UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructuras de Datos I

Tiempo de Ejecución de Peor Caso de un Programa

En este taller vamos a implementar algoritmos de búsqueda y en algunos casos vamos a medir sus tiempos de cómputo con la intención de obtener intuición respecto de su complejidad temporal. Recuerden que cuando se habla en términos de complejidad lo que se está haciendo es señalar de qué depende principalmente el número de operaciones que realiza, e ignorar todos los elementos de "menor relevancia".

Por ejemplo, un algoritmo que hace $2n^3+n^2+5$ operaciones en peor caso, y otro que hace $\frac{1}{100}n^3$, en ambos casos decimos que tiene complejidad cúbica, $O(n^3)$. Claramente no hacen la misma cantidad de operaciones. Pero en ambos la eficiencia depende aproximadamente $c.n^3$ milisegundos en terminar en el peor caso, con c una constante que no depende de n.

$$2n^3 < 2n^3 + n^2 + 5 < 4n^3 \Longrightarrow 2n^3 + n^2 + 5 \approx c\,n^3 \qquad \text{(en el peor caso)}$$

En este taller, a diferencia de los anteriores, nos interesa particularmente resolver los ejercicios mediante algoritmos eficientes. Por eso pedimos no usar búsqueda lineal o cualquier algoritmo que tenga que recorrer toda la secuencia en caso de que exista una alternativa posible. Y además vamos a medir tiempos para estimar la constante c que se ignora cuando hablamos en términos de complejidad, que es muy difícil conocer en términos analíticos por la cantidad de capas de abstracción que hay entre el código que escribimos y lo que la computadora realmente hace al final de cuentas.

Para los siguientes ejercicios recomendamos utilizar `contruir_vector`, que recibe un entero n y un `string` *disposicion* y devuelve un vector de n elementos en la disposición especificada (opciones para disposición: "asc", "desc", "azar", "iguales")

Ejercicio 1. *Medición de tiempos*
En el archivo ejercicios .cpp encontrarán la implementación de hayDuplicados(vector<int> v) que dado un vector v demora aproximadamente $c.n^2$ segundos en terminar en el peor caso (siendo n el tamaño del vector de entrada y c una constante que no depende de n). Es decir, su tiempo de ejecución de peor caso pertenece a $O(n^2)$.

1. Estimar el valor de c mediante simulaciones. ¿Cambia el valor encontrado dependiendo el valor de los elementos del vector? ¿Cambia el valor encontrado según desde qué n se comienza a medir?
2. Calcular cuántos segundos demoraría (sin correrlo!) aproximadamente en el peor caso si $n = 2200000000$ (# de usuarios de Facebook)

Ejercicio 2. *Graficar*
Escribir los tiempos calculados en un archivo con n desde 1 hasta 10000 (paso 500). Este archivo deberá contener una columna para n (con encabezado "n") y una columna (con cualquier encabezado declarativo) para el tiempo para dicho n en segundos.

Generar un gráfico con eje x: n y eje y: tiempo de ejecución en segundos. Para graficar, pueden utilizar cualquier programa que deseen o el script provisto por la cátedra. Para ver el modo de uso del script: `python3 graficador.py --help`. Si se desea visualizar más de una curva a la vez, pueden agregar columnas al archivo con nombres declarativos como encabezado de la columna.

Ejercicio 3. *Opcional*
Modificar las guardas del programa del punto 1 para que los ciclos terminen en caso de haber encontrado un duplicado. Medir nuevamente. ¿Cambió la complejidad?

Ejercicio 4.
Escribir programas con tiempo de ejecución de peor caso pertenecientes a $O(1)$, $O(n)$, $O(n^3)$ y $O(n.log(n))$. Utilizar si se desea la función provista en el ejercicio 1 y la función busqueda_binaria(vector<int> v, int e) incluida en los archivos de la clase.

1. Medir el tiempo en segundos y completar la siguiente tabla

	$O(1)$	$O(n)$	$O(n^3)$	$O(n.log(n))$
n=100				
n=1000				
n=10000				
n=100000				

Ejercicio 5. *STL*
Realizar simulaciones para estimar la complejidad temporal de las siguientes operaciones sobre vectores en C++.

1. v. size ()
2. v.push_back(e)
   Tip: Análizar que pasa cuando se supera el tamaño del vector en memoria `http://www.cplusplus.com/reference/vector/vector/capacity/` `http://www.cplusplus.com/reference/vector/vector/reserve/`
3. v.pop_back()
4. v[i] (¿cambia la complejidad estimada según el valor de i?)
5. v[i] = e; (¿cambia la complejidad estimada según i?)
6. v. flip () (sólo para v de tipo vector<bool>)
7. v. clear ()

¿Por qué sucede que las mediciones con O(1) contienen dos lineas marcadas? ¿Por qué el push_back tiene saltos?

Ejercicio 6.

Realizar simulaciones y gráficos para determinar el tiempo de ejecución de **peor caso** del programa algunSubconjSuma (se encuentra en ejercicios.cpp).