2024 | Estructura de Datos

TRAVEL MAP GT

Manual Técnico

TRAVEL MAP GT

ÍNDICE

Contenido

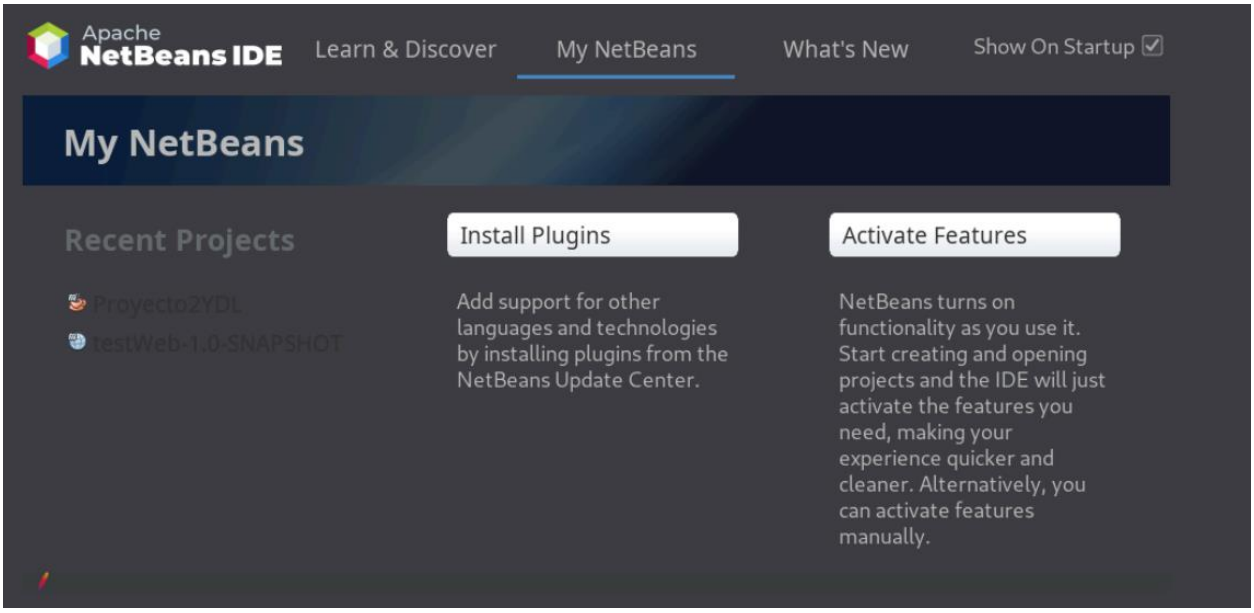
TECNOLOGÍA USADA

Lenguaje de Programación usado:

- Java
- OpenJDK 17.0.10
- OpenJDK Runtime Environment (build 17.0.10+7-Debian-1)
- OpenJDK 64-Bit Server VM (build 17.0.10+7-Debian-1, mixed mode, sharing)

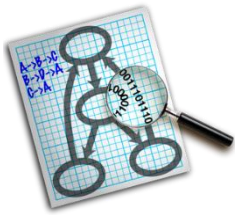
IDE

- Apache netbeans 21



Graphviz

- Version 8.1.0



USO DE ESTRUCTURAS

Árbol B

Para construir el árbol B se necesitó de dos clases, se especifican a continuación:

Página de árbol

Se trata de una clase genérica que necesita de un objeto <T> que implementa la interfaz comparable.

La clase es: BTreePage<T extends Comparable<T>>

Atributos

Modificador de acceso	Tipo	Nombre	Descripción
private	LinkedList<T>	nodes	Son los nodos de la pagina actual del árbol, es una lista donde se almacenan los objetos tipo T
private	LinkedList <BTreePage<T>>	punteros	Son los punteros a los que la pagina del árbol puede apuntar, son punteros a otra estructura BTreePage<T>
private	int	totalElements	El total de elementos o nodos en la página actual
private	int	order	El orden de la página, sirve para poder hacer crecer el árbol.
private	BTreePage<T>	father	Una referencia al padre de la pagina.

Métodos

Modificador de Acceso	Tipo de retorno	Nombre	Parámetros	Descripción
public	Constructor		int order	Crea una página raíz
public	Constructor		int order BTreePage<T> father	Crea una página con un padre asociado
public	void	insert	T content	Inserta un nodo en la página del árbol

private	void	insertInLevel	T content	Inserta el nodo en el nivel del árbol
private	boolean	isFull		Evalua si el arbol supera su orden y necesita dividirse
private	void	internalInsert	LinkedList<T> orderedNodes	agrega una lista de nodos estrictamente ordenados
private	void	internalAdjust	LinkedList <BTreePage<T>> punteros	Hace un ajuste interno de punteros, básicamente agregar los punteros
private	void	growUp		Hace crecer la página, haciendo división celular.
private	void	concatReference		Agrega una referencia de un nivel más bajo en el árbol
private	void	onFixFatherRef		Arregla recursivamente las referencias a los padres de la pagina actual y posibles asociadas.
private	int	getIndexToInsert	T content	Obtiene el índice dónde se debe ingresar un objeto, de manera ordenada.
private	void	reset		Elimina todos los datos de la página, es decir, los objetos.
public	boolean	isEmpty		Retorna “true” si la pagina está vacía, o sea que no tiene objetos almacenados, de lo contrario retorna “false”
public	T	get	T comparable	Busca un objeto a partir de un comparable y lo retorna si existe. De lo contrario lanza una excepción.

public	LinkedList <T>	getNodes		Retorna la lista de nodos u objetos almacenados.
public	LinkedList <BTreePage <T> >	getPunteros		Retorna los punteros de la página.
public	boolean	hasFather		Retorna “true” si el padre de la pagina es diferente de “null”, de lo contrario retorna “false”

Clase Árbol

Se trata de una clase genérica que necesita de un objeto <T> que implementa la interfaz comparable, para poder crear sus páginas.

El nombre de la clase es BTree<T extends Comparable<T>> y se encuentra en el paquete de BTree

Atributos

Modificador de acceso	Tipo	Nombre	Descripción
private	BTreePage<T>	raiz	Es la página raíz del árbol.
private	int	size	El total de elementos contenidos en el árbol.

Métodos

Modificador de Acceso	Tipo de retorno	Nombre	Parámetros	Descripción
public	Constructor		int order	Crea un nuevo árbol vacío.
public	void	insert	T content	Inserta un nodo en el árbol.
public	boolean	isEmpty		Retorna “true” si el árbol está vacío, de lo contrario “false”.

public	T	find	T comparable	Busca un objeto a partir de un comparable y lo retorna si existe. De lo contrario lanza una excepción.
public	BTreePage <T>	getRaiz		Retorna la raíz del árbol
public	boolean	exist	T comparable	Retorna “true” si el elemento existe en el árbol, “false” de lo contrario
public	int	getProfundidad		Retorna la profundidad del árbol, es decir, cuántos niveles tiene.

Grafo

Se necesitaron 3 clases para construir un grafo que puede servir para un recorrido dirigido como no dirigido.

Clase Grafo

Se trata de una clase genérica que necesita de un objeto <K> que implementa la interfaz comparable y un objeto cualquiera tipo W

El nombre de la clase: Grafo<K extends Comparable<K>, W>

Donde K se puede entender como “Key” y W como “Weight”

Atributos

Modificador de acceso	Tipo	Nombre	Descripción
private	BTree <NodeGraph<K>>	nodos	Es un árbol B que almacena los nodos del grafo respecto a una clave especifica.
private	BTree <NodeNum<K>>	nodesByNumber	Árbol B que almacena los nodos del grafo respecto a un número que la clase le asigna incrementalmente.

private	List<List<W>>	paths	Es una lista dinámica, que siempre debe ser cuadrada (o sea de n*n) que almacena los objetos de pesos. Un null en la lista significa que no hay un camino disponible de un nodo a otro. La primera indicación es del nodo desde y la otra es a dónde.

Métodos

Modificador de Acceso	Tipo de retorno	Nombre	Parámetros	Descripción
public	Constructor			Crea un nuevo grafo sin información.
public	void	addNode	K key	Agrega un nuevo nodo para el grafo, debe insertarlo en los dos árboles B correspondientes e ir incrementando la lista de aristas.
private	void	refillList	List<W> list int number	Agrega nulos a una lista especifica, se usa para mantener la lista de aristas cuadrada.
public	void	updateAllLists		Actualiza todas las listas que lo necesiten con nulos.
public	void	addPath	K fromKey K toKey W weight	Agrega un camino entre dos nodos. Lanza una excepción si alguno de los dos nodos no se encuentra.
public	boolean	isEmpty		Retorna “true” si el grafo está vacío, “false” de lo contrario
public	K	getNode	K key	Retorna un nodo a partir de una clave específica

public	K	getNode	int code	Retorna un nodo a partir de su clave numérica
public	NodeNum<K>	getNodeNum	int code	Retorna un nodo de grafo a partir de su clave numérica
public	NodeGraph<K>	getNodeNum	K key	Retorna un nodo de grafo a partir de una clave.
public	W	getPath	K fromKey K toKey	Retorna el objeto W que representa el peso de una arista del grafo, lanza una excepción si alguna de las claves no existe.
public	W	getPath	int codeFrom int codeTo	Retorna el objeto W que representa el peso de una arista del grafo, lanza una excepción si alguna de las claves se sale del rango adecuado.
public	W	getPath	K fromKey int codeTo	Retorna el objeto W a partir de una clave especificada y un código de ubicación. Lanza una excepción si alguna de las claves se sale del rango adecuado o la clave no existe.
public	W	getPath	int codeFrom K toKey	Retorna el objeto W a partir de una clave especificada y un código de ubicación. Lanza una excepción si alguna de las claves se sale del rango adecuado o la clave no existe.

Clases de Nodo de Grafo

Son objetos genéricos simples que necesitan que implementan la interfaz Comparable.

Tienen la capacidad de convertirse de uno a otro.

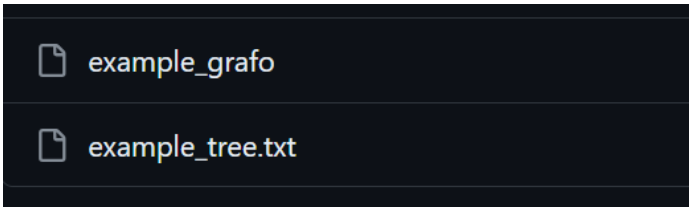
Estas clases son:

- NodeGraph<K extends Comparable<K>> implements Comparable<NodeGraph<K>>
  - Se ordena por medio del objeto tipo K
- NodeNum<K extends Comparable<K>> implements Comparable<NodeNum<K>>
  - Se ordena por medio de un número.

Ejemplo del uso de estructuras

En la carpeta de documentación del proyecto se incluye un código en java de ejemplo para poder usar las estructuras de Grafos como de árboles B especificadas antes

El path de los archivos de referencia es ../TravelMapGT/documentation/use_structures



ALGORITMOS DE FUNCIONES IMPORTANTES

Algoritmo para encontrar todos los recorridos de A hasta B sin pasar dos veces por el mismo destino (no tiene sentido)

empezar en el punto A

BUSCAR_CAMINO(A)

```
BUSCAR_CAMINO(nodo):
        guardar en la lista de ya recorridos el número del nodo actual
        para cada nodo que conecte con A hacer
                verificar si el nodo subcamino es el destino
                 si es el destino
                        agregar el nodo a la lista de recorridos
                        guardar la lista de recorridos en el arbol
                de lo contrario
                        verificar que no este en la lista de ya recorridos
                        si esta en la lista
                                no ir por ahi
                        de lo contrario
                                recorrer subcamino: BUSCAR_CAMINO(subcamino);
        fin para
        si no hay otros nodos, el camino no llega al destino y no se hace nada.
Fin del método
```