

# Programa 03

## grafos

PROFESOR: NOÉ ORTEGA SANCHEZ

MARTINEZ OROZCO VICTOR MANUEL



El programa es un grafo dinámico el cuál almacena un tipo de dato abstracto nombrado alumno. Este dato estructurado se compone de nombre, código y carrera. Realmente no termino de encontrarle sentido a los datos que almacena, pues el grafo solo relaciona las distancias entre estos alumnos, pero técnicamente el programa es totalmente funcional.

Cabe aclarar que los métodos implementados propios del grafo son los más básicos e indispensables para poder operar con él y estos son públicos para operar de la manera más fácil con ellos.

- **Buscar vértice:** Este método realmente solo opera internamente con los demás métodos del grafo. Recibe un dato tipo alumno y se enfoca en el atributo nombre del mismo. Prácticamente no es más que un recorrido simple de los vértices en el grafo tal como el nombre lo denota, esto es; usando una iteración preprueba, verifica si la variable *i* (inicializada en la raíz del grafo) es diferente de una dirección nula. En caso de ser diferente de nula, verifica si la dirección en que apunta ese puntero a vértice en su atributo nombre es igual al dato tipo alumno en su atributo nombre que recibió como parámetro el mismo método y en caso de que sí sea igual, este retorna la variable *i*, siendo esta la dirección de memoria que aloja la dirección del vértice a buscar. Caso contrario, si se llegó a una dirección nula, solo la retorna y culmina el algoritmo.
- **Insertar vértice:** Este método recibe de parámetro un dato tipo alumno. Se enfoca únicamente en el nombre para asignar el vértice, pero solicita al usuario que ingrese completamente los atributos del mismo objeto. Inicialmente verifica ya existe el vértice a insertar, para ello llama al método de buscar vértice. En caso de ya existir solo imprime en pantalla que el vertice ya existe. En caso de que no exista, verifica si el grafo está vacío (si el ancla al grafo es una dirección nula), si está vacío únicamente inserta el vértice en la misma ancla, sino recorre las direcciones de memoria del grafo hasta llegar a una dirección nula y procede a insertar el dato en ese lugar, para esto usa una iteración preprueba.
- **Insertar arista:** Este método recibe dos datos tipo alumno que serán los que se tomarán como referencia para insertar la arista y un dato tipo entero que será el precio entre ellos (este precio tómesese como distancia entre ellos). Primeramente busca si existen los vértices brindados como parámetro al mismo método y los asigna a las variables *vori* (tómese como vertice origen) y *vdest*(vértice destino). Si alguna de estas variables apunta a nulo, significa que no existe tal referencia y será imposible insertar, por ello solo muestra en pantalla “el vértice no existe” y culmina con las demás instrucciones. En caso de sí existir los vértices, crea la nueva arista con los parámetros dados y únicamente inserta la arista a lista de aristas mediante un recorrido simple o crea una nueva arista según sea el caso.
- **Mostrar lista de adyacencia:** Este método funciona muy simple, inicializa una variable en la dirección de la raíz a los vértices del grafo y mientras no se llegue a una dirección nula, recorre cada vértice junto con todas las aristas e imprime los parámetros que guarda cada dato alumno. En otras palabras, son dos iteraciones preprueba anidadas, una recorre los vértices y otra recorre las aristas del grafo. Una vez que apunta a una dirección nula solo termina con el algoritmo.
- **Eliminar aristas:** Este método recibe un dato tipo alumno, mismo que sería el dato a eliminar. Primeramente, verifica si el vértice dado es una dirección nula, si es así significa que el vértice no existe y solo cancela las demás instrucciones. De lo contrario, se crea un puntero "*i*" que se establece inicialmente en el primer elemento de la lista de aristas del vértice. Luego, en un

bucle while, el método recorre la lista de aristas del vértice y elimina cada una de ellas. El puntero "i" se actualiza para apuntar al siguiente elemento de la lista, y el bucle continúa hasta que se eliminan todas las aristas de la lista.

Dentro del bucle, el método muestra en la consola un mensaje que indica qué arista se está eliminando. Finalmente, se libera la memoria asignada a cada arista eliminada.

- Eliminar vértice: Este método elimina un vértice específico junto con todas las aristas que se originan desde ese vértice. El vértice que se eliminará se logra ubicar mediante su nombre, que se pasa como parámetro al método (el dato tipo alumno).

Primeramente, este comienza verificando si el vértice que se eliminará es el primer vértice del grafo (el vértice apuntado por "hGrafo"). Si es así, el puntero ancla o "hGrafo" se actualiza para apuntar al siguiente vértice, luego se llama al método "Eliminar Aristas" para eliminar todas las aristas que se originan desde el vértice que se eliminará, y finalmente se libera la memoria asignada para ese vértice. Si el vértice que se eliminará no es el primer vértice del grafo, el método recorre la lista de vértices del grafo para encontrar el vértice especificado. Si se encuentra el vértice, el puntero "i" se actualiza para apuntar al vértice anterior, el puntero "j" se actualiza para apuntar al vértice que se eliminará y se actualiza el puntero "i" para saltar al siguiente vértice.

Luego, el método llama al método "Eliminar Aristas" para eliminar todas las aristas que se originan desde el vértice que se eliminará, actualiza el puntero "i" para saltar el vértice que se eliminará y finalmente libera la memoria asignada para ese vértice.

Si el vértice que se eliminará no se encuentra en la lista de vértices del grafo, se muestra un mensaje indicando que el vértice especificado no existe.

- Eliminar arista: Este es un método que elimina una arista específica entre dos vértices. El método recibe como parámetros los nombres de los dos vértices que están conectados por la arista. Primeramente, el método busca los vértices de origen y destino correspondientes a los nombres dados usando el método "buscarVertice". Si uno o ambos vértices no existen, se muestra un mensaje de error y se cancelan las posteriores instrucciones. Si ambos vértices existen, el método busca la arista que conecta los dos vértices. Si la arista existe, se actualiza el puntero "i" para apuntar a la arista anterior, luego se actualiza el puntero "j" para apuntar a la arista que se eliminará y se actualiza el puntero "i" para saltar la arista que se eliminará.

Finalmente el método libera la memoria asignada para la arista que se eliminará y muestra un mensaje indicando que la arista ha sido eliminada. Si la arista no se encuentra en la lista de aristas del vértice de origen, se muestra un mensaje indicando que la arista especificada no existe.

- Eliminar todo: Este método es muy simple, comienza con el puntero "i" apuntando al primer vértice de la lista de vértices.

Luego, entra en un bucle mientras el puntero "hGrafo" no sea nulo, lo que indica que aún hay vértices en la lista. Dentro del bucle, se actualiza el puntero "i" para apuntar al vértice que se eliminará, se actualiza el puntero "hGrafo" para saltar el vértice que se eliminará y se llama al método "Eliminar Aristas" para eliminar todas las aristas que conectan con el vértice que se eliminará. Después de eliminar todas las aristas del vértice, el método libera la memoria asignada al vértice y repite el proceso hasta que no haya más vértices en la lista.

Finalmente, el método muestra un mensaje indicando que se han eliminado todos los datos correctamente.

Guardar en archivo: Para guardar simplemente se reutilizó el algoritmo de impresión de la lista de adyacencia. Se crea un dato tipo ofstream para poder escribir en un archivo txt llamado file01Vertices y se declara una variable (en este caso "i") para iterar en un bucle preprueba mientras no se llegue a una dirección nula. Entonces, en cada iteración, se escriben en el archivo los vértices junto con toda la información insertada en los atributos del objeto alumno. Cuando i apunte a nulo, se sabrá que ya no hay más vértices por insertar al archivo, entonces se procede a aplicar la misma metodología para guardar las aristas, es decir, el método abre el archivo "file01Aristas.txt" en modo de escritura y recorre cada vértice del grafo. Para cada vértice, se recorre su lista de adyacencia y se guarda la información de cada arista en una línea separada. La información de cada arista se guarda en el formato "origen|precio|destino\*", donde "origen" es el nombre del vértice de origen, "precio" es el peso de la arista y "destino" es el nombre del vértice de destino. Después de guardar la información de todas las aristas, se cierra el archivo "file01Aristas.txt".

Como es evidente, se utilizaron dos archivos para poder guardar la información del grafo, puesto que utilizar solo uno a mi parecer fue más complicado, además recursos hay diría el profe :)

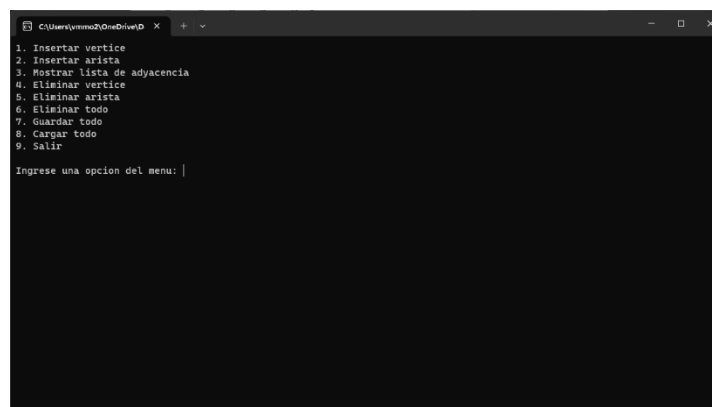
- Cargar de archivo: Para este método, simplemente se reutiliza la misma técnica aplicada desde el primer trabajo para cargar de un archivo. Se declaran dos variables tipo fstream para poder leer el archivo y un objeto alumno para poder operar con los atributos del mismo.

Posteriormente se abren los documentos de texto en donde se almacenan los vértices y las aristas y con dos iteraciones se insertan los datos.

Primero se cargan los vértices al grafo con ayuda de una iteración preprueba la cual se detendrá cuando el archivo llegue a su fin y luego se cargan las aristas, reciclando de forma descarada el algoritmo anterior para cargar los vértices. Una vez cargados los datos en el grafo, se muestra un mensaje en pantalla aclarando que ya fueron cargados los datos en el grafo.

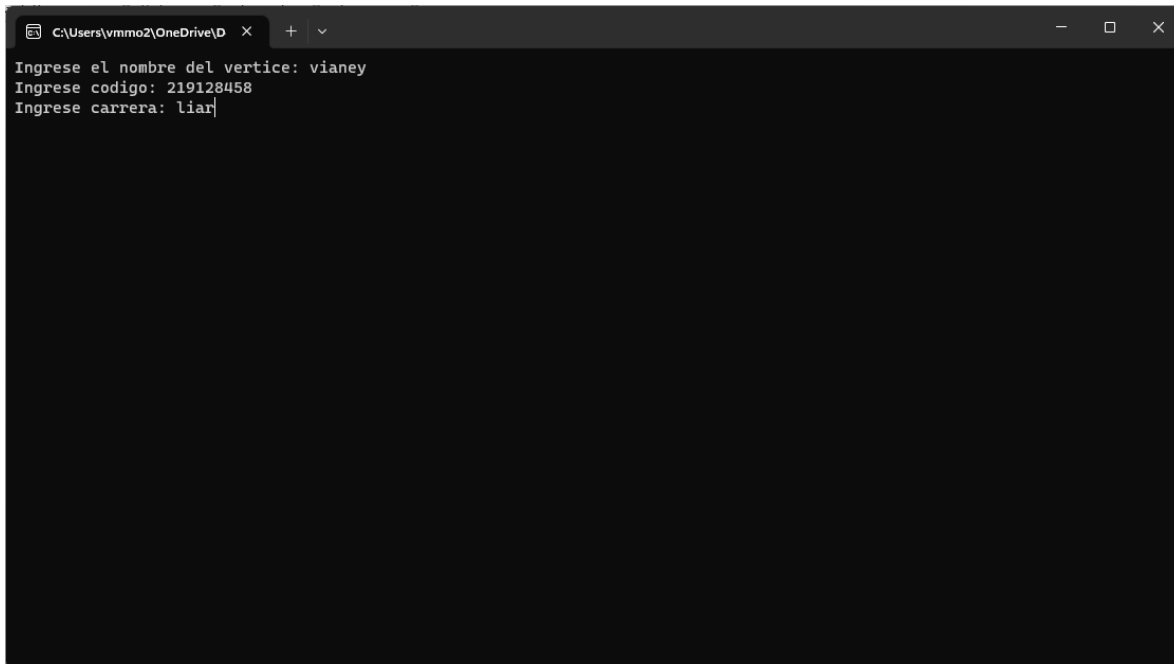
Impresiones de pantalla

Menu principal:



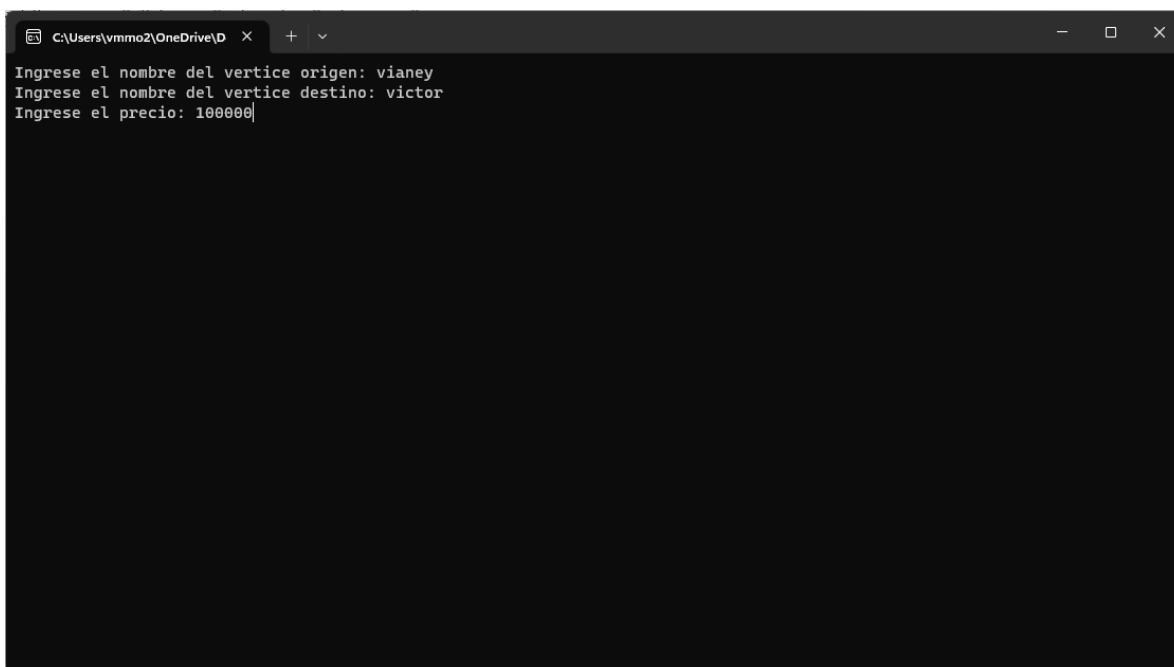
```
C:\Users\ymma2\OneDrive\ID >
1. Insertar vertice
2. Insertar arista
3. Mostrar lista de adyacencia
4. Eliminar vertice
5. Eliminar arista
6. Eliminar todo
7. Guardar todo
8. Cargar todo
9. Salir
Ingrese una opcion del menu: |
```

Insertar vertice:



```
C:\Users\vmimo2\OneDrive\D  X + v
Ingrese el nombre del vertice: vianey
Ingrese codigo: 219128458
Ingrese carrera: liar
```

Insertar arista:



```
C:\Users\vmimo2\OneDrive\D  X + v
Ingrese el nombre del vertice origen: vianey
Ingrese el nombre del vertice destino: victor
Ingrese el precio: 100000
```

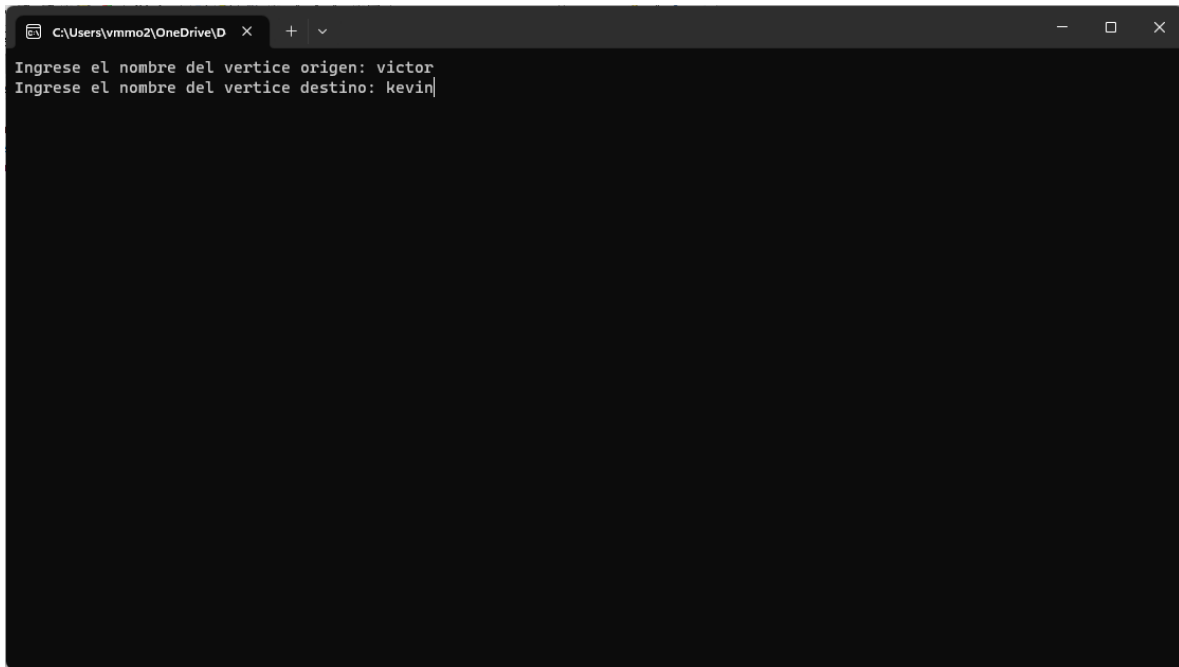
Mostrar lista de adyacencia:

```
C:\Users\vmimo2\OneDrive\D  X + v
victor = victor|inni|32324|distancia: 500-->kevin|23232|jojojo*
kevin =
vianey = vianey|liar|219128458|distancia: 100000-->victor|32324|inni*
Presione una tecla para continuar . . . |
```

Eliminar vértice:

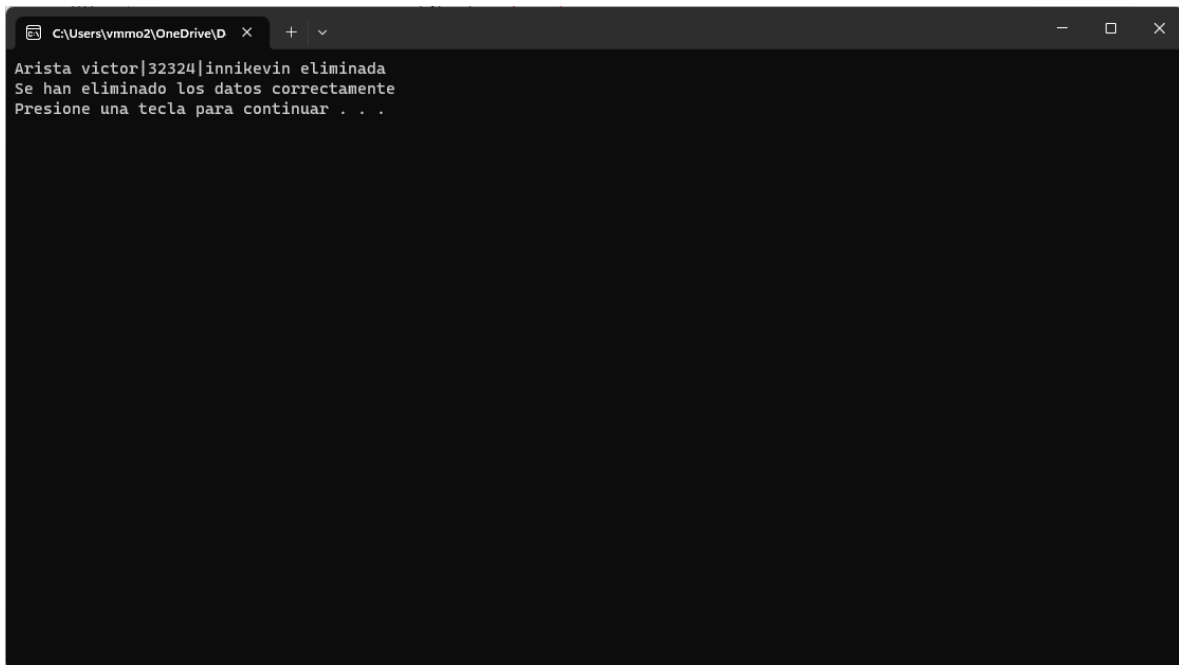
```
C:\Users\vmimo2\OneDrive\D  X + v
Ingrese el nombre del vertice a eliminar: victor
Arista victor|32324|innikevin eliminada
Vertice victor fue eliminado
Presione una tecla para continuar . . . |
```

Eliminar arista:



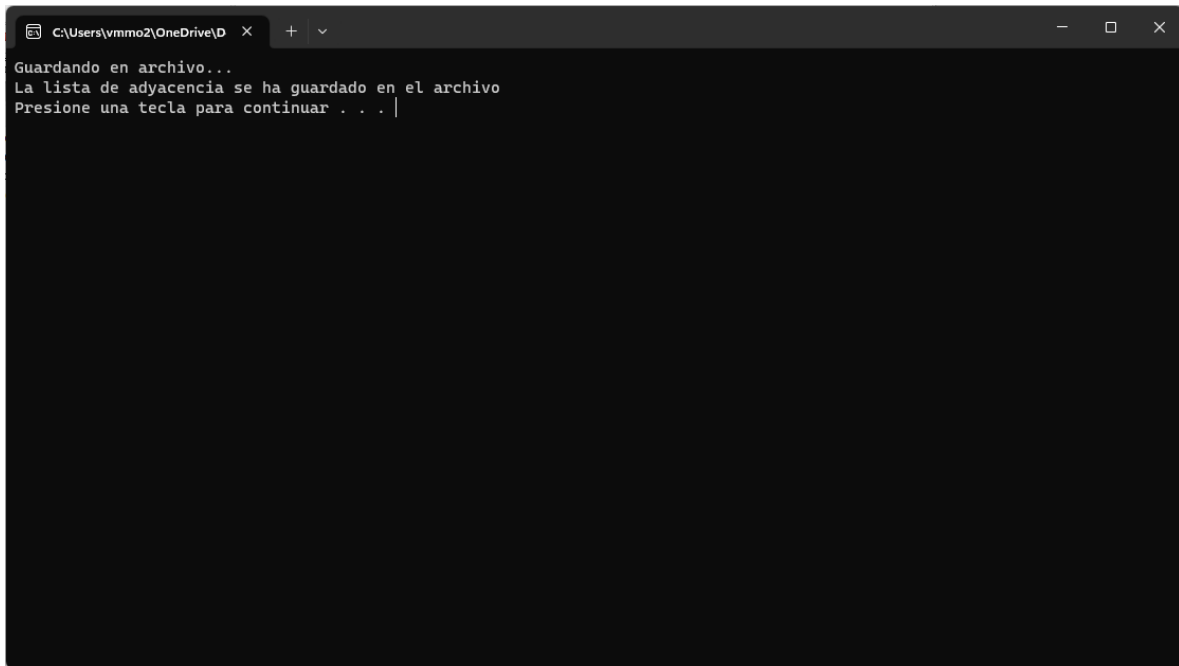
```
C:\Users\vmimo2\OneDrive\D >
Ingrese el nombre del vertice origen: victor
Ingrese el nombre del vertice destino: kevin
```

Eliminar todo:



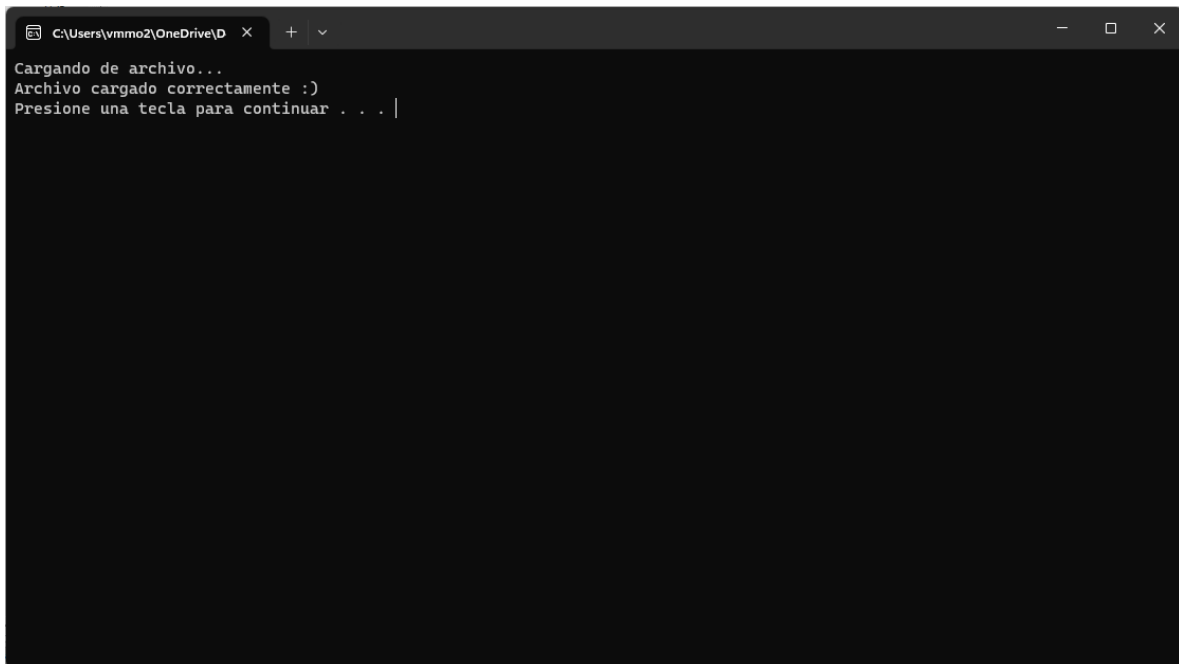
```
C:\Users\vmimo2\OneDrive\D >
Arista victor|32324|innikevin eliminada
Se han eliminado los datos correctamente
Presione una tecla para continuar . . .
```

Guardar en archivo:



```
C:\Users\vmimo2\OneDrive\D >
Guardando en archivo...
La lista de adyacencia se ha guardado en el archivo
Presione una tecla para continuar . . . |
```

Cargar de archivo:



```
C:\Users\vmimo2\OneDrive\D >
Cargando de archivo...
Archivo cargado correctamente :)
Presione una tecla para continuar . . . |
```