DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Laboratorio Semana 7: GRAFOS

Semestre 2 - 2024

Análisis de Algoritmos y Estructura de Datos

Departamento de Ingeniería Informática

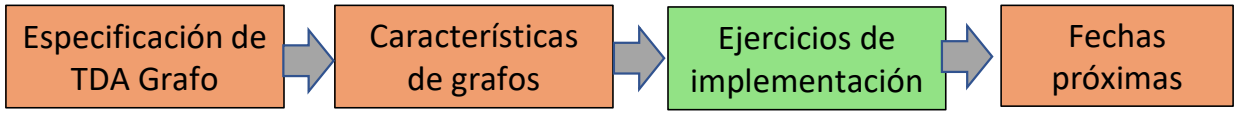
U3 -S8

Contenido

- Representación de Matriz de Adyacencia

Objetivos

- Familiarizarse con la representación de Matriz de adyacencia.



Refuerzo

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Grafo: Definición Formal

- Grafo G=(V, A)
- Grafo G=(V, A, W)

- V Conjunto de vértices o nodos
- V={v_1, v_2, … , v_n} , $|V|=n$ número de vértices

- A, conjunto de arcos o aristas
- A={a_{ij}, …, a_{kl}}, $|A|=m$ número de aristas, a_{ij} =(v_i, v_j)

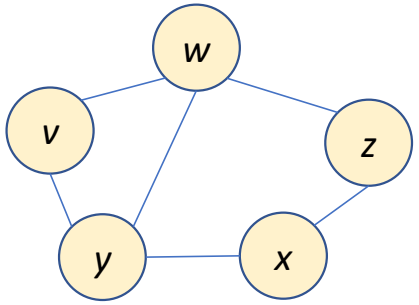
- W, conjunto de pesos
- W={w_{ij},…, w_{kl}}

Representación

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

TDA Grafo: Matriz de Adyacencia

No dirigido



Grafo G=(V, A)

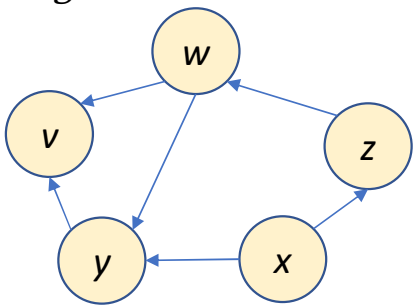
$V=\{ v, w, x, y, z \}$

$A=\{(v,w),(v,y),(w,y),(w,z),(x,y),(x,z)\}$

	v	w	x	y	z
v	0	1	0	1	0
w	1	0	0	1	1
x	0	0	0	1	1
y	1	1	1	0	0
z	0	1	1	0	0

Matriz bidimensional que 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 si existe la arista y 0 en caso contrario.

Dirigido



Grafo G=(V, A)

$V=\{ v, w, x, y, z \}$

$A=\{(w,v),(y,v),(w,y),(z,w),(x,y),(x,z)\}$

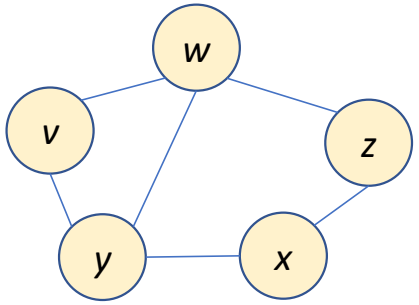
	v	w	x	y	z
v	0	0	0	0	0
w	1	0	0	1	0
x	0	0	0	1	1
y	1	0	0	0	0
z	0	1	0	0	0

Representación

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

TDA Grafo: Matriz de Adyacencia

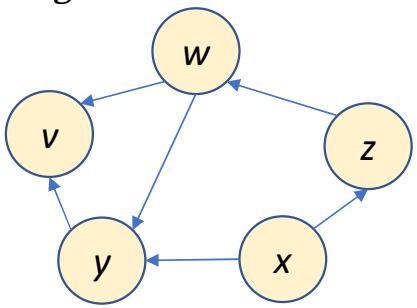
No dirigido




```
REGISTRO grafo:
   num c_vertices
   num A[c_vertices][c_vertices]
```

	v	w	x	y	z
v	0	1	0	1	0
w	1	0	0	1	1
x	0	0	0	1	1
y	1	1	1	0	0
z	0	1	1	0	0

Dirigido



	v	w	x	y	z
v	0	0	0	0	0
w	1	0	0	1	0
x	0	0	0	1	1
y	1	0	0	0	0
z	0	1	0	0	0

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Implementación de estructura de datos de TDA Grafo

- La estructura de datos que representa un grafo como una matriz de adyacencia puede ser la siguiente

```
typedef struct{
    int cvertices;
    int **adyacencias;
    }grafo;
```

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Operaciones de TDA Grafo

- `crea_grafo_vacio(num v): grafo*`
- `imprime_matriz_grafo(grafo *G)`
- `lee_grafo_nodirigido(char archivo[n])`

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Implementación de operaciones de TDA Grafo

- **crea_grafo_vacio(num v): grafo*** crea grafo de n vértices.

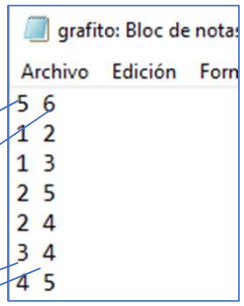
```
grafo* crea_grafo_vacio(int vertices){
    grafo *g = (grafo*)malloc(sizeof(grafo));
    g->cvertices = vertices;
    g->adyacencias = (int**)malloc(vertices * sizeof(int*));
    int i, j;
    for (i = 0; i < vertices; i++){
        g->adyacencias[i] = (int*)malloc(vertices * sizeof(int));
        //Inicializa en cero
        for(j = 0; j < vertices; j++){
            g->adyacencias[i][j] = 0;
            }
        }
    return g;
    }
```

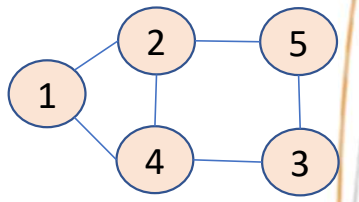
	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Implementación de operaciones de TDA Grafo


```
grafo* lee_grafo_nodirigido(char *nombre_archivo){
    FILE *pf;
    pf = fopen(nombre_archivo, "r");
    int n_vertices, m_aristas;
    int i,j,k;
    if (pf == NULL){
        printf("Error de archivo\n");
        return NULL;
        }
    else{
        fscanf(pf, "%d %d", &n_vertices,&m_aristas);
        grafo *G = crea_grafo_vacio(n_vertices);
        for(k = 0; k < m_aristas; k++){
            fscanf(pf,"%d %d",&i, &j);
            G->adyacencias[i - 1][j - 1] = 1;
            G->adyacencias[j - 1][i - 1] = 1;
            }
        fclose(pf);
        return  G;
        }
}
```





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

	0	1	2	3	4
0	0	1	0	1	0
1	1	0	0	1	1
2	0	0	0	1	1
3	1	1	1	0	0
4	0	1	1	0	0

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Implementación de operaciones de TDA Grafo

- **imprime_matriz_grafo(grafo *G)**

Muestra/Imprime los elementos de matriz de adyacencia

```
void imprime_matriz_grafo(grafo *g){
    int i, j;
    for (i = 0; i < g->cvertices; i++){
        for (j = 0; j < g->cvertices; j++){
            printf("%d ", g->adyacencias[i][j]);
            }
        printf("\n");
        }
    }
```

```
Matriz de adyacencia:
0 1 1 0 0
1 0 0 1 1
1 0 0 1 0
0 1 1 0 1
0 1 0 1 0
```

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

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	1	0	0	1	0
3	0	1	1	0	1
4	0	1	0	1	0

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

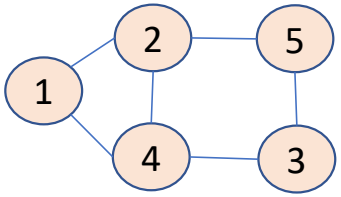
Trabajo individual

- Instrucciones:
  - Implementación individual.
  - Si se presentan dudas, avisar para apoyar.

Ejercicios de implementación

Actividad nro. 1

- Descargar los archivos **TDAgrafo.h, TDAgrafo_implementacion.c** y **lab08-grafos.c**.
- Compilar (cliente e implementación) y ejecutar.
- Revisar y probar implementaciones.
- Visualizar despliegue de matriz identificando los elementos del grafo que la origina.



Actividad de Implementación nro. 1

- Para un grafo no ponderado implementar la operación :

obtiene_grado(grafo* G, num v): num

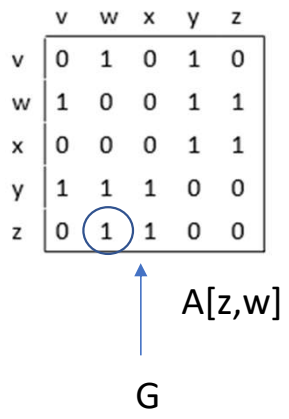
La operación devuelve la cantidad de vértices adyacentes del vértice dado.

- Probar funcionalidad **obtiene_grado** desde main() de **lab07-grafos.c**.

Ejercicios de implementación

Actividad de Implementación nro. 1

```
obtiene_grado(grafo *G, num v): num
    num grado ← 0
    num i ← 1
    while i <= G→c_vertices do
        if G→A[v][i] > 0 entonces
            grado ← grado + 1
        i ← i + 1
    return grado
```



Ejercicios de implementación

Actividad de Implementación nro. 2

- Un punto de articulación de un grafo conexo G es un vértice cuya eliminación (eliminación del vértice y de todas sus conexiones) convierte a G en un grafo no conexo. Construya un programa en C que dado un grafo no dirigido G (V, E), representado como matriz de adyacencias, muestre los vértices de G que sean puntos de articulación.

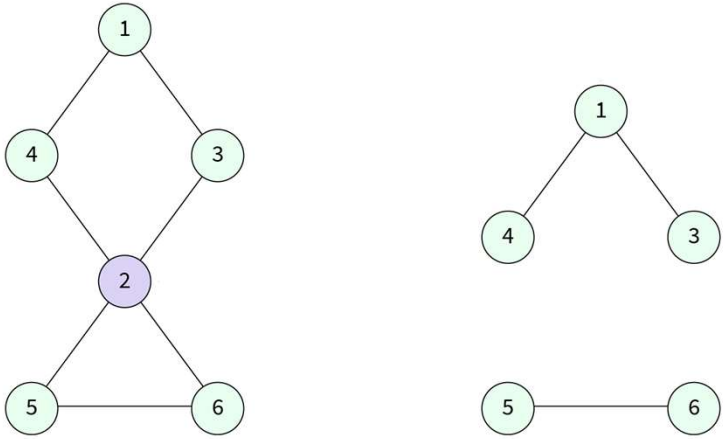


Figura: grafo *G* y punto de articulación 2.

Ejercicios de implementación

Actividad de Implementación nro. 2

- Para su solución considere la existencia de la función int es_conexo(grafo *g) que retorna 1 si el grafo es conexo y 0 si el grafo es no conexo.
- Probar funcionalidad desde main() de **lab07-grafos.c**.

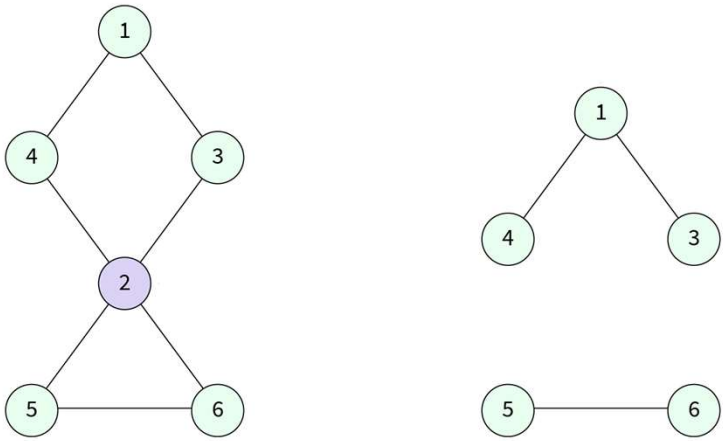
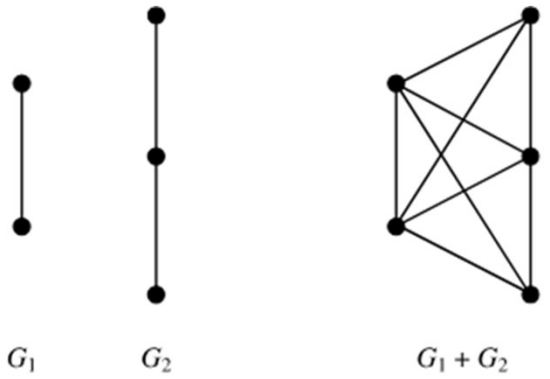


Figura: grafo *G* y punto de articulación 2.

Ejercicios de implementación

Actividad de Implementación nro. 3

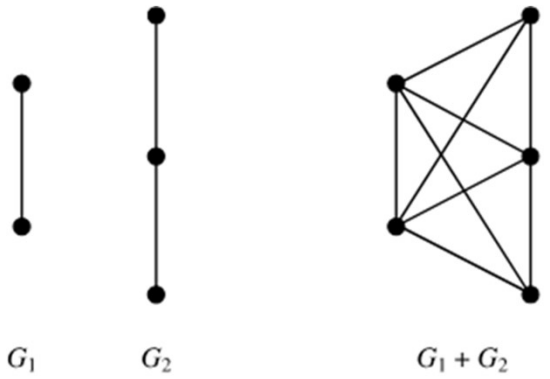
- La operación unión de Harary G = G1 + G2 sobre los grafos G1 y G2, con conjuntos disjuntos de vértices V1 y V2, y conjuntos de aristas E1 y E2 entrega como resultado el grafo con conjunto de vértices V = V1 ∪ V2 y conjunto de aristas E = E1 ∪ E2, además de todas las aristas que unan V1 y V2. En la figura se muestra un ejemplo de la operación unión de Harary sobre dos grafos G1 y G2.



Ejercicios de implementación

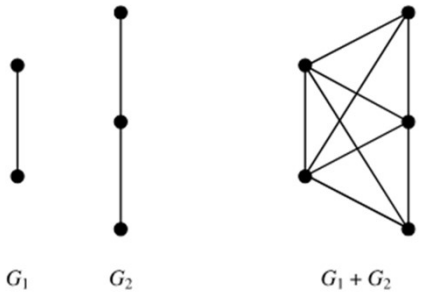
Actividad de Implementación nro. 3

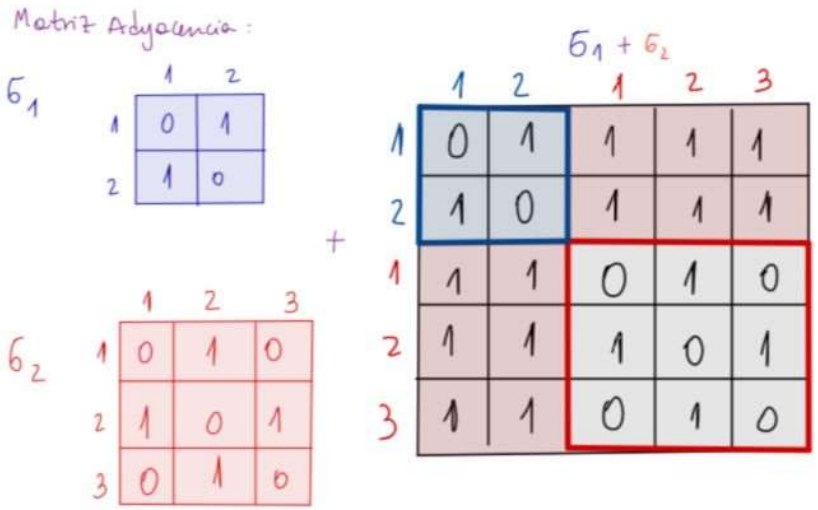
- Implemente una operación que reciba como entrada dos grafos no dirigidos G1 y G2, representados como matriz de adyacencia, y que retorne el grafo resultante al aplicar operación unión de Harary sobre los grafos de entrada.
- Probar funcionalidad desde main() de **lab07-grafos.c**.



Ejercicios de implementación

Actividad de Implementación nro. 3





Ejercicios de implementación

Actividad de Implementación nro. 3

```
union_harary(grafo *G_1, grafo *G_2): grafo*
    grafo *G_harary ← crea_grafo_vacio(G_1→c_vértices + G_2→c_vertices)
    num i, j
    for i ← 1 to G_1→c_vertices
        for j ← G_1→c_vértices + 1 to G_harary→c_vertices
            G_harary→A[i][j] ← 1
            G_harary→A[j][i] ← 1
    for i ← 1 to G_1→c_vértices
        for j ← 1 to G_1→c_vertices
            G_harary→A[i][j] ← G_1→A[i][j]
    for i ← 1 to G_2→c_vértices
        for j ← 1 to G_2→c_vértices
            G_harary→A[G_1→c_vértices + i][G_1→c_vértices + j] ← G_2→A[i][j]
    return G_harary
```

Entrega de actividad de laboratorio

- Entrega *obligatoria*
- Subir actividades 2 y 3 de esta sesión en buzón de Campus Virtual, en único archivo **s7_coordinacion_apellido_nombre.zip**
- Se espera TDAgrafo.h, TDAgrafo_implementacion.c y lab07-grafos.c comprimidos en archivo .zip
- Plazo: durante el laboratorio