Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación
ICC2133 – Estructuras de Datos y Algoritmos – Sección 2
Profesores Yadrán Eterovic y Cristobal Gazali
Ayudante Jefe Trinidad Vargas
Primer Semestre 2021

Alumno: José Antonio Castro

Informe Tarea 0:
Propagación del Kevin-21

Calculo de Complejidades Teóricas

0.1 Algoritmos

- Iteración del for del *depth*: En el peor de los casos va hasta el final recorriendo todos los casos, sería $\mathcal{O}(n)$.
- *world_search*: En el peor de los casos va hasta el final recorriendo todos los casos, sería $\mathcal{O}(n)$. (Tambien se puede ver como $\mathcal{O}(n \cdot k)$).
- Iterar por los hijos de alguien (*child*): En el peor de los casos va hasta el final recorriendo todos los casos menos en padre (sería si el nodo padre tiene todos los hijos él y recorre $n-1$), sería $\mathcal{O}(n)$.
- Iterar de la forma *delete_person*: Esta forma itera el árbol primero a los hijos y luego a los hermanos de la forma DFS. Tiene complejidad $\mathcal{O}(n)$ en el peor de los casos.
- Recursion informe: En el peor de los casos este recorre todo una vez. Así que es $\mathcal{O}(n)$.

0.2 Eventos

(a) Add Contact: Tiene una suma de complejidades de *depth*, *search_world* y un $\mathcal{O}(1)$. La suma por aproximación asintótica es del orden de $\mathcal{O}(n)$.

(b) Positive: Tiene una suma de las complejidades de *depth*, *serach_world* y una iteración a los hijso del tipo *child*. Serian $\mathcal{O}(3n)$, pero en aproximacion asitotica se simplifica a $\mathcal{O}(n)$.

(c) Negative: Tiene una suma de las complejidades de *depth*, *serach_world* y *delete_person*. Serian $\mathcal{O}(3n)$, pero en aproximacion asitotica se simplifica a $\mathcal{O}(n)$.

(d) Recoverd: Tiene la suma de las complejidades *depth* y *world_search*. Por lo que corresponde al orden de complejidad $\mathcal{O}(n)$.

(e) Correct: Este evento recorre 2 veces los pasos de los algoritmos *depth* y *world_search* y 4 veces el algoritmo de *child*. Tendria una suma de compleidades de $8n$, lo que corresponde al orden de $\mathcal{O}(n)$.

(f) Inform: Basicamente sigue el algoritmo de recursion del informe, que como ya dijimos en el peor de los casos recorre todo n una vez, por lo que es del orden $\mathcal{O}(n)$.

(g) Statistics: Este evento recorre recursivamente todos los casos de una forma identica al algoritmo *peron_delete*, solo que en vez de borrarlos, los agrega a un contador según su estado. Es por ello que tiene una complejidad de $\mathcal{O}(n)$.

Ventajas y Desventajas en el nuevo modelado de Persona

En el nuevo modelo de persona, los contactos estrechos pasan de listas ligadas a un arreglo.

- La complejidad de busquedas pasa a ser $\mathcal{O}(1)$, debido a que en un arreglo se puede acceder de forma instantanea con el indice (*id* en este caso). Pero se debe llegar a este arreglo en un principio y recorrer otras personas en la linea. Es por ello que igual sería una complejidad de $\mathcal{O}(n)$.

- La complejidad de inserción de forma similar, daría un total de $\mathcal{O}(n)$. Pues la inserción sería automática, pero la búsqueda de llegar a ese caso tendría n.

- Y por último la complejidad de eliminación sería igual a insercion. es decir, $\mathcal{O}(n)$.

Eso si tener en cuenta que esa es la complejidad en el peor de los casos, pero en el caso medio mejoraria, pues es más facli que acceda a los casos en general debido a la dirección automática en arreglos.

- Ventaja: Ahora es mucho más facil acceder a cada elemento si se tiene el *id*, esto baja la complejidad de busqueda. En cambio en listas ligada se itera por cada nodo. Si bien en el peor caso ambos de $\mathcal{O}(n)$, en el mediano con arreglos es mas rapido.

- Desventaja: Ahora se pide la momoria toda junta a la hora de crear los arreglos y puede ser que en el PC no siempre haya tanta memoria junta disponible.

Contraste Python vs C

Corrí todos los testcases de la carpeta **compare/** tanto en **Python** como en **C** y dan diferentes tiempo en mi PC. Los gráfique en una tabla para compararlos.

Tests	Python (seg)	C (seg)	N° allocs	Duración Python/C
Test 1	1.219	0.218	70,178	5.591743119
Test 2	2.53	0.43	140,272	5.88372093
Test 3	3.442	0.601	209,779	5.727121464
Test 4	4.51	0.792	279,569	5.694444444
Test 5	5.839	0.977	350,172	5.976458547
Test 6	7.075	1.201	420,142	5.89092423

Figure 1: Tabla Contraste de tiempo Python v/s C

Como podemos ver, C es casi 6 veces más rapido que Python.

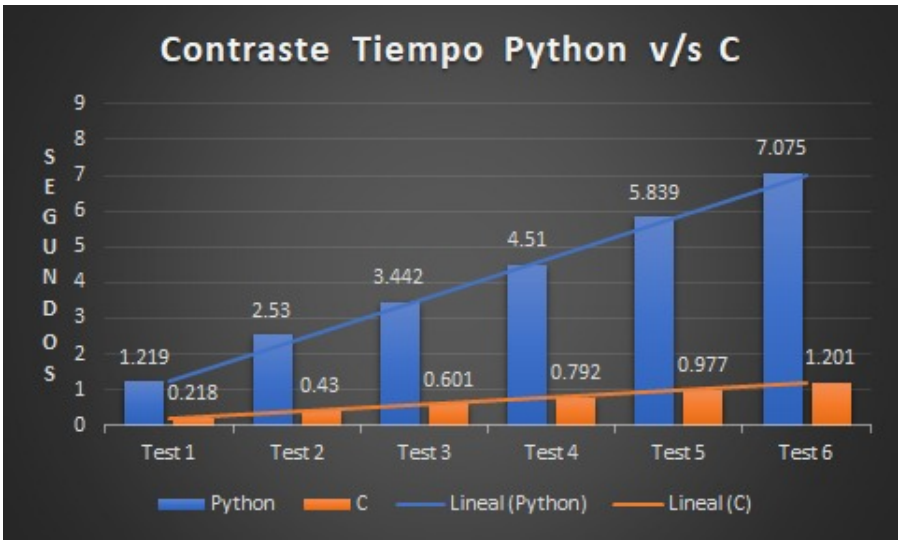

Figure 2: Grafico Contraste de tiempo Python v/s C

Ambos son presentan crecimientos lineales, como muestra el gráfico. Pero se puede ver la clara diferencia y eficiencia de *C* sobre *Python*. Esto se puede explicar por diversas razones en su conjunto:

(a) El lenguaje C es un lenguaje compilado, en cambio *Python* es un lenguaje interpretado por lo cual debe ejecutar un paso más antes de pasar al lenguaje de máquina, es donde se convierte en BYTE CODE y es ejecutado por una Máquina virtual eso toma su tiempo y lo hace mas lento, que tanto dependerá de que se ejecute.

(b) *Python* por defecto viene con hartos build-ins y funciones que son memoria extra constante en cada archivo. Tambien al inicializar una clase, vienen muchas cosas por detrás para cada una de las instancias (que aparentan ser pocas) pero al trabajar con grandes programas se nota su peso y tiempo extra que requiere

inicializarlo y procesarlo. En la misma linea los 'import' de *Python* vienen con muchas cosas que pesan computacionalmente a la hora de cargar.

(c) El Garbage-collector, los lenguajes de interpretado lo tienen. Básicamente se encarga de eliminar los objetos que no se encuentran en uso por el programa, sin duda este proceso es beneficioso para la memoria; sin embargo, vuelve mas lento el software.

(d) El tipado de datos que tiene *C*, *Python* no es un lenguaje tipado, si no más bien lo que conocemos como type-juggling. El tipado de datos permite tener un mejor control de la información y hace más efectivo la interpretación de la información, así el compilador o el intérprete no tienen que gastar tiempo determinando el tipo de datos y evitar comportamiento erráticos.