Unidad 1 - Parte 2 – Representación de Grafos

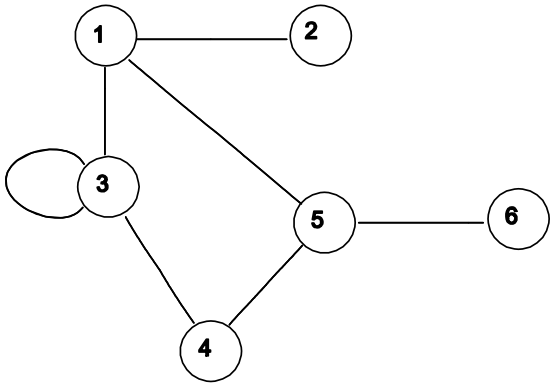
La representación matemática más común para grafos es la denominada matriz de adyacencia.

Si G = (V,A) y |V| = n, se tendrá una matriz de n filas y n columnas, una por cada vértice.

En cada lugar de la matriz, para el caso general de multigrafos (es decir, grafos que admiten más de una arista entre dos vértices dados) se coloca la cantidad de aristas que hay entre el par de vértices correspondientes.

En particular si no se trata de multigrafos, es decir que se admite a lo sumo una única arista entre dos vértices cualesquiera (que es el tipo de grafos que se ha presentado en el curso) la matriz contendrá sólo ceros y unos.

Ejemplo:



	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	0	0	0	0
3	1	0	1	1	0	0
4	0	0	1	0	1	0
5	1	0	0	1	0	1
6	0	0	0	0	1	0

Matriz de adyacencia

Representación en Lenguajes de Programación

Cuando se quiere representar grafos en los lenguajes de programación, básicamente se suele trabajar con dos representaciones posibles:

- Matriz de adyacencia
- Listas de adyacencia.

Matriz de adyacencia

Esta representación en un lenguaje de programación corresponde a la representación matemática vista en el apartado anterior. Se manejan dos variantes para esta representación:

1. Matriz de adyacencia de enteros.

```
const int N = /* cantidad de vertices */
typedef int Grafo[N][N];
```

En esta representación si A es una variable de tipo Grafo se tiene:

`A[i][j] = 0` si no existe $a \in$ A tal que $a = \{i,j\}$.
`A[i][j] = p` si existen p aristas entre los vértices *i* y *j*.

2. Matriz de adyacencia de booleanos

Si el grafo G no es un multigrafo (o sea, existe a lo sumo una arista entre cada par de vértices), es posible usar la representación anterior con p = 1 si existe $a \in$ A tal que $a = \{u,v\}$.

Sin embargo puede resultar más cómodo utilizar booleanos en lugar de 0's y 1's, de la siguiente forma:

`A[i][j] = true` si existe $a \in$ A tal que $a = \{i,j\}$.
`A[i][j] = false` en otro caso.

La representación de matriz de adyacencia es útil en aquellos algoritmos sobre grafos que necesiten saber frecuentemente si una determinada arista existe. Sin embargo en grafos con pocas aristas hay mala utilización de la memoria, ya que se destinan muchas celdas en memoria con valor cero, representando aristas que no existen en el grafo.

Observar que para los grafos vistos (que son no dirigidos), hay información redundante en la matriz pues tenemos el mismo dato en la celda [i,j] y en [j,i]. Decimos que un grafo es dirigido cuando las aristas del mismo tienen un sentido asociado, en cuyo caso se representa a las aristas mediante pares ordenados en vez de mediante conjuntos a nivel de la definición matemática de grafo.

Listas de Adyacencia

Esta representación consiste en un arreglo de N posiciones (una por cada vértice), desde el cual se apunta a listas de adyacencia. Una lista de adyacencia para un determinado vértice *i*, consiste en una lista conteniendo los identificadores de los vértices que son adyacentes al vértice *i*.

Esta representación es de utilidad cuando el grafo cuenta con pocas aristas o cuando no interesa saber con mucha frecuencia si una determinada arista existe, ya que consume memoria solamente para representar aquellas aristas que efectivamente existen en el grafo.

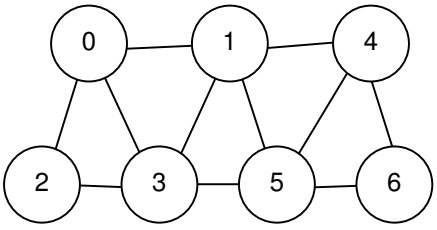
```
typedef struct nodo{ int vert;
                     nodo * sig;
                   }

typedef nodo * ListaAdy;

const int N = /* cantidad de vertices */
typedef ListaAdy Grafo [N];
```

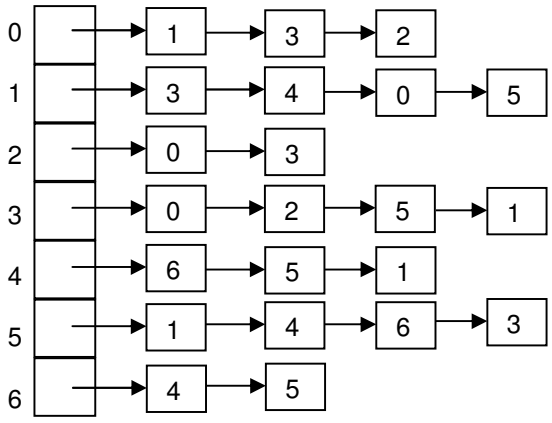
En un grafo no dirigido, tenemos nuevamente información redundante en las listas de adyacencia. Por ejemplo, para representar la arista {1,2} tendremos que en la lista de adyacencia correspondiente al vértice 1 hay un nodo con el valor 2. A su vez, en la lista de adyacencia correspondiente al vértice 2 hay un nodo con el valor 1.

Si el grafo fuera dirigido, es decir en el caso en que sí importe el sentido de las aristas, esta redundancia de información no ocurre. Por ejemplo, si tenemos la arista (1,2) (nótese la representación mediante un par ordenado, donde sí importa el sentido) pero no tenemos la arista (2,1) no tendremos información redundante, ya que en la lista de adyacencia correspondiente al vértice 1 habrá un nodo con el valor 2 pero en la lista de adyacencia correspondiente al vértice 2 no habrá un nodo con el valor 1.

El siguiente ejemplo ilustra la representación mediante listas de adyacencia para el caso de un grafo no dirigido (que es el tipo de grafos que se ha presentado en el curso). Obsérvese que la forma en que se ordenan los nodos en las listas de adyacencia es, en principio, irrelevante. Lo importante es que en cada lista se encuentren efectivamente todos los identificadores de los vértices adyacentes a un vértice dado, no importando tanto en qué orden se guardan dentro de la lista.

Ejemplo:





Listas de adyacencia