# Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

## RTP1

Segundo Cuatrimestre 2014



| Integrante | LU | Correo electrónico |
|---|---|---|
| Ricardo Colombo | 156/08 | ricardogcolombo@gmail.com.com |
| Federico Suarez | 610/11 | elgeniofederico@gmail.com |
| Juan Carlos Giudici | 827/06 | elchudi@gmail.com |
| Franco Negri | 893/13 | franconegri2004@gmail.com |



**Reservado para la cátedra**

| Instancia | Docente | Nota |
|---|---|---|
| Primera entrega | | |
| Segunda entrega | | |

# Contenidos

# Capítulo 1

# Puentes sobre lava caliente

### 1.0.1. Introducción

Cada participante de una competencia debe cruzar un puente dando saltos de tablón en tablón, con la limitación de que pueden saltar como máximo una cantidad fija de tablones por vez. Sin embargo algunos de estos tablones están rotos. El objetivo del ejercicio es dar un algoritmo que nos devuelva un recorrido por los tablones del puente el cual sea mínimo en la cantidad de saltos requeridos para poder cruzarlo con una complejidad temporal de O(n) donde n es la cantidad de tablones del puente.

### 1.0.2. Ejemplos y Soluciones

Sea un puente de 14 tablones representado por el siguiente vector de 0 y 1, `[0,0,0,1,1,0,0,1,0,1,0,0,1,0]` llamado `vector_puente`, donde 1 representa que el tablón esta roto y 0 sano, y la primer posición del mismo el inicio del puente y la última el fin del mismo. Adicionalmente sabemos que la cantidad máxima de tablones que el participante puede saltar son 3 tablones.

Nuestro algoritmo va a encontrar la solución de la siguiente forma: En cada paso el algoritmo goloso busca quedarse con la solucion optima, esta es el salto maximo que el participante puede saltar, con lo cual en cada paso de nuestro algoritmo lo que realiza es intetar este salto si es factible, en caso que el tablon esta sano saltamos a ese tablon y continuamos de manera iterativa, caso contrario comenzamos a revisar desde el salto maximo acercandonos tablon por tablon hasta la posicion actual hasta encontrar un tablon sano. Esto podria no ocurrir y en ese caso quiere decir que hay una cantidad de tablones rotos mayor al salto maximod el participante y en esta situacion no tenemos solucion.

### 1.0.3. Desarrollo

Para la solución de este problema recurrimos a la tecnica de algoritmos voraces, donde en cada paso en la construccion a la solucion obtenemos el mejor salto. Por el enunciado sabemos que tenemos n tablones, el participante puede saltar C tablones de una sola vez y cuales son las posiciones de los tablones rotos.Nos armamos un arreglo donde cada posición representa un tablón, en el cual guardamos 0 y 1 para indicar su estado (sano o roto respectivamente) llenando el mismo según la entrada, llamemoslo puente. Nuestro algoritmo ira recorriendo el puente tratando de hacer el mayor salto cuando este es posible, dependiendo si el tablon al que iria a parar el participante esta sano, en caso contrario comenzamos a iterar los tablones desde este salto maximo hacia donde se encuentra parado el participante uno por uno hasta encontrar un tablon sano.
Para cada al cual se salta se guarda su posicion en una lista siempre insertando atras de la misma, una vez que cruzo participante cruzo todo el puente se devuelve esta lista como la sucesion de saltos realizada por el participante, en caso de no ser factible ya que en una seccion del puente la cantidad de tablones rotos consecutivos es mayor a el salto maximo se devuelve no, ya que no tiene solucion.

### 1.0.4. Complejidad

El siguiente es un pseudo-código de nuestro algoritmo.

**Algorithm 1**
Saltos( salto_Maximo : natural, puente : arreglo(1's y 0's), distancias: arreglo(naturales) , cantidadTablones : natural)

```
1: Si saltoMax > cantidadTablones     O(1)
2:    devolver 1
3: SI NO
4: SI existe una cantidad de tablones rotos mayor o igual a salto_maximo en algun lugar del puente
5:    no existe solucion
6: nueva listaDeSaltos
7:    mientras posActual < n && posActual >= 0     O(n)
8:       SI puente(posActual) == 0
9:          Agrego posActual a Lista de Saltos
10:         le sumo a posActual el largo del salto
11:      SI NO
12:      Le resto uno a PosActual
13:      Devolver listaDeSaltos
```

Todas las asignaciones y comparaciones son en O(1) como esta marcado en el pseudocodigo, ya que son números naturales y están acotados por la cantidad de tablones
En la linea 4 para revisar si la cantidad de tablones rotos es mayor o igual a el salto maximo en alguna parte del puente se realiza un ciclo que cuenta la cantidad de tablones rotos consecutivos, en caso de que se encuentre con uno sano resetea el contador y comienza de nuevo a contar, en el caso que el contador llegue a superar el salto maximo se devuelve que no tiene solucion. Esto se realiza con un solo ciclo y la complejidad es O(n). El ciclo de las lineas 7 - 13 se realiza n veces en el peor caso con lo cual la complejidad total del algoritmo es O(n), ya que los dos posibles peores casos son, cuando el puente esta sano y salta de a un tablon por vez, o cuando los unicos tablones sanos son en las posiciones 1, Salto Maximo +1 , 2*(salto Maximo +1)..., de esta manera siempre salta el maximo y recorre los tablones hasta el primero para encontrar el sano dando como resultado que recorre todo el puente para encontrar los saltos.

### 1.0.5. Demostración

Para la prueba de correctitud, queremos demostrar que nuestro algoritmo siempre encuentra una sucesión de saltos que es la menor posible.

Luego para demostrar por absurdo, supongamos que no es así. Dicho mas formalmente, existe una sucesión $V = \{v_1, v_2, ..., v_i\}$ de saltos tal que es menor a la sucesión $W = \{w_1, w_2, ..., w_i, w_{i+1}, ..., w_j\}$ que encuentra nuestro algoritmo.

Por condiciones del problema sabemos el tablon al que queremos saltar no puede estar a distancia mayor a $k$ (el salto maximo posible), esto significa que:

$$v_1 \leq k$$

$$v_2 - v_1 \leq k$$

$$...$$

$$v_i - v_{i-1} \leq k$$

Sumando:

$$v_1 \leq k$$

$$v_2 \leq 2.k$$

$$...$$

$$v_i \leq i.k$$

Pero nuestro algoritmo, en cada paso, salta la mayor cantidad de tablones posibles y de allí comienza a retroceder hasta encontrar uno sano, dado que $v_1, v_2, ..., v_i$ deben ser tablones sanos, los tablones elegidos por nuestro algoritmo estarán acotados de esta manera:

$$v_1 \leq w_1 \leq k$$

$$v_2 \leq w_2 \leq 2.k$$

$$...$$

$$v_i \leq w_i \leq i.k$$

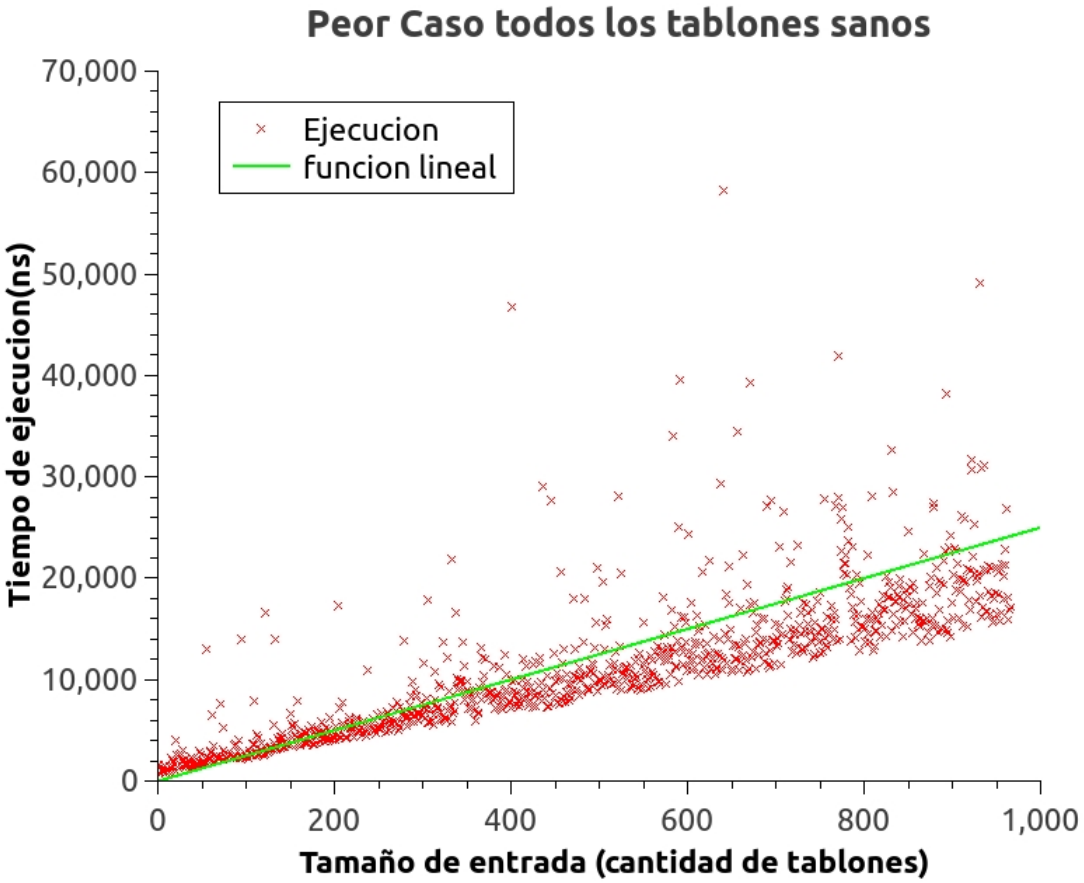
Pero entonces, esto quiere decir que nuestro algoritmo encuenta una sucesión de saltos, para los cuales siempre se salta por lo menos, la misma cantidad de tablones que en la sucesión $V$. Luego nuestro algoritmo encuentra una solución $W$ que tiene, como mucho, $k$ saltos.

Dado que supusimos que nuestro algoritmo encuntraba una sucesión que era peor a $V$, abusrdo.

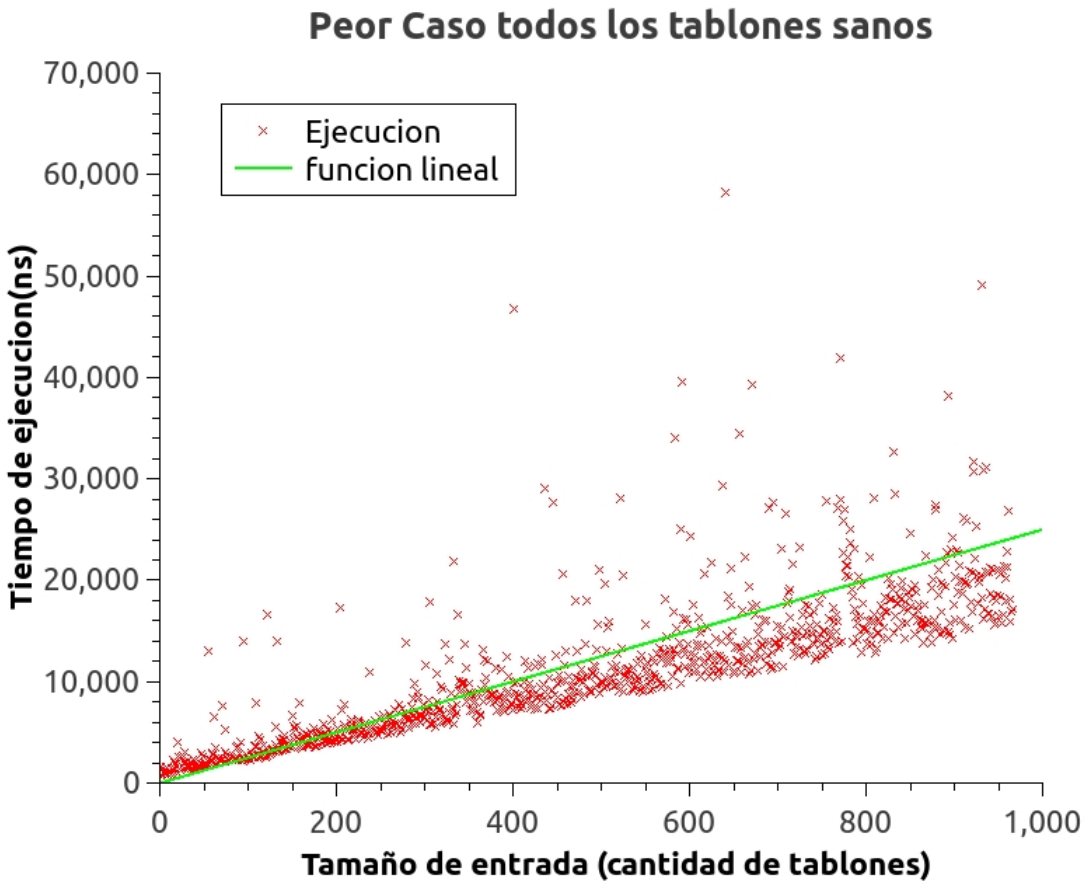
Luego nuestro algoritmo encuentra siempre una sucesión de saltos que es minima.

### 1.0.6. Experimentacion

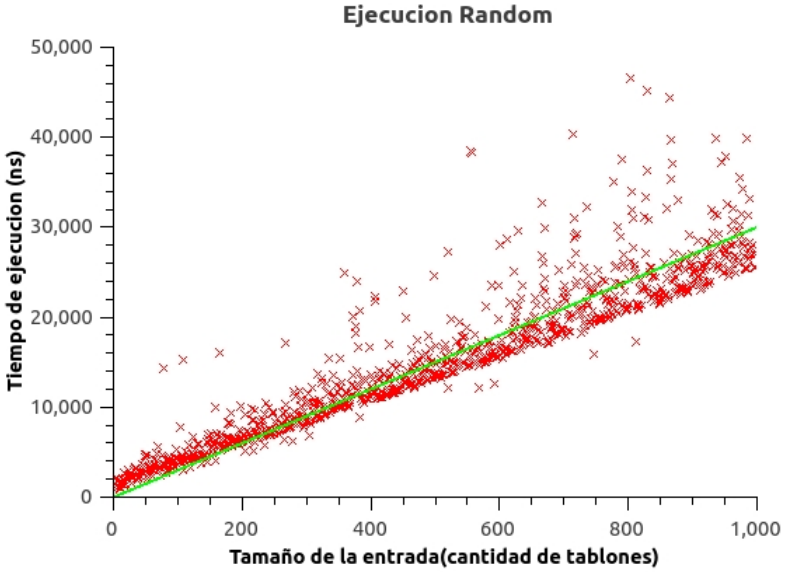
Para la experimentación del problema en cuestión se realizaron los dos test de peores casos como se menciono en la seccion de complejidad, el primero teniendo en cuenta el caso que todos los tablones del puente estén sanos y el salto máximo del participante sea uno, para esto se fijo el salto máximo y se crearon 100 instancias de puentes sanos en tamaños que van del 1 al 100 luego se midieron los tiempos de ejecución dando como resultado el siguiente gráfico




Como puede verse en el gráfico su complejidad esta en el orden lineal como puede compararse con la función, en el gráfico puede observar que algunos casos excede el orden, este ruido se debe a que puede haber ovaciones donde el sistema operativo este realizando otras tareas y afecte el orden de ejecución. En el segundo test, se realizo el segundo caso mencionado en la seccion complejidad y dando como resultado un grafico con complejidad del orden lineal como se puede observar en el siguiente grafico.




Para el tercer test se realizo un test aleatorio, para poder observar el comportamiento de todos los casos posibles, para esto se crearon 1000 instancias, en las cuales el tamaño del puente fue creciendo de 1 a 1000 y el salto máximo es un numero random entre 1 y la cantidad de tablones del puente para poder obtener también casos que no sean solución, dando comer resultado el siguiente gráfico.




Puede observarse que al igual en el test anterior el orden de complejidad esta en el orden lineal, de esta manera podemos concluir que nuestro algoritmo cumple con los ordenes de complejidad impuestos por el enunciado.

# Capítulo 2

# Apéndice