Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE 0217 – Estructuras de Datos Abstractas y Algoritmos
III ciclo 2024

Laboratorio III

Danna Guevara C23562
Grupo 01

Profesor: Karen Dayana Tovar Parra

2 marzo

Índice

1. Ejercicio 2

Guiándome por medio de la página geeksforgeeks, encontré una implementación a la función deleteNode() para eliminar un nodo de un árbol binario de búsqueda, donde este básicamente lo que hace es si el árbol está vacío o que el valor no existe no elimina nada, porque no hay que eliminar, luego si el valor que se quiere eliminar es menor que el nodo actual, se debe de buscar en el lado izquierdo de este, entonces si es mayor, se debe de buscar en el lado derecho, y cuando ya se encuentra quien se quiere eliminar, se debe de tomar en cuenta si el nodo solo tiene un hijo derecho, izquierdo o si tiene los dos, ya que dependiendo del caso se eliminará y puede que se reemplace un número. Finalmente, le enseñé a chatgpt cómo se veía mi código para que este pudiera calzar con el tipo de método que se estaba pidiendo en la instrucción, para que de igual manera así pudiese funcionar en el main. Por lo que simplemente realizó otro método donde utilizó el tipo que se debía para llamar a la función que realizaba la tarea correcta de eliminar el nodo, pero no como se indicaba ni con los parámetros que se indicaba.

`https://www.geeksforgeeks.org/deletion-in-binary-search-tree/`
`https://chatgpt.com/share/67ae6856-276c-8001-a23c-f766e6dd0c52`

2. Ejercicio 3

Para este ejercicio quería comprender de mejor manera cómo utilizar el algoritmo y qué métodos iban a ser de ayuda para incluirlos en el .h, por lo que utilicé la página que incluí al final, donde se añade una función de utilidad para encontrar el vértice con el valor de distancia mínima y la función que implementa el algoritmo de Dijkstra que busca la ruta más corta para un gráfico mediante la representación de matriz de adyacencia. Lo implementé de una manera similar, donde se representa un grafo usando una lista de adyacencia y una lista de aristas con "pesos". Además, el algoritmo de Dijkstra funciona con el fin de encontrar las distancias más cortas desde un vértice de inicio hasta todos los demás vértices del grafo. Se añade desde el inicio un vector de tuplas que almacena las aristas del grafo. También se logra agregar una arista con peso, de manera en que se almacena la arista en G con su peso y actualiza la lista de adyacencia para hacer la conexión. Se busca en la función de distancia mínima el nodo con la menor distancia en el arreglo dist[] que aún no ha sido visitado, donde solamente se alteró que tomase en cuenta el número de vértices. Finalmente, el algoritmo crea un ünordered_map (vertexIndex)"para mapear cada vértice a un índice entero, esto para que se facilite el acceso a los arreglos dist[] y visited[], así encuentra el vértice con la menor distancia aún no visitado, marca ese vértice como visitado, y de ser el caso de que v no ha sido visitado y el camino a través de u es más corto, actualiza dist[v].

`https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/?ref=lbp/`

3. Ejercicio 4

De igual manera para este ejercicio quería comprender de mejor manera cómo utilizar el algoritmo, por lo que utilicé la página que añadí al final, a fin de cuentas el código lo que hace es almacenar todas las aristas en edges y ordenadas por peso, utiliza la estructura de conjuntos disjuntos (DSU) para tener información sobre la conexión entre nodos y de esta manera recorrer

las aristas en orden creciente de peso, porque si los nodos aún no están conectados, los une y añade la arista al MST, esto se repite hasta formar un árbol con V-1 aristas (donde V es el número de nodos). Para demostrar cómo resulta el MST y su costo total.

`https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/`

4. Instrucciones Makefile

El laboratorio cuenta con un archivo tipo `Makefile`, el cual está diseñado para gestionar los diferentes ejercicios del laboratorio, estableciendo variables de entorno y rutas a directorios específicos.

Para ejecutar este `Makefile`, es necesario abrir una terminal en el directorio raíz del proyecto (donde está ubicado el archivo `Makefile`) y utilizar el comando:

```
make <objetivo>
```

Reemplazando `<objetivo>` con el nombre de la tarea que se desea ejecutar, los objetivos definidos en el `Makefile` se presentan a continuación. Cada objetivo especifica un conjunto de comandos para compilar los ejercicios según las rutas definidas en las variables.

- `ej1`
- `ej2`
- `ej3`
- `ej4`
- `ej5`

Después de ejecutar make, los ejecutables se crean en la carpeta build/; por lo que se debe de utilizar el siguiente comando seguidamente del make;

```
./build/<objetivo>
```

- `ej1`: ejecuta el ejercicio 1 que busca el valor mínimo y máximo en BST.
- `ej2`: ejecuta el ejercicio 2 que implementa la eliminación de nodos en BST.
- `ej3`: ejecuta el ejercicio 3 que implementar Dijkstra en grafos.
- `ej4`: ejecuta el ejercicio 4 que utiliza Kruskal para encontrar el Árbol de Expansión Mínima (MST).
- `ej5`: ejecuta el ejercicio 5 que utiliza el HeapSort usando un Max-Heap.