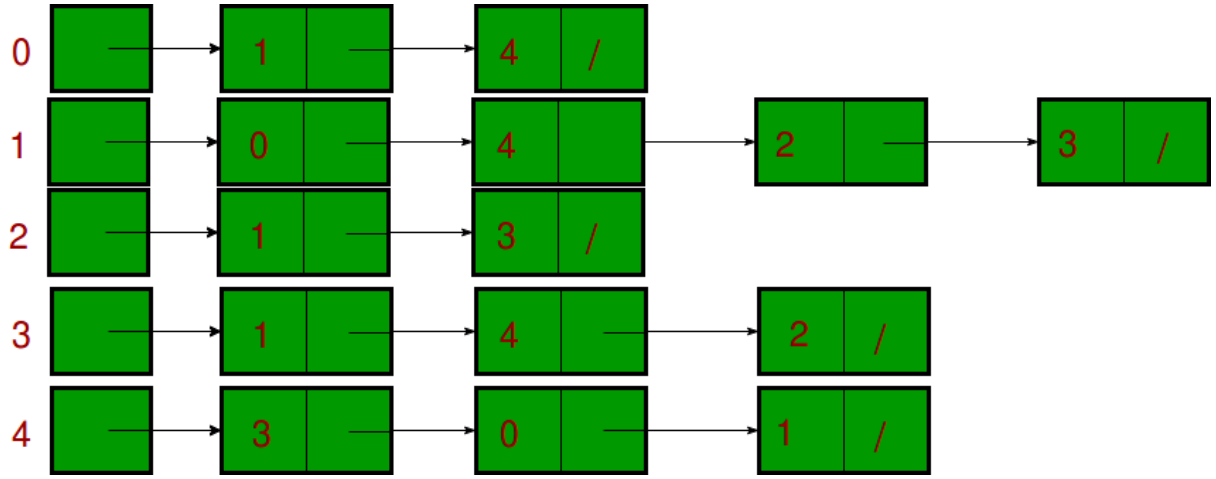
Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887

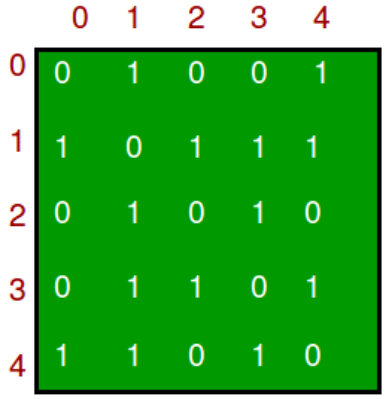
Introducción

Muchos problemas del mundo real se resuelven mediante gráficos. Las redes se representan mediante gráficos. Los gráficos también se emplean en sitios de redes sociales como LinkedIn y Facebook. En Facebook, por ejemplo, cada individuo se representa como un vértice (o nodo). Cada nodo es una estructura que contiene información como la identificación, el nombre, el género, la ubicación de una persona, etc.

Hay varias maneras de representar grafos. La primera es utilizar listas de adyacencia, las cuales se ven de la siguiente manera:



Por el lado de la matriz de adyacencia, esta es una matriz que contiene un nodo por índice, entonces los valores en los que haya ejes se representan como 1 en el índice de cada uno de los nodos.



Basándonos en la anterior representación podríamos decir que el nodo 1 se conecta con el nodo 0, y con el nodo 1, también el nodo dos con el nodo 1 y así con todos los otros valores encendidos en la matriz.

Cíclos:

Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887
Un ciclo en un grafo se define como una conexión entre un nodo y algún ancestro del nodo. Para encontrar ciclos en grafos se utilizan los siguientes pasos:

- Crear un gráfico con el número especificado de aristas y vértices.
- Crear una función recursiva que establezca el índice o vértice actual, la pila de visitas y la recursividad en sus valores iniciales.
- Marcar el nodo actual como visitado y el índice en la pila de recursividad también.
- Encontrar todos los vértices que no se han visitado pero que están cerca del nodo actual.
- Llamar a la función para esos vértices de forma recursiva. Si la función recursiva devuelve verdadero, devolver verdadero.
- Devolver verdadero si los vértices cercanos ya están etiquetados en la pila de recursividad.

```
bool* visited = new bool[V];

   bool* recStack = new bool[V];

   for (int i = 0; i < V; i++)

   {

       visited[i] = false;

       recStack[i] = false;

   }
```

En esta parte del código se marcan los vértices como no visitados en ambos arreglos.

```
   for (int i = 0; i < V; i++)

       if (isCyclicUtil(i, visited, recStack))

       {

           return true;

       }
```

Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887

```
   return false;
```

Esta es la rutina base donde se pregunta si hay un ciclo a cada valor, esto se hace con el fin de encontrar todos los ciclos. Si no se revisará cada valor arbitrariamente se podría detener el algoritmo en un ciclo local.

```
bool Graph::isCyclicUtil(int v, bool visited[], bool* recStack)

{

   if (visited[v] == false)

   {

       // Mark the current node as visited and part of recursion stack

       visited[v] = true;

       recStack[v] = true;



         //Iterators are used to point at the memory addresses of STL containers. They are primarily used in sequence of numbers, characters etc. They reduce the complexity and execution time of program.

       list<int>::iterator i;

       for (i = adj[v].begin(); i != adj[v].end(); ++i)

       {

               if (!visited[*i] && isCyclicUtil(*i, visited, recStack))   // ya lo visite y ademas pude regresar y por lo que encontre un ciclo

           {

               return true;

           }

               else if (recStack[*i])   // esta esta visitado y lo encontre en el recstack, por lo tango, pos ya
```

Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887

```
            {

                return true;

            }

        }



    }

    recStack[v] = false;  // remove the vertex from recursion stack

    return false;

}
```

La función isCyclicUntil primero revisa si el valor está visitado, si no lo está, lo marca como visitado.
Los casos clave son los siguientes:
Cuando ya se visitó y pudo regresar. Al cumplirse este condicional retorna verdadero al haber encontrado un cíclo. Se sabe que se pudo regresar ya que se hizo una llamada recursiva en el condicional.
El otro caso es cuando se encuentra en el recstack, en ese caso ss retorna verdadero ya que se sabe que hay un cíclo.
Cuando se acaba el recorrido, se remueve el vertice del recStack.

```
void loadGraph()
{
    string line;
    ifstream myfile("casos_prueba.txt");

    if (myfile.is_open())
    {

        getline(myfile, line);
        std::cout << "Casos : "  << line << std::endl;

        int casos = toInt(line[0]);
        cout << casos <<endl;
        for (int i =0; i< casos; i+=1)
        {
            getline(myfile, line);
```

Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887

```
            auto V = toInt(line[0]);
            cout << V << endl;
            auto g = Graph(V);
            getline(myfile, line);



            auto ejes = toInt(line[0]);
            cout << ejes << endl;
            for (int j =0; j<ejes; j+=1)
            {
                getline(myfile, line);
                cout << line[0] << line[2] << endl;
                g.addEdge(toInt(line[0]), toInt(line[2]) );


            }
            cout<<i<<endl;
            if (g.isCyclic())

                    cout << "Graph contains cycle" << endl;

                else

                    cout << "Graph doesn't contain cycle" << endl;

        }

        myfile.close();

    }

    else
    {
        std::cout << "Unable to open file";


}
}
```

Por el lado de la función loadgraph, se utiliza el código que implementamos en veces anteriores para obtener la fecha y se adaptó para cargar los casos de prueba. El input se veía de la siguiente forma:
3 #Número de casos prueba
5 # Número de Vértices en prueba 1
7 # Número de conexiones

Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887
0,1
1,0
0,2
1,2
2,0
2,3
3,3
5
7
0,1
1,0
0,2
1,2
2,0
2,3
3,3
5
7
0,1
1,0
0,2
1,2
2,0
2,3
3,3
Utilizando estos datos obtuvimos los siguientes resultados:
Casos : 3
3
5
7
01
10
02
12
20
23
33
0
Graph contains cycle
5
7
01
10
02
12
20
23

Reflexión 4.1
Juan Pablo Montoya Estevez
A01251887
33
1
Graph contains cycle
5
7
01
10
02
12
20
23
33
2
Graph contains cycle

Conclusión:

La implementación de grafos es de alta importancia a la vida cotidiana. Esta estructura de datos se está utilizando para resolver desde problemas triviales hasta para modelar situaciones físicas complejas (Teoría de Wolfram). El estudio de los grafos ha permitido desarrollos muy significativos en el área computacional. Esta actividad me ayudó a comprender la implementación de los grafos así como los algoritmos fundamentales para manipularlos.