Instituto Tecnológico y de Estudios Superiores de Occidente

Programación con memoria dinámica

Reporte Contenedor DGraph

ITESO

Universidad Jesuita de Guadalajara

Profesor: Dr. Francisco Cervantes

José Manuel Lira Aguas nt730200@iteso.mx

Fecha de entrega: 1 de diciembre de 2022

Contenido

Introducción

Un grafo es una estructura compuesta por vértices (o nodos) y aristas que conectan vértices, un grafo G consta de dos partes:

- Un conjunto V, cuyos elementos se denominan vértices (o nodos)
- Un conjunto E, cuyos elementos se denominan aristas, que son un par ordenado de vértices

En un grafo dirigido, cada vértice puede tener tantas aristas como vértices haya, y las aristas tienen direcciones como se muestra en la siguiente figura.

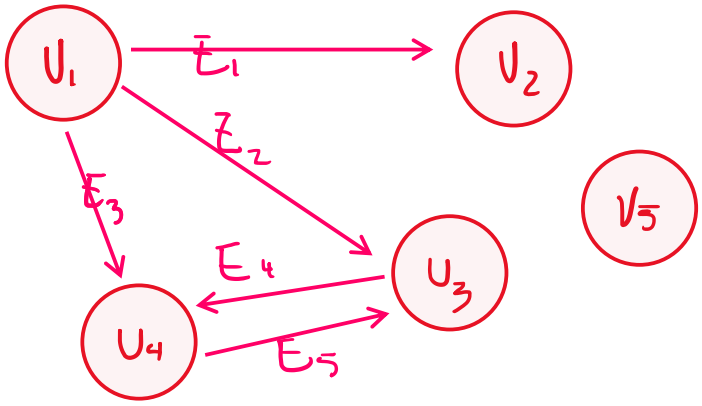

Fig1. Diagrama de grafo dirigido.

En la figura 1 se muestra un grafo dirigido formado por 5 vértices:

V_1, V_2, V_3, V_4, V_5 y se tienen las aristas

$E_1 = (V_1, V_2), E_2 = (V_1, V_3), E_3 = (V_1, V_4), E_4 = (V_4, V_3), E_5 = (V_3, V_4)$

Como se puede observar, las aristas quedan definidas por el orden de los vértices, y puedes haber nodos no conectados.

Algunas definiciones que se usaran, para $X\ y\ Y$ vertices:

$X\ y\ Y$ son adyacentes si existe $E = (X, Y), (Y, X)$

$X\ es\ vecino\ de\ Y$ si y solo si existe $E = (Y,\ X)$

En este proyecto se presenta una implementación de un contenedor que permita a un usuario crear y utilizar grafos dirigidos donde se puedan almacenar datos genéricos.

Análisis del problema

Para poder generar el grafo se necesitan generar los dos componentes:

- Vértices, que almacenen datos genéricos
- Aristas, que “unan” vértices y tengan una etiqueta (label)

Además, debe contar con algunas funciones básicas como

- Crear el grafo
- Crear vértices nuevos, se debe revisar que no exista el vértice que se quiere crear
- Crear aristas, se debe revisar antes que no exista la arista que se busca crear
- Remover vértices, esto implica remover las aristas que surgen y que van hacia ese vértice
- Remover aristas
- Eliminar grafo

Diseño de la solución

La solución propuesta en este trabajo es el uso de listas de adyacencias, aunque la idea es similar a una matriz de adyacencia, solo se genera un vector de vértices y en cada vector una lista donde se guardan sus vértices vecinos. Esta lista está formada de la siguiente manera:

Se crea un vector V, donde se guardan los vértices del grafo

Cada vértice V_i cuenta con una lista de aristas donde se guardan los vértices V_j que son vecinos de V_i como se muestra en el siguiente diagrama

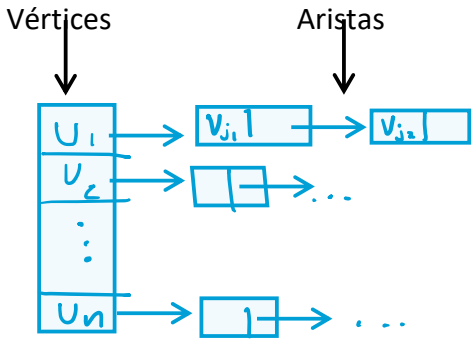

Fig2. Diagrama de propuesta de solución.

De esta manera resulta fácil implementar una solución ya que para agregar un vértice solo se agrega un elemento nuevo en el vector y para las aristas se agrega un elemento nuevo en el vértice correspondiente.

A continuación, se muestra un diagrama con el que se puede representar el grafo dirigido de la figura 1:

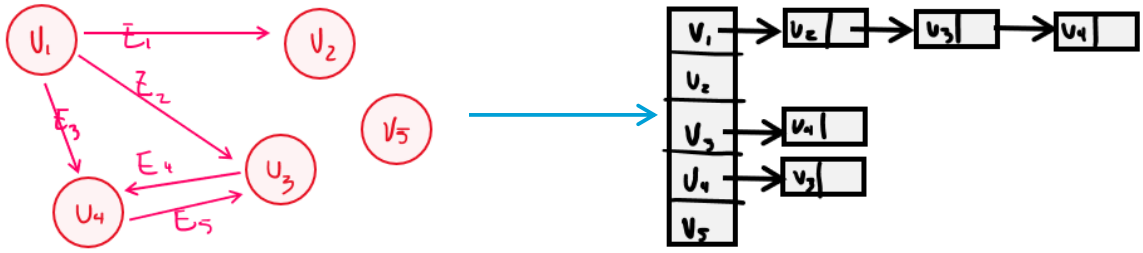

Fig3. Representación del grafo 1 mediante listas de adyacencia.

Descripción de funciones

- ### CreateDGraph
  Crea el grafo de la siguiente manera:

  - Primero crea un vector donde se guardarán vértices
  - Para cada vértice genera una lista de adyacencia

  Regresa el contendedor DGraph.

- ### SizeDGraph
  Regresa el número de vértices.

- ### Adjacent
  Esta función determina si existe una arista entre dos vértices X y Y, como es dirigido hay dos posibilidades: $E_1 = (X, Y), E_2 = (Y, X)$, funciona de la siguiente manera (caso 1):

  - Hace un recorrido buscando el vértice X, si lo encuentra
  - Hace un recorrido en su lista de adyacencia buscando el vertice Y, si lo encuentra entonces hay adyacencia
  - Se repite lo mismo para el caso 2

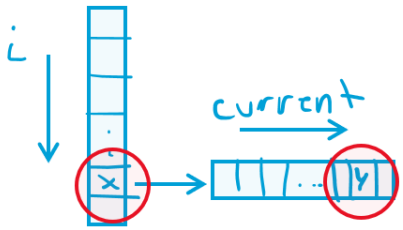

Fig4. Diagrama función de adyacencia.

Si se encontró adyacencia regresa True, si no False.

- Neighbors

  Funciona de manera similar a la funciona anterior, como busca los vecinos de X, basta con buscar X en el vector y todos los elementos en su lista de adyacencia son sus vecinos, con la única diferencia que al encontrarlos cuenta cuantos hay y genera una copia.

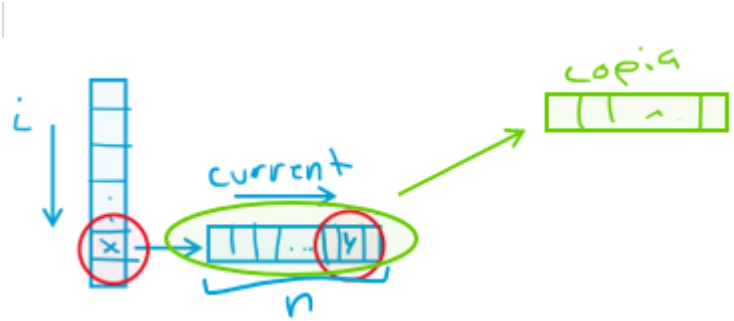

Fig5. Diagrama función neighbors.

Retorna la copia.

- Addvertex

  Primero se recorre el vector para verificar que no exista el vertice que se quiere agregar, en caso de que no se agrega al final con un nodo que es el inicio de su lista de adyacencia

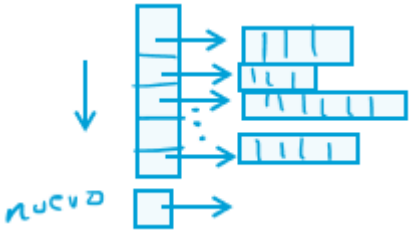

Fig6. Diagrama funcion Addvertex

- RemoVertex

  Se hacen dos cosas:

  - Se recorre el vector para buscar el vértice X, al encontrarlo se eliminan las aristas que salen de X.

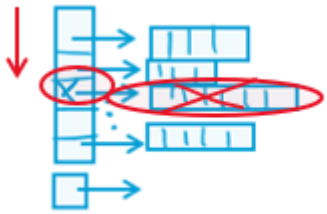

Fig7. Remove Vertex; eliminar aristas que salen de X y luego X.

- En cada vértice se buscan aristas que apunten al vértice X, al encontrarlas se cambian los enlaces en la lista de adyacencia para borrar las aristas que apuntan a X.

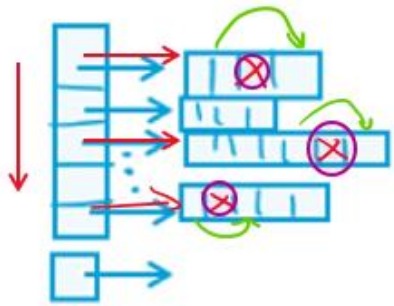

Fig8. RemoveVertex: Eliminar aristas que van a X.

- AddEdge

  Se agregan de manera similar que los vértices; se busca primero el vértice origen, se revisa que no exista la arista en su lista de adyacencia y finalmente agrega.

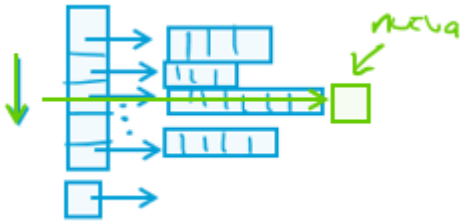

Fig9. Agregar arista

- RemoveEdge

  Primero, se verifica que exista el vértice destino, luego se busca ese mismo en la lista de adyacencias del vértice origen, luego para eliminar la arista se hace algo parecido a lo que se muestra en la figura8: se cambian los enlaces para poder eliminar la arista de la lista.

- GetEdgeLabel

  Se busca la arista en todas las listas y se le cambia la etiqueta.

- DestroyDgraph

  Eliminar las listas y finalmente el vector de nodos.

Pruebas

Se creará el siguiente grafo, cuyos vértices serán caracteres y las aristas enteras.

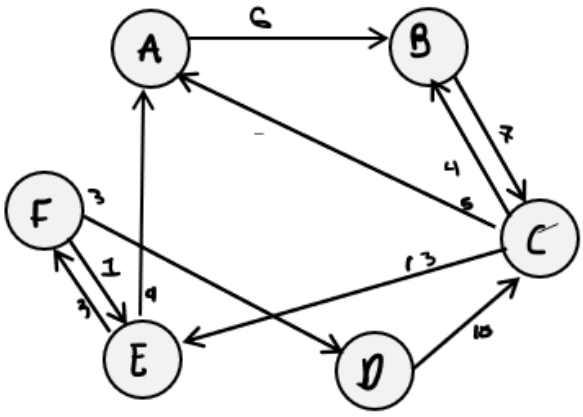

Fig10. Grafo de prueba

Aquí se crea el grafo y se agregan vértices (caracteres de A a F), también se muestra que no se pueden agregar repetidos

```
PS C:\Users\jos3l\OneDrive - ITESO\ITESO\semestre7\memoriadinamica\DG> .\test.exe
Grafo construido
Tama├▒o:: 0 v

Agregar vertice'A'
Tama├▒o:: 1 v

Agregar vertice'B'
Tama├▒o:: 2 v

Agregar vertice'C'
Tama├▒o:: 3 v

Agregar vertice'B'
El vertice ya existe
Tama├▒o:: 3 v

Agregar vertice'D'
Tama├▒o:: 4 v

Agregar vertice'E'
Tama├▒o:: 5 v

Agregar vertice'A'
El vertice ya existe
Tama├▒o:: 5 v

Agregar vertice'E'
El vertice ya existe
Tama├▒o:: 5 v

Agregar vertice'F'
Tama├▒o:: 6 v
```

Ahora se agregan aristas, solo se pueden agregar aristas nuevas y entre vértices que existan, muestra un erro si se intentan agregar aristas repetidas o entre vértices que no existen.

```
Agregar arista '6', 'A' -> 'B'

Agregar arista '7', 'B' -> 'C'

Agregar arista '4', 'C' -> 'B'

Agregar arista '5', 'C' -> 'A'

Agregar arista '10', 'D' -> 'C'

Agregar arista '3', 'E' -> 'F'

Agregar arista '1', 'F' -> 'E'
```

```
Agregar arista '1', 'F' -> 'G'
El segundo vertice no existe.

Agregar arista '1', 'H' -> 'E'
El primer vertice no existe.

Agregar arista '4', 'E' -> 'A'

Agregar arista '3', 'F' -> 'D'

Agregar arista '4', 'E' -> 'A'
La arista ya existe.

Agregar arista '3', 'F' -> 'D'
La arista ya existe.

Agregar arista '13', 'C' -> 'E'
```

Adyacencias (1 son adyacentes, 0 no lo son) y vecinos (Y es vecino de X si y solo si hay una arista de X hacia Y):

```
'A' & 'B' son adyacentes? 1

'C' & 'D' son adyacentes? 1

'A' & 'D' son adyacentes? 0

vecinos de 'A': 'B'

vecinos de 'E': 'F', 'A'

vecinos de 'F': 'E', 'D'
```

Se eliminan los vértices A y F y aristas

```
Eliiminar vertice 'A'
Tama▒o:: 5 v

vecinos de 'A': El vertices no existe.
None

vecinos de 'E': 'F'

Eliiminar vertice 'F'
Tama▒o:: 4 v

vecinos de 'E': Ninguno

Eliiminar vertice 'A'
El vertice no existe.
Tama▒o:: 4 v
```

```
vecinos de 'C': 'B', 'E'

Eliiminar arista'C' -> 'E'

vecinos de 'C': 'B'

Eliiminar arista'E' -> 'C'
La arista no existe

Eliiminar arista'C' -> 'E'
La arista no existe
```

Resultando el siguiente grafo:

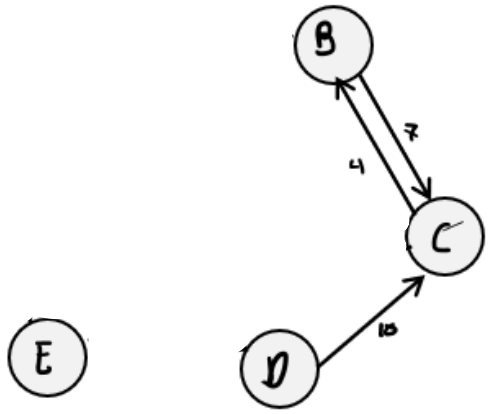

Fig11. Grafo de prueba 2

Finalmente, se muestran las etiquetas del nuevo grafo y se destruye

```
Etiqueta de 'C' -> 'D': La arista no existe

Etiqueta de 'D' -> 'C': 10

Etiqueta de 'A' -> 'Z': El segundo vertice no existe

Etiqueta de 'Z' -> 'A': El segundo vertice no existe
Grafo destruido
```

Conclusiones

Utilizar listas de adyacencia para representar un grafo resulta muy practico y resulta sencillo asociar un grafo (como concepto) con su representación mediante estas listas. Aunque es muy parecido a una matriz de adyacencias, una ventaja que tiene sobre estas es que solo se necesita la memoria que se utilizara a diferencia de las matrices donde al generar un vértice se necesitan generar espacios para sus aristas, aunque no se utilicen. Una desventaja que encontré de esta implementación es que para crear/buscar/eliminar vértices o aristas se deben recorrer todas las listas hasta encontrar lo que se busca.

Referencias

Cormen, T. C., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms.*

Garro, V. (s.f.). *Representacion de grafos*. Obtenido de https://algoritmos.victorgarro.com/grafos/Grafos3.htm