Algoritmos y Estructuras de Datos

Examen Parcial
Pregrado
2019-2

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 (6 puntos) (habilidad b, c)

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

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

Escriba una función rearrenge con la siguiente firma: **"*void rearrenge(Node* head)*"** que reordene una lista simplemente enlazada de una manera específica como se muestra en los ejemplos. Los elementos de la primera mitad de la lista se irán intercalando con los elementos de la segunda mitad empezando del final. El tiempo de ejecución debe ser O(n) y el espacio constante. Por tanto no pueden utilizar estructuras de apoyo.

Por ejemplo,

Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6
Output: 1 -> 6 -> 2 -> 5 -> 3 -> 4

Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
Output: 1 -> 7 -> 2 -> 6 -> 3 -> 5 -> 4

void rearrenge(Node* head) {

}

B. **Haciendo uso de la misma estructura presentada en la pregunta anterior** (2 pts)

Escriba una función findKthFromTheEnd con la siguiente firma: ***"int getKthFromtheEnd(Node* head, int k)".*** Donde dada una lista simplemente enlazada y una posición k, se retorne el elemento en la posición k empezando desde el final. O(n), se tomará eficiencia en la calificación.

Por ejemplo:

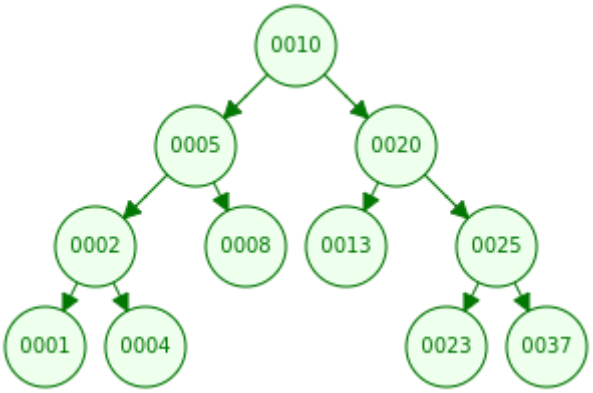
Input: 8 -> 7 -> 3 -> 1 -> 9 -> 5 k = 2
Output: 9

int getKthFromtheEnd(Node* head, int k)

C. Dado un árbol binario, con la siguiente estructura Node: (2 pts)

```
struct Node {
        int data;
        Node* left;
        Node* right;
};
```

Implemente la función print con la firma: ***"void print(Node *root)"***. Que imprima los nodos al extremo de cada nivel:



Output:
10
5 20
2 25
1 37

- Si, el nodo con valor 37 no estuviera, entonces imprimiría:

Output:
10
5 20
2 25
1 23

Pregunta 2 (2 puntos) (habilidad a)

1. Qué es un grafo disperso? De un ejemplo

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

3. Supongamos que tenemos los números del 1 al 100 en un árbol binario de búsqueda y queremos buscar el número 44. Cuál de las siguientes opciones (puede ser más de una) podrían ser secuencias de nodos examinados?

   I. 5, 2, 1, 10, 39, 34, 77, 63
   II. 1, 2, 3, 4, 5, 6, 7, 10
   III. 9, 8, 63, 0, 4, 3, 2, 1
   IV. 8, 7, 6, 5, 4, 3, 2, 1
   V. 50, 25, 26, 27, 40, 43, 42

VI. 50, 25, 26, 27, 40, 45

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

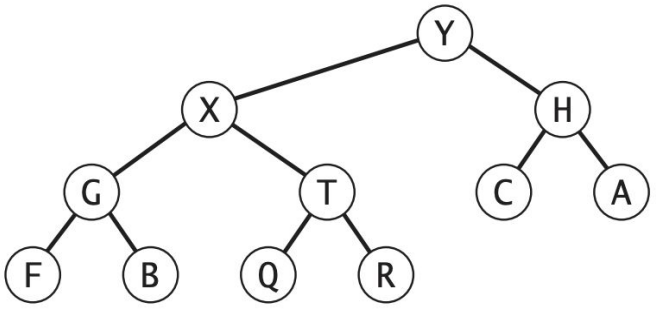
A. Dibuje el árbol binario resultante después de realizar las siguientes operaciones **(los cambios deben realizarse con el elemento anterior, NO con el siguiente)**: (2 pts)

Insertar: 67, 23, 48, 57, 36, 64, 51, 78, 51, 43, 81, 26, 32, 72
Remover: 51, 23, 67

Escriba el orden de los nodos visitados en recorrido:

1. In-order:
2. Pre-order:
3. Post-order:

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



De el arreglo que representaría el heap

0	1	2	3	4	5	6	7	8	9	10
Y	X	H	G	T	C	A	F	B	Q	R

Inserte el valor P en el max-heap y complete los valores del nuevo arreglo representante

0	1	2	3	4	5	6	7	8	9	10	11
Y	X	P	G	T	H	A	F	B	Q	R	C

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:

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

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

falta *

B.
```
void fun(Node* head) {
        if (head== NULL) {
                return;
        }

        printf("%d  ", head->data);

        if (head->next != NULL ) {
           fun(head->next->next);
        }

        printf("%d  ", head->data);
}
```

1 5 8 8 5 3 1

Para la Linked List: 1->2->3->4->5->6->8->7

C.
```
int q = 10;

void function(int *p) {
        p = &q;
}

int main(int argc, char *argv[]) {
        int r = 4;
        int *p = &r;
        function(p);
        cout << *p << endl;
        return EXIT_SUCCESS;
}
```

4

D.
```
int main(int argc, char *argv[]) {
        int n = 6;
        cout << ++n << " ";
        cout << n << " ";
        cout << n++ << " ";
        return EXIT_SUCCESS;
}
```

7 7 7

Pregunta 5 (4 puntos) (habilidad b, c)

Dada la siguiente estructura **Node** y clase **DisjointSet**:

```
struct Node {
        int data;
        int rank;
        Node* parent;

        Node(int data) : data(data), rank(0), parent(this) {};
};

class DisjointSet {
        map<int, Node*> nodes;

        public:
                ...
}
```

Implemente dentro de la clase DisjointSet los métodos **makeSet**, **link** y **find**. El *map nodes* representa el universo de los conjuntos donde la key (int) es la data. Debe utilizar path compression en la operación find.