Ing. de Sistemas e Informática
Estructuras de Datos y Algoritmos
Jorge Mario Londoño P.

Ejercicios de repaso 4

Estructura Unión-Búsqueda

1. Asi como se tiene la operación `unión(a,b)` que conecta dos componentes de la colección, sería posible tener una operación `separar(a,b)`? Pensar en que sería necesario hacer para implementarla e identificar dificultades para realizar esta operación.

2. Se tienen los elementos 0..9 y se hacen la siguiente secuencia de uniones:

9-0 3-4 5-8 7-2 2-1 5-7 0-3 4-2

a. Ilustrar la representación al final de esta secuencia que se obtiene aplicando la estructura QuickFind.
b. Repetir para la estructura QuickUnion. Cuál es la mayor altura?
c. Repetir para la estructura QuickWeightedUnion. Cuál es la mayor altura?
d. Cuantas componentes conexas hay al final.

3. El método de compresión de caminos propone conectar todos los nodos a la raíz del árbol para mejorar la eficiencia de las búsquedas. Analizar:

- Como implementar la compresión de caminos al momento de realizar la unión? Cuál sería el orden de crecimiento de la operación unión resultante?
- Como implementar la compresión de caminos al momento de realizar la búsqueda? Cuál sería el orden de crecimiento de la operación búsqueda resultante?

Métodos de Ordenación

1. Sea el siguiente arreglo de Chars:

M E T O D O S D E O R D E N A C I O N

a. Ilustrar la secuencia de pasos que se obtiene aplicando el algoritmo de ordenación por selección. Contabilizar el número de comparaciones e intercambios realizados.
b. Ilustrar la secuencia de pasos que se obtiene aplicando el algoritmo de ordenación por inserción. Contabilizar el número de comparaciones e intercambios realizados.
c. Ilustrar la secuencia de pasos que se obtiene aplicando el algoritmo de ordenación shellsort. Contabilizar el número de comparaciones e intercambios realizados.

2. *Natural Mergesort*

Una idea para organizar los elementos de un arreglo aprovechando que algunos de ellos pueden estar en orden, es buscar subrangos en orden y hacer la operación *merge* entre ellos. Más exactamente:

1. Incrementar un primer contador mientras se encuentre una secuencia creciente de elementos. El contador para en el primer elemento que cumpla que es menor que su predecesor.
2. A partir de la posición en donde termino el recorrido anterior, iniciar un segundo contador y hacer un recorrido identificando un segundo rango en orden.
3. A los dos rangos anteriores hacerles la operación merge. Luego del merge volver al paso 2 y repetir hasta organizar todo el vector.

Se pide entonces:

- Dar la implementación del algoritmo *Natural Mergesort*.
- Estimar el tiempo (# de comparaciones) en función del número de secuencias crecientes en el arreglo.
- Cuál sería el peor caso para este algoritmo. Cómo sería el tiempo en este caso.

3. *Aleatorización de una lista enlazada*

La estrategía “Divide y Vencerás” es frecuentemente utilizada para diseñar algoritmos. Como ejercicio para familiarizarse con esta estrategía se propone aleatorizar los elementos de una lista ordenada. Este tipo de operaciones son frecuentemente utilizadas en los programas (e.g. barajar un juego de cartas).

Se pide:

- Dar una implementación del metodo *shuffle* para listas enlazadas haciendo uso de la estrategía divide y vencerás.
- Estimar el tiempo (# de comparaciones, # de intercambios) requeridos por el algoritmo propuesto en función de la longitud de la lista N.
- (* Opcional) Mejorar el desempeño del algoritmo propuesto garantizando que el tiempo es linearitmético y el espacio adicional es logaritmico.

4. Algoritmos de ordenación no recursivos

Un principio fundamental de la programación establece que la recursión y la iteración son equivalentes: Cualquier programa recursivo se puede hacer iterativo o viceversa.

- (*) Dar una implementación iteractiva de mergesort
- (*) Dar una implementación iterativa de quicksort. Es la implementación dada un algoritmo *in-situ*?

5. Se tiene un arreglo genérico ordenado de menor a mayor. Convertir eficientemente el arreglo a orden mayor a menor. Reto adicional: Hacer el proceso *in-situ*.

6. Los algoritmos *naïve* para encontrar la intersección/diferencia de dos conjuntos son algoritmos $\sim N^2$. Si los conjuntos se encuentran en orden, es posible resolver estos problemas de forma más eficiente:

- Dar un algoritmo para encontrar la intersección de dos arreglos ordenados. Determinar el orden de crecimiento del algoritmo.
- Dar un algoritmo para encontrar la diferencia de dos arreglos ordenados. Determinar el orden de crecimiento del algoritmo.