**Universidad de San Carlos de Guatemala**

Facultad de Ingeniería

Laboratorio Estructura de Datos

Sección B

**Manual técnico**

**FASE III**

Franklin Orlando Noj Perez

202200089

5to Semestre

Ing. En Ciencias y Sistemas

**FORTRAN**

Fortran, acrónimo de "Formula Translation", es un lenguaje de programación de alto nivel diseñado principalmente para cálculos numéricos y científicos. Fue desarrollado por IBM en la década de 1950 y ha experimentado varias versiones desde entonces. Fortran es conocido por su eficiencia en el manejo de operaciones matemáticas y su capacidad para trabajar con grandes conjuntos de datos.

En términos de soporte de computadoras, Fortran ha sido ampliamente adoptado en sistemas mainframe y supercomputadoras, así como en plataformas más modernas. Es compatible con una variedad de arquitecturas de procesadores, incluyendo x86, ARM y POWER, lo que permite su ejecución en una amplia gama de hardware.

En cuanto al consumo de recursos, Fortran es conocido por ser eficiente en términos de rendimiento y uso de memoria. Su diseño está orientado a la velocidad y la capacidad de procesar grandes cantidades de datos, lo que lo hace adecuado para aplicaciones científicas y de ingeniería que requieren cálculos intensivos.

**Consejos para Programar en Fortran:**

Optimización de Cálculos Numéricos: Fortran es conocido por su capacidad para realizar cálculos numéricos eficientes. Aprovecha las características específicas del lenguaje, como las matrices y las operaciones vectorizadas, para optimizar el rendimiento.

Uso de Módulos: Fortran soporta la programación modular, lo que facilita la organización del código en módulos. Utilizar esta característica puede mejorar la legibilidad y mantenibilidad del código.

Gestión de Memoria: Aunque Fortran maneja automáticamente la gestión de memoria en muchos casos, es importante entender cómo funciona para evitar posibles problemas de rendimiento.

**Ventajas de Fortran:**

Eficiencia: Fortran está diseñado para ser eficiente en términos de velocidad de ejecución y uso de memoria, lo que lo hace ideal para aplicaciones científicas y de ingeniería.

Herencia en Computación Científica: Dado que ha sido utilizado históricamente en el ámbito científico, Fortran tiene una amplia base de código existente y numerosas bibliotecas especializadas.

Compatibilidad con Hardware Diverso: Fortran es compatible con diversas arquitecturas de hardware, permitiendo su ejecución en una amplia variedad de sistemas, desde supercomputadoras hasta computadoras personales.

**Desventajas de Fortran:**

Sintaxis Antigua: La sintaxis de Fortran puede parecer anticuada en comparación con lenguajes más modernos, lo que puede dificultar la adopción para nuevos programadores.

Menos Orientado a Objetos: Aunque se han introducido características orientadas a objetos en versiones más recientes, Fortran tradicionalmente carece de las características avanzadas de programación orientada a objetos presentes en algunos lenguajes más modernos.

Arquitectura y Evolución:

Fortran fue diseñado originalmente por IBM en la década de 1950. A lo largo de las décadas, ha experimentado múltiples revisiones y actualizaciones para adaptarse a las necesidades cambiantes de la computación científica y de ingeniería.

**Inventor e Historia:**

John W. Backus lideró el equipo de desarrollo de Fortran en IBM en la década de 1950. Fortran I, la primera versión, se lanzó en 1957. Backus recibió el Premio Turing en 1977 por su contribución al diseño de lenguajes de programación, incluido Fortran.

En resumen, Fortran sigue siendo una herramienta poderosa para cálculos numéricos y científicos, con una rica historia, una amplia base de usuarios y una evolución continua para adaptarse a las demandas de la computación moderna.

***TABLAS HASH***

¡Claro! Las tablas hash son estructuras de datos que se utilizan para almacenar y recuperar datos de manera eficiente. Están compuestas por una matriz (o arreglo) y una función hash que mapea claves de búsqueda a índices de la matriz.

***Función Hash***

Una función hash toma una clave de búsqueda como entrada y produce un valor hash, que es un número entero. Esta función debe ser determinista, es decir, para la misma clave siempre produce el mismo valor hash. La función hash distribuye uniformemente las claves a lo largo del rango de índices de la tabla hash, idealmente evitando colisiones (dos claves distintas que tienen el mismo valor hash).

***Tabla Hash***

Una tabla hash es una estructura de datos que consiste en una matriz (o arreglo) y una función hash. La matriz tiene un tamaño predefinido y generalmente es mucho más grande que la cantidad de elementos que se espera almacenar en ella. Cada elemento de la matriz se denomina "cubo" o "slot".

***Almacenamiento de Datos***

Para almacenar un par clave-valor en una tabla hash, se aplica la función hash a la clave para determinar el índice en la matriz donde se almacenará el valor asociado. Si hay colisiones (dos claves que tienen el mismo valor hash), se resuelven utilizando una estrategia de resolución de colisiones, como encadenamiento o sondeo.

***Encadenamiento***

Cada cubo de la tabla contiene una lista enlazada de elementos que tienen el mismo valor hash. Cuando ocurre una colisión, el nuevo elemento se agrega al final de la lista enlazada en el cubo correspondiente.

***Sondeo***

Cuando ocurre una colisión, se busca otro cubo en la tabla hash para almacenar el elemento. Esto se hace iterativamente utilizando una secuencia predefinida de índices calculados a partir del valor hash original.

***Búsqueda y Recuperación de Datos***

Para buscar un valor en una tabla hash, se aplica la función hash a la clave de búsqueda para determinar el índice en la matriz donde se espera encontrar el valor. Si hay colisiones, se sigue el mismo proceso que se utilizó para almacenar los datos. Una vez en el cubo correcto, se busca el valor dentro de la lista enlazada (en el caso de encadenamiento) o se realiza una búsqueda secuencial en los cubos (en el caso de sondeo).

***Eficiencia***

La eficiencia de una tabla hash depende en gran medida de la calidad de la función hash y de la estrategia de resolución de colisiones utilizada. En el mejor caso, la búsqueda y recuperación de datos en una tabla hash puede tener una complejidad de tiempo constante O(1). Sin embargo, en el peor caso, si hay muchas colisiones, la complejidad puede ser lineal O(n).

En resumen, las tablas hash son una herramienta poderosa para el almacenamiento y recuperación eficientes de datos, particularmente útiles cuando se necesita acceso rápido a los datos utilizando claves de búsqueda.

***Árboles de Markov:***

Los árboles de Markov, también conocidos como cadenas de Markov de orden 1, son estructuras de datos probabilísticas que representan una secuencia de eventos donde la probabilidad de que ocurra un evento depende únicamente del evento inmediatamente anterior. Estos árboles son una forma de modelo probabilístico que se utiliza en una variedad de aplicaciones, como procesamiento del lenguaje natural, modelado de sistemas estocásticos y análisis de datos.

***Nodos y Transiciones***

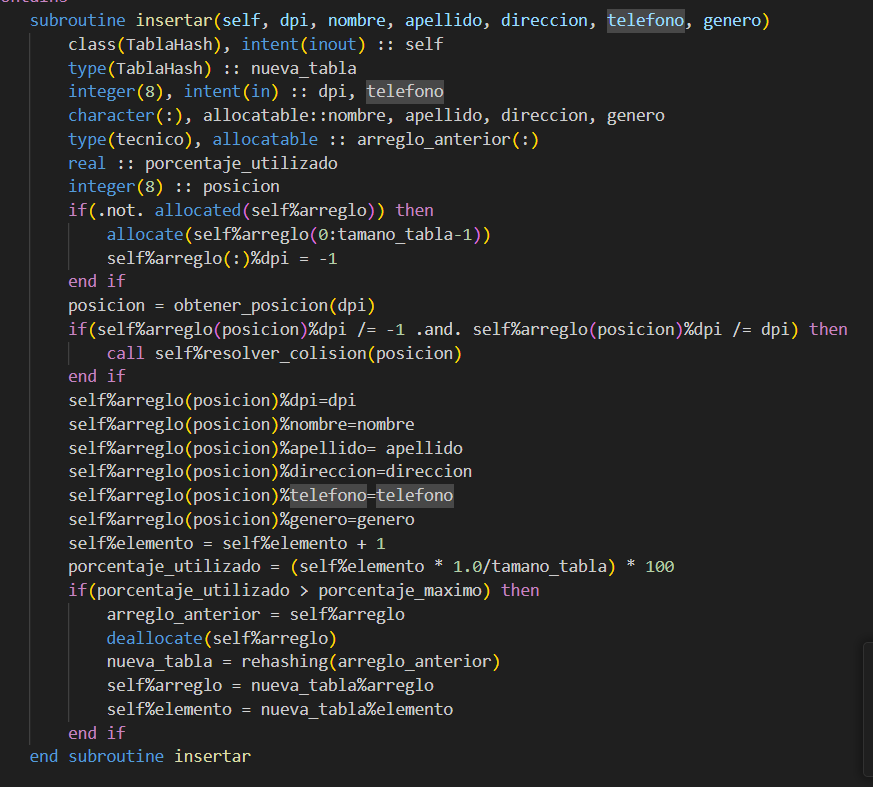
Cada nodo en el árbol de Markov representa un estado posible, y las transiciones entre los estados se modelan como arcos dirigidos con probabilidades asociadas. La probabilidad de transición desde un estado a otro se puede representar mediante una matriz de probabilidades de transición.

***Matriz de Transición***

Una matriz de transición es una matriz cuadrada donde cada fila y cada columna representan un estado en el árbol de Markov. Los elementos de la matriz representan las probabilidades de transición entre los estados. En un árbol de Markov de orden 1, estas probabilidades dependen solo del estado actual y no de los estados anteriores.

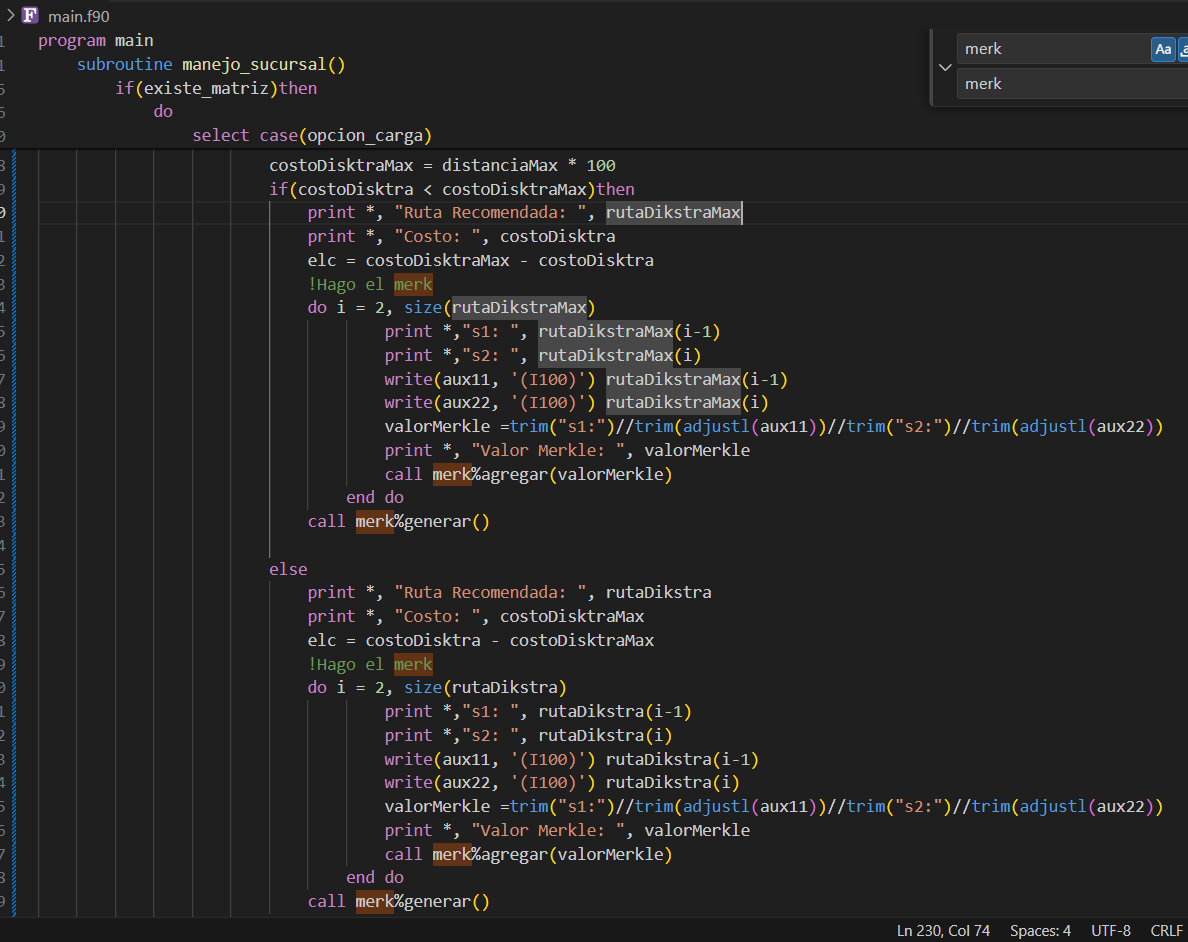
Propiedades: Los árboles de Markov tienen propiedades interesantes, como la propiedad de Markov, que establece que la probabilidad de transición a un estado futuro solo depende del estado actual y no de la secuencia de estados anteriores. Esto los hace útiles para modelar sistemas donde las transiciones son independientes de la historia pasada, pero dependen solo del estado actual.

***Algunas partes de mis Algoritmos***

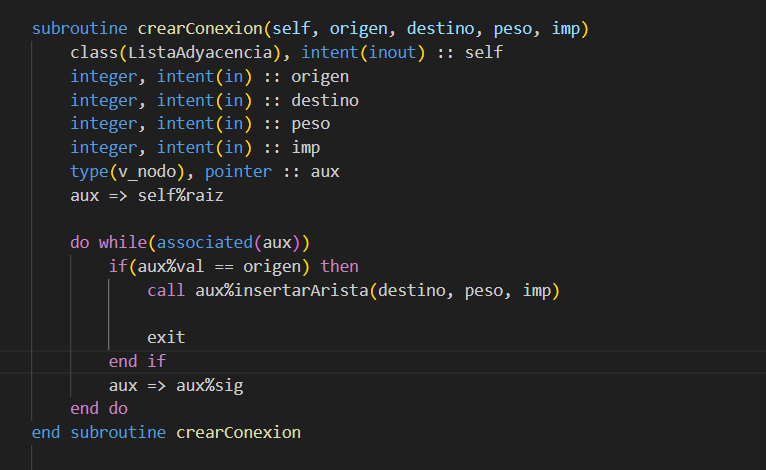
******

***TABLA HASH DE LOS TECNICOS***

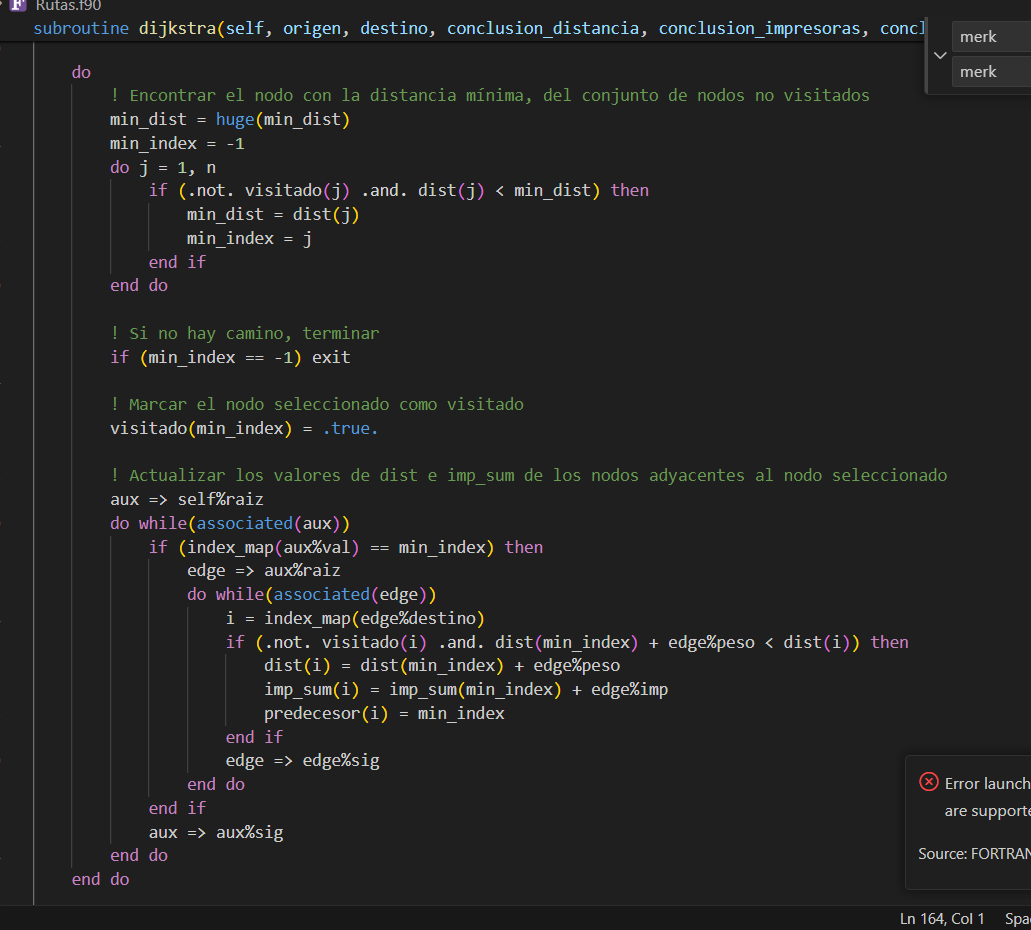
***ACA SE HACE USO DE LAS ESTRUCTURAS***

******

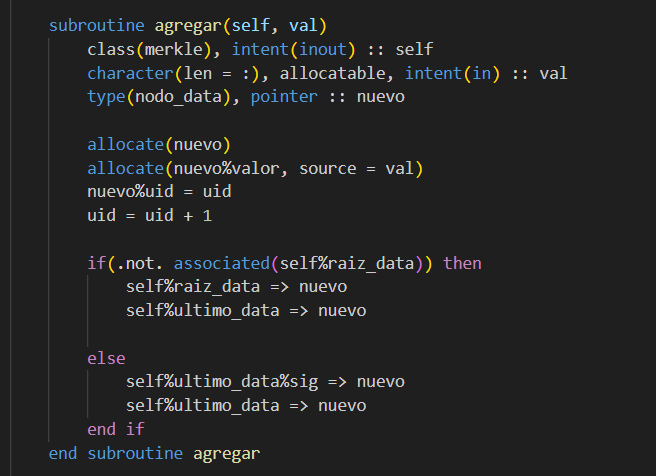
***SUBRUTINA PARA CREAR LAS CONEXIONES EN LA LISTA DE ADYACENCIA***

******

***ESCENCIA DEL ALGORITMO DE DIKSTRA***

******

***SUBRUTINA DE AGREGACION DEL ARBOL DE MERKLE***

******



**Se utilizo el IDE Visual studio code**

**Se utilizo la extensión C/C++**

**Modern Fortran**

**Code Runner**

***Para el versiona miento de hizo uso de Git Hub***

***Repositorio del proyecto***

***https://github.com/FranklinONP/EDD\_PROYECTO\_202200089***