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) {
```

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

Por ejemplo:

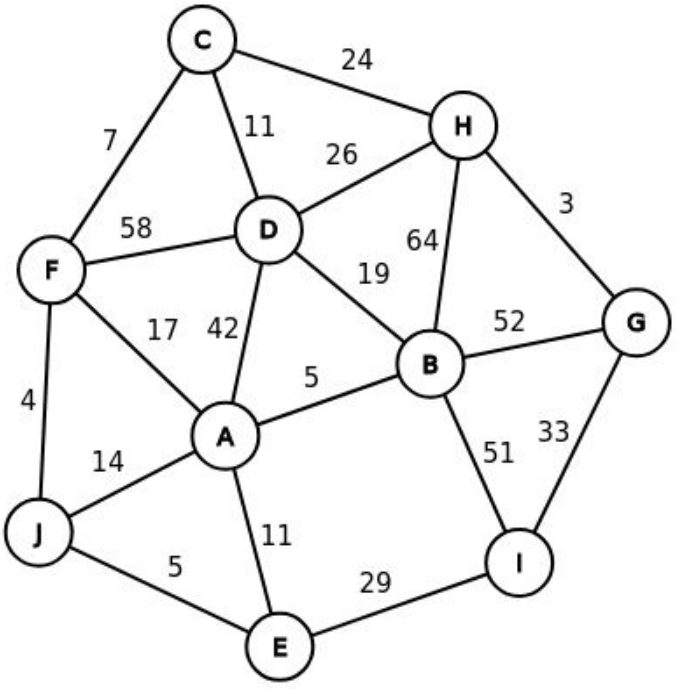
Input: [-2 1, -3, 3, -2, 2, 1, -5, 4]
Output: [3, -2, 2, 1]

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

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

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

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

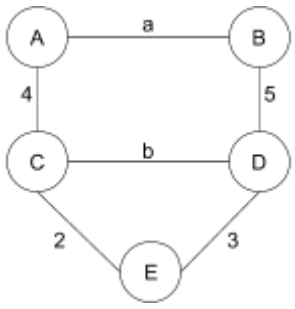


A. **Árbol de expansión máxima** es el árbol de expansión con el mayor peso posible (contrario al árbol de expansión mínima). Indique cómo se podrían usar los algoritmos de Prim y Kruskal **(sin modificarlos)** sobre un grafo *g* para poder encontrarlo (1 pts)

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)

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)

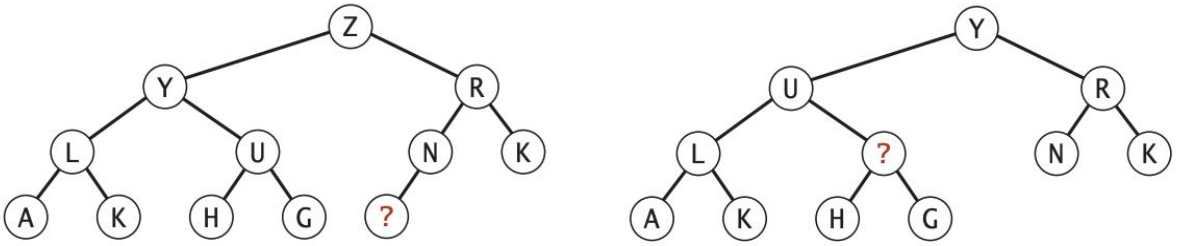
D. **Dado el siguiente grafo, sabiendo que {{A, C}, {C, E}, {E, D}, {D, B}} conforman un árbol de expansión mínima y que el peso de las aristas son de tipo entero.** ¿Cuáles son los posibles valores que al menos pueden tomar las variables a y b? Justifique su respuesta para *PRIM* (empiece por A) y *KRUSKAL*: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



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

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

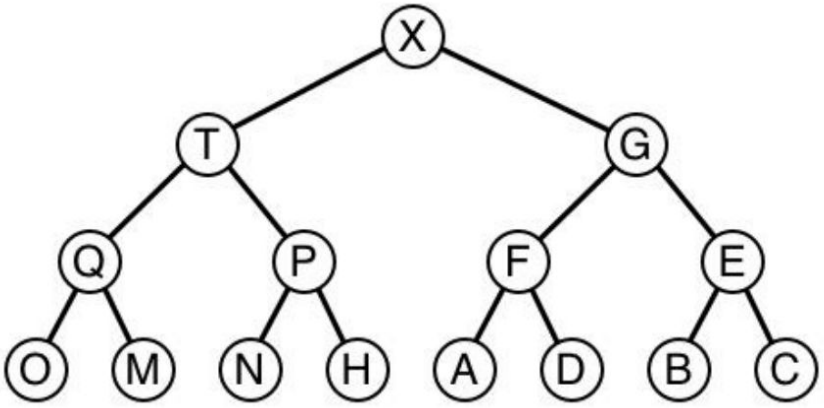
A. Dada la siguiente representación de un max-heap: (2 pts)



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;
}
```

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

        return EXIT_SUCCESS;
}
```

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;
}
```

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;
}
```

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?

2. Qué es un algoritmo voraz? Liste sus propiedades

3. Qué es un grafo disperso? De un ejemplo

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

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)