

REPRESENTACIÓN DE GRAFOS

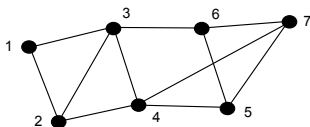
Tecnología Digital V: Diseño de Algoritmos
Universidad Torcuato Di Tella

Nos interesa programar algoritmos que trabajen sobre grafos. Para esto, debemos **representar** adecuadamente grafos (cómo lo hacemos?).

Nos interesa programar algoritmos que trabajen sobre grafos. Para esto, debemos **representar** adecuadamente grafos (cómo lo hacemos?).

Definición

La **matriz de adyacencia** de un grafo G es una matriz $A = (a_{ij}) \in \mathbb{R}^{|V| \times |V|}$ tal que $a_{ij} = 1$ si $ij \in E$ y $a_{ij} = 0$ en caso contrario.



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

```
class Grafo
{
    public:

    Grafo(int n);
    ~Grafo();

    void agregarArista(int i, int j);
    void eliminarArista(int i, int j);
    bool existeArista(int i, int j);

    private:

    int _vertices;
    bool** _adj;
}
```

```
// Constructor
Grafo::Grafo(int n)
{
    _vertices = n;
    _adj = new bool*[n];

    for (int i=0; i<n; ++i)
        _adj[i] = new bool[n];

    for (int i=0; i<n; ++i)
        for (int j=0; j<n; ++j)
            _adj[i][j] = false;
}

// Destructor
Grafo::~Grafo()
{
    for (int i=0; i<_vertices; ++i)
        delete [] _adj[i];

    delete [] _adj;
}
```

```
// Agrega una arista
void Grafo::agregarArista(int i, int j)
{
    _adj[i][j] = _adj[j][i] = true;
}

// Elimina una arista
void Grafo::eliminarArista(int i, int j)
{
    _adj[i][j] = _adj[j][i] = false;
}

// Consulta por arista
bool Grafo::existeArista(int i, int j)
{
    return _adj[i][j];
}
```

- Ventajas de esta representación:

1. Agregar una arista: $O(1)$.
2. Eliminar una arista: $O(1)$.
3. Consultar si existe arista: $O(1)$.

- Desventajas de esta representación:

1. Agregar un vértice: $O(n^2)$.
2. Eliminar un vértice: $O(n^2)$.
3. Obtener todos los vecinos de un vértice: $O(n)$.

Definición.

La **matriz de incidencia** de un grafo G es una matriz $M = (m_{ij}) \in \mathbb{R}^{|V| \times |E|}$ tal que $m_{ie} = 1$ si el vértice i es uno de los extremos de la arista e , y $m_{ie} = 0$ en caso contrario.

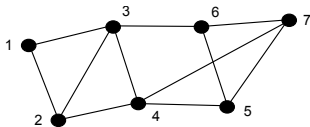
- En nuestro ejemplo, $E = \{12, 13, 23, 24, 34, 36, 45, 47, 56, 57, 67\}$.

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Pregunta

¿Es una representación conveniente?

- Una tercera opción es por medio de **listas/conjuntos de vecinos** asociados con cada vértice:



1	→	{2, 3}
2	→	{1, 3, 4}
3	→	{1, 2, 4, 6}
4	→	{2, 3, 5, 7}
5	→	{4, 6, 7}
6	→	{3, 5, 7}
7	→	{4, 5, 6}

```
class Grafo
{
    ...

    private:
        vector< set<int> > _vecinos;
}

// Constructor
Grafo::Grafo(int n)
{
    for(int i=0; i<n; ++i)
        _vecinos.push_back(set<int>());
}
```

```
// Agrega una arista
void Grafo::agregarArista(int i, int j)
{
    _vecinos.at(i).insert(j);
    _vecinos.at(j).insert(i);
}

// Elimina una arista
void Grafo::eliminarArista(int i, int j)
{
    _vecinos.at(i).erase(j);
    _vecinos.at(j).erase(i);
}

// Consulta por arista
bool Grafo::existeArista(int i, int j)
{
    return _vecinos.at(i).find(j) != _vecinos.at(i).end();
}
```

- Ventajas de esta representación:

1. Obtener todos los vecinos de un vértice: $O(1)$.
2. Agregar un vértice: $O(n)$ en el peor caso, $O(1)$ amortizado.

- Desventajas de esta representación:

1. Agregar una arista: $O(\log n)$ (depende de la implementación de set).
2. Eliminar una arista: $O(\log n)$.
3. Consultar si existe arista: $O(\log n)$.
4. Eliminar un vértice: $O(n^2)$.

Operación	Matriz de adyacencia	Conjunto de Vecinos
Complejidad espacial	$O(n^2)$	$O(E)$
Agregar vértice	$O(n^2)$	$O(n)/O(1)$ amortizado
Eliminar vértice	$O(n^2)$	$O(n^2)$
Agregar arista	$O(1)$	$O(\log n)$
Eliminar arista	$O(1)$	$O(\log n)$
Consultar arista	$O(1)$	$O(\log n)$
Obtener vecinos	$O(n)$	$O(1)$

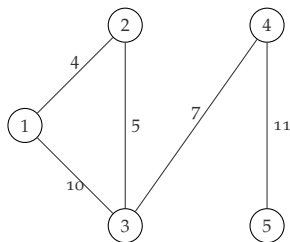
Pregunta

¿Cómo representamos el grafo si cada arista tiene un peso asociado?

Matriz de adyacencia

$$A = \begin{bmatrix} 0 & 4 & 10 & 0 & 0 \\ 4 & 0 & 5 & 0 & 0 \\ 10 & 5 & 0 & 7 & 0 \\ 0 & 0 & 7 & 0 & 11 \\ 0 & 0 & 0 & 11 & 0 \end{bmatrix}$$

Grafo



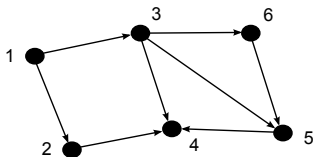
Lista de vecinos

- 1 → {{2 → 4}, {3 → 10}}
- 2 → {{1 → 4}, {3 → 5}}
- 3 → {{1 → 10}, {2 → 5}, {4 → 7}}
- 4 → {{3 → 7}, {5 → 11}}
- 5 → {{4 → 11}}

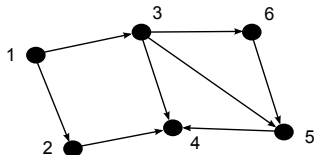
Definición

Un **grafo dirigido** (o digrafo) es un par $D = (N, A)$ tal que ...

1. N es un conjunto finito (llamado el conjunto de **nodos** de D), y
2. $A \subseteq \{(i, j) \in V \times V : i \neq j\}$ es un conjunto de pares ordenados, llamados **arcos**.



- A diferencia de los grafos (no dirigidos), ahora las aristas son **pares ordenados**.



- Diferenciamos entre el **grado de entrada** y el **grado de salida** de cada nodo.

1. $d^-(i) = |N^-(i)| = |\{j \in N : (j, i) \in A\}|$.
2. $d^+(i) = |N^+(i)| = |\{j \in N : (i, j) \in A\}|$.

- Un camino entre dos nodos es una secuencia de arcos entre ellos, respetando el sentido de los arcos.

Pregunta

¿Cómo representamos un grafo dirigido?