Algoritmos y Estructuras de Datos

Examen Parcial
Pregrado
2020-1

Indicaciones específicas:

Duración: 90 minutos
Número de preguntas: 5

- Se permite el uso de calculadoras, y hojas en blanco de ayuda.
- No se permitirá el uso de laptops, notas o libros.
- Lea las preguntas cuidadosamente y responda de manera clara. Respuestas que no sean legibles o claras no tendrán ningún puntaje.
- Si no sabes la respuesta, deja el espacio en blanco y coloca "no sé la respuesta". Se concederá el 10% del puntaje. (Recuerden de todas formas resolver las preguntas que no entendieron luego de la prueba)

Pregunta 1 (4 puntos) (habilidad b, c)

A. Dada la siguiente estructura Node de una lista simplemente enlazada (2 pts)

```
template <typename T>
struct Node {
        T data;
        Node<T>* next;
};
```

Implementa una función reverseLinkedList que revierta una lista simplemente enlazada en O(n). No puedes crear y retornar una lista nueva, debe ser la misma lista la que sea revertida. Ni usar estructuras de apoyo, solo variables temporales (memoria O(1)).

Por ejemplo:

Input: 10 -> 5 -> 17 -> 2 -> nullptr
Output: 2 -> 17 -> 5 -> 10 -> nullptr

```
template<typename T>
void reverseLinkedList(Node<T>** head) {
```

no se la respuesta

B. Dado un arreglo de números, encuentre y retorne el sub-arreglo contiguo con suma máxima O(n). (2 pts)

Por ejemplo:

Input: [-2 1, -3, 3, -2, 2, 1, -5, 4]
Output: [3, -2, 2, 1]

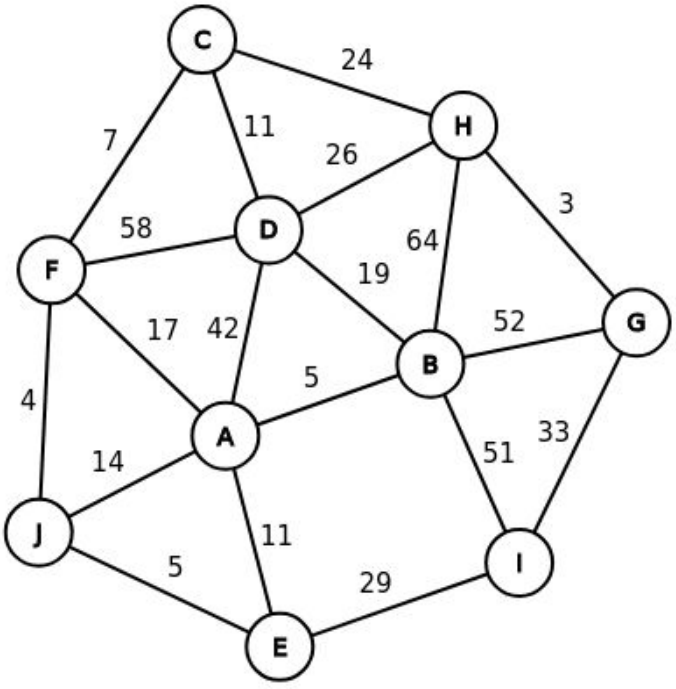
Input: [-2, 4, -3, 5, 1, -1, 2, -5, 4]
Output: [4, -3, 5, 1, -1, 2]

int* **findMaxSubArray**(int arr[], int size) {

no se la respuesta

Pregunta 2 (5 puntos) (habilidad a, b, c)

Árbol de expansión mínima (MST) Considere el siguiente grafo de 10 vértices y 19 aristas



A. **Árbol de expansión máxima** es el árbol de expansión con el mayor peso posible (contrario al árbol de expansión mínima). Indique cómo se podrían usar los algoritmos de Prim y Kruskal **(sin modificarlos)** sobre un grafo *g* para poder encontrarlo (1 pts)

Para obtenerlo aplicando los algoritmos tal cual, se podría realizar una copia del grafo ponderando los pesos de las aristas de manera inversa. Se realiza una lista de estos pesos asignados a una posición en específico (para relacionarlo con el grafo) y se realiza otra lista invirtiendo esta lista. Se reemplazan estos pesos en el nuevo grafo dependiendo de la posición asociada en la primera lista creada. Al realizar el algoritmo, dará la secuencia de aristas del árbol de expansión máxima que será correspondiente al grafo original.

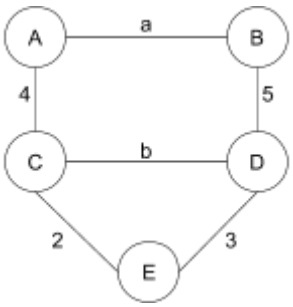
B. Ubique debajo la secuencia de aristas del **árbol de expansión máxima** en el **orden** en que el algoritmo de **Kruskal** las selecciona. Recuerde la notación de las aristas, por ejemplo: *{G, B}, {B, I},...* (1.5 pts)

{H,B}, {F, D}, {B, G}, {B, I}, {D, A}, {E, I}, {D, H}, {C,H}, {J, A}

C. Ubique debajo la secuencia de aristas del **árbol de expansión máxima** en el **orden** en que el algoritmo de **Prim** las selecciona. Empiece por el vértice A. Recuerde la notación de las aristas, por ejemplo: *{G, B}, {B, I},...* (1.5 pts)

{A, D}, {F, D}, {D, H}, {H, B}, {B, G}, {B, I}, {E, I}, {C, H}, {J, A}

D. **Dado el siguiente grafo, sabiendo que {{A, C}, {C, E}, {E, D}, {D, B}} conforman un árbol de expansión mínima y que el peso de las aristas son de tipo entero.** ¿Cuáles son los posibles valores que al menos pueden tomar las variables a y b? Justifique su respuesta para *PRIM* (empiece por A) y *KRUSKAL*: (1 pt)



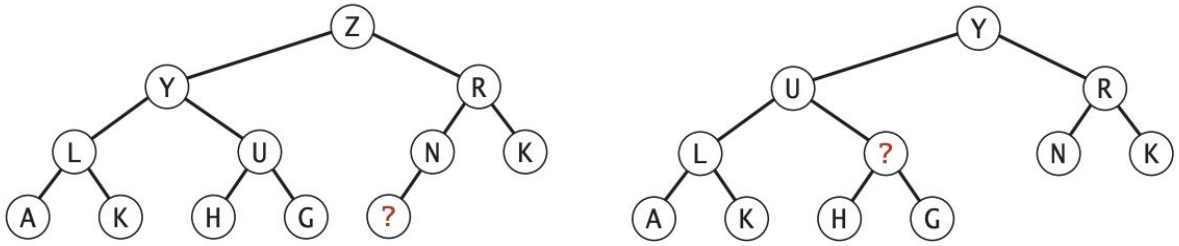
Prim: se escoge {A, C} se asume que 'a' será mayor a 4 (tomando en cuenta la nota). Luego, se va por el camino {C, E}: se asume que 'b' es mayor a 2. Luego, {E, D} con lo cual se concluye que será mayor a 3. Debe escogerse el camino {A, B}, {D, B} o {D, C} para acabar; pero {D, C} forma un ciclo, por eso es que se asume que 'b' será mayor a 3. Se toma {B, D} en lugar de {A,B}: se concluye que 'a' es mayor a 5.

Kruskal: se toma inicialmente {C, E} y {E,D}; se concluye que 'b' debe ser mayor a 3 y por ello no fue escogido ese camino. Se toma {A, C}. Podría tomarse {A, B} o {B, D}, porque ambas son opciones para el MST, pero se toma {B, D}; por ello 'a' será mayor a 5 (tomando en cuenta la Nota)

Nota: Debe ser el único MST posible (mismas aristas).

Pregunta 3 (4 puntos) (habilidad a, b, c)

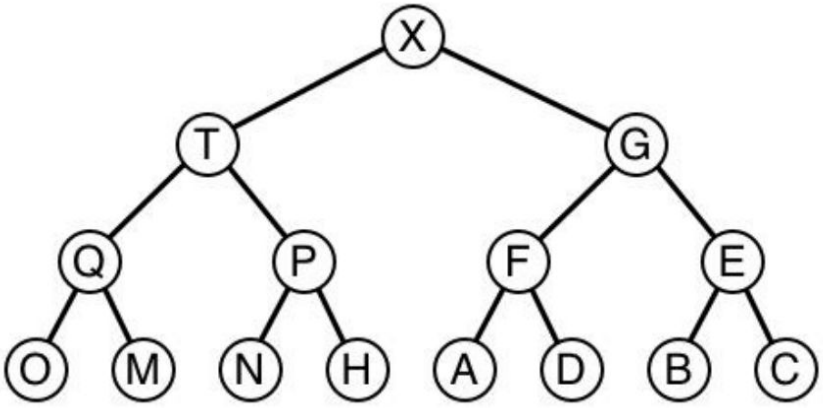
A. Dada la siguiente representación de un max-heap: (2 pts)



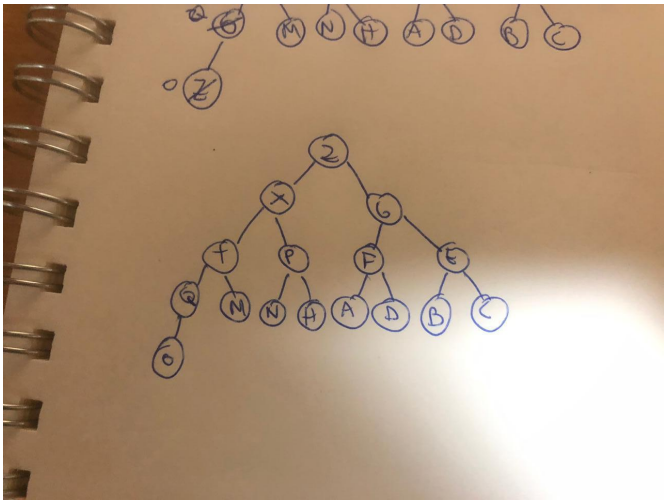
Al borrar un elemento del heap de la izquierda resulta en el heap de la derecha. Cuáles de las siguientes keys, podrían estar en la posición del símbolo de pregunta:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

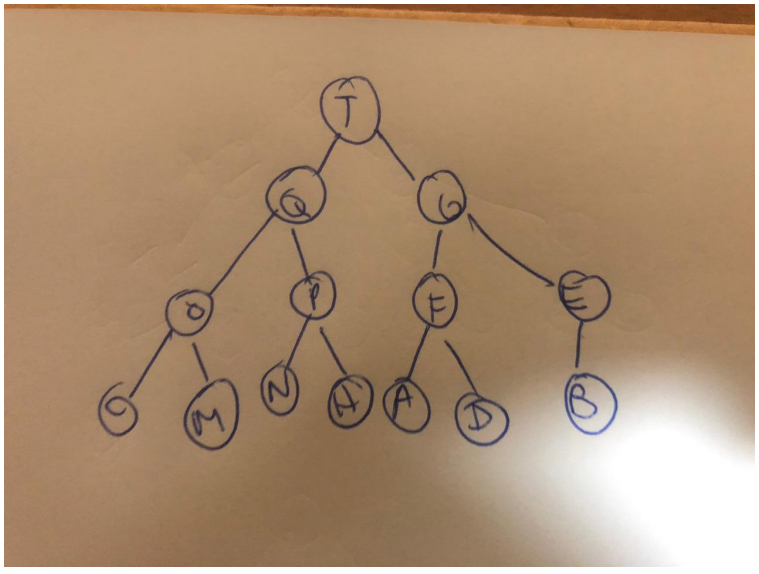
B. Dada la siguiente representación de un max-heap: (2 pts)



Dibuje el resultado de insertar Z



Dibuje el resultado de borrar el máximo elemento del max-heap original **(antes que Z fuera insertado)**



Pregunta 4 (4 puntos) (habilidad a)

Escribe lo que imprimirá cada programa (en caso que sea una dirección de memoria, colocar Ax01, AX02, ...) o explica en caso genere un error:

A.
```
int main(int argc, char *argv[]) {
        int n[6] = {0, 0, 0, 0, 0, 0};
        int size = sizeof(n) / sizeof(int);
        int *p = n;
        *p = 12;
        p++;
        *p = 8;
        *(p++) = size;
        *(++p) = (6 * *p) + 1;
        p = n + 4;
        *p = 24;
        p = &n[4];
        *p = 4;
        p = n;
        *(p + 5) = 36;

        for (int i = 0; i < size; i++) {
                cout << n[i] << " ";
        }

        return EXIT_SUCCESS;
}
```
0 6 1 36 0

B.
```
int main(int argc, char *argv[]) {
        int *p, n;
        n = 23;
        *p = n;
        cout << *p << endl;

        return EXIT_SUCCESS;
}
```
error, *p tendría que igualarse a la dirección de memoria de n (&n) o realizarse un new

C.
```
struct Node {
        int data;
        Node* next;
};

int main(int argc, char *argv[]) {
        Node* temp1, temp2;
        temp2.data = 10;
        *temp1 = temp2;
        cout << temp1->data << endl;

        return EXIT_SUCCESS;
}
```
error, *temp1 no puede ser igualado a temp2 temp1 es un puntero a Node y temp 2 es un nodo. Tendría que igualarse a &temp2

D.
```
void func(int **ptr) {
        static int number = 5;
        *ptr = &number;
         number++;
}

int main(int argc, char *argv[]) {
        int q = 10;
        int *ptr = &q;
        func(&ptr);
        func(&ptr);
        cout << *ptr << endl;

        return EXIT_SUCCESS;
}
```

no se la respuesta

Pregunta 5 (3 puntos) (habilidad a)

Escriba respuestas cortas. Respuestas largas no recibirán ningún puntaje.

1. Cuál es la diferencia entre los algoritmos de Prim y Kruskal?

   El algoritmo Kruskal funciona para grafos no conexos mientras Prim no, porque si es que el grafo es así, el algoritmo Prim no podría llegar a todos los nodos ya que hay uno(s) que no serán adyacentes

2. Qué es un algoritmo voraz? Liste sus propiedades

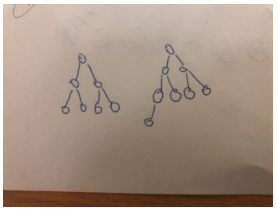
   - No realiza backtracking
   - Toma una decisión para generar una 'buena' solución local, con lo cual se espera que sea la mejor global pero no necesariamente asegura ello
   - Su finalidad es arrojar una respuesta con un tiempo mínimo, por ello no necesariamente dará la más óptima de todas

3. Qué es un grafo disperso? De un ejemplo

   Un grafo disperso es aquel que tiene un número de aristas pequeño. Por lo general responde a ser menor que una cota que evidencie la densidad; por ello se puede determinar el número de aristas como 'pocas aristas'. Un ejemplo de ello por lo general podría ser una red social con muchos usuarios

4. Qué es un árbol binario balanceado? De dos ejemplos

   Un árbol balanceado es aquel al cual si restas la altura de los hijos izquierdo y derecho, el número resultante debe ser igual a 1 o a 0



5. **Social networking** (1 pt.)
   Supón que una página de red social llamada AMIGOS, necesita soportar dos operaciones: (i) definir que A y B son amigos (por tanto, hacer todos los amigos de A y B, amigos entre ellos); y (ii) determinar si A y B son amigos:

   Qué APIs deberían usar en la red AMIGOS para poder soportar esas operaciones (marque 2)?

   A. Queue.
   B. Union-find.
   C. Stack.
   D. Priority queue.
   E. Symbol table. (e.g. map)
   F. Binary Heaps. (e.g. max & min)

   En una o dos líneas, justifique sus respuestas (describa como AMIGOS deberían implementar/usar las dos operaciones)

Amigos es un grafo no dirigido; cada usuario es un vértice y cada relación de amistad es una arista. La operación i debe crear aristas entre aquellos que tienen una relación con A, con B y todos aquellos que tienen una relación con B. La operación ii verifica si es q el grafo es completo.