Complejidad en función de cantidad de películas

Llamamos:

- n a la cantidad total de películas.
- r a la cantidad total de relaciones entre las películas.

Todas las operaciones individuales son $O(1)$.

Las siguientes líneas se ejecutan $O(n)$ veces

[5 - 6 -15 -16]
A lo sumo hay $O(n)$ iteraciones, ya que el camino más largo que puede existir entre dos películas es n (si existe un camino entre ellos)

[7 - 10 - 11]
Porque cada película puede ser agregada a la listaActiva/Pasiva una sola vez (ya que al ser agregada, es marcada)
Las siguientes líneas se ejectuan $O(r)$ veces:
[8 - 9]
Por cada elemento x de la listaActiva se revisan todas las películas relacionadas con x. Cada elemento está en la listaActiva como máximo una única iteración. Entonces concluimos que se revisan todas las relaciones que existen entre las películas un total de a lo sumo 2 veces.

Entonces podemos decir que la complejidad temporal del algoritmo con respecto a la cantidad de películas (n) es $O(n + r)$. Si tenemos n peliculas, la cantidad de relaciones que pueden existir (r) está acotada superiormente por

$$\sum_{j=1}^{n} j = \frac{n \times (n+1)}{2} \in O(n^2)$$

Entonces la complejidad del algoritmo con respecto a n es $O(n^2)$

2.3.2. Complejidad en función del tamaño de la entrada

Analicemos el tamaño de la entrada para expresar la complejidad en función de ella. Consideremos un conjunto de películas de cardinalidad n. La entrada consiste en n, p, q y para cada película, la cantidad de elementos con los que se relaciona y cuáles son, lo que ocupa:

$$log_2(n) + log_2(p) + log_2(q) + \sum_{i=1}^{n}(log_2(r_i) + \sum_{j=1}^{r_i} log_2(a_{ij}))$$

Como para todo $i\ tal\ que 1 \leq i \leq n$ vale que $log_2(r_i) \geq 1$, luego $\sum_{i=1}^{n} log_2(r_i) \geq n$ y

$$log_2(n) + log_2(p) + log_2(q) + \sum_{i=1}^{n}(log_2(r_i) + \sum_{j=1}^{r_i} log_2(a_{ij})) \geq n$$

Por lo tanto, el tamaño de la entrada en función de n, $E(n)$, pertenece a $\Omega(n)$.

Sabemos por el punto anterior que el costo del algoritmo, $C(n)$ es $O(n^2)$, es decir, $C(n) \leq k * n^2$. Entonces:

$$E(n) \geq n \Leftrightarrow (E(n))^2 \geq n^2 \Leftrightarrow k * (E(n))^2 \geq n^2 * k \geq C(n)$$

Luego $C(n) \in O((E(n))^2)$

2.4. Detalles de implementación

El input lo cargamos en una estructura auxiliar que llamamos *tabla* en la cual tenemos un array de tupla(lista[pelicula] , bool). La posición (i-1)-ésima representa la película i. La primera componente de la tupla es una lista de todas las películas con las cuales i está relacionada, y la segunda indica si la película ya fué revisada durante el transcurso de la ejecución del algoritmo.

Como en el Ejercicio 1 y 2 utilizamos los mismos formatos de archivo input/output, y las funciones tienen la misma aridad, decidimos que la función que se encarga de cargar el archivo, ejecutar el algoritmo, y guardar la salida, fuera la misma. Para esto, creamos la función **cargar** que toma como parámetros:

- El nombre del archivo de entrada
- El nombre del archivo de salida
- La función que se ejecutará

El valor de retorno de la funcioón *cargar* indica si se pudo ejecutar el algoritmo o no.

Como la estructura que utilizamos es esencialmente la misma que tiene el archivo de entrada, la complejidad de generarla es lineal con respecto al tamaño de la entrada, o cuadrática con respecto a la cantidad de películas (en el peor caso).

2.5. Detalles de implementación

Las instancias generadas para realizar las mediciones del algoritmo de longitud del camino más corto fueron variadas. Buscamos reflejar los peores casos mediante dos tipos de instancias: aquellas en las que todas las películas, **excepto** q (que no tiene ninguna relación), están relacionadas entre sí y aquellas en las que nuevamente todas las películas están relacionadas *"todas con todas"* y q posee una única relación con alguna de ellas. Por la forma en que cargamos los datos, dado que en este caso p es la película de la primer línea y q la n-ésima, conviene que la película relacionada con q sea la anteúltima en la lista de relaciones de la entrada. Para poder analizar el comportamiento de nuestro algoritmo en otros casos, generamos las instancias mediante una matriz simétrica M $(n \times n)$ en la que la relación entre la película de la i-ésima fila y la película de la j-ésima columna (con $1 \leq i, j \leq n$ y $j \neq i$) es indicada en la posición M_{ij} con un 1 si están relacionadas y con un 0 si no lo están. Hicimos que el porcentaje de relaciones sea de un 20 % (es decir, una probabilidad del 20 % de colocar un 1), de un 40 % y de un 60 % con el número de películas incrementándose de 5 en 5 desde $n = 10$ hasta llegar a $n = 995$.

Las siguientes figuras ilustran las mediciones de tiempo de nuestro algoritmo en los peores casos. En ambas podemos observar que existe una gran precisión respecto al ajuste cuadrático realizado. A modo ilustrativo se incluyó un ajuste lineal, pero claramente el comportamiento ilustrado por las mediciones se asemeja a una función polinomial de grado 2.

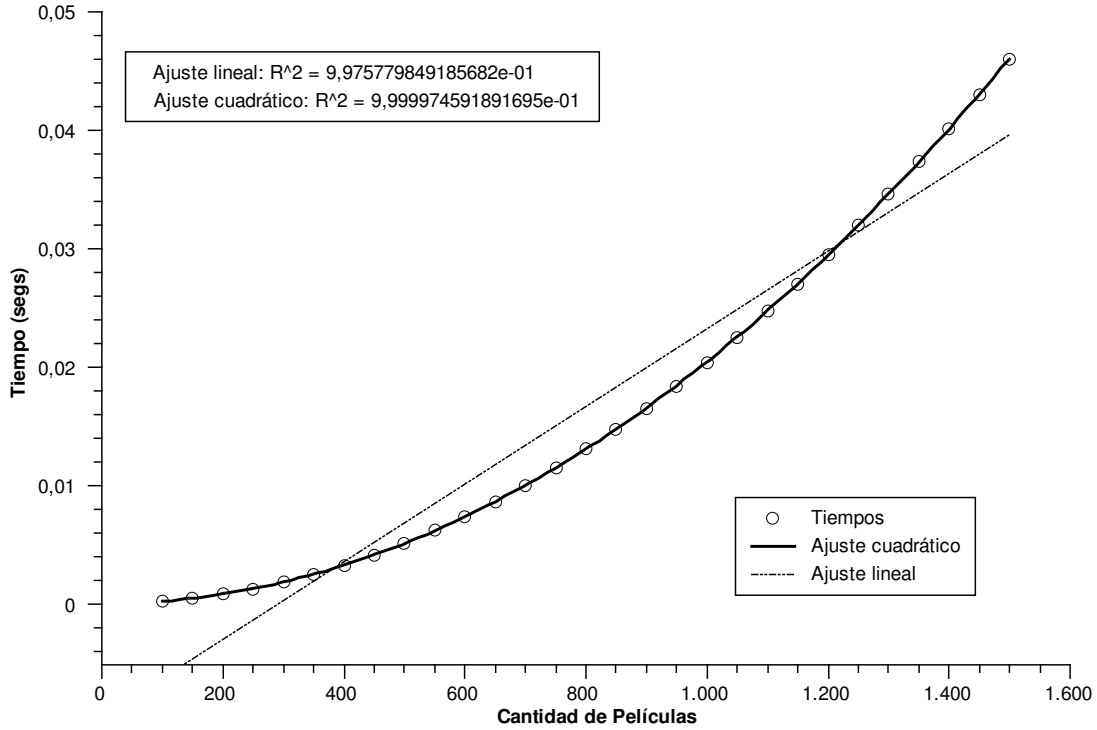

Figura 1: Gráfico para peores casos con q con una única relación y con n = cantidad de películas de 100 hasta 1500

Para observar el comportamiento en un espectro más grande de casos, generamos la próxima figura que incluye las mediciones que resultaron de las distintas matrices con probabilidades de relacionarse entre películas de un 20 %, 40 % y 60 %.

A su vez, incluimos un ajuste cuadrático respecto al peor caso para contrastarlo con los casos generados de forma aleatoria.

Para obtener datos más precisos, evitando errores que dependan de factores externos y aleatorios, repetimos las mediciones corriendo 300 veces un mismo caso (una instancia de *"todas con todas"* con q sin niguna relación) y pudimos observar las siguientes frecuencias de apariciones para cada rango de tiempo.

2.6. Conclusión

Los análisis realizados tanto de forma teórica/analítica como empírica reflejaron los resultados esperados para este algoritmo. Las mediciones plasmaron en las gráficas ajustes precisos con respecto a lo calculado teóricamente para nuestro algoritmo de complejidad $O(n^2)$ en un modelo uniforme.

Dadas las características de los datos que utilizamos (tanto entrada como parámetros en instancias intermedias), la elección del modelo y enfoque teórico del problema resultó acertada en contraste con los resultados obtenidos en los gráficos.

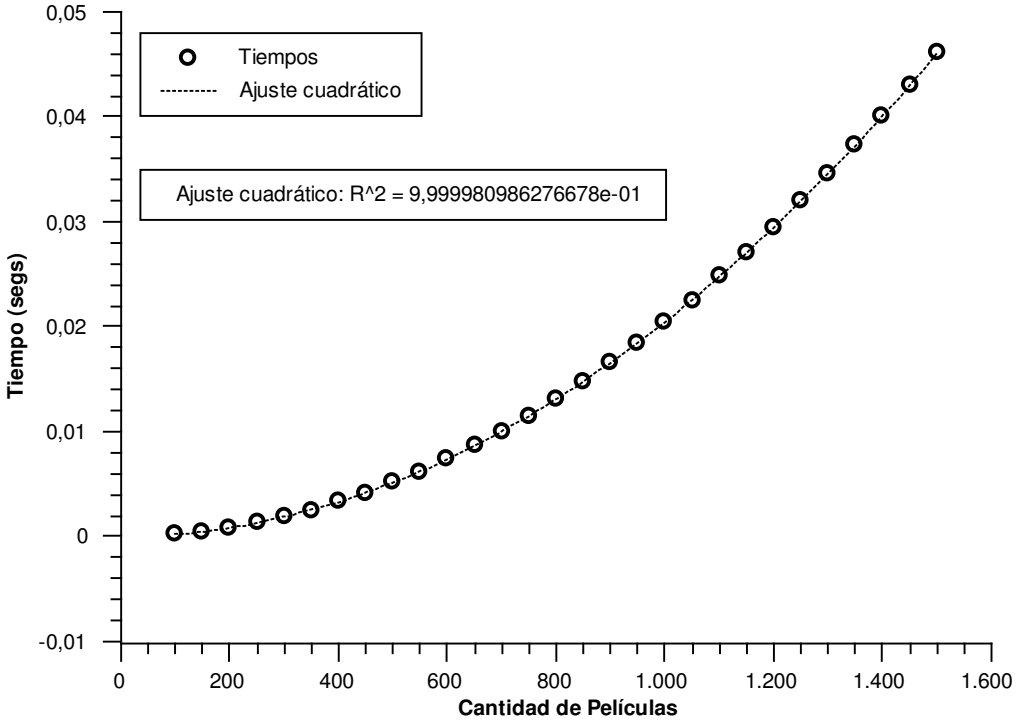

Figura 2: Gráfico para peores casos con q sin ninguna relación y con n = cantidad de películas de 100 hasta 1500

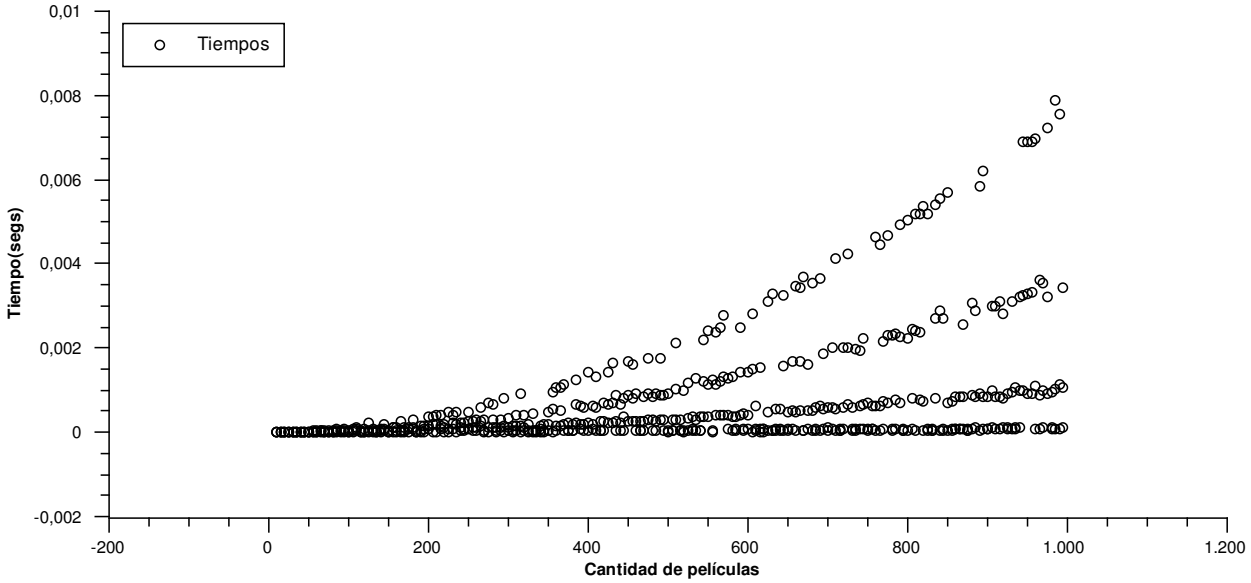

Figura 3: Gráfico de relaciones con un 20, 40 y 60 porciento de probabilidades

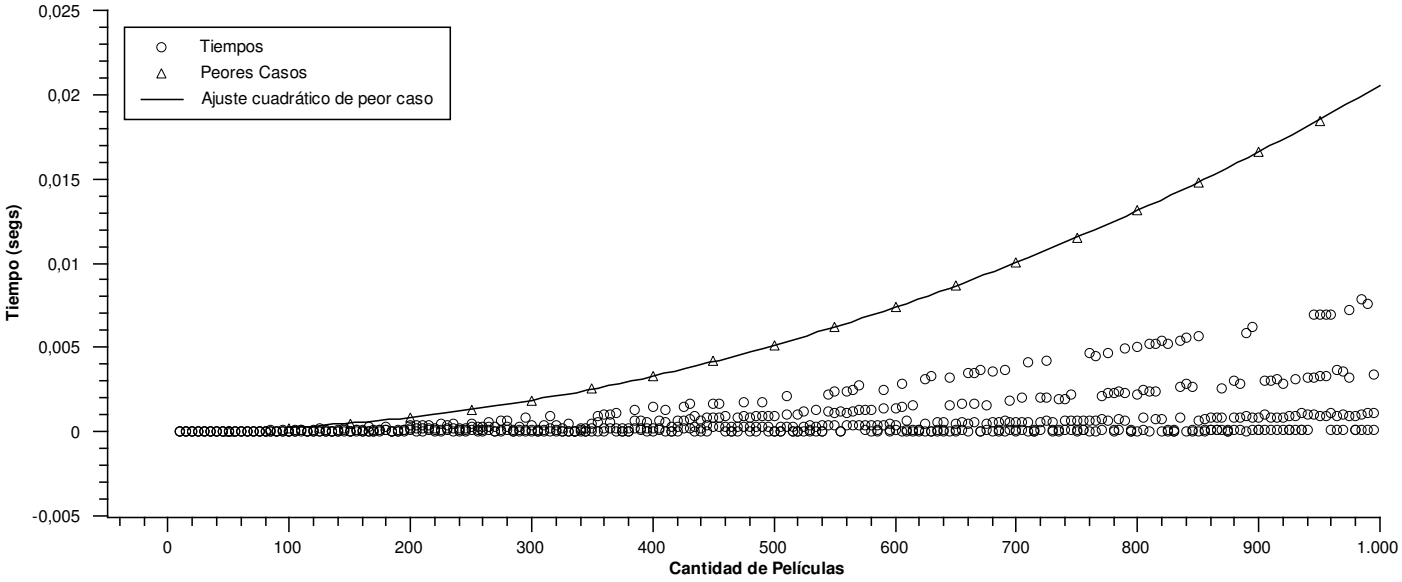

Figura 4: Gráfico de relaciones con un 20, 40 y 60 porciento de probabilidades y ajuste comparativo con peores casos

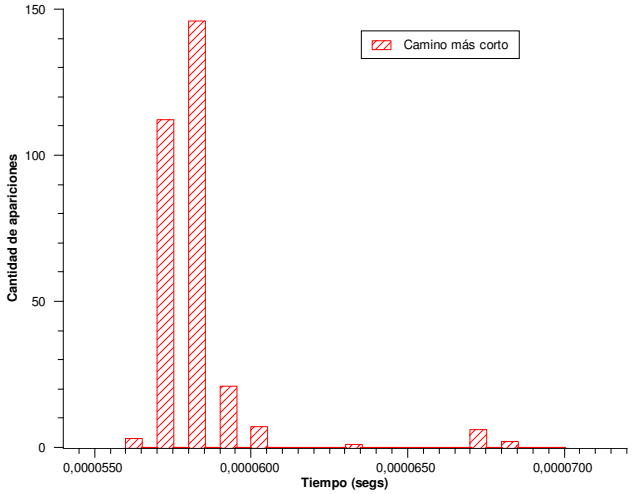

Figura 5: Histograma que ilustra la distribución de los tiempos para una misma instancia de 50 películas para peor caso (q sin relaciones) corrida 300 veces

3. Ejercicio 2

3.1. Introducción

Se cuenta, al igual que en el ejercicio anterior, con un conjunto de $n \geq 2$ películas, sus relaciones y el concepto de *camino*. Además, definimos que un camino es *simple* cuando no contiene películas repetidas. Dadas dos películas distintas p y q del conjunto, se desea determinar la cantidad de películas en el camino *simple más largo* que vaya de p a q y contenga sólo películas del conjunto. Si no existe tal camino, se retornará el valor -1. De aquí en adelante nos referiremos a este camino como *camino máximo* y nos restringiremos a trabajar con caminos formados por películas pertenecientes al conjunto dado.

3.2. Explicación

Diremos que un camino simple C es *extensible* (o diremos que C se puede extender) si y sólo existe una película que no pertenece a C y está relacionada con la última de C. Además, *extender* un camino simple C con una película a es equivalente a agregar a al final de C de manera que C siga siendo simple. Es decir, un camino simple C es extensible si y sólo si se puede extender. Debido a que pueden existir múltiples caminos simples de p a q, exploramos todos ellos para decidir cuál es el más largo. Para esto, partimos de p y alguna de las películas que se relacionan con p (todas ellos, una a la vez) y tratamos de extender ese camino de todas las maneras posibles. Repetidamente, veremos todas las maneras posibles de extender los caminos obtenidos. Si en el proceso encontramos que un camino se puede extender con q, habremos descubierto un camino simple de p a q, candidato a ser el camino máximo. Si su longitud es mayor que la de todos los caminos buscados vistos hasta el momento, la almacenamos en una variable. No será necesario tratar de extender los caminos que contengan a q porque no es posible formar un camino simple a q de esa manera. La manera en que se extenderán los caminos será la siguiente. Comenzamos con un camino formado por p y una película a relacionada con p, si es que tal a existe (si no es así, entonces habremos analizado todos los caminos simples de p a q).

- En el caso de que $a = q$, procederemos como se detalla en el párrafo anterior y luego trataremos de extender un camino formado por p y otra película b relacionada con p con $b \neq a$. Si no existe tal b, entonces habremos analizado todos los caminos buscados.
- De lo contrario, si $a \neq q$, trataremos de extender este camino, que llamaremos C. Si es posible, lo hacemos y vemos si el camino obtenido a su vez se puede extender. Si C no es extensible, pasaremos a estudiar los caminos que se pueden formar a partir de uno constituido por p y otra película b relacionada con p que no hayamos considerado previamente. Si tal b no existe, significa que ya agotamos todos los caminos simples.

Si el camino C que queremos extender tiene longitud mayor que 2, pueden darse tres posibilidades. Las dos últimas no son excluyentes y pueden realizarse en cualquier orden.

- C no es extensible. Luego, quitamos la última película de C y vemos si el camino resultante se puede extender de alguna manera distinta a las que ya han sido probadas.
- C se puede extender con q. Actualizamos el acumulador si la longitud de este camino de p a q es el más largo visto hasta el momento y luego vemos si C se puede extender de alguna manera aún no considerada.
- C se puede extender con a y $a \neq q$. Extendemos C con a y observamos si el camino resultante puede extenderse.

3.3. Pseudocódigo

Algunas observaciones que podemos realizar sobre este algoritmo son:

- Si la longitud de *camino* es mayor que uno, la secuencia que resulta de leer su contenido desde la base hasta el tope es el camino que se desea extender. Llamemos a esa secuencia el camino actual, aun cuando la longitud de camino sea uno.
- La longitud de *camino* es una unidad más pequeña que la longitud de *arbol*, salvo al final de la ultima iteración.
- En *arbol* se apila una referencia a la lista de películas relacionadas con la última del camino actual para luego recorrerla buscando extender este camino. Notemos que alcanza con recorrer esta lista porque si agregamos al camino actual una película que no esté relacionada con la última, éste deja de ser un camino.
- Las únicas películas marcadas en la tabla son las que figuran en el camino actual.
- En una iteración, dentro de *arbol* sólo se modifica la lista que se encuentra en el tope al inicio.

3.4. Complejidad

Usaremos el modelo uniforme para calcular la complejidad de este algoritmo por la misma razón que lo utilizamos en el Ejercicio 1.

Algoritmo 2 maximo(tabla, p, q)

```
1:  pila(pelicula) camino
2:  camino ← vacia
3:  pila(secu(peliculas)) arbol
4:  secu(pelicula) ← s
5:  agregar(s, p)
6:  arbol ← vacia
7:  arbol ← apilar(arbol, s)
8:  int max ← -1
9:  while !vacia(arbol) do
10:   if Ya se vieron todos los elementos de tope(arbol) then
11:     desapilar(arbol)
12:     if !vacia(camino) then
13:       Desmarco el tope de camino
14:       desapilar(camino)
15:     end if
16:   else
17:     if La próxima película de tope(arbol) es q then
18:       if longitud(camino) >= max then
19:         max ← longitud(camino) + 1
20:         Avanzo un lugar en la lista de tope(arbol)
21:       end if
22:     else if La próxima película de tope(arbol) está marcada then
23:       Avanzo un lugar en la lista de tope(arbol)
24:     else
25:       x ← proxima(tope(arbol))
26:       Avanzo un lugar en la lista de tope(arbol)
27:       apilar(arbol, relacionadas(tabla, x))
28:       Marco a x
29:       apilar(camino, x)
30:     end if
31:   end if
32: end while
33: return max
```

3.4.1. Complejidad con respecto a la cantidad de películas

En primer lugar, notemos que todas las operaciones consideradas individualmente tienen complejidad temporal de orden constante. La manera obtener las secuencias de películas relacionadas con otra y de operar con ellas (apilarlas, recorrerlas y despilarlas de *arbol*) es a través de referencias. Por lo tanto, el enfoque que adoptaremos será estimar la cantidad de veces que se ejecuta cada uno de los siguientes cuatro bloques de instrucciones:

10 Ya han sido vistas todas las películas relacionadas con la última del camino actual. Además, q no está en el camino.

17 Se encontró un camino simple de p a q.

22 La próxima película en la lista de candidatos para extender el camino ya está en el camino. Además, q no está en el camino.

24 Se puede extender el camino actual con la próxima película en la lista de candidatos. Además, q no está en el camino.

Dados un camino C y una película p, sólo puede suceder que se intente extender C con p lo sumo una vez. Esto nos permite acotar la cantidad de iteraciones del ciclo sumando la cantidad de ejecuciones de los bloques mencionados, ya que en cada iteración se entra a sólo uno de los cuatro. Cuando escribamos n nos referiremos a la cantidad de películas del conjunto.

Entrar al bloque 17 equivale a encontrar un camino simple de p a q. Si tenemos n películas en total, los nodos por los que podemos pasar para llegar de p a q son un subconjunto de entre 0 y $n-2$ películas distintas, que a su vez pueden estar ordenadas de todas las maneras en las que sea posible. Por lo tanto, podemos entrar a este bloque a lo sumo

$$\sum_{i=0}^{n-2} \binom{n-2}{i} i! = \sum_{i=0}^{n-2} \frac{(n-2)!i!}{i!(n-2-i)!} = \sum_{i=0}^{n-2} \frac{(n-2)!}{(n-2-i)!} \leq \sum_{i=0}^{n-2} (n-2)! = (n-1)! < n!$$

Entrar al bloque 22 equivale a tener un camino simple sin q que empieza en p más una película repetida. Además de p el camino estará formado por entre 1 y $n-2$ películas en algun orden, y al final una película contenida en el camino pero

distinta a la penúltima. A lo sumo, la cantidad de caminos de este tipo es

$$\sum_{i=1}^{n-2} \binom{n-2}{i} i! \times i = \sum_{i=1}^{n-2} \frac{(n-2)!i!i}{i!(n-2-i)!} = \sum_{i=1}^{n-2} \frac{(n-2)!i}{(n-2-i)!} \leq \sum_{i=1}^{n-2} (n-2)!(n-2) = (n-2)!(n-2)^2 < n!$$

Entrar al bloque 24 equivale a tener una secuencia de películas que empieza en p, no contiene a q y no tiene repetidos. Puede tener ninguna película además de p o desde 1 hasta $n-2$ películas distintas de p.y q en algún orden. Esta cantidad de caminos se puede acotar por

$$\sum_{i=0}^{n-2} \binom{n-2}{i} i! = \sum_{i=0}^{n-2} \frac{(n-2)!i!}{i!(n-2-i)!} = \sum_{i=0}^{n-2} \frac{(n-2)!}{(n-2-i)!} \leq \sum_{i=0}^{n-2} (n-2)! = (n-1)! < n!$$

Observemos que el bloque 10 es el único en el que se desapila *arbol*. Además, el ciclo empieza con *arbol* de longitud 1 y termina cuando se vacía. Por lo tanto, la cantidad de veces que se ejecuta este bloque debe ser igual a la cantidad de veces que se apilan elementos más uno. Como el único bloque en el que se apila es el 24, aquella cantidad se puede acotar por

$$(n-1)! + 1 < n!$$

Llamemos I_1, I_2, I_3 e I_4 a la cantidad de veces que se ejecuta 10, 17, 22 y 24 respectivamente. La guarda del ciclo y la de 10 se ejecutan $I_1 + I_2 + I_3 + I_4$ veces; la guarda de 17, $I_2 + I_3 + I_4$ veces; la guarda de 22, $I_3 + I_4$ veces. Llamemos k_p, k_1, k_2, k_3 y k_4 a la cantidad de operaciones previas al ciclo y las de los bloques 10, 17, 22 y 24 respectivamente. En conclusión, la cantidad de operaciones de la funcion *maximo* está acotada por

$$k_p + k_1 \times I_1 + k_2 \times I_2 + k_3 \times I_3 + k_4 \times I_4 + 2 \times (I_1 + I_2 + I_3 + I_4) + I_2 + I_3 + I_4 + I_3 + I_4 =$$

$$= k_p + (k_1 + 2) \times I_1 + (k_2 + 3) I_2 + (k_3 + 4) \times I_3 + (k_4 + 4) \times I_4 \leq$$

$$\leq k_p + (k_1 + k_2 + k_3 + k_4 + 13) \times n! \in O(n!)$$

3.4.2. Complejidad en función del tamaño de la entrada

El tamaño de la entrada es el mismo que analizamos en el Ejercicio 1. Es decir $E(n) \geq n$

Sabemos por el punto anterior que el costo del algoritmo, $C(n)$ es $O(n!)$, es decir, $C(n) \leq k * n!$. Entonces:

$$E(n) \geq n \Leftrightarrow (E(n))! \geq n! \Leftrightarrow k * (E(n))! \geq n! * k \geq C(n)$$

Luego $C(n) \in O((E(n))!)$

3.5. Detalles de implementación

(Ver 2.4)

3.6. Pruebas, resultados y mediciones

Para realizar un análisis de nuestro algoritmo para calcular la longitud del camino más largo generamos una variedad de instancias similiares a las del ejercicio 1. Dado que la complejidad de este algoritmo es muchísimo mayor que la del ejercicio 2, la cantidad de películas en las relaciones y el n de entrada deben ser menores. Los peores casos fueron generados por aquellas instancias en las que absolutamente todas las películas de la entrada están relacionadas entre sí *"todas con todas"*. Para poder analizar el comportamiento de nuestro algoritmo en otros casos, generamos las instancias de la misma forma que en el ejercicio 1, mediante una matriz simétrica M $(n \times n)$ en la que la relación entre la película de la i-ésima fila y la película de la j-ésima columna (con $1 \leq i, j \leq n y j \neq i$) es indicada en la posición M_{ij} con un 1 si están relacionadas y con un 0 si no lo están. Hicimos que el porcentaje de relaciones sea de un 20 % (es decir, una probabilidad del 20 % de colocar un 1), de un 40 % y de un 60 % con instancias de 12 películas para cada porcentaje.

Reflejamos en el gráfico siguiente nuestras mediciones para peores casos, que apuntaban a un algoritmo de una complejidad de O(n!). Al realizar un ajuste por medio de una función exponencial, el coeficiente de determinación daba un número considerablemente cercano a 1. Razón por la cual graficamos las mismas entradas para el eje X, pero en el eje Y reflejamos el logaritmo en base 2 de los tiempos correspondientes a cada entrada. De ésta forma nuestro nuevo gráfico ajustado linealmente ilustra si efectivamente se trata de un caso exponencial o mayor. El ajuste cuadrático más acertado que el lineal nos confirma que efectivamente el algoritmo (en términos gráficos y de mediciones) posee un crecimiento mayor al de una función exponencial.

Para observar el comportamiento en un espectro más grande de casos, generamos la próxima figura que incluye las mediciones para peores casos y las resultantes de las distintas matrices con probabilidades de relacionarse entre películas de un 20 %, 40 % y 60 %.

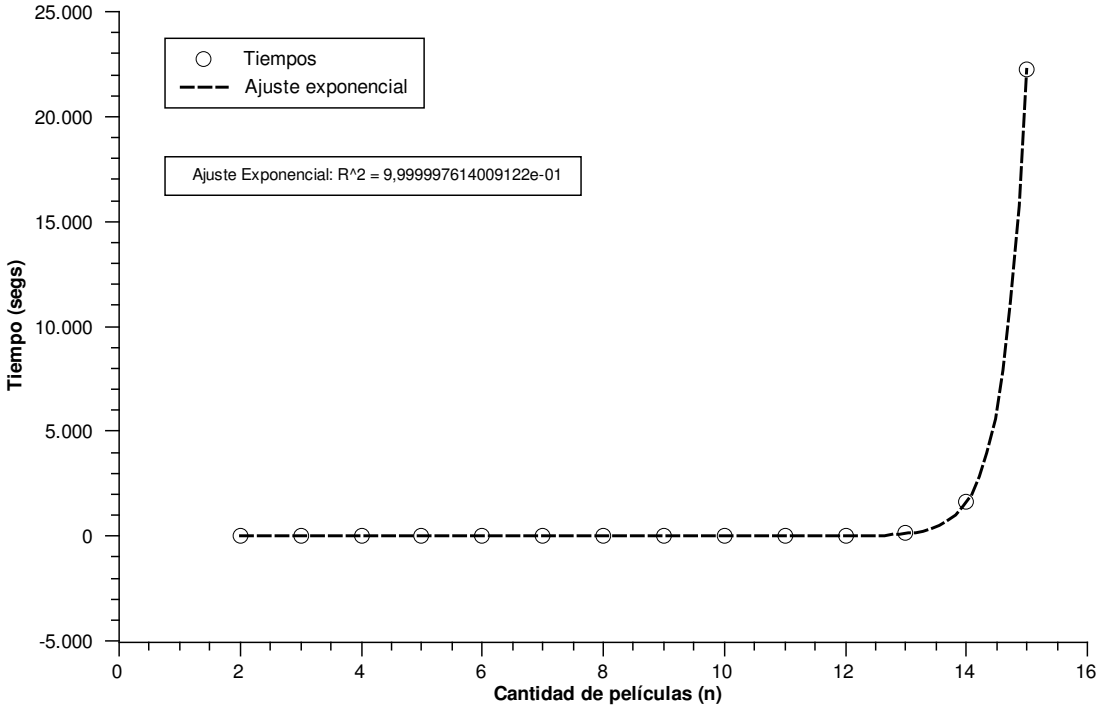

Figura 6: Gráfico para peores casos con n = cantidad de películas de 2 hasta 15 con ajuste exponencial

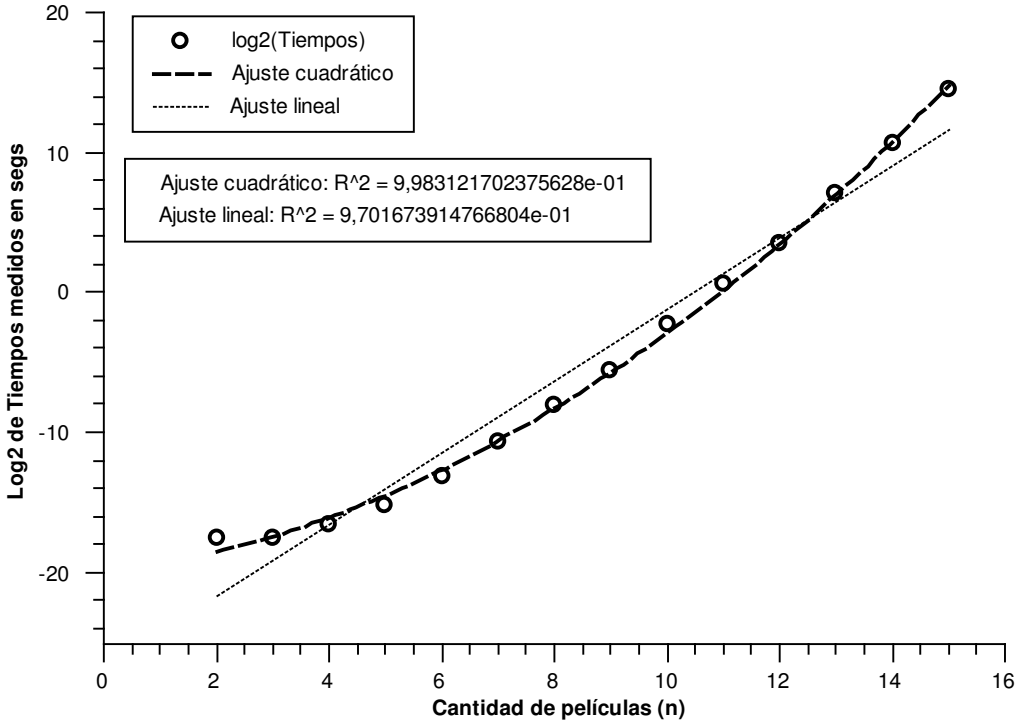

Figura 7: Gráfico para peores casos graficando en el eje Y (ordenadas) log2(Tiempo) y comparando con ajuste lineal y cuadrático

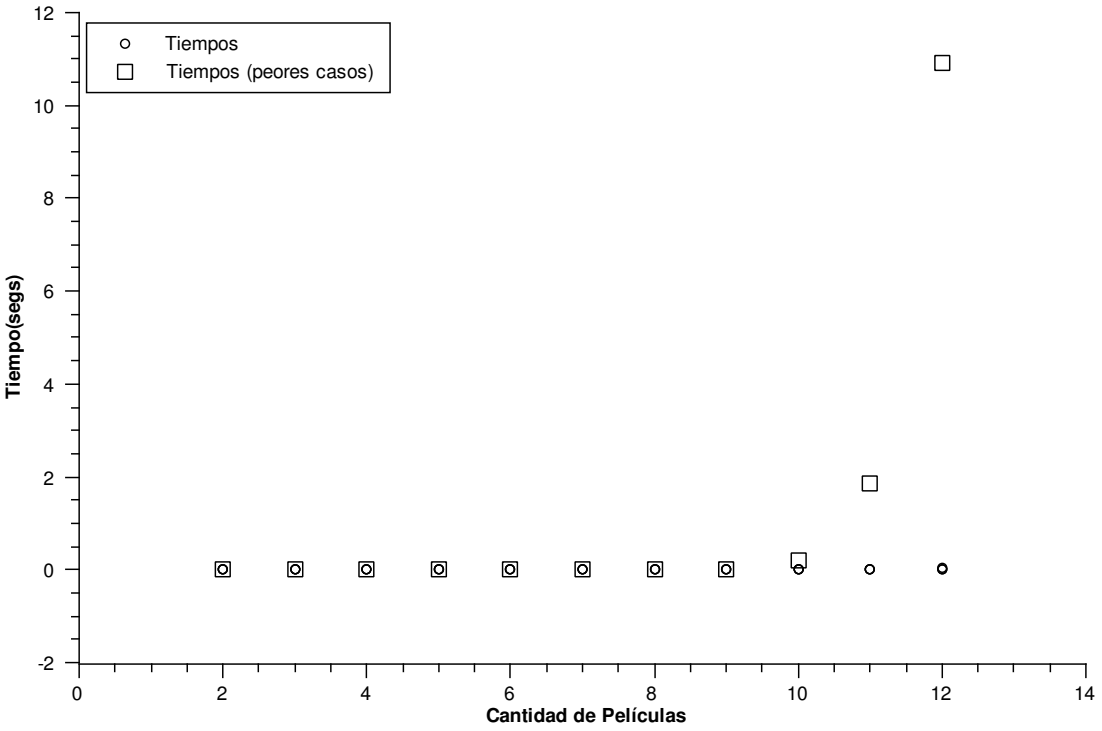

Figura 8: Gráfico de relaciones con un 20, 40 y 60 porciento de probabilidades y peores casos

Debido a que comparar casos con pocas relaciones frente a otros con muchas más (como ocurriría con las instancias generadas con un 20 % de probabilidades de relación contrastadas con los peores casos) no permite análisis precisos pues los casos con muchas relaciones se alejan muy velozmente de los otros. Para el gráfico a continuación utilizamos las instancias generadas con un 60 % y los peores casos para ilustrar comportamientos de forma más precisa. Utilizamos un ajuste exponencial respecto a los dos conjuntos de mediciones.

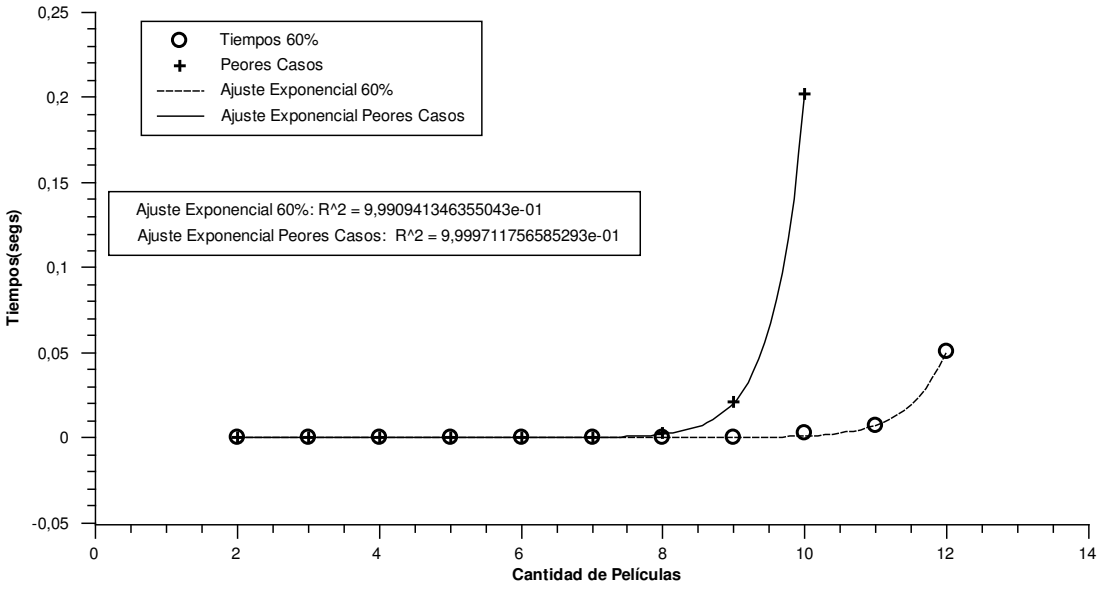

Figura 9: Gráfico de relaciones con un 60 porciento de probabilidades y peores casos

Nuevamente nos encontramos frente al inconveniente de no poder deducir del análisis gráfico si nuestro algoritmo refleja en las mediciones una complejidad exponencial o mayor. Para corroborarlo recurrimos nuevamente a una modificación de los valores del eje de las ordenadas (con log2(tiempo)) y contrastamos con ajustes líneales que nos indicaron que tal como suponíamos el crecimiento era mayor que exponencial.

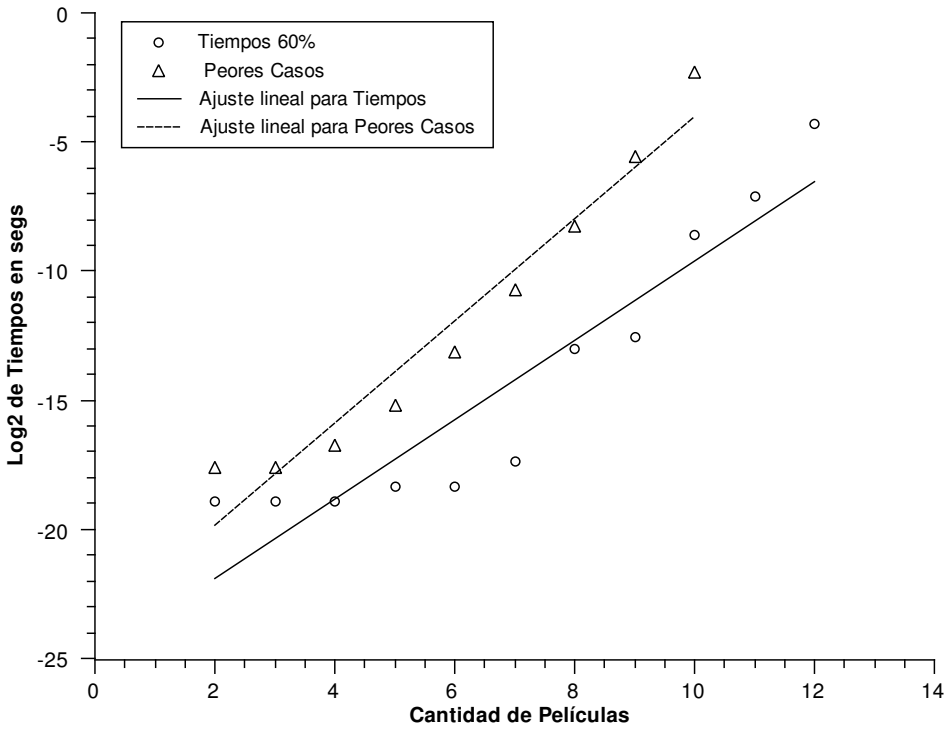

Figura 10: Gráfico de relaciones con un 60 porciento de probabilidades y peores casos graficando en el eje de las ordenadas log2(Tiempo) y comparando con ajuste lineal para ambos casos

3.7. Conclusión

Los cálculos esperados y análisis realizados se vieron felizmente correspondidos con los resultados de los análisis empíricos. Este ejercicio nos permitió el estudio de un problema de mayor costo computacional que los que estábamos acostumbrados a tratar. Vimos claramente ilustrado en las mediciones como frente a una pequeña variación en la entrada se disparaban los tiempos de medición.

4. Ejercicio 3

4.1. Introducción

El problema que tenemos que resolver en este ejercicio es elevar una matriz cuadrada de dimensión 2 a la n y devolver el elemento que está en la fila 1 y la columna 2. Un primer enfoque es multiplicar la matriz n veces, con complejidad O(n) (modelo uniforme). Sin embargo, se puede plantear un algoritmo Divide and Conquer que haga $M^{n/2} \times M^{n/2}$, cuya complejidad es $O(\log_2(n))$.

Nuestro algoritmo trata de dividir el problema de la misma manera que el algoritmo D&C, sin ser recursivo. Además, la matriz que se eleva siempre es la misma, y tiene la propiedad de que, para n $\geq$ 2, los elementos de esta matriz son fib(n - 2), fib(n - 1), fib(n - 1) y fib(n), por lo tanto, podemos obtener ese elemento usando sólo la primera columna.

Además, nos aprovechamos del hecho de que un término de la sucesión de Fibonacci es la suma de los dos anteriores para calcular la *(n-1)-ésima* potencia de la matriz dada y obtener el coeficiente buscado sumando los elementos de la primera fila de ese resultado. La función *costo(n)* es la que resuelve el problema de calcular el presupuesto, y la función *fibo(n)* calcula el *n-ésimo* número de Fibonacci.

4.2. Pseudocódigo

Algoritmo 3 costo(n)

```
1: fibo(n-1)
```

Algoritmo 4 fibo(n)

```
1: acum ← {0, 1}
2: pila ← vacia
3: while n > 1 do
4:    if esPar?(n) then
5:       apilar(pila, false)
6:    else
7:       apilar(pila, true)
8:    end if
9:    n ← n div 2
10: end while
11: while La pila no esta vacia do
12:    b ← tope(pila)
13:    parcial ← acum[1]
14:    acum[1] ← acum[1]^2 + acum[2]^2
15:    acum[2] ← acum[2] × (2 × parcial + acum[2])
16:    if b = true then
17:       parcial ← acum[1]
18:       acum[1] ← acum[2]
19:       acum[2] ← parcial + acum[2]
20:    end if
21:    desapilar(pila)
22: end while
23: return acum[1] + acum[2]
```

4.3. Correctitud

Nota:

Como a la función costo(n) se invoca con $n \geq 1$, la función fibo(x) se ejecutará con $x \geq 0$. Si el parámetro de fibo() es 0 o 1, no se entrará a ninguno de los 2 ciclos, y es fácil ver que el resultado es correcto. Todo el análisis posterior se hará suponiendo que $n > 1$.

Llamamos:

- estado *inter-ciclo* al estado después de salir del primer ciclo de la línea 3 (de ahora en más *primer ciclo*) y antes de entrar al segundo de la línea 11 (de ahora en más *segundo ciclo*).

- L a la longitud de la pila en el estado *inter-ciclo*.
- $p[i]$ (con $1 \leq i \leq L$) al elemento *i-ésimo* contando desde la base de la *pila* en el estado *inter-ciclo*.
- N al valor de la variable n antes de entrar al primer ciclo.
- d_i (con $1 \leq i \leq \lfloor log(N) \rfloor$) al i-ésimo dígito de la representación binaria de N.

Sea $w : bool \to \mathbb{N}$ definida como:

$$w(b) = \begin{cases} 1 & b = true \\ 0 & b = false \end{cases}$$

Sea $a(i)$ una función definida:

$$a(i) = \begin{cases} 2 + \ w(\ p[\ L - i + 1\ ]\ ) & i = 1 \\ 2 \times a(i-1) + \ w(\ p[\ L - i + 1\ ]\ ) & i \geq 1 \end{cases}$$

Vamos a demostrar que:

1. La pila contiene el desarrollo en binario de N, exceptuando el dígito más significativo. Por ejemplo si $N = 11$ (cuyo desarrollo en binario es 1011), la pila sería *tope* [0 , 1 , 1] *base*
2. Después de la i-ésima iteración del segundo ciclo, la variable **acum** es la primer fila de la matriz $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{a(i)}$.
3. $a(L)$ es el desarrollo en base decimal de N.

4.3.1. Primer ciclo

Recordemos que:

$$\sum_{j=0}^{\lfloor log(N) \rfloor} d_j 2^j = N \qquad y \qquad d_{\lfloor log(N) \rfloor} = 1$$

Proposición:

Para i tal que $1 \leq i \leq \lfloor log(N) \rfloor$ vale que al terminar el primer ciclo $p[i] = d_{i-1}$.

Además, al final de la i-ésima iteración

$$n = \sum_{j=i}^{\lfloor log(N) \rfloor} d_j 2^{j-i} = \sum_{j=0}^{\lfloor log(N) \rfloor - i} d_{j+i} 2^j$$

Probémoslo por inducción.

Caso base

$i = 1$

Al entrar al primer ciclo vale que

$$n = N = \sum_{j=0}^{\lfloor log(N) \rfloor} d_j 2^j = \sum_{j=1}^{\lfloor log(N) \rfloor} d_j 2^j + d_0 = 2 \times \sum_{j=0}^{\lfloor log(N) \rfloor - 1} d_{j+1} 2^j + d_0$$

Pero entonces, como el resto y el cociente en la división entera son únicos y $d_0 < 2$ tenemos que: $n \ mod \ 2 = d_0$ y

$$n \ div \ 2 = \sum_{j=0}^{\lfloor log(N) \rfloor - 1} d_{j+1} 2^j$$

Luego, la base de la pila será d_0 y al final de la iteración del ciclo

$$n = \sum_{j=0}^{\lfloor log(N) \rfloor - 1} d_{j+1} 2^j$$

Paso inductivo

$1 \leq i < \lfloor log(N) \rfloor$

Por hipótesis inductiva, al final de la iteración i (y en consecuencia al inicio de la *(i + 1)-ésima*)

$$n = \sum_{j=0}^{\lfloor log(N) \rfloor - i} d_{j+i} 2^j$$

Pero entonces

$$n = \sum_{j=1}^{\lfloor log(N) \rfloor - i} d_{j+i} 2^j + d_i = 2 \times \sum_{j=0}^{\lfloor log(N) \rfloor - (i+1)} d_{j+(i+1)} 2^j + d_i$$

Deducimos por unicidad de cociente y resto en la división entera que $n \ mod \ 2 = d_i$ y

$$n \ div \ 2 = \sum_{j=0}^{\lfloor log(N) \rfloor - (i+1)} d_{j+(i+1)} 2^j$$

Luego $p[i+1] = d_i$ y al final de la *(i + 1)-ésima* iteración n tomará el valor de

$$\sum_{j=0}^{\lfloor log(N) \rfloor - (i+1)} d_{j+(i+1)} 2^j$$

.

Observemos que el ciclo itera $\lfloor log(N) \rfloor$ veces. En efecto, por la proposición anterior, al final de la iteración $\lfloor log(N) \rfloor$ vale que

$$n = \sum_{j=0}^{\lfloor log(N) \rfloor - (\lfloor log(N) \rfloor)} d_{j+\lfloor log(N) \rfloor} 2^j = \sum_{j=0}^{0} d_{j+\lfloor log(N) \rfloor} 2^j = d_{\lfloor log(N) \rfloor} \leq 1$$

Como la guarda es $n > 1$, el ciclo termina.

Esta proposición implica que al terminar el cilo se encuentren en la pila todos los dígitos de la representación binaria de N excepto por el más significativo, que es 1. La longitud de la pila luego del primer ciclo es entonces $\lfloor log(N) \rfloor$.

4.3.2. Segundo Ciclo

Proposición: para toda iteración i ≥ 1, se cumple que al final del ciclo, la variable ***acum*** representa la primera fila de la matriz $M^{a(i)}$, siendo $M = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$

Demostración

Observaciones preliminares:

- Se puede demostrar que al comienzo de la i-ésima iteración del ciclo, $tope(pila) = p[L - i + 1]$.
- Cuando escribimos $*@*$, nos referimos al valor de la variable:

  $variable@pre$ - antes de la iteración del ciclo que estamos analizando.

  $variable@final$ - después de terminar la iteración del ciclo, y antes de iniciar la próxima (o salir del mismo).

  $variable@\#linea$ - luego de haberse ejecutado la línea número $\#linea$.

Caso base

Sea $i = 1$. Es decir, entramos al segundo ciclo por primera vez. Tenemos dos posibilidades: el tope de la pila es *false*, o bien es *true*.

Caso I: el tope de la pila es *false*

Qvq: Después de terminar la iteración, la variable ***acum*** representa la primera fila de la matriz

$$M^{a(1)} = M^{2+w(\ false\ )} = M^{2+0} = M^2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Hagamos un pequeño seguimiento del programa:

- $acum[1]@pre = 0$
- $acum[2]@pre = 1$
- $b@12 = tope(pila@preCiclo)$
- $parcial@13 = acum@12$
- $acum[1]@14 = (acum[1]@13)^2 + (acum[2]@13)^2$
- $acum[2]@15 = (acum[2]@14 \times (2 \times parcial@14 + acum[2]@14))$

Entonces en el estado correspondiente a la línea 16, vale que:

- $acum[1]@16 = (acum[1]@pre)^2 + (acum[2]@pre)^2$
- $acum[2]@16 = acum[2]@pre \times (2 \times acum[1]@pre + acum[2]@pre)$

Obs: no se entra al *SI* pues $b@16 = tope(pila@pre) = false$

Luego, en el estado posterior a la iteración vale que:

- $acum[1]@16 = acum[1]@final = (acum[1]@pre)^2 + (acum[2]@pre)^2 = 0^2 + 1^2 = 1 = m^2_{(1,1)}$
- $acum[2]@16 = acum[2]@final = acum[2]@pre \times (2 \times acum[1]@pre + acum[2]@pre) = 1 \times (2 \times 0 + 1) = 1 = m^2_{(1,2)}$

Entonces vale que la variable *acum* representa la primer fila de la matriz

$$M^2 = M^{2+0} = M^{2+w(false)} = M^{2+w(\ p[L-(i+1)+1]\ )} = M^{a(1)}$$

Caso II: el tope de la pila es *true*

Qvq: Después de terminar la iteración, la variable ***acum*** representa la primera fila de la matriz

$$M^{a(1)} = M^{2+w(\ true\ )} = M^{2+1} = M^3 = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

El seguimiento hasta la línea 16 va a ser el mismo, con la única diferencia que se entrará al bloque SI, pues $b@16 = tope(pila@pre) = true$. Entonces el seguimiento continuaría de esta manera:

- $parcial@17 = acum[1]@16$
- $acum[1]@18 = acum[2]@17$
- $acum[2]@19 = parcial@18 + acum[2]@18$

Luego, en el estado posterior a la iteración vale que:

- $acum[1]@19 = acum[1]@final = acum[2]@16 = acum[2]@pre \times (2 \times acum[1]@pre + acum[2]@pre) = 1 \times (2 \times 0 + 1) = 1 = m^3_{(1,1)}$
- $acum[2]@19 = acum[2]@final = acum[1]@16 + acum[2]@16 = (acum[2]@pre \times (2 \times acum[1]@pre + acum[2]@pre)) + ((acum[1]@pre)^2 + (acum[2]@pre)^2) = (1 \times (2 \times 0 + 1)) + (0^2 + 1^2) = 1 + 1 = 2 = m^3_{(1,2)}$

Entonces vale que la variable *acum* representa la primer fila de la matriz

$$M^3 = M^{2+1} = M^{2+w(true)} = M^{2+w(\ p[L-(i+1)+1]\ )} = M^{a(1)}$$

PI

Supongo que vale para algun $i > 1$ tal que $i + 1 < log(n)$

Qvq: $P(i) \Rightarrow P(i+1)$

Como la iteración anterior fue la número i, por hipótesis inductiva se cumple que la variable **acum** antes de entrar a la iteración *i+1-ésima*, es la primera fila de $M^{a(i)}$

Por la demostración 5.1.1, sabemos que la matriz $M^{a(i)}$ es de la forma:

$$\begin{pmatrix} a & b \\ b & (a+b) \end{pmatrix}$$

Y por la nota 5.1.2, también sabemos entonces que $(M^{a(i)})^2 = M^{2 \times a(i)}$ es de la forma:

$$\begin{pmatrix} a^2+b^2 & b \times (2 \times a + b) \\ b \times (2 \times a + b) & b^2 + (a+b)^2 \end{pmatrix}$$

Por el seguimiento mostrado en el Caso Base, sabemos que en el estado correspondiente a la línea 16, vale que:

- $acum[1]@16 = (acum[1]@pre)^2 + (acum[2]@pre)^2$
- $acum[2]@16 = acum[2]@pre \times (2 \times acum[1]@pre + acum[2]@pre)$

Y por hipótesis inductiva $acum[1]@pre = a$ y $acum[2]@pre = b$. Entonces podemos ver que:

- $acum[1]@16 = (acum[1]@pre)^2 + (acum[2]@pre)^2 = a^2 + b^2 = M^{2 \times a(i)}_{(1,1)}$
- $acum[2]@16 = acum[2]@pre \times (2 \times acum[1]@pre + acum[2]@pre) = b \times (2 \times a + b) = M^{2 \times a(i)}_{(1,2)}$

Ahora tenemos 2 casos posibles: el tope de la pila es *true* o bien es *false*

Caso I

Si el tope de la pila es *false*, (por el mismo razonamiento que utilizamos en el Caso Base), no se ejecuta el bloque SI de la línea 16, por lo que no hay más cambios de estado. Entonces vale que:

$$acum[1]@final = a^2 + b^2 = M^{2 \times a(i)}_{(1,1)}$$
$$acum[2]@final = b \times (2 \times a + b) = M^{2 \times a(i)}_{(1,2)}$$

Por lo que podemos decir que la variable *acum* representa la primer fila de la matriz:

$$M^{2 \times a(i)} = M^{2 \times a(i)+0} = M^{2 \times a(i)+w(false)} = M^{2 \times a(i)+w(\ p[\ L-(i+1)+1\ ]\ )}$$

Que es lo que queríamos demostrar.

Caso II

Veamos primero que:

$$\begin{pmatrix} x & y \\ y & x+y \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} y & x+y \\ x+y & 2y+x \end{pmatrix}$$

Ahora, si el tope de la pila es *true*, (por el mismo razonamiento que utilizamos en el Caso Base), se ejecuta el bloque SI de la línea 16. Entonces el seguimiento sería el mismo hasta la línea 16, y continuaría de la misma manera que en el Caso II del Caso Base. Por lo que tendríamos que en el estado posterior a la iteración vale que:

- $acum[1]@19 = acum[1]@final = acum[2]@16 = M_{(1,2)}^{2\times a(i)}$ *(por lo demostrado en el Caso I)*
- $acum[2]@19 = acum[2]@final = acum[1]@16 + acum[2]@16 = M_{(1,1)}^{2\times a(i)} + M_{(1,2)}^{2\times a(i)}$

Entonces, utilizando el razonamiento de la observación del comienzo:

$$M_{(1,2)}^{2\times a(i)} = M_{(1,1)}^{2\times a(i)+1}$$
$$M_{(1,1)}^{2\times a(i)} + M_{(1,2)}^{2\times a(i)} = M_{(1,2)}^{2\times a(i)+1}$$

Vale entonces que la variable *acum* representa la primer fila de la matriz

$$M^3 = M^{2+1} = M^{2+w(true)} = M^{2+w(\ p[L-(i+1)+1]\ )} = M^{a(1)}$$

Que es lo que queríamos demostrar.

4.3.3. Desarrollo decimal de N

Ahora veamos que si $1 \leq i \leq L$ entonces $a(i) = \sum_{j=0}^{i} d_{L-j}2^{i-j}$ por inducción

Caso base: $i = 1$

$$\sum_{j=0}^{1} d_{L-j}2^{1-j} = d_L 2 + d_{L-1} = 2 + w(p[L]) = a(1)$$

Paso inductivo: $1 \leq i < L$

$$\sum_{j=0}^{i+1} d_{L-j}2^{i+1-j} = \sum_{j=0}^{i} d_{L-j}2^{i+1-j} + d_{L-(i+1)}2^{i+1-(i+1)} =$$

$$= 2 \times \sum_{j=0}^{i} d_{L-j}2^{i-j} + d_{L-i-1)} = 2 \times a(i) + w(p[L-i]) = a(i+1)$$

De esto se deduce que:

$$a(L) = \sum_{j=0}^{L} d_{L-j}2^{L-j} = \sum_{j=0}^{L} d_j 2^j = N$$

Por lo tanto al final del segundo ciclo, la variable *acum* representa la primera fila de la matriz M^N.

4.4. Complejidad

4.4.1. Modelo Uniforme

- En el primer ciclo, todas las operaciones son constantes y la entrada se divide por 2 en cada iteración. Por lo tanto, se ejecutará a lo sumo $O(log(n))$.
- En el segundo ciclo, todas las operaciones también son constantes y se ejecuta tantas veces como elementos hay en la pila en el estado interciclo. Entonces el ciclo se ejecuta como máximo $O(log(n))$.

Por lo tanto, la complejidad temporal del algoritmo, según el modelo uniforme es $O(log(n))$.

4.4.2. Modelo Logarítimico

Observaciones:

- Consideramos que la división y tomar resto entre dos numeros a y b cuesta $log(a) * log(b)$
- Consideramos que **true** y **false** tienen tamaño constante.
- Consideramos que el manejo de punteros es $O(1)$, por lo que verificar si una pila está vacia, o desapilar una pila de bool, cuesta $O(1)$.
- Apenas se ingresa al primer ciclo, vale que $acum[1] \leq acum[2] \leq F(2^{(i+1)} - 1)$ donde i es el número de iteración.

- Vale que $log(F(n)) \in O(n)(\forall n \in \mathbb{N})$ *(ver 5.1.3)*
- Cuando escribimos $log(a)$ nos referimos a $log_2(a)$.

Análisis de complejidad:

3 La comparación es $O(log(n))$.

3 El ciclo se ejecuta $O(log(n))$ veces.

[4] esPar?(x) es equivalente a tomar el resto de dividir x por 2. Luego, es $O(log(x))$.

[5] Como apilo siempre booleanos, la operación es $O(1)$.

[9] La división de x por 2 es $O(log(x))$.

Luego, el ciclo es $O(log^2(n))$.

11 Verificar si la pila está vacía es $O(1)$.

11 El ciclo se ejecuta $O(log(n))$ veces, pues la pila fué armada por el ciclo anterior (que itera $O(log(n))$ veces agregando un elemento a la vez). Sea i el número de iteración del ciclo, entonces:

[12] Asignar el tope de la pila a una variable es $O(1)$, pues los elementos de la pila son de tipo bool.

[13] La asignación es $O(2^i)$, pues $acum[1] \leq F(2^{i+1})$.

[14] El costo de la potencia es $log^2(acum[1]) + log^2(acum[2]) \leq 2 * log^2(F(2^{i+1})) \leq 2 * 4^{i+1} \in O(4^i)$.

[14] El costo de la suma es $log(acum[1]^2) + log(acum[2]^2) \leq 4 * log(F(2^{i+1})) \leq 4 * 2^{i+1} \in O(2^i)$.

[15] El costo del primer producto es $log(2) * log(parcial) = log(2) * log(acum[1]@13) \leq log(2) * log(F(2^{i+1})) \leq log(2) * 2^{i+1} \in O(2^i)$.

[15] El costo de la suma es $log(2 * parcial) + log(acum[2]) \leq log(2) * log(acum[1]@13) + log(acum[2]) \leq log(2) * log(F(2^{i+1})) + log(F(2^{i+1})) \leq (log(2) + 1) * 2^{i+1} \in O(2^i)$.

[15] El costo del último producto es $log(acum[2]) * log(2 * parcial + acum[2]) \leq log(acum[2]) * log(2 * parcial + 2 * acum[2]) \leq log(acum[2]) * log(2 * (parcial + acum[2])) = log(acum[2]) * log(2 * (acum[1]@13 + acum[2])) \leq log(acum[2]) * log(2 * (2 * acum[2])) \leq log(acum[2]) * (log(4) + log(acum[2])) \leq log(acum[2]) * log(4) + log^2(acum[2]) \leq log(F(2^{i+1})) * log(4) + log^2(F(2^{i+1})) \leq 2^{i+1} * log(4) + 4^{i+1} \in O(4^i)$

[17] Utilizando los razonamientos de las cotas anteriores podemos decir que la complejidad de esta operación es $O(2^i)$.

[18] Por lo visto en la línea 15, esta complejidad es $O(2^i)$.

[19] La complejidad de esta operación es $O(2^i)$.

[21] Desapilar es $O(1)$

Luego la complejidad del ciclo es

$$O(\sum_{j=1}^{\lfloor log(n) \rfloor} 4^j) = O(\frac{4^{\lfloor log(n) \rfloor + 1} - 1}{4 - 1}) = O((2^{\lfloor log(n) \rfloor + 1})^2) = O(n^2)$$

23 Se suman $F(n-2)$ y $F(n-1)$, luego la complejidad de esta suma es $O(2n-3) = O(n)$.

Entonces la complejidad temporal del algoritmo en función de n es $O(n^2)$.

4.4.3. Complejidad en función del tamaño de la entrada

El tamaño de la entrada, $E(n)$ es $log(n)$, luego: $2^{E(n)} = 2^{log(n)} = n$.

Entonces la complejidad en función del tamaño de la entrada es $O((2^{E(n)})^2) = O(4^{E(n)})$.

4.5. Detalles de implementación

Usamos la librería **GMPlib** para trabajar con números grandes, que nos garantiza en la multiplicación una complejidad temporal $O(n^2)$ donde n es la cantidad de dígitos del número.

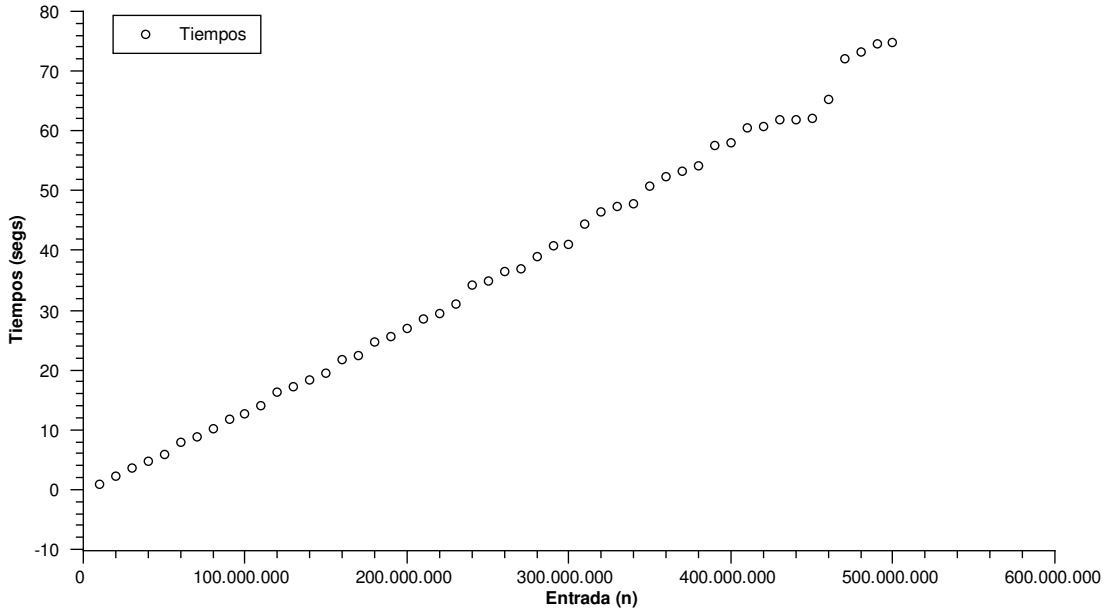

Figura 11: Gráfico de Tiempo en función de la entrada para valores de n entre 10000000 y 500000000

4.6. Pruebas, resultados y mediciones

El análisis de este algoritmo nos requirió trabajar con números realmente grandes. Para entradas pequeñas las mediciones de tiempo devolvían tiempos demasiado cortos como para poder hacer un análisis significativo. Los sets más representativos que utilizamos estuvieron generados por entradas con n entre 10000000 y 500000000 incrementándose de 10000000 en 10000000. Otro caso particular que encontramos fue el de las potencias de 2, dada la implementación del algoritmo, este tipo de números genera casos particularmente costosos.

El siguiente gráfico está generado con potencias de 2 entre 2048 y 1073741824, un ajuste del mismo refleja una cota cuadrática con mucha precisión (dado que consideramos las potencias de 2 entre los peores casos para esta implementación), aunque considerando la cantidad y distribución de los valores de la entrada no se puede hacer una afirmación del todo precisa al respecto aunque claramente sí es una cota superior.

Realizando mediciones y ajustes para el caso de las entradas entre 10000000 y 500000000 pudimos observar que frente a ajustes lineales las mediciones tendían a distribuirse por encima de la gráfica de la recta en los extremos de la misma y por debajo en las instancias intermedias. Este comportamiento nos da la pauta de un crecimiento mayor que lineal. A su vez, otro gráfico con ajuste preciso respecto a un polinomio cuadrático nos muestra una distribución con un crecimiento más lento que dicha función (puede observarse como al principio las mediciones son parejas pero luego las instancias van curvándose sobre el gráfico de la función cuadrática).

Para obtener datos más precisos, evitando errores que dependan de factores externos y aleatorios, repetimos las mediciones corriendo 300 veces un mismo caso (n = 16777216) y pudimos observar las siguientes frecuencias de apariciones para cada rango de tiempo.

4.7. Pruebas, resultados y mediciones

En este ejercicio, a diferencia de los anteriores, pudimos observar MUY claramente la importancia de una correcta elección al momento de utilizar un modelo de complejidad. Este algoritmo en un primer análisis teórico bajo el modelo uniforme parecía insinuar costos temporales que difirieron mucho con los resultados reflejados en los gráficos para entradas considerablemente grandes. En cambio, el estudio de las mediciones para esas mismas plasmado en los gráficos fue mucho más representativo de lo indicado en un análisis bajo el modelo logarítmico.

Para contrastar con una primera idea de complejidad sin considerar memoria acotada, incluímos este gráfico que deja clara evidencia entre el modelo teórico y el análisis gráfico, así como de la importancia de una buena elección de modelo en función de las características del problema.

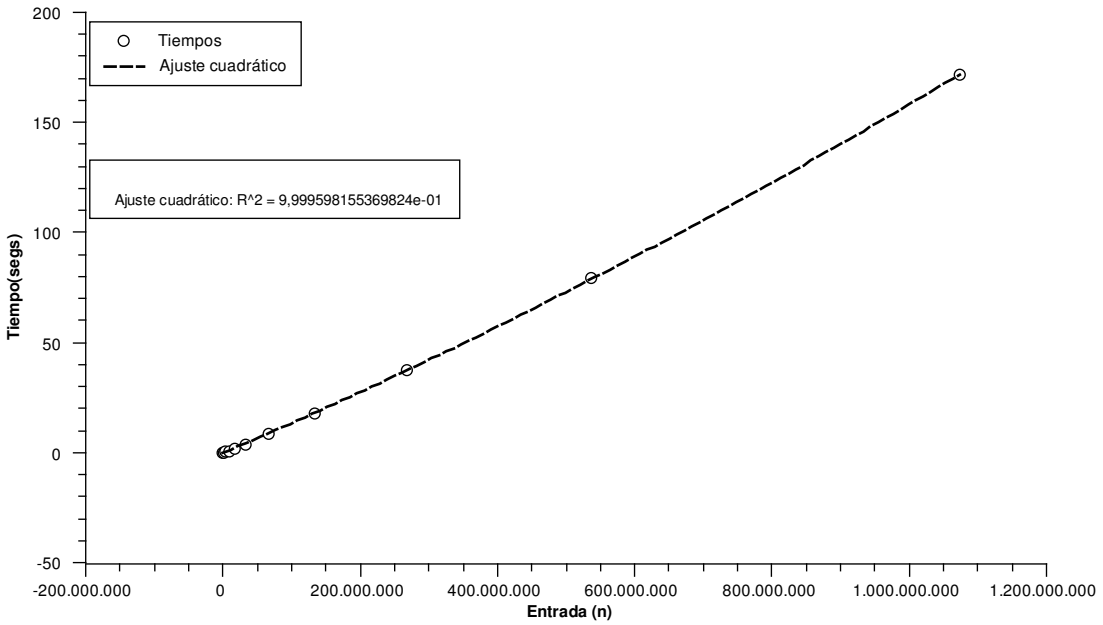

Figura 12: Gráfico para casos costosos con n potencia de dos y ajuste cuadrático

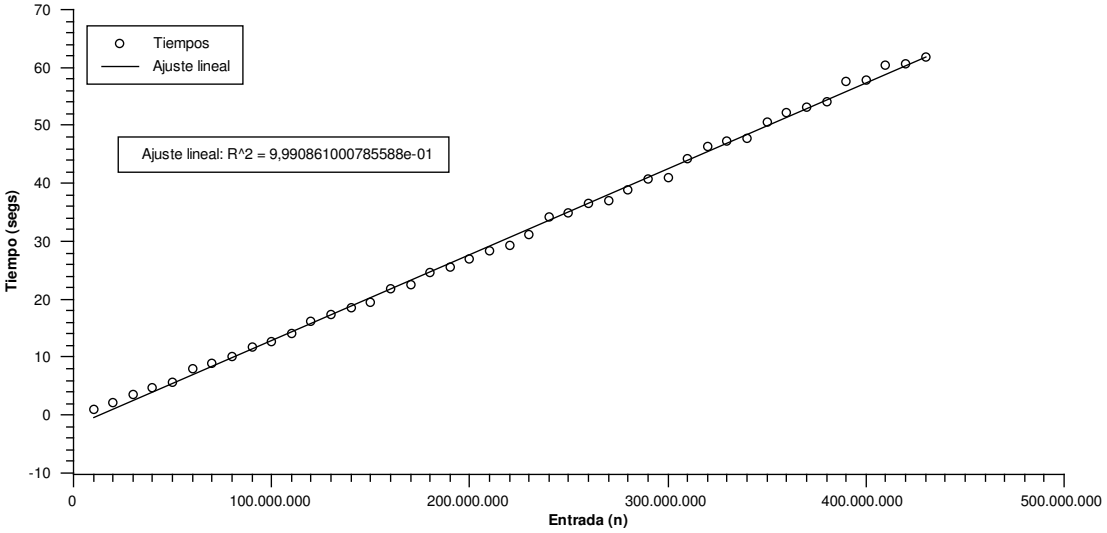

Figura 13: Gráfico de Tiempo en función de la entrada con ajuste lineal

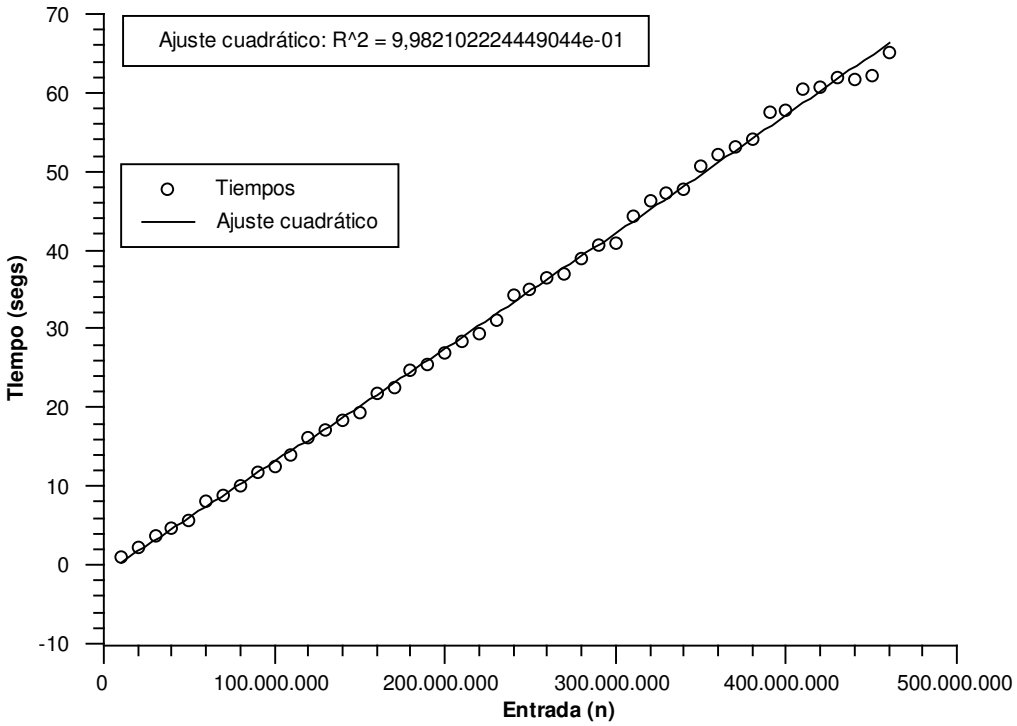

Figura 14: Gráfico de Tiempo en función de la entrada con ajuste cuadrático

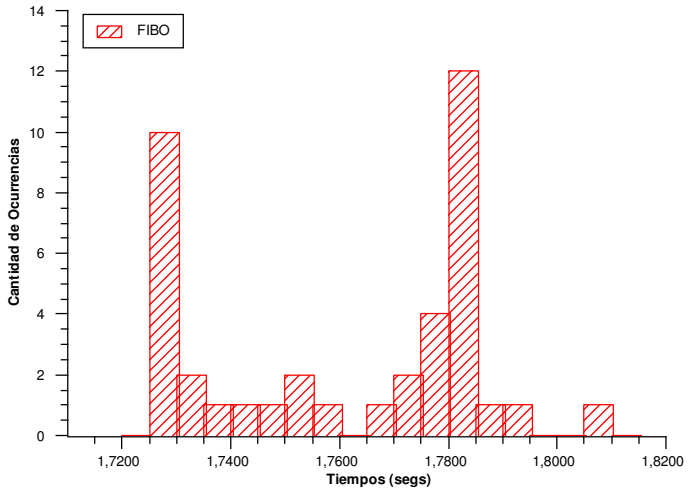

Figura 15: Histograma que ilustra la distribución de los tiempos para una entrada n = 16777216 corrida 300 veces

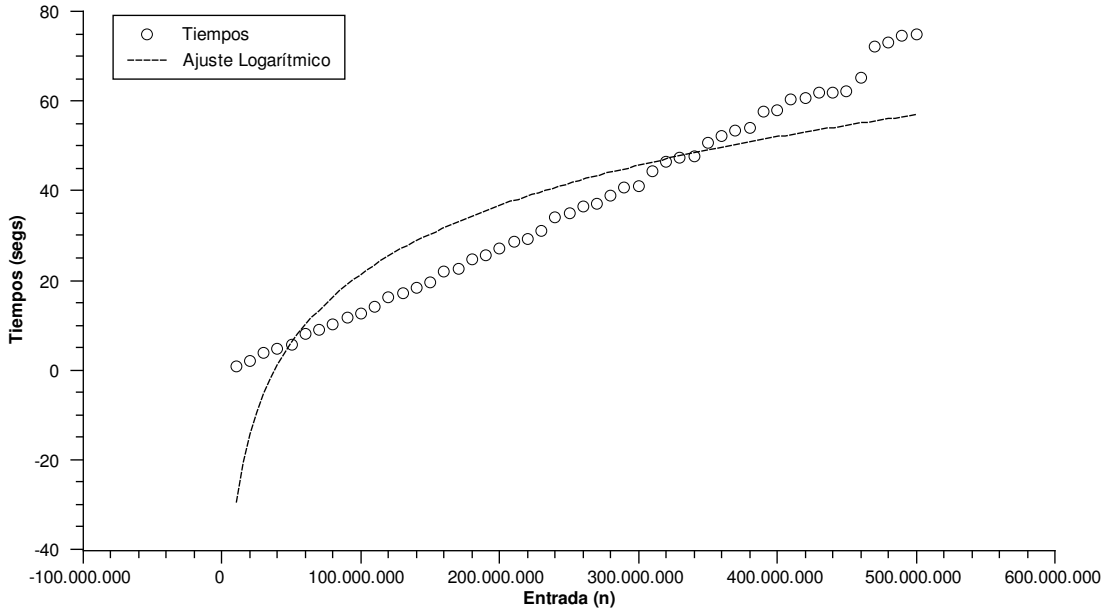

Figura 16: Gráfico de Tiempo en función de la entrada con ajuste logarítmico permite observar las discrepancias entre un primer análisis teórico y uno empírico.

5. Anexo

5.1. Demostraciones

5.1.1. Los elementos de la matriz son números de Fibonacci

Dado M = $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$, n $\in$ N $\Rightarrow (\forall n \geq 2)\ M^n = \begin{pmatrix} fib(n-2) & fib(n-1) \\ fib(n-1) & fib(n) \end{pmatrix}$

Demostración por inducción:

Sea n = 2, $M^2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} fib(0) & fib(1) \\ fib(1) & fib(2) \end{pmatrix}$

Sea n tal que P(n), quiero ver que P(n) $\Longrightarrow$ P(n+1)

$$M^{(n+1} = M^n \times M = \begin{pmatrix} fib(n-2) & fib(n-1) \\ fib(n-1) & fib(n) \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} fib(n-1) & fib(n-1)+fib(n-2) \\ fib(n) & fib(n-1)+fib(n) \end{pmatrix} = \begin{pmatrix} fib(n-1) & fib(n) \\ fib(n) & fib(n+1) \end{pmatrix}$$

5.1.2. Pinta de las matrices

Por lo visto en 5.1.1, se cumple que si $k \geq 2$ tenemos que: $H^k = \begin{pmatrix} fib(n-2) & fib(n-1) \\ fib(n-1) & fib(n) \end{pmatrix}$

Entonces en todas las iteraciones vale que H es de la forma: $\begin{pmatrix} a & b \\ b & (a+b) \end{pmatrix}$

Veamos que:

$$H^2 = \begin{pmatrix} a & b \\ b & (a+b) \end{pmatrix}^2 = \begin{pmatrix} a^2+b^2 & a \times b + b \times (a+b) \\ b \times a + (a+b) \times b & b^2 + (a+b)^2 \end{pmatrix} = \begin{pmatrix} a^2+b^2 & b \times (a+a+b) \\ b \times (a+a+b) & b^2 + (a+b)^2 \end{pmatrix}$$

5.1.3. Cota superior de Fibonacci(n)

Sabemos que $F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$

Entonces podemos decir que $F_n \leq \frac{\varphi^n}{\sqrt{5}} \leq \varphi^n$

Luego: $log_b(F_n) \leq log_b(\varphi^n) \Leftrightarrow log_b(F_n) \leq n * log_b(\varphi) \Rightarrow log_b(F_n) \in O(n)$

6. Bibliografía

- **Fundamentals of Algorithmics**. Gilles Brassard, Paul Bratley. Prentice Hall, 1996.
- `en.wikipedia.org/wiki/Fibonacci_number`
- `gmplib.org/manual/`
- `www.opengroup.org/onlinepubs/000095399/functions/gettimeofday.html`