7541 - 20201C - 3er Final

EJERCICIO 1 | EJERCICIO 2 | EJERCICIO 3 | EJERCICIO 4 | EJERCICIO 5

Version 1

1. Explique como es que se pueden comparar dos algoritmos sin tener en cuenta el Hardware en el que estan siendo ejecutados. ¿Cómo es posible que se puedan comparar?. Ponga un ejemplo.
2. Ordene en orden creciente de complejidad los algoritmos con las siguientes ecuaciones de recurencia asociadas:

   a. T(n)= 4T(n/4)+n^2

   b. G(n)= 16T(n/4) + n

   c. F(n)=3T(n/2)+ n^2

7541 - 20201C - 3er Final

Version 1

Dada la siguiente definición de *lista doblemente enlazada* recursiva:

```
typedef struct lista{
    struct lista* siguiente;
    struct lista* anterior;
    void* elemento;
}lista_t;
```

Implemente la siguiente función de forma completamente recursiva (**no se admiten for/while/do, etc**).

```
lista_t* lista_insertar(lista_t* lista, void* elemento);
```

Nota: A los efectos de este ejercicio, puede asumir que malloc/calloc nunca fallan.

Version 2

Dada la siguiente definición de *lista doblemente enlazada* recursiva:

```
typedef struct lista{
    struct lista* siguiente;
    struct lista* anterior;
    void* elemento;
}lista_t;
```

Implemente la siguiente función de forma completamente recursiva (**no se admiten for/while/do, etc**).

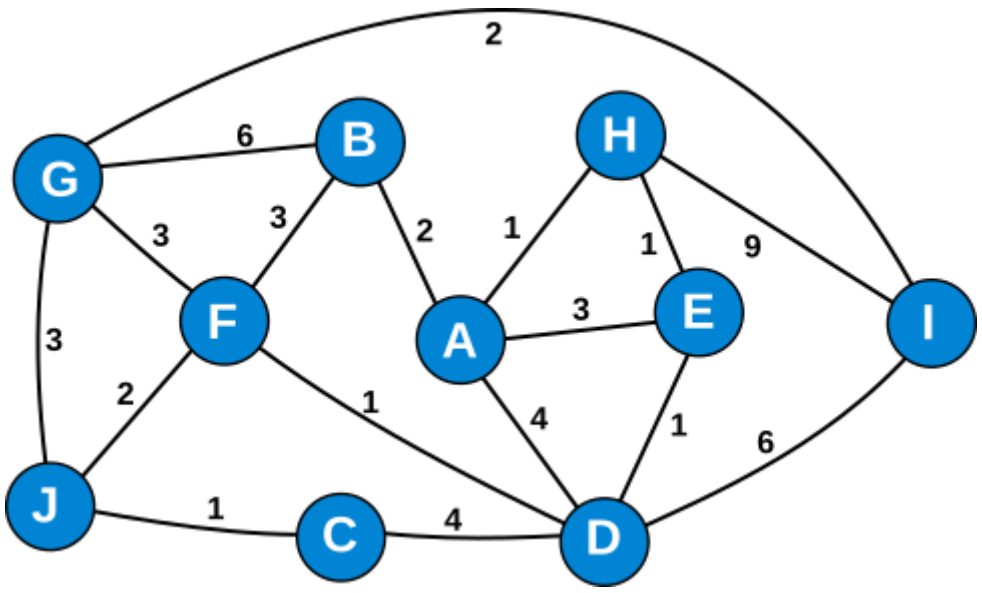
```
lista_t* lista_eliminar(lista_t* lista, void* elemento, int (*comparador)(void*,void*));
```

Nota: A los efectos de este ejercicio, puede asumir que malloc/calloc nunca fallan.

7541 - 20201C - 3er Final

Version 1

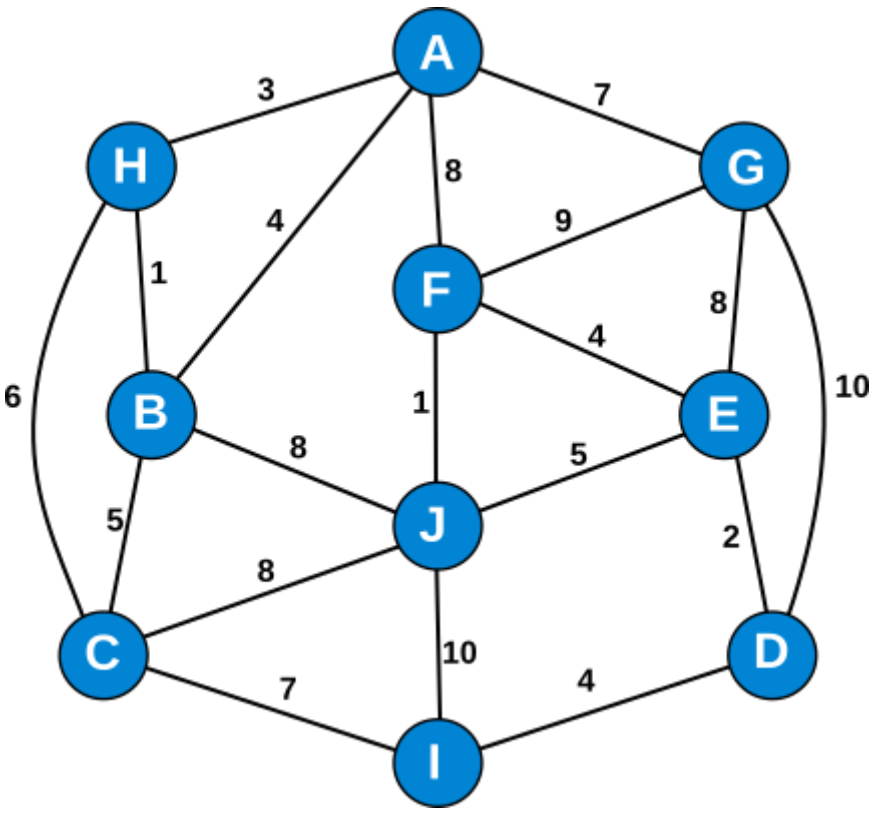
Dado el siguiente grafo:



1. Aplique **Dijkstra** desde **A** hasta **I** mostrando la tabla a cada paso.
2. Aplique **DFS**, empezando por el vértice **A**, explicando el procedimiento.

Version 2

Dado el siguiente grafo:



1. Aplique **Kruskal** explicando el procedimiento.
2. Aplique **BFS**, empezando por el vértice **F**, explicando el procedimiento.

7541 - 20201C - 3er Final

Version 1

1. Justifique si la siguiente **función de hashing** es buena o no para ser utilizada en una tabla hash:

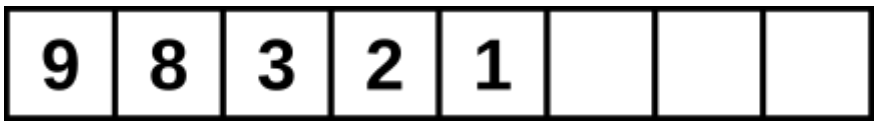
```c
size_t funcion_hash(const char* string){
   if(!string || !*string) return 0;

   size_t acumulado = 0;

   for(size_t i=0;string[i];i++)
       acumulado = acumulado + string[i]*(i+1);

   return acumulado/(size_t)string;
}
```

2. Dado el siguiente **Heap binario**:



Insertar los valores **4** y **7** y luego eliminar 2 veces la raíz. Muestre el estado del heap luego de cada operación y justifique.

Version 2

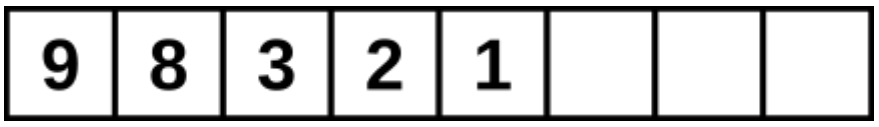
1. Justifique si la siguiente **función de hashing** es buena o no para ser utilizada en una tabla hash:

```c
size_t funcion_hash(const char* string){
   if(!string || strlen(string)<3) return 0;

   size_t factor1 = string[0];
   size_t factor2 = string[1];
   size_t factor3 = string[2];

   return ((factor1*NUMERO_MAGICO)+(factor2*factor3))/FACTOR_NORMALIZADOR;
}
```

2. Dado el siguiente **Heap binario**:



Insertar los valores **4** y **7** y luego eliminar 2 veces la raíz. Muestre el estado del heap luego de cada operación y justifique.

Version 3

1. Justifique si la siguiente **función de hashing** es buena o no para ser utilizada en una tabla hash:

```c
size_t funcion_hash(const char* string){
   if(!string || !*string) return 0;

   size_t acumulado = 0;

   for(size_t i=0;string[i];i++){
       for(size_t j=0;string[j];j++)
           acumulado = acumulado + i + string[j]/(2*j+1);

       acumulado = acumulado / (i+1);
   }

   return acumulado;
}
```

2. Dado el siguiente **Heap binario**:

1	2	8	3	9			

Insertar los valores **4** y **7** y luego eliminar 2 veces la raíz. Muestre el estado del heap luego de cada operación y justifique.

7541 - 20201C - 3er Final

EJERCICIO 1 | EJERCICIO 2 | EJERCICIO 3 | EJERCICIO 4 | EJERCICIO 5

Version 1

Dados los siguientes recorridos de un **ABB** cuya forma es desconocida:

Inorden: `!` , `%` , `=` , `#` , `@` , `&` , `$` , `+`

Preorden: `#` , `!` , `=` , `%` , `@` , `$` , `&` , `+`

Reconstruya el **ABB** y explique el procedimiento.

Nota: Tiene todos los datos necesarios en los recorridos dados. No es necesaria ninguna conversion de ningún tipo y no hay ningún significado escondido en los símbolos.

Version 2

Dados los siguientes recorridos de un **ABB** cuya forma es desconocida:

Inorden: `!` , `%` , `=` , `#` , `@` , `&` , `$` , `+`

Preorden: `&` , `%` , `!` , `=` , `#` , `@` , `$` , `+`

Reconstruya el **ABB** y explique el procedimiento.

Nota: Tiene todos los datos necesarios en los recorridos dados. No es necesaria ninguna conversion de ningún tipo y no hay ningún significado escondido en los símbolos.

Version 3

Dados los siguientes recorridos de un **ABB** cuya forma es desconocida:

Inorden: `!` , `%` , `=` , `#` , `@` , `&` , `$` , `+`

Preorden: `&` , `=` , `%` , `!` , `@` , `#` , `+` , `$`

Reconstruya el **ABB** y explique el procedimiento.

Nota: Tiene todos los datos necesarios en los recorridos dados. No es necesaria ninguna conversion de ningún tipo y no hay ningún significado escondido en los símbolos.
