Universidad Nacional Autónoma de México

Facultad de Ciencias

Complejidad Computacional

Tarea 1

García Ponce José Camilo - 319210536

1. **Ejercicio 1**

   a) Estima el tiempo de ejecución para una lista de $5,000$ elementos
   Primero usaremos el texto del ejercicio 1 para poder obtener la c en $T(n) = cf(n) = c(n\log(n))$
   Sustituimos los datos que conocemos, 0,5 segundos $= T(1000) = c(1000\log(1000))$
   Entonces tenemos $c(1000\log(1000)) = 0{,}5$
   Luego $c = \frac{0{,}5}{1000\log(1000)}$
   Y usando una calculadora obtenemos que $c = \frac{1}{6000}$ (usando logaritmo base 10)
   Con esto obtenemos la siguiente formula $T(n) = \frac{1}{6000}(n\log(n))$
   Ahora sustituimos n por 5000, obteniendo $T(5000) = \frac{1}{6000}(5000\log(5000))$
   Y otra vez usando la calculadora obtenemos que $T(5000) = 3{,}0824$, entonces son 3.0824 segundos

   b) Estima el tiempo de ejecución para una lista de $18,000$ elementos
   Vamos a usar la formula $T(n) = \frac{1}{6000}(n\log(n))$ (obtenida en el ejercicio de arriba)
   Sustituimos n por 18000, obteniendo $T(18000) = \frac{1}{6000}(18000\log(18000))$
   Usando la calculadora obtenemos que $T(18000) = 12{,}7658$, entonces son 12.7658 segundos

   c) Determina el tamaño máximo de la lista que puede ser ordenada en menos de 10 segundos
   En este ejercicio intente despejar la n de $T(n) = \frac{1}{6000}(n\log(n))$, pero no lo logre (ya no me acuerdo como se hace), entonces vamos a usar código de Python para resolverlo
   El código es el siguiente:

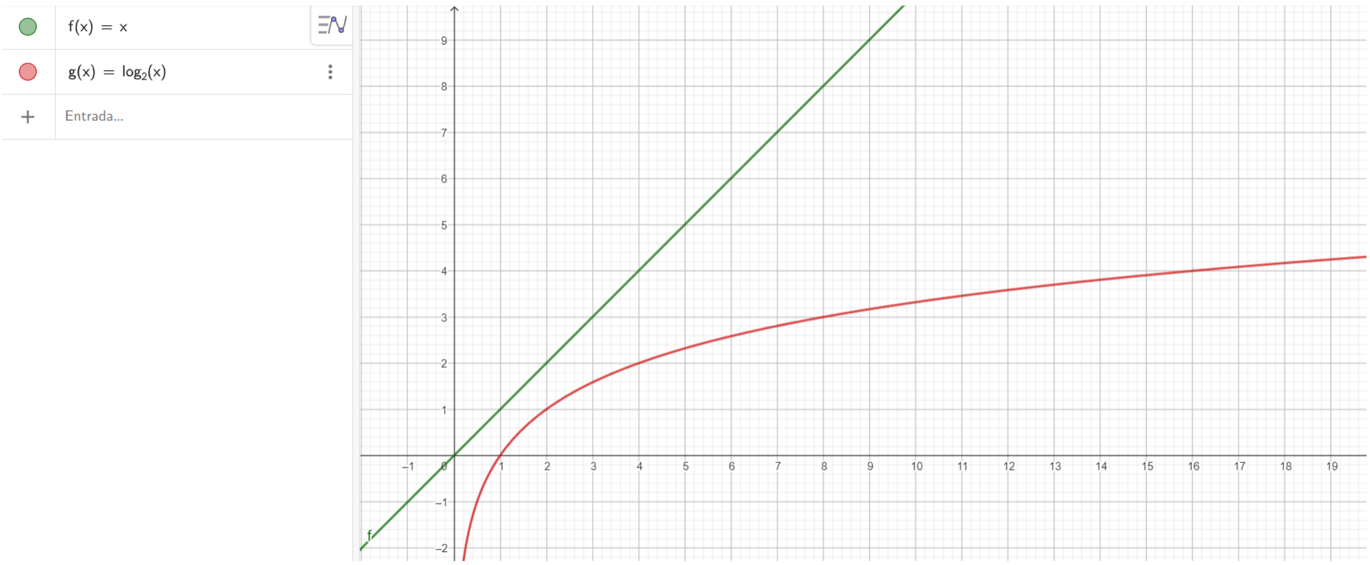
```python
import math

def encontrar_n(segundos):
    cantidad = 1
    tiempo = 0
    while tiempo < segundos:
        cantidad += 1
        tiempo = (1/6000) * (cantidad * math.log10(cantidad))
        if tiempo > segundos:
            cantidad -= 1
    print(cantidad)

encontrar_n(10)
```

   En el código iniciamos con una variable cantidad en 1 (que es el numero de elementos), luego tenemos otra variable tiempo en 0 (el tiempo que tarde con cantidad de elementos), entonces mientras tiempo sea menor que los segundos que buscamos, vamos incrementando cantidad y calculando tiempo hasta romper el ciclo, además cada vez que calculamos el tiempo, revisamos si ya es mayor que segundos y al final se imprime cuantos elementos son para el tiempo dado
   Ejecutando el código obtuve que el tamaño de la lista es 14426

2. **Ejercicio 2**

   a) ¿Qué algoritmo es mejor?

   El algoritmo B es mejor, veamos la razón de que pase esto.
   Primero recordemos que n es mayor que $\log_2(n)$ para $n \geq 0$. Para este mini análisis vamos a tomar al algoritmo A como n^3 (ignorando el $10n + 16$ y el 20 multiplicando al n^3) y al algoritmo B como $n^2 \log_2(n)$ (ignorando el 22 multiplicando al n^2 y el 8 multiplicando al n), todo esto para poder facilitar el análisis y también ya que con valores de n algo grandes, estos números constantes no afectan mucho y lo que más importa es la n (en el caso de $10n$, lo ignoramos ya que n^3 crece mucho más rápido que n, entonces no tiene tanto peso en el tiempo de ejecución). Veamos que n^3 lo podemos ver como $n^2 * n$, entonces ya sabemos que $\log_2(x)$ es mejor que n, por lo tanto como n^2 y n^2 crecen de la misma forma, podemos ver que $n^2 * n$ va a crecer más rápido (va a ser peor) que $n^2 \log_2(n)$, todo causado debido a que n^2 al multiplicarse por n genera un numero más grande que al multiplicar n^2 y $\log_2(n)$ (para valores de n mayores a 0). Entonces concluimos que el mejor algoritmo va a ser el B.



   b) ¿Para qué valores de n el algoritmo A es mejor que el B?

   Para los valores n del 1 al 6, el algoritmo A es mejor, esto lo podemos notar en nuestra tablita de valores, ya que en los valores del 1 al 6 el algoritmo A tiene valores menores y luego esto ya no se cumple, además como nuestros valores de n son enteros positivos, tendremos que A solo sera mejor en estos valores.

n	$20n^3 + 10n + 16$	$22n^2 \log_2(8n)$
1	46	66
2	196	352
3	586	907.82
4	1336	1760
5	2566	2927.06
6	4396	4423.29
7	6946	6260.33
8	10336	8448
9	14686	10994.81
10	20116	13908.24

   c) ¿Para qué valores de n el algoritmo B es mejor que el A?

   Para los valores n mayores o iguales a 7, el algoritmo B es mejor, esto lo podemos notar en nuestra tablita de valores (la misma usada en el ejercicio de arriba), debido a que en los valores del 7 al 10 el algoritmo B tiene valores menores y como nuestros valores de n son enteros positivos, tendremos que B sera mejor con valores más grandes (estrictamente) que 6.

n	$20n^3 + 10n + 16$	$22n^2 \log_2(8n)$
1	46	66
2	196	352
3	586	907.82
4	1336	1760
5	2566	2927.06
6	4396	4423.29
7	6946	6260.33
8	10336	8448
9	14686	10994.81
10	20116	13908.24

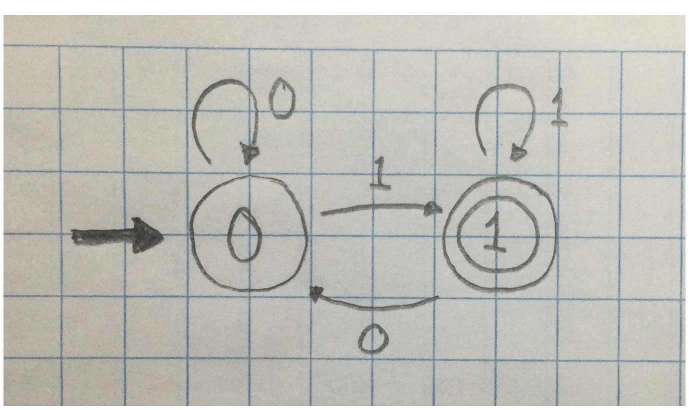
3. **Ejercicio 3**

   a) $(1^*0^*)^*1$
   El lenguaje de la expresión regular son todas las cadenas del alfabeto $\{0, 1\}$ que terminan con un "1".
   Esto debido a que tenemos $(1^*0^*)^*$, por lo tanto sabemos que esa expresión regular hace referencia a todas las cadenas (incluyendo la vacía) de este alfabeto pero en nuestra expresión regular tenemos un 1 al final, por lo tanto todas las cadenas del lenguaje deben terminar con un "1".
   Algunos ejemplos de cadenas del lenguajes son: 1, 011, 0001 y 1111

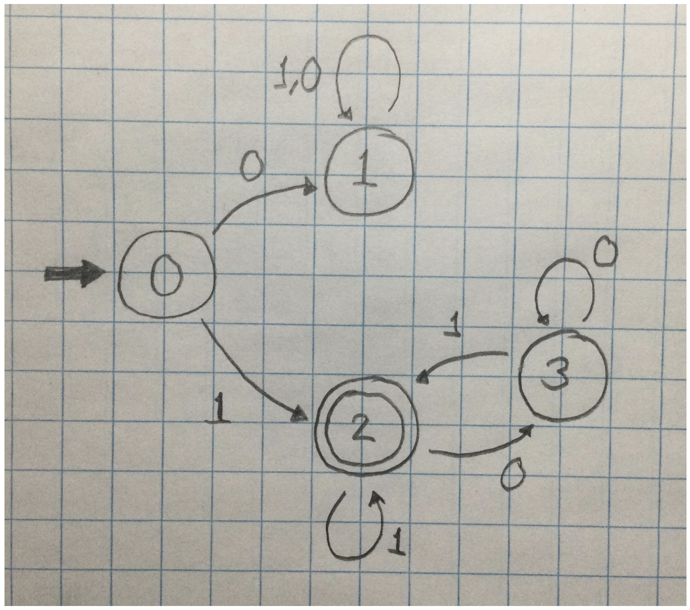
   b) $(10^*)^*1$
   El lenguaje de la expresión regular son todas las cadenas del alfabeto $\{0, 1\}$ que inician con un "1" y terminan con un "1".
   Esto debido a que, como en la expresión anterior, tenemos un 1 al final de la expresión por lo tanto las cadenas deben terminar con un "1", pero en la expresión tenemos $(10^*)^*$ entonces veamos que cadenas son. Primero la expresión 10^* son las cadenas de la forma: un "1" y ningún o varios "0"s, entonces $(10^*)^*$ son la cadena vacía o varias cadenas de la forma: un "1" y ningún o varios "0"s, por lo tanto son todas las cadenas que inician con un "1" o la cadena vacía. Y juntando lo que revisamos arriba, concluimos que $(10^*)^*1$ son las cadenas del alfabeto que inician con un "1" y terminan con un "1".
   Algunos ejemplos de cadenas del lenguajes son: 1, 101 y 100011

   c) Diferencia entre los lenguajes anteriores
   La principal diferencia es que las cadenas del lenguajes de $(10^*)^*1$ necesitan tener un "1" al inicia, pero esta condición no es obligatoria para las cadenas del lenguaje de $(1^*0^*)^*1$, esto debido a que $(10^*)^*1$ tiene el primer 1 sin * por lo tanto las cadenas necesitan un "1" al inicio, a diferencia de $(1^*0^*)^*1$ que puede o no iniciar con "1", todo gracias a la estrella de Kleene (*).

   d) Representación gráfica
   - $(1^*0^*)^*1$
     Esta la representación, ya que para aceptar la cadena se necesita un "1" al final, en otro caso no aceptamos

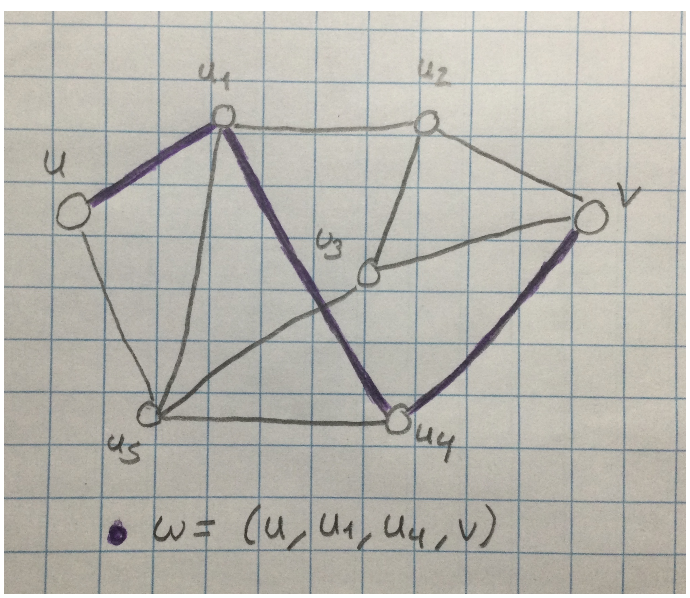


- $(10^*)^*1$
  Esta la representación, debido a que si al inicio de la cadena hay un "0" entonces nunca la vamos a aceptar y el resto es como la representación la expresión regular de arriba para solo aceptar con un"1" al final

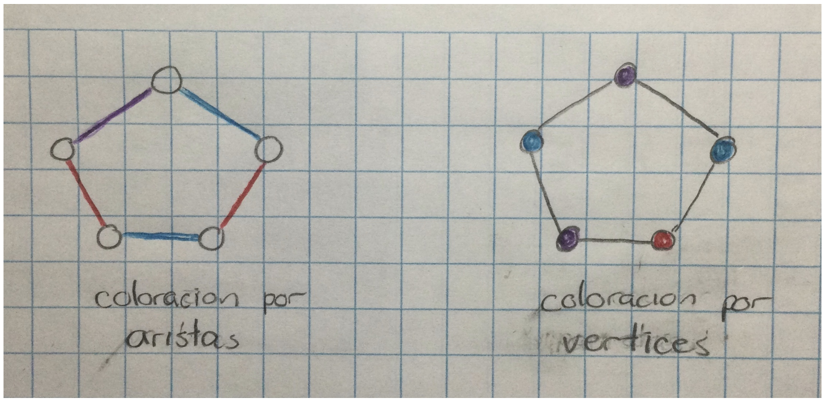


4. **Ejercicio 4**

   a) ¿Qué es un camino en gráficas?
   Para dos vértices (no necesariamente distintos) u y v en una gráfica G, un $u - v$ camino W en G es una sucesión de vértices en G, empezando en u y terminando en v tal que vértices consecutivos en W son adyacentes en G. Los vértices no consecutivos en W no necesariamente son distintos.



   b) ¿Qué es una coloración en gráficas?
   Una coloración (por vértices) de una gráfica G es una asignación de colores a los vértices de G, un color a cada vértice. Si no hay dos vértices adyacentes que se les asigne el mismo color, la coloración es una coloración por vértices propia.
   Una coloración (por aristas) de una gráfica G es una asignación de color a las aristas de G. Si aristas adyacentes tienen diferentes colores, entonces la coloración es una coloración por arista propia.



5. **Fuentes**

- Geogebra, para graficar algunas cosas del ejercicio 2
- Mis apuntes de la materia Graficas y Juegos 2022-2, impartida por el Profesor Juan Carlos García Altamirano, apuntes de las clases del 22/02/22, del 25/5/22 y 7/6/22