

Estructuras de Datos

UNIDAD 2: Grafos – parte 1

LABORATORIO Semana 7

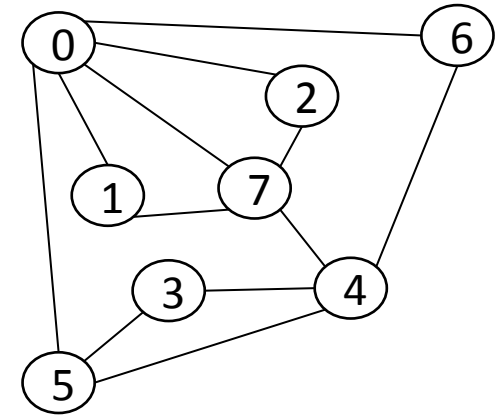
Contenido

- TDA Grafo – Representaciones (Lista de adyacencia y Matriz de adyacencia)
- TDA Grafo – Operaciones
- Tareas

TDA Grafo – Representación

Matriz de adyacencia

- Una representación de un grafo mediante una matriz de adyacencia es una matriz V por V de valores booleanos, con la entrada en la fila v y la columna w definida como 1 si hay una arista que conecta el vértice v y el vértice w en el gráfico, y 0 en otro caso.
- Con una matriz de adyacencia, es posible determinar eficientemente si hay o no una arista desde el vértice i al vértice j , simplemente verificando si la fila i y la columna j de la matriz es diferente de cero.
- Para un grafo no dirigido, si hay una entrada en la fila i y la columna j , entonces también debe haber una entrada en la fila j y la columna i , por lo que la matriz es simétrica.
- Pregunta: ¿Como se ve la matriz de adyacencia en el caso de un grafo denso y disperso?



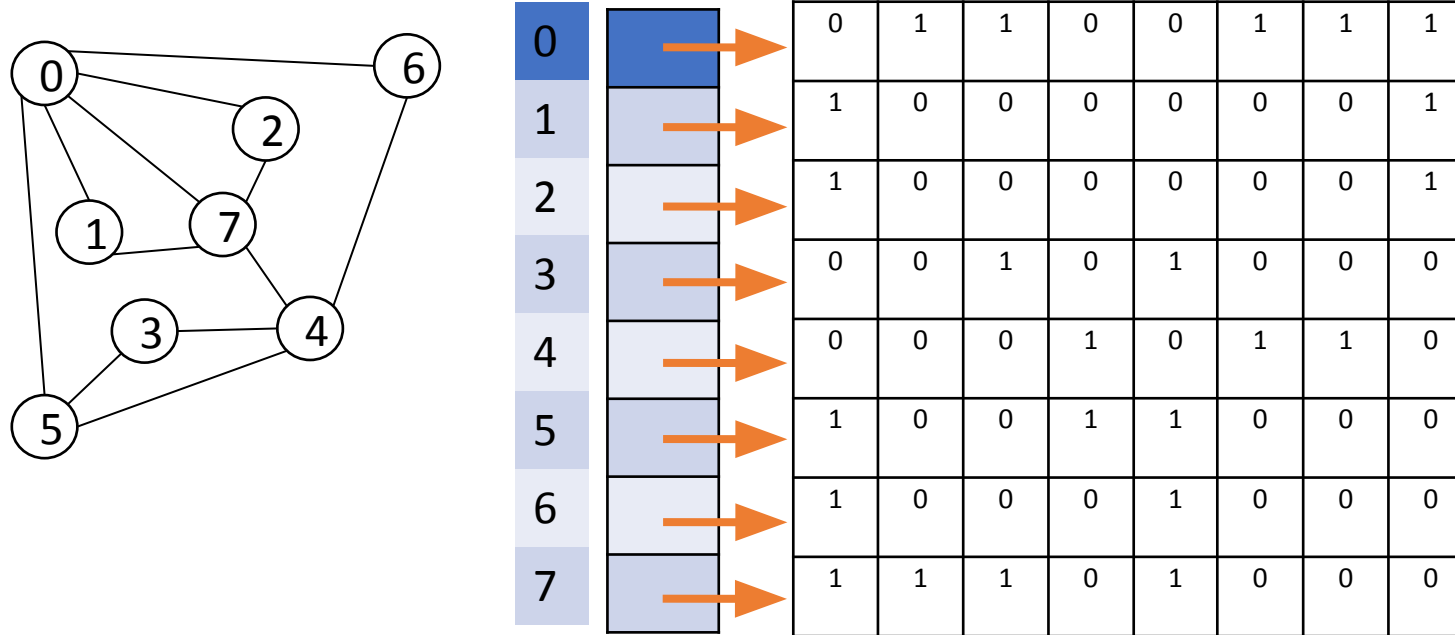
Matriz de 8 x 8

	0	1	2	3	4	5	6	7
0	0	1	1	0	0	1	1	1
1	1	0	0	0	0	0	0	1
2	1	0	0	0	0	0	0	1
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	1	0
5	1	0	0	1	1	0	0	0
6	1	0	0	0	1	0	0	0
7	1	1	1	0	1	0	0	0

TDA Grafo – Representación

Matriz de adyacencia

- Grafo arreglo de arreglos.



TDA grafo no dirigido

VALORES: Dos valores enteros y una matriz de enteros.

Arista: struct (o registro)

v: Entero //vértice 1

w: Entero //vértice 1

Grafo: struct (o registro)

n: Entero; // cantidad de vértices

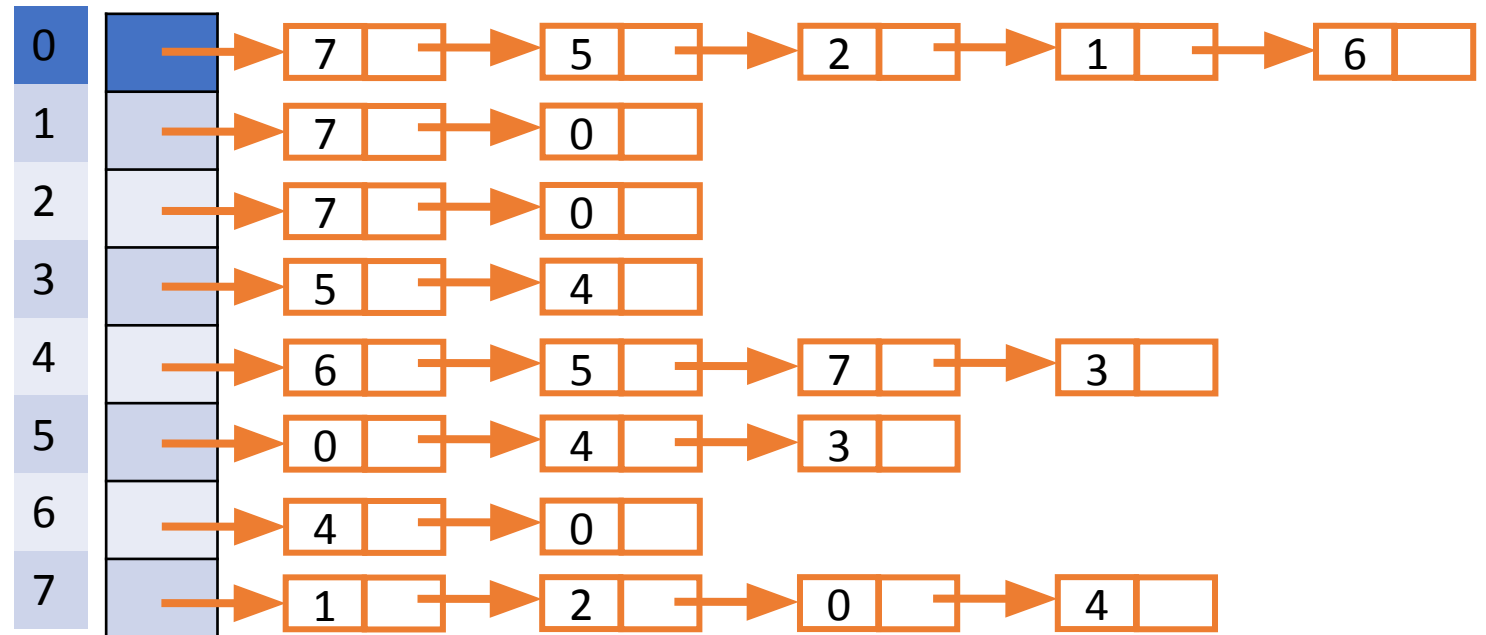
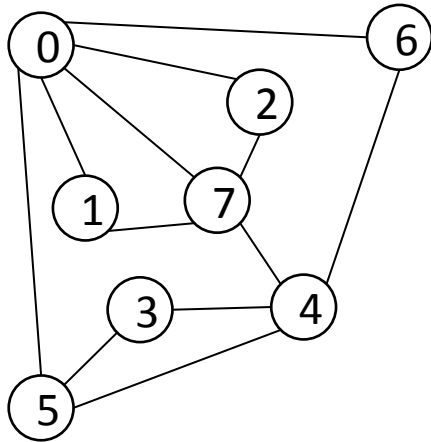
m: Entero // cantidad aristas;

**Madj: Entero //matriz de adyacencia

TDA Grafo – Representación

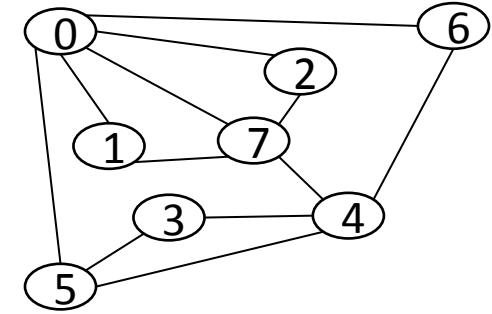
Lista de adyacencia

- Es posible representar un grafo mediante una matriz de listas enlazadas, llamadas listas de adyacencia. Se mantiene una lista enlazada para cada vértice, con un nodo para cada vértice conectado a ese vértice. Para un grafo no dirigido, si hay un nodo para j en la lista de i , entonces debe haber un nodo para i en la lista j .



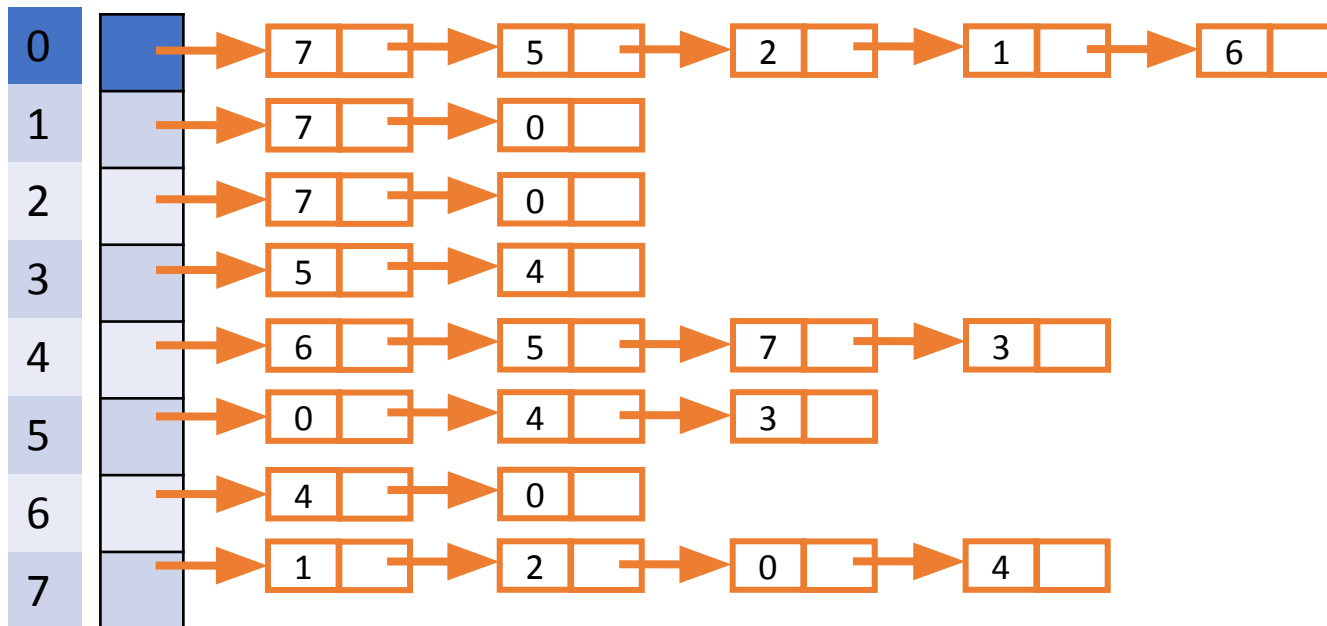
TDA Grafo – Representación

Lista de adyacencia



- Grafo arreglo con n vértices. Cada vértice apunta a una lista.

ArrV[8]:



TDA grafo no dirigido

Nodo: struct (o registro)

info: int

*sig: Nodo

Lista: struct (o registro)

*head: Nodo

Grafo: struct (o registro)

n: Entero; // cantidad de vértices

m: Entero // cantidad aristas;

*ArrV: Lista //arreglo con los vértices
que apuntan a una lista

TDA Grafo – Representación

Lista de adyacencia

- La principal ventaja de la representación mediante listas de adyacencia sobre la representación mediante matriz de adyacencia es que siempre utiliza un espacio proporcional a $E + V$, en oposición a V^2 en la matriz de adyacencia.
- La principal desventaja es que verificar la existencia de aristas específicas puede tomar un tiempo proporcional a V , a diferencia del tiempo constante en una matriz de adyacencia.

TDA Grafo – *Operaciones*

- ***crear_grafo***(número vértices) → grafo
- ***insertar_arista***(grafo, arista) → void
- ***mostrar_grafo***(grafo) → void (muestra la matriz o lista de c/vértice según corresponda)
- ***remover_arista***(grafo, arista) → void
- ***pertenece_arista***(grafo, arista) → booleano
- ***crear_arista***(vértice, vértice) → arista
- ***obtener_aristas***(grafo) → arreglo de aristas
- ***obtener_grado_vertice***(grafo, vértice) → entero
- ***obtener_adyacentes_vertice***(grafo, vértice) → arreglo de vértices adyacentes

TDA Grafo – otras *operaciones*

- *Verifica si un conjunto de vértices es una clique*
- *Verificar si un grafo es k regular (indicar el valor de k)*
- *Generar el grafo complemento (guardar en archivo)*
- *Verificar si un grafo es simple*
- *Obtener el grado de cada nodo*
- *Obtener grado máximo y mínimo*
- *Verificar si un conjunto ordenado de vértices es un tour*

TDA Grafo — Cargar grafo usando matriz de adyacencia

lea desde un archivo de texto un conjunto de aristas de un grafo no dirigido y construya una representación de un grafo mediante matriz de adyacencia

```
#include <stdio.h>
#include "TDA-Grafo-M.h"
int main() {
    char nombre[200];
    int vertices, vert1, vert2;

    printf("Ingrese el nombre del archivo a leer\n");
    scanf("%s", nombre);
    FILE *fp;
    fp = fopen(nombre, "r");
    fscanf(fp, "%d", &vertices);

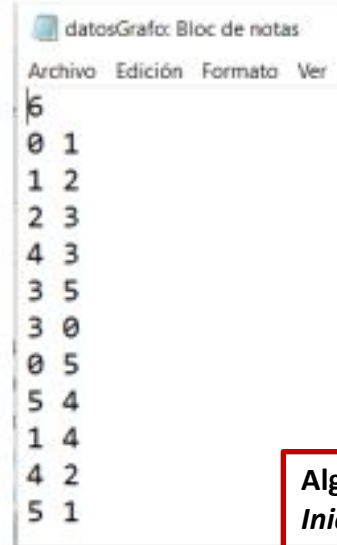
    Grafo *graf = crear_grafo(vertices);

    while (fscanf(fp, "%i %i", &vert1, &vert2) == 2){
        a: Arista
        a = crear_arista(vertice1, vertice2)
        insertar_arista(graf, a);
    }
    return 0;
}
```

(*) La función main está escrita en lenguaje C

(*) Las operaciones crear_grafo, inicializar_matriz, crear_arista e insertar_arista están escritas con pseudocódigos

Ejemplo archivo entrada:



```
datosGrafo: Bloc de notas
Archivo Edición Formato Ver
6
0 1
1 2
2 3
4 3
3 5
3 0
0 5
5 4
1 4
4 2
5 1
```

Arista: struct

v: Entero //vértice 1 de la arista

w: Entero //vértice 2 de la arista

Grafo: struct

n: Entero; // cantidad de vértices

m: Entero // cantidad aristas;

****Madj:** Entero //matriz de ady

Alg. *crear_Grafo(numV: Entero): Grafo

Inico

*G: Grafo

G = asignar memoria

G->n = numV

G->m = 0

G->Madj = inicializar_matriz(numV, numV, 0)

retornar G

Fin

Alg. crear_arista(ver1, ver2: Entero): Arista

Inico

e: Arista

e.v = ver1

e.w = ver2

retornar e

Fin

Alg. insertar_arista(*g: Grafo, e: Arista)

Inico

Si (g->Madj[e.v][e.w] == 0

g->m = g->m + 1

g->Madj[e.v][e.w] = 1

g->Madj[e.w][e.v] = 1

Fin

Alg. **inicializar_matriz(f, c, val: Entero): Entero

Inico

****M, i, j:** Entero

M = asignar memoria para **f** filas

PARA i = 0 HASTA f HACER

M[i] = asignar memoria para **C** columnas

PARA i = 0 HASTA f HACER

PARA j = 0 HASTA c HACER

M[i][j] = val //val es igual a 0

retornar M

Fin

TDA Grafo – Cargar grafo usando lista de adyacencia

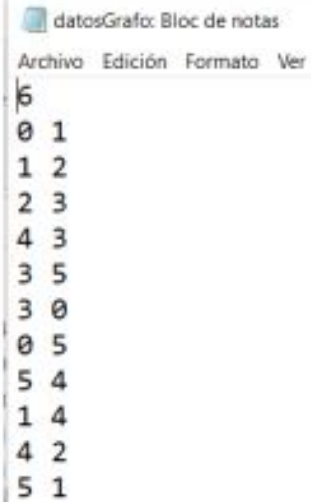
Lea desde un archivo de texto un conjunto de aristas de un grafo no dirigido y construya una representación de un grafo mediante lista de adyacencia.

```
#include <stdio.h>
#include "TDA-Grafo-L.h"
#include "TDA-Lista-D.h"
int main() {
    char nombre[200];
    int vertices, vert1, vert2;
    printf("Ingrese el nombre del archivo a leer\n");
    scanf("%s", nombre);
    FILE *fp;
    fp = fopen(nombre, "r");
    fscanf(fp, "%d", &vertices);

    Grafo *graf = crear_grafo(vertices);

    while (fscanf(fp, "%i %i", &vert1, &vert2) == 2){
        insertar_nodo_fin(graf->arrV[vert2], vert1);
        insertar_nodo_fin(graf->arrV[vert1], vert2);
    }
```

Ejemplo archivo entrada:



```
datosGrafo: Bloc de notas
Archivo Edición Formato Ver
6
0 1
1 2
2 3
4 3
3 5
3 0
0 5
5 4
1 4
4 2
5 1
```

Alg. *crear_Grafo(numV: Entero): Grafo

Inico

```
*G: Grafo
G = asignar memoria
G->n = numV
G->m = 0
G->arrV = inicializar_arreglo_listas(numV, 0)
retornar G
```

Fin

Nodo: struct

info: Entero

*sig: Nodo

Nodo: struct

*head: Nodo

n: Entero

Grafo: struct

n: Entero; // cantidad de vértices

m: Entero // cantidad aristas;

*arrV: Lista //arreglos de los vértices

TDA Lista

Alg. *inicializar_arreglo_listas(f, val: Entero): Lista

Inico

*arrLis : Lista

i: Entero

arrLis = asignar memoria para f filas

PARA i = 0 HASTA f HACER

arrLis[i] = crear_lista()

retornar arrLis

Fin

(*) La función main está escrita en lenguaje C

(*) Las operaciones crear_grafo e inicializar_arreglo_listas están escritas con pseudocódigos

Tareas

- Implementar el TDA grafo, con operaciones usando matriz de adyacencia:
Inicializar grafo, insertar arista, remover arista, obtener aristas, etc.

CONSULTAS