EJERCICIOS TEMA 4

Jaime Ramírez

2023

Índice

1. EJERCICIOS TEMA 4

1.1. EJERCICIO 1 - TEXTO CODIFICADO

El código comprime dos mensajes utilizando el algoritmo de Huffman y luego los descomprime para mostrar los mensajes originales.

El algoritmo de Huffman es un algoritmo de compresión de datos sin pérdida que se utiliza para comprimir datos de texto, y funciona asignando códigos de bits más cortos a caracteres que ocurren con más frecuencia y códigos de bits más largos a caracteres que ocurren con menos frecuencia.

La función `crear-arbol-huffman` toma un diccionario de frecuencias de caracteres y devuelve un árbol de Huffman, que es una lista anidada que representa el árbol y los códigos asignados a cada caracter.

La función `descomprimir-mensaje` toma un mensaje comprimido y un diccionario de códigos de Huffman y devuelve el mensaje original descomprimido.

La variable `mensaje1` es un mensaje comprimido y se descomprime utilizando la función `descomprimir-mensaje` y el diccionario de códigos `huffman-codes` que se genera a partir del árbol de Huffman. El mensaje descodificado se almacena en la variable `mensaje1-descodificado`.

La variable `mensaje2` es otro mensaje comprimido que se descomprime de la misma manera que `mensaje1`. El mensaje descodificado se almacena en la variable `mensaje2-descodificado`.

La función `calcular-espacio-memoria` está incompleta y no se utiliza en el código proporcionado.

1.2. EJERCICIO 2 - RECURSOS STAR WARS

El código es un ejemplo de asignación de recursos para misiones militares en el universo de Star Wars.

La función asignar-recursos toma tres argumentos: tipo, planeta-destino y general. Dependiendo del tipo de misión, asigna diferentes recursos. Si el general encargado de la misión es Palpatine o Darth Vader, la prioridad de la misión es .ªltaz los recursos asignados son "manuales". De lo contrario, se asignan recursos específicos según el tipo de misión. Para la misión de exploración, se asignan 15 Scout Troopers y 2 speeder bikes. Para la misión de contención, se asignan 30 Stormtroopers y 3 vehículos de contención elegidos al azar de una lista predefinida. Para la misión de ataque, se asignan 50 Stormtroopers y 7 vehículos de ataque elegidos al azar de otra lista predefinida.

La función mostrar-recursos toma una lista de misiones y las muestra en la consola. Imprime el tipo de misión, el planeta de destino, el general encargado, la prioridad y los recursos asignados para cada misión.

En la función main, se crea una lista de misiones llamada misiones y se agregan varias misiones utilizando la función asignar-recursos. Luego, se llama a la función mostrar-recursos para imprimir los detalles de cada misión en la consola.

El código muestra cómo se pueden asignar recursos para misiones militares utilizando Python y cómo se pueden personalizar los recursos en función del tipo de misión y el general encargado.

1.3. EJERCICIO 3 - ESTACIONES, GRAFO

La clase Vertex representa un vértice del grafo, con su nombre, tipo (ya sea station o junction) y sus vecinos, guardados en un diccionario llamado adjacent.

La clase Graph representa el grafo en sí, utilizando un diccionario llamado vert-dict para guardar los vértices y su información, y un contador num-vertices para llevar la cuenta del número de vértices.

El método add-vertex añade un vértice al grafo, y el método add-edge añade una conexión entre dos vértices.

El algoritmo de Dijkstra se implementa en el método dijkstra. Este método utiliza un diccionario llamado dist para guardar las distancias de cada vértice al vértice inicial. El método utiliza una cola de prioridad (implementada con la función heapq) para ir extrayendo los vértices con menor distancia y actualizando las distancias de sus vecinos.

Finalmente, se crean las estaciones y las intersecciones del grafo, se establecen las conexiones entre ellas, y se encuentra el camino más corto entre ciertos pares de estaciones utilizando el método dijkstra. Los resultados se imprimen en la consola con la función print.

1.4. EJERCICIO 4 - MÉTODOS

El código utiliza los métodos numéricos de bisección, secante y Newton-Raphson para encontrar la raíz de una función matemática dada. La función que se utiliza en este caso es

$$f(x) = x^3 + x + 16.$$

Primero, se define la función f(x) en el código y luego se definen las funciones de bisección, secante y Newton-Raphson. La función de bisección implementa el método de bisección para encontrar la raíz de f(x) en un intervalo dado. La función de secante implementa el método de la secante para encontrar la raíz de f(x) a partir de dos valores iniciales. La función de Newton-Raphson implementa el método de Newton-Raphson para encontrar la raíz de f(x) a partir de un valor inicial.

A continuación, se define una tolerancia (tol) y un número máximo de iteraciones (max-iter) que se utilizarán en los métodos de bisección, secante y Newton-Raphson.

Luego, el código utiliza un bucle while para encontrar dos valores a y b en el intervalo [-10, 10] que tengan signos opuestos de la función f. Esto se hace porque el método de bisección solo funciona si hay un cambio de signo en la función dentro del intervalo. Si no hay un cambio de signo, el método de bisección no funcionará.

Después de encontrar los valores a y b, se aplican los métodos de bisección, secante y Newton-Raphson utilizando estos valores, la tolerancia y el número máximo de iteraciones definidos anteriormente. Los resultados de los métodos se imprimen en la consola, incluyendo la cantidad de iteraciones que se necesitaron para encontrar la raíz y la solución encontrada por cada método.

Finalmente, se calculan las diferencias de decimales entre los métodos utilizando la función abs() para obtener el valor absoluto de la diferencia. Estos valores se imprimen en la consola para mostrar que la diferencia de decimales entre los métodos es prácticamente nula, lo que sugiere que los métodos de bisección, secante y Newton-Raphson son muy similares en este caso.