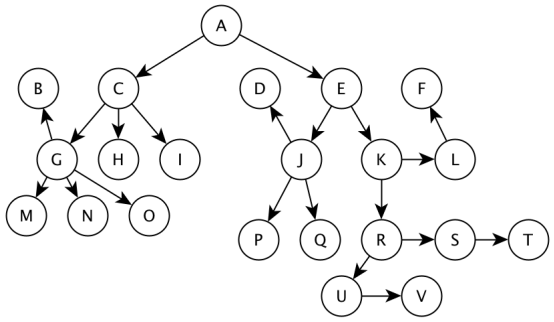
IIC2115 – Programación como Herramienta para la Ingeniería (I/2019)

Ejercicios Capítulo 2

1. La información de una empresa se encuentra almacenada en el siguiente grafo que tiene a A como nodo raíz.



   Se le pide verificar si las funciones desarrolladas para acceder a ciertos datos es correcta.

   a) Si se implementa un método *BFS* para encontrar al nodo R. ¿Cuál es una posible secuencia de visitas previas a encotrar el nodo R?

      i. A - C - E - J - P - Q - K - L - F - R

      ii. A - E - J - P - Q - K - L - F - R

      iii. A - E - C - J - K - G - H - I - D - P - Q - R

      iv. A - C - E - K - R

   b) Si se implementa un método *Pre-order traversal* para encontrar al nodo N. ¿Cuál es una posible secuencia de visitas previas a encotrar el nodo N?

      i. A - C - E - B - G - H - I - D - J - K - M - N

      ii. A - C - G - B - H - I - O - M - N

      iii. A - E - J - D - C - G - E

      iv. A - C - H - I - G - O - N

2. Como parte de los laboratorios de un curso de programación, los alumnos deben implementar un programa que tome código escrito en Python y verifique la correcta sintáxis de este. En caso de encontrar errores, el programa debe informárselo al usuario, indicando la línea donde se encontró.

a) El primer proceso a realizar por el programa consiste en almacenar la línea en que se **define** cada variable (para luego poder encontrarla rapidamente), y verificar que no haya uso de variables que no hayan sido previamente definidas. ¿Qué estructura de datos recomienda para facilitar este proceso?

   i. Cola

   ii. Set

   iii. Stack

   iv. Diccionario

b) ¿Cuál es la estructura de datos más adecuada para verificar si cada paréntesis abierto en una línea tiene su correspondiente cierre?

   i. Cola

   ii. Set

   iii. Stack

   iv. Diccionario

3. Marque la opción más correcta en las siguientes preguntas respecto a la materia del curso:

a) ¿Cuál es la diferencia entre una Lista y una Tupla?

   i. La tupla almacena los elementos de una lista pero sin repetición

   ii. La tupla posee una cantidad máxima de elementos, mientras que la lista puede contener tantos como se quiera

   iii. La tupla no puede ser modificada una vez creada, mientras que la lista siempre puede modificarse

   iv. No existe diferencia significativa entre estas dos estructuras de datos

b) En una matriz incidencia ¿Qué representan las filas, las columnas y los valores de las celdas?

   i. Las filas y columnas representas nodos y cada valor en la matriz indica si el par de nodos se encuentran conectados directamente (por un arco)

   ii. Posee solo una columna que almacena las conexiones disponibles entre nodos y arcos

   iii. Las filas representan los nodos y las columnas representas los arcos. Cada valor en la matriz representa si existe contacto entre el arco y nodo

   iv. Las filas y columnas representas nodos y cada valor en la matriz indica el coste existente en cada conexión

4. ¿Cuál de las siguientes es la estructura de datos más adecuada para modelar la cola de un supermercado, donde un cliente puede retirarse de la cola en cualquier posición de esta?

   a) cola.

   b) diccionario.

   c) stack.

   d) lista ligada.

5. ¿En cuál de las siguientes estructuras se necesitan menos pasos para obtener el elemento buscado?

   a) árbol binario.

   b) diccionario.

   c) stack.

   d) lista ligada.

6. Si **G** es un grafo dirigido con 20 vértices, ¿cuántos coeficientes debe tener su matriz de adyacencia?

   a) 20.

   b) 40.

   c) 200.

   d) 400.

7. ¿Cuál es el tamaño de una matriz de incidencia que representa un árbol binario completo de profundidad n?

   a) $2^n - 1 \times 2^n - 2$.

   b) $2^n - 1 \times 2^{n+1} - 2$.

   c) $2^{n+1} - 1 \times 2^n - 2$.

   d) $2^{n+1} - 1 \times 2^{n+1} - 2$.

8. ¿Para cuál de lo siguientes problemas es un *stack* una estructura adecuada?

   a) verificación de paréntesis balanceados en una fórmula.

   b) orden de atención de los clientes de un supermercado.

c) almacenar amistades en una red social.

d) todas las anteriores.

9. La profundidad máxima de un árbol binario estricto (cada nodo tiene cero o dos hijos) con 6 nodos es:

   a) 2

   b) 3

   c) 4

   d) no existen árboles binarios estrictos con 6 nodos.

10. La notación *infix* es la notación matemática que generalmente se usa. En ella, los operadores se escriben entre sus operandos. Existen otras notaciones. Una de ellas es la notación *postfix*, que ha sido muy usada por Hewlett-Packard para hacer más eficientes algunos cálculos, en la que se escriben los operadores después de sus operandos. Por ejemplo, lo siguiente es la misma expresión escrita usando ambas notaciones:

$$\textit{infix}\colon\ 12 + (3 \times 3.14)$$
$$\textit{postfix}\colon\ 1233.14 \times +$$

Como pueden notar, en la expresión en *postfix* no se puede determinar cuáles son los operandos (e.g., la multiplicación podría interpretarse como 3.1×4 y la suma, como $(3.1 \times 4) + 123$. Por esta razón, es necesario manejar el concepto de *tokens* (o usar paréntesis, pero no serán utilizados para la notación *postfix* en esta actividad).

Los *tokens*, en este contexto, son los elementos de la expresión matemática: los operadores y los operandos. En el ejemplo en *postfix* anterior ($1233.14 \times +$), hay cinco *tokens*: 12, 3, 3.14, $\times$ y $+$.

En este ejercicio deberá implementar un algoritmo que, dados los *tokens* ordenados de una expresión en *postfix*, pueda calcular el valor de tal expresión. Para lograrlo, tendrán que apoyarse particularmente en colas (*queues*) y pilas (*stacks*).

Su algoritmo deberá soportar el cálculo de expresiones que:

- Tengan largo, cantidad de operaciones y cantidad de *tokens* indefinidos.
- No tengan paréntesis.

- Solo tengan números reales como operandos.
- Solo tengan operaciones que pertenezcan a cualquier subconjunto del conjunto de operaciones de suma, resta, multiplicación, división, potencia y factorial.

En su programa, por ejemplo, pueden implementar este algoritmo mediante una función que reciba una lista (o cola) de *tokens* y que evalúe la expresión postfix compuesta de aquellos *tokens*, llamada *evaluate_postfix(tokens)*. A continuación se muestran ejemplos de su uso:

```
evaluate_postfix(["12", "3", "3.14", "*", "+"])  # Retorna 21.42
evaluate_postfix(["4", "2", "**", "2", "3", "**", "/"])  # Retorna 2
evaluate_postfix(["4", "!", "1", "!", "3", "!", "*", "/"])  # Retorna 4
evaluate_postfix(["42", "-33", "-"])  # Retorna 75
```

Pueden usar los símbolos que quieran para representar las operaciones, siempre y cuando usen **al menos dos símbolos para cada operación** (por ejemplo, "-" y "menos" para representar la resta) y los expliciten en un diccionario que relacione cada símbolo con la función que representa. Este diccionario les servirá para realizar las operaciones necesarias en su implementación; porque dado un símbolo, se puede obtener una función que realice la operación correcta, mediante el uso del diccionario.

Algunos módulos de Python útiles para realizar este ejercicio son los siguientes:

(a) Es recomendable que importes el módulo `math`, para que así puedas utilizar la función factorial.

(b) Puedes hacer uso del módulo `operator`, que implementa operadores en forma de funciones.