DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Algoritmos y Estructuras de Datos III

Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Aldasoro Agustina	86/13	agusaldasoro@gmail.com
Noriega Francisco	660/12	frannoriega.92@gmail.com
Zimenspitz Ezequiel	155/13	ezeqzim@gmail.com
Zuker Brian	441/13	brianzuker@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Resumen

Poner resumen.

Índice

1. Dakkar

1.1. Descripción de la problemática

La problemática trata de una travesía, la cual cuenta con n cantidad de etapas. Para cada una de las etapas, se puede elegir recorrerla en alguno de los tres vehículos disponibles: una BMX, una motocross o un buggy arenero. Cada uno de ellos permite concretar cada etapa en cantidades de tiempo diferentes. Además, la cantidad de veces que se pueden usar la motocross y el buggy arenero está acotada por k_m y k_b respectivamente.

Los *tiempos* que le llevan a los vehículos recorrer el trayecto varían por cada etapa y son datos conocidos pasados por parámetro.

Se pide recorrer la travesía, dentro de las restricciones, de modo que se utilice la menor cantidad de tiempo posible. Si existen dos (o más) maneras de atravesarla dentro del tiempo óptimo, se pide devolver sólo una.

Se exige resolver la problemática con una complejidad temporal de $O(n.k_m.k_b)$.

Dibujitos con ejemplos :)

1.2. Resolución propuesta y justificación

Para resolver esta problemática, optamos por implementar un algoritmo de *Programación Dinámica*.

Con el fin de encontrar el recorrido factible que emplee menos tiempo; debemos comparar, para cada etapa, cuál es el menor tiempo con el que puede recorrer el camino faltante eligiendo en la instancia actual uno de los tres vehículos disponibles. Dado que la formulación de este problema es muy extensa, se realizó una formulación recursiva de modo que para cada problema se le asigna un valor dependiendo de un subproblema menor.

Formulación Recursiva

Optamos por comenzar recorriendo desde la etapa n hasta la etapa 0; *n* va a indicar la etapa actual, k_m la cantidad de motos y k_b la cantidad de boogys restantes que se pueden utilizar.

- Cuando llegamos a la etapa *n*$= 0$ es porque terminamos todo el recorrido, de modo que el tiempo devuelto va a ser 0.
- Cuando $k_m = 0$ y $k_b = 0$ es porque la etapa actual (*n*) y el recorrido restante (las *n-1* etapas) lo vamos a tener que hacer sólo en bicicleta, sin importar el tiempo que conlleve ya que nos quedamos sin motos y boogys para usar.
- Cuando $k_m = 0$ y $k_b \neq 0$ es porque utilizamos la mayor cantidad de motos posibles y las *n-1* etapas restantes -conjunto a la actual(*n*)- las vamos a tener que recorrer con Bicicleta o Boogy. Por este motivo se elige la opción con tiempo menor usando Bicicleta o Boogy en la etapa *n* y llamando recursivamente a la función para *n-1* considerando esta elección.
- De modo análogo, cuando $k_m \neq 0$ y $k_b = 0$ sólo vamos a contar con Motos y Bicicletas para la etapa actual y las *n-1* etapas faltantes.
- En cambio, en caso contrario, todavía tenemos disponible cantidad de los tres vehículos. Por este motivo, se comparan los tres casos: empleando la Bicicleta en la etapa n, la Moto o el Boogy llamando recursivamente a la función para *n-1* de modo que va a devolver el menor tiempo posible considerando la elección llevada a cabo.

$$
func(n, k_m, k_b) = \begin{cases}
0 & \text{si } n = 0 \\
tiempoBici(n) + f(n-1, 0, 0) & \text{si } k_m = 0 \wedge k_b = 0 \\
min \begin{pmatrix} tiempoBici(n) + func(n-1, 0, k_b), \\ tiempoBoogy(n) + func(n-1, 0, k_b - 1) \end{pmatrix} & \text{si } k_m = 0 \wedge k_b \neq 0 \\
min \begin{pmatrix} tiempoBici(n) + func(n-1, k_m, 0), \\ tiempoMoto(n) + func(n-1, k_m - 1, 0) \end{pmatrix} & \text{si } k_m \neq 0 \wedge k_b = 0 \\
min \begin{pmatrix} tiempoBici(n) + func(n-1, k_m, k_b), \\ tiempoMoto(n) + func(n-1, k_m - 1, k_b), \\ tiempoBoogy(n) + func(n-1, k_m, k_b - 1) \end{pmatrix} & \text{sino}
\end{cases}
$$

Dado que los n, k_m y k_b iniciales van a ser los dados por parámetro y en el planteo de nuestra ecuación en la llamada recursiva n siempre decrementa en 1 y los demás o bien quedan iguales o uno de ellos decrementa en uno, estos parámetros van a estar acotados por:

$$
\begin{array}{ccccc}
0 & \leq & n & \leq & n_{parametro} \\
0 & \leq & k_m & \leq & k_{m_{parametro}} \\
0 & \leq & k_b & \leq & k_{b_{parametro}}
\end{array}
$$

1.3. Análisis de la complejidad

1.3.1. Complejidad Temporal

1.3.2. Complejidad Espacial

Si bien, ya no piden ningun requisito, pongamos cuanta memoria usa :)

1.4. Código fuente

1.5. Experimentación

1.5.1. Constrastación Empírica de la complejidad

2. Zombieland II

2.1. Descripción de la problemática

2.2. Resolución propuesta y justificación

2.3. Análisis de la complejidad

2.4. Código fuente

2.5. Experimentación

2.5.1. Constrastación Empírica de la complejidad

3. Refinando petróleo

3.1. Descripción de la problemática

3.2. Resolución propuesta y justificación

3.3. Análisis de la complejidad

3.4. Código fuente

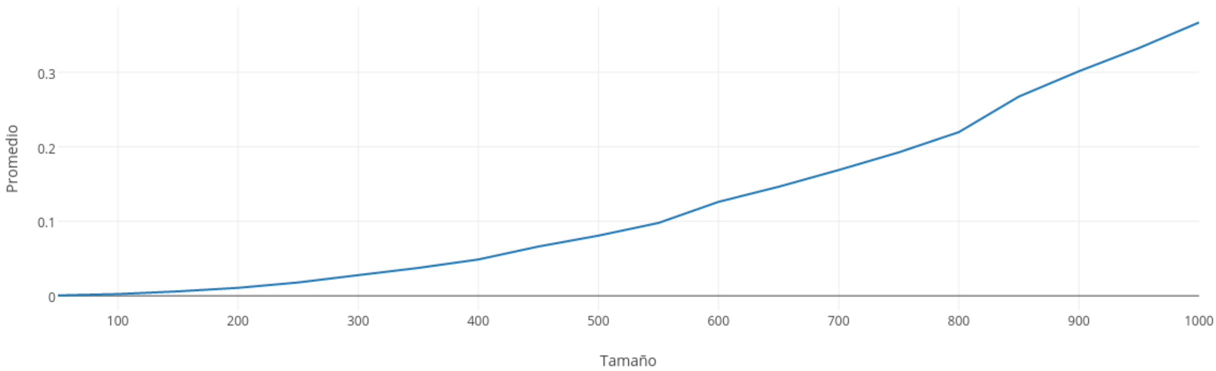
3.5. Experimentación

3.5.1. Constrastación Empírica de la complejidad

Para llevar a cabo esta experimentación, consideramos el peor caso posible de cantidad de ejes del grafo, es decir que habrá exactamente $n.(n-1)$ ejes (cada nodo puede conectarse con cualquier nodo del grafo, excepto consigo mismo), variando la cantidad de nodos.

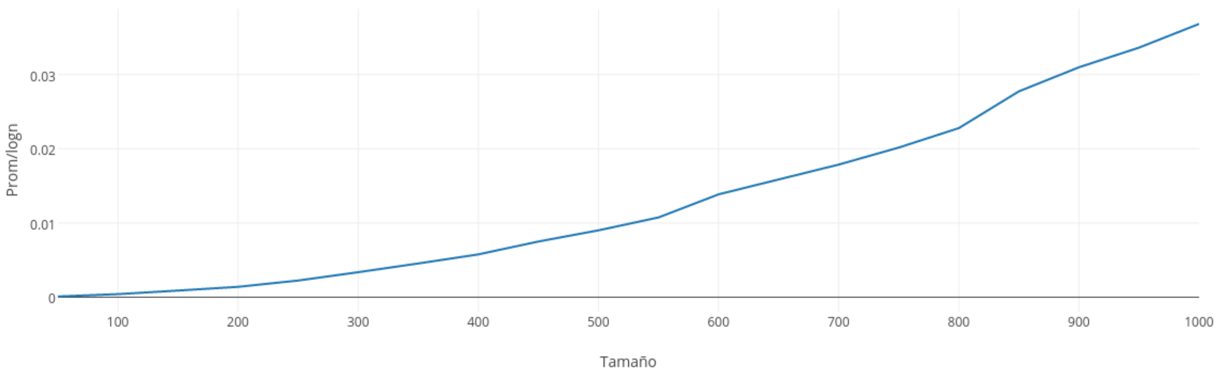
Los tiempos de ejecución para cada n (cantidad de nodos) fueron los siguientes:

n	Tiempo en segundos
50	0.0005254864
100	0.002332719
150	0.0059328642
200	0.01065945
250	0.0178728144
300	0.0277215188
350	0.0373807766
400	0.0487278266
450	0.0661190683
500	0.0807527896
550	0.0978193478
600	0.126140013
650	0.146378511
700	0.168847877
750	0.192539493
800	0.219705288
850	0.267457097
900	0.301437881
950	0.332602623
1000	0.366929358

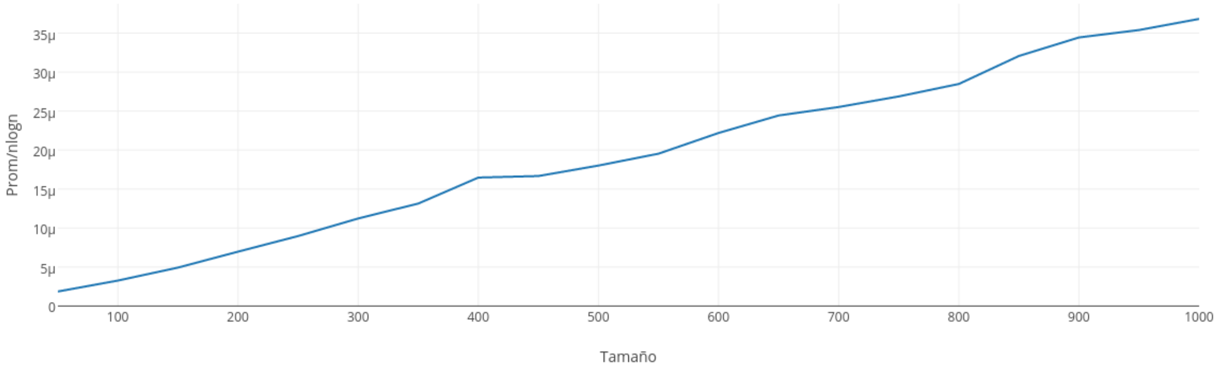


Dado que la Cota de Complejidad planteada teóricamente es de $O(n^2.log(n))$, era esperable que la curva sea una parábola creciente.

A simple vista, no se puede apreciar si la relación que tienen respecto de tamaño/tiempo es efectivamente la que buscamos (pues casi todas las curvas polinomiales tienen gráficos similares). Por este motivo, como siguiente paso decidimos comenzar a linealizar los tiempos, dividiendo a cada uno por $log(n)$.



La morfología de este gráfico es similar a la anterior, sigue siendo una parábola creciente, por lo tanto terminaremos de linealizar, dividiendo a cada uno por n, para ver si se trata de la parábola cuadrática.



Efectivamente puede observarse que el comportamiento es lineal. Por lo tanto, podemos afirmar que nuestra experimentación condice a la Cota Teórica planteada de $O(n^2.log(n))$.