Tecnológico de Monterrey
Edgar Cruz Vázquez - (A01730577)
TC1031.11: Programación de estructura de datos y algoritmos.

Act 4.1 – Grafo: sus representaciones y sus recorridos.

GRAFO

Los grafos dentro del ámbito de las ciencias de la computación se tratan de una estructura de datos, un tipo de datos abstractos (TAD), que consiste en un conjunto de nodos (también conocidos como vértices) y un conjunto de arcos (aristas) en las cuales se establecen relaciones entre los nodos.

Los grafos se podrían definir informalmente como G = (V, E), siendo los elementos de V los vértices, y los elementos de E, las aristas (edges en inglés). Formalmente, un grafo G se define como un par ordenado, G = (V, E), donde V es un conjunto finito y E es un conjunto que consta de dos elementos de V.

Los grafos se pueden clasificar en dos grupos:

- Dirigidos
- No dirigidos.

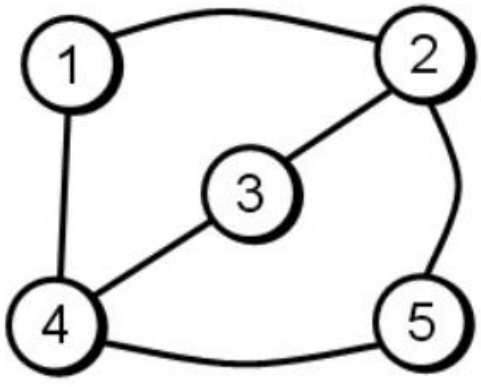
Un grafo dirigido cada arco está representado por un par ordenado de vértices, de forma que representan dos arcos diferentes.

En un grafo no dirigido el par de vértices que representan un arco no está ordenado.

FORMAS DE REPRESENTACIÓN

Existen dos formas de representar grafos: con una matriz de adyacencia y listas de adyacencia.

Matriz de adyacencias: Se asocia cada fila y cada columna a cada nodo del grafo, siendo los elementos de la matriz la relación entre los mismos, tomando los valores de 1 si existe la arista y 0 en caso contrario.

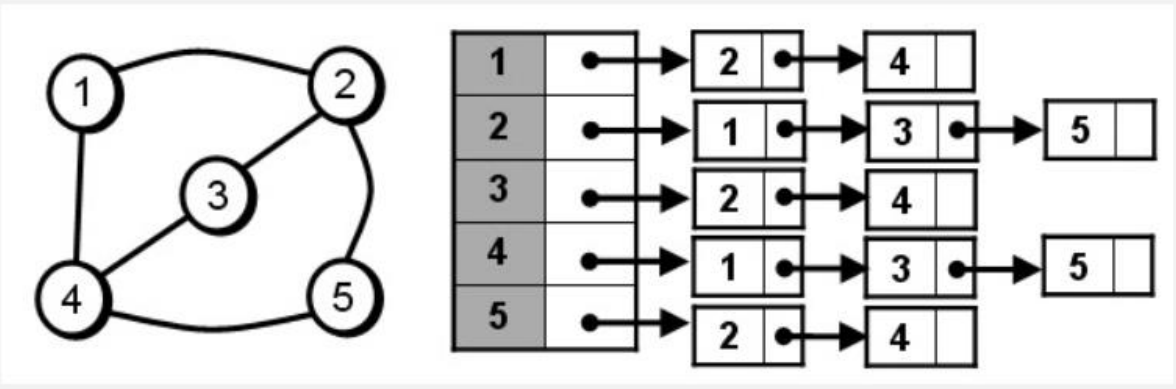


M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Tecnológico de Monterrey
Edgar Cruz Vázquez - (A01730577)
TC1031.11: Programación de estructura de datos y algoritmos.

Act 4.1 – Grafo: sus representaciones y sus recorridos.

Listas de adyacencias: Se asocia a cada nodo del grafo una lista que contenga todos aquellos nodos que sean adyacentes a él.



Gracias a la teoría de grafos se pueden resolver diversos problemas como por ejemplo la síntesis de circuitos secuenciales, contadores o sistemas de apertura. En áreas como el dibujo computacional, en la mayoría de las áreas de ingeniería. Los grafos se utilizan para modelar trayectos como el de una línea de autobús a través de las calles de una ciudad, en el que podemos obtener caminos óptimos para el trayecto aplicando diversos algoritmos, ha servido hasta de inspiración para las ciencias sociales, en la biología, en el desarrollo de IA, etc.

Al momento de declarar la matriz de adyacencia este tiende a tener una complejidad de de memoria de O(n^2) ya sea dirigido o no dirigido pero en tiempo es O(1), mientras que en las listas de adyacencias en grafos no dirigidos la complejidad de memoria puede ser O(n+2a) y en dirigidos O(n+a), en cuestión de recorridos la de BFS si se representa con una lista de adyacencia seria O(V + E) y si fuera con una matriz de adyacencia sería O(V^2) y para el DFS son las mismas complejidades.

FUNCIONES:

```cpp
void Graph::BFS(Graph const& graph, int v, vector<bool>& discovered)
{
    queue<int> q;

    discovered[v] = true;

    q.push(v);

    while (!q.empty())
    {
        v = q.front();
        q.pop();
        cout << v << " ";

        // do for every edge (v -> u)
        for (int u : graph.adjList[v])
            if (!discovered[u])
            {
                discovered[u] = true;
                q.push(u);
            }
    }
}
```

Tecnológico de Monterrey
Edgar Cruz Vázquez - (A01730577)
TC1031.11: Programación de estructura de datos y algoritmos.

Act 4.1 – Grafo: sus representaciones y sus recorridos.

El BFS o (Breath-first search), basa el orden de visita de los nodos del grafo en un queue, incorporándole en cada paso los adyacentes al nodo actual, esto implica que se visitarán todos los hijos de un nodo antes de proceder con sus demás descendientes.

```
void Graph::DFS(int start, vector<bool>& visited)
{

    // Print the current node
    cout << start << " ";

    // Set current node as visited
    visited[start] = true;

    // For every node of the graph
    for (int i = 0; i < v; i++) {

        // If some node is adjacent to the current node
        // and it has not already been visited
        if (adj[start][i] == 1 && (!visited[i])) {
            DFS(i, visited);
        }
    }
}
```

El DFS o (Depth-first search), basa el orden de visita de los nodos del grafo en una manera recursiva, buscando e imprimiendo en cada paso los adyacentes al nodo actual, esto hace que agote los nodos accesibles desde un hijo antes de proceder con sus hermanos.

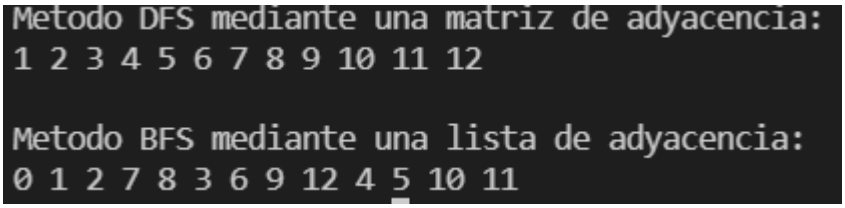
	CASOS DE PRUEBA ACT 3.4				
No.	Precondiciones	Resumen	Pasos	Resultado Esperado	Resultado obtenido
1	Tener el código y tener un grafo válido	El usuario usa la función loadGraph(), e introduce vértices y arcos	1. Iniciar el programa. 2. Llamar a la función e insertar los nodos y edges válidos.	Una vez insertado la función creará los grafos e imprimirá	El programa crea los grafos y imprime ambos recorridos el BFS y el DFS.

Tecnológico de Monterrey
Edgar Cruz Vázquez - (A01730577)
TC1031.11: Programación de estructura de datos y algoritmos.

Act 4.1 – Grafo: sus representaciones y sus recorridos.

		mayores de 0		ambos recorridos	
2	Tener el código y tener un grafo inválido.	El usuario llama la función loadGraph() e inserta un número de vértices y aristas alguna siendo iguales a 0	1. Correr el programa. 2. Introducir en la función número de vértices o aristas inválidos.	El programa debería imprimir un texto diciendo que no puede hacer grafos con tales datos.	El programa imprime el texto avisando que no puede hacer el grafo con tales datos enviados.

CASO 1:



CASO 2: