NOMBRES: Isidora Guelfand 16635108

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2133 — — I' 2021

Tarea 1 – Respuesta Pregunta 1

Complejidades
Asumiendo que la cantidad de países es n, el número total de regiones entre todos los países es r

1. create_world : $O(1)$
   No se debe realizar ningun iteración. Son un número limitado (5) de operaciones siempre.

2. create_region : $O(r)$
   Se debe iterar sobre la cantidad de regiones creadas. Es decir, es del orden de la cantidad de regiones.

3. encontrar raíz : $O(1)$

   Es conocido que en C encontrar el elemento con índice n de un arreglo es de $O(n)$. Esto se debe realizar dos veces, una para cada arreglo, por lo que la operación es de $2 * O(1)$. Sin emabrgo, se sabe que $2 * O(1) \in O(1)$

4. ADD CONTACTS : $O(m + n)$

   Sea m todos los nodos del árbol correspondiente La complejidad para encontrar al nodo al cual se deben añadir los contactos depende de cual es el nodo. En el caso del nodo 0, la complejidad es de $O(1)$, ya que la función *search* no realiza ninguna iteración. Este evento entrega la ruta, por lo que se debe aplicar la función *search*. Esta función tiene d iteraciones, donde d representa la profundidad de la ruta. Además, dentro de cada una de esas iteraciones se llama a la función *search_contact* . Esta función puede no realizar ninguna iteración, o iterar sobre todos los hijos del nodo actual de la ruta. Por lo tanto, encontrar el contacto en el peor de los casos puede recorrer todos los nodos del árbol (si están todos en la ruta). Sin embargo, se puede asegurar que a lo más se recorre cada nodo una vez, ya que $search_contact$ solo recorre los contactos estrechos, y *search* solo itera sobre los padres de los contactos estrechos. Esta lógica (encontrar el árbol y luego encontrar el nodo sobre el cual se hará una operación), es utilizada para los eventos NEGATIVE, POSITIVE, RECOVERED y CORRECT.

   En el mejor de los casos, no recorre ningún nodo para encontrar el final de la ruta(ya que se añaden al nodo 0). En el caso general, por cada nodo de la ruta, se recorren c_j nodos, que son los contactos estrechos correspondientes para encontrar el siguiente nodo en el ruta dada cierta profundidad. Es decir, para cada nivel en la ruta, se deben iterar sobre los contactos estrechos, para encontrar al nodo siguiente en la ruta. Si la ruta tiene solo nodos que son la cabeza de las listas ligadas de los contactos estrechos, solo se deben hacer d iteraciones para encontrar el final de la ruta. Si esto no ocurre, se den realizar $n * c_j$ iteraciones. Por lo tanto, si en cada nivel de la ruta (entiendase nivel por una iteración de la función *search*, una distinta profundidad) se sabe que hay c_j contactos estrechos, entonces para encontrar el nodo final de la ruta desde el nodo inicial se deben realizar a lo más $d * c_j$ iteraciones, tomando c_j como el máximo de contactos estrechos hasta un nivel d. Es decir, no se puede garantizar que se realizarán menos iteraciones que el total de nodos que existen hasta esa profundidad. Por temas

prácticas, se tomará que p_d es la cantidad de nodos hasta (inclusive) la profundidad d . Así, se deben realizar a lo más p_d iteraciones para encontrar un nodo a profundidad d.Sin embargo, esa información no se conoce inmediatamente, por lo que no se puede garantizar en primera instancia una operación menor a $O(n)$

En casos mejores que el peor caso, no se debe iterar sobre todos los nodos, ya que basta que sea un árbol binario la región sobre la cual se está buscando cierto nodo, y al existir "sub árboles", para cada nivel de profundidad se descarta el otro sub-árbol.

Además, por cada contacto agregado existe una complejidad de $O(1)$. Por lo tanto, la complejidad de este evento es $O(p_d + n)$, donde n es el número de nuevos contactos a agregar. Esta puede disminuir dependiendo de la cantidad de contactos estrechos en cada nivel. Sin embargo, como no se conoce p_d, el orden es de $O(m + n)$

5. RECOVERED: $O(m)$
   Para encontrar el nodo, se debe realizar una operación de $O(p_d)$, donde d es la profundidad y p_d la cantidad de nodos estrechos existentes en cierta profundidad. Por último, una vez encontrado el nodo, solo se debe cambiar su estado, por lo que se debe realizar una operación de $O(1)$. Así, en resumen, este evento tiene una complejidad que esta determinado por la profundidad del nodo: $O(p_d)$.Sin embargo, no se puede asegurar que encontrar el nodo del recuperado sea menor a $O(m)$, donde m es todos los nodos del árbol correspondiente. Además, por como están construidas las funciones $search_contact$ y $search$ los nodos se recorren solo una vez.

6. POSITIVE: $O(m)$
   La complejidad para encontrar al positivo es igual a los dos eventos anteriores, en el peor de lo casos se debe recorrer todo el árbol, en el mejor de los casos no se realiza ninguna iteración extra sobre el nodo raíz. Sin embargo, en este evento se debe iterar además sobre todos los hijos del nodo que se está buscando. Siempre se debe iterar sobre todos los hijos. En el caso general, se deben realizar a lo más operaciones de orden $O(p_d)$ para encontrar al nodo, y luego j operaciones, donde j son los hijos del nodo buscado. Es decir, es del orden $O(p_d + j)$, que no se puede asegurar sea menor a $O(m)$.

7. NEGATIVE: $O(m)$
   De nuevo, no se puede asegurar que en esta caso no se recorran todos los nodos del árbol correspondiente. Para encontrar el árbol se necesitan solo dos operaciones. Sin embargo, se puede asegurar que a lo más se recorre cada nodo una vez (por lo explicado anteriormente de las funciones $search_contact$ y $search$. En el mejor de los casos, se podría recorrern solo el primer nodo. Sí se puede asegurar que se deben recorrer todos los hijos(y los hijos recursivos) del nodo sobre el cual se realiza la operación, pero estos podrían ser todos los nodos del árbol menos la raíz o no ser ninguno. Como a lo más se realiza una operación sobre todos los nodos del árbol , es de $O(m)$.

8. CORRECT: $O(m)$
   Acá se deben encontrar dos nodos distintos para realizar la corrección. como fue mencionado anteriormente, para encontrar cada uno de estos nodos, se deben realizar operaciones del orden $O(m)$. Como son dos, en conjunto se realizan operaciones del orden $2 * O(n)$. Se sabe que $2 * O(m) \in O(m)$. Luego de haber encontrado los dos nodos cuyos hijos deben ser intercambiados, se itera sobre las dos listas ligadas de hijos. cada una de estas iteraciones está en en el orden $O(m)$.Esto ocurre aunque sea imposible iterar sobre un nodo más de una vez (si se encontraba en la ruta no es un constacto estrecho). Luego, se sabe que esta función en total realiza operaciones del orden $2 * O(m) + 2 * O(m)$, como se conoce que $4 * O(m) \in O(m)$, se puede concluir que la función se encuentra en $O(m)$

9. INFORM: $O(m)$
   Esta función lo que hace es recorrer todo el árbol siempre. Siempres tendrá m iteraciones en total, por lo que siempre será $O(m)$. la forma de realizar esto es llamar a la función recursiva sobre los hijos de un nodo, y luegi avanzar en la lista ligada. Así, recorrecore el árbol desde arriba y desde la izquierda, y pasa por cada nodo una sola vez.

10. STATISTICS: $O(m)$
    Al igual que el evento anterior, el rol de esta función es recorrer todo un árbol completo, pero pasando por cada nodo solo una vez. Por esto, siempre se recorre una vez cada nodo, y sobre cada nodo se realizar una sola operación. . Por esto, se hacen m iteraciones y por lo tanto, el evento es de $O(m)$.

11. destroy_world: $O(m)$
    Misma explicación anterior. Se recorre cada nodo una sola vez y no más.

NOMBRES: Isidora Guelfand 16635108

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2133 — — I' 2021

Tarea 1 – Respuesta Pregunta 2

Nueva estructura

1. Búsqueda : $O(d)$
   Al conocer el índice de cada ID, para encontrar un nodo en un arból será solamente necesario encontar el arreglo de los contactos estrechos al cual pertenece, ya que encontrar el elemento número n de un arreglo es de $O(1)$, por lo que es irrelevante para el cálculo de la complejidad. Para encontrar este arreglo, se debe iterar desde la raíz del nodo por los hijos hasta la profundidad requerida (asumo que todavía esta es entregada en la ruta). Sin embargo, al iterar sobre cierta profundidad, no será necesario iterar también sobre sus contactos estrechos, ya que el siguiente nodo de la ruta podrá ser obtenido con una sola operación: recuperar el elemento j del arreglo correspondiente. Por lo tanto, para encontrar un nodo se necesitara n iteraciones, la profundidad a lo cual se encuentra el nodo. Sin embargo, no se puede asegurar que el árbol es una línea recta y que se deban recorrer todos los nodos. Por lo que sin conocer la profundiad de la operación es $O(m)$. Es decir, estrictamente la complejidad no cambia en el peor de los casos, pero si lohace en la práctica, ya que será un una operación de $O(n)$, lo que a lo más será de $O(m)$. Anteriormente, solo se podían garantizar a lo más p_d iteraciones, donde esta parámentro representa la cantidad de nodos hasta la profundidad d.

2. Inserción
   Depende como se construya el arreglo. Asumiendo que es de largo variable, luego habrá que recorrer el arreglo completo para insertar un elemento al final. Por lo tanto, la complejidad no cambia. Se debe recorrer toda la lista para insertar un elemento al final, para cada persona que se inserta. Sin embargo, se debe encontrar el arreglo de contactos estrechos antes. Nuevamente, no se puede asegurar que esto será menor a $O(n)$ para agregar un solo elemento.

3. Eliminación $O(m)$
   Si bien se mantiene la complejidad, por que el peor de los casos no cambia (eliminar el último nodo de árbol), la eliminación si cambia el tiempo promedio de ejecución y lo hace menos eficiente. Al encontrar el nodo a eliminar, si este forma parte de un arreglo de contactos estrechos y no es el último, se deberá iterar también sobre todos los ndoos siguientes para desplazarlos dentro del arreglo. Esto, no ocurría en el caso de la lista liagada, ya que solo se debían cambiar atributos del nodo anterior y del siquiente. por lo tanto, al tener una lista ligada y tener que eliminar el primero nodo de los contactos estrechos a una profundidad n, se debían realizar n iteraciones. Sin emabrgo, al tener que eliminar el nodo que es el primero de un arreglo se deberán realizar $n + c$ iteraciones, donde c es la cantidad de nodos en el arreglo que se encuentra el nodo a eliminar.

NOMBRES: Isidora Guelfand 16635108

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2133 — — I' 2021

Tarea 1 – Respuesta Pregunta 2

C y Python

Table 1: Comparación en segundos

test	C	Python
1	0.95	2.1
2	2.17	4.1
3	2.77	9.3
4	3.76	11.4
5	4.9	11.9
6	6.5	13.7

C es un lenguaje de más bajo nivel por lo que permite la creación de lenguajes más eficientes. En general, en los test realizados el tiempo de ejecución es de alrededor de la mitad en C que en Python. C se debe compilar, por lo que C puede correr directamente sus instrucciones. Python es un lenguaje de más alto nivel, por lo que debe interpretarse mientras está corriendo, lo que lo hace menos eficiente.

Además, en Python existe un uso menos eficiente de memoria y variables, ya que estas no deben ser declaradas antes de ser utilizadas. Por otro lado, no se conoce de que tipo es una variable en Python ("dynamic typing"), por lo que esto lo hace también menos eficiente que C. Fuente: https://stackoverflow.com/questions/3033329/w
are-python-programs-often-slower-than-the-equivalent-program-written-in-c-or