Algoritmos y Programación II (75.41, 95.15) – Curso Buchwald –

3.ra fecha de examen final – 18/07/2019

1. Dados dos arreglos ordenados y sin repetidos `A` y `B`, donde `B` *es igual a* `A` *pero con un elemento menos*, implementar un algoritmo de División y Conquista que permita encontrar el elemento faltante de `A` en `B`, en $\mathcal{O}(\log n)$ (siendo n la cantidad de elementos en dichos arreglos). Ejemplo: `A = {2, 4, 6, 8, 9, 10, 12}`, `B = {2, 4, 6, 8, 10, 12}`. La salida del algoritmo debe ser el valor 9. Justificar el orden del algoritmo, utilizando el Teorema Maestro.

2. Implementar en C una primitiva que reciba un árbol binario que representa un heap (árbol binario izquierdista, que cumple la propiedad de heap), y devuelva la representación en arreglo del heap. La firma de la primitiva debe ser `void** ab_repr_arreglo(const ab_t*)`. Indicar y justificar el orden de la primitiva. La estructura del árbol binario es:

```c
typedef struct ab {
    struct ab* izq;
    struct ab* der;
    void* dato;
} ab_t;
```

3. Queremos comparar el heap `d`-ario (en el que cada nodo tiene hasta `d` hijos) con el heap binario común, que tiene hasta 2 hijos.

   a. ¿Cómo se representa un heap `d`-ario en un arreglo? ¿Cómo calculo el padre de un nodo, o los `d` hijos de un nodo?
   b. ¿Cuál es la altura de un heap `d`-ario con `n` elementos, en términos de `n` y `d`?
   c. Dar una implementación eficiente de encolar (para un heap `d`-ario de máximos). Analizar en términos de `n` y `d` su eficiencia. Definir si tiene sentido la implementación de un heap `d`-ario por sobre uno binario.

4. Se tiene un laberinto compuesto por casilleros. En dicho laberinto se permite el movimiento en cualquier dirección, pero moverse de un casillero a otro puede producir **una cantidad de cansancio diferente**. Explicar cómo se puede modelar este problema con un grafo. Diseñar un algoritmo que nos permita llegar desde un casillero inicial hasta el casillero final (salida, de la cual conocemos ya su posición), con el menor cansancio posible. ¿Cambiaría la elección del algoritmo si el cansancio fuera el mismo en cualquier dirección e igual para todos los casilleros? Si es así, explique por qué haría dicho cambio, y si no es así, explique por qué no resulta conveniente o necesario.

5. Implementar un algoritmo que, por **backtracking**, dado un Grafo dirigido y dos vértices s y t, nos devuelva todos los caminos simples para ir desde s a t en el grafo. Se recomienda pensar en un algoritmo similar a un DFS.

Algoritmos y Programación II (75.41, 95.15) – Curso Buchwald –

3.ra fecha de examen final – 18/07/2019

1. Dados dos arreglos ordenados y sin repetidos `A` y `B`, donde `B` *es igual a* `A` *pero con un elemento menos*, implementar un algoritmo de División y Conquista que permita encontrar el elemento faltante de `A` en `B`, en $\mathcal{O}(\log n)$ (siendo n la cantidad de elementos en dichos arreglos). Ejemplo: `A = {2, 4, 6, 8, 9, 10, 12}`, `B = {2, 4, 6, 8, 10, 12}`. La salida del algoritmo debe ser el valor 9. Justificar el orden del algoritmo, utilizando el Teorema Maestro.

2. Implementar en C una primitiva que reciba un árbol binario que representa un heap (árbol binario izquierdista, que cumple la propiedad de heap), y devuelva la representación en arreglo del heap. La firma de la primitiva debe ser `void** ab_repr_arreglo(const ab_t*)`. Indicar y justificar el orden de la primitiva. La estructura del árbol binario es:

```c
typedef struct ab {
    struct ab* izq;
    struct ab* der;
    void* dato;
} ab_t;
```

3. Queremos comparar el heap `d`-ario (en el que cada nodo tiene hasta `d` hijos) con el heap binario común, que tiene hasta 2 hijos.

   a. ¿Cómo se representa un heap `d`-ario en un arreglo? ¿Cómo calculo el padre de un nodo, o los `d` hijos de un nodo?
   b. ¿Cuál es la altura de un heap `d`-ario con `n` elementos, en términos de `n` y `d`?
   c. Dar una implementación eficiente de encolar (para un heap `d`-ario de máximos). Analizar en términos de `n` y `d` su eficiencia. Definir si tiene sentido la implementación de un heap `d`-ario por sobre uno binario.

4. Se tiene un laberinto compuesto por casilleros. En dicho laberinto se permite el movimiento en cualquier dirección, pero moverse de un casillero a otro puede producir **una cantidad de cansancio diferente**. Explicar cómo se puede modelar este problema con un grafo. Diseñar un algoritmo que nos permita llegar desde un casillero inicial hasta el casillero final (salida, de la cual conocemos ya su posición), con el menor cansancio posible. ¿Cambiaría la elección del algoritmo si el cansancio fuera el mismo en cualquier dirección e igual para todos los casilleros? Si es así, explique por qué haría dicho cambio, y si no es así, explique por qué no resulta conveniente o necesario.

5. Implementar un algoritmo que, por **backtracking**, dado un Grafo dirigido y dos vértices s y t, nos devuelva todos los caminos simples para ir desde s a t en el grafo. Se recomienda pensar en un algoritmo similar a un DFS.