Análisis de Algoritmos y Estructura de Datos

Implementación de TDA grafo

Prof. Violeta Chang C

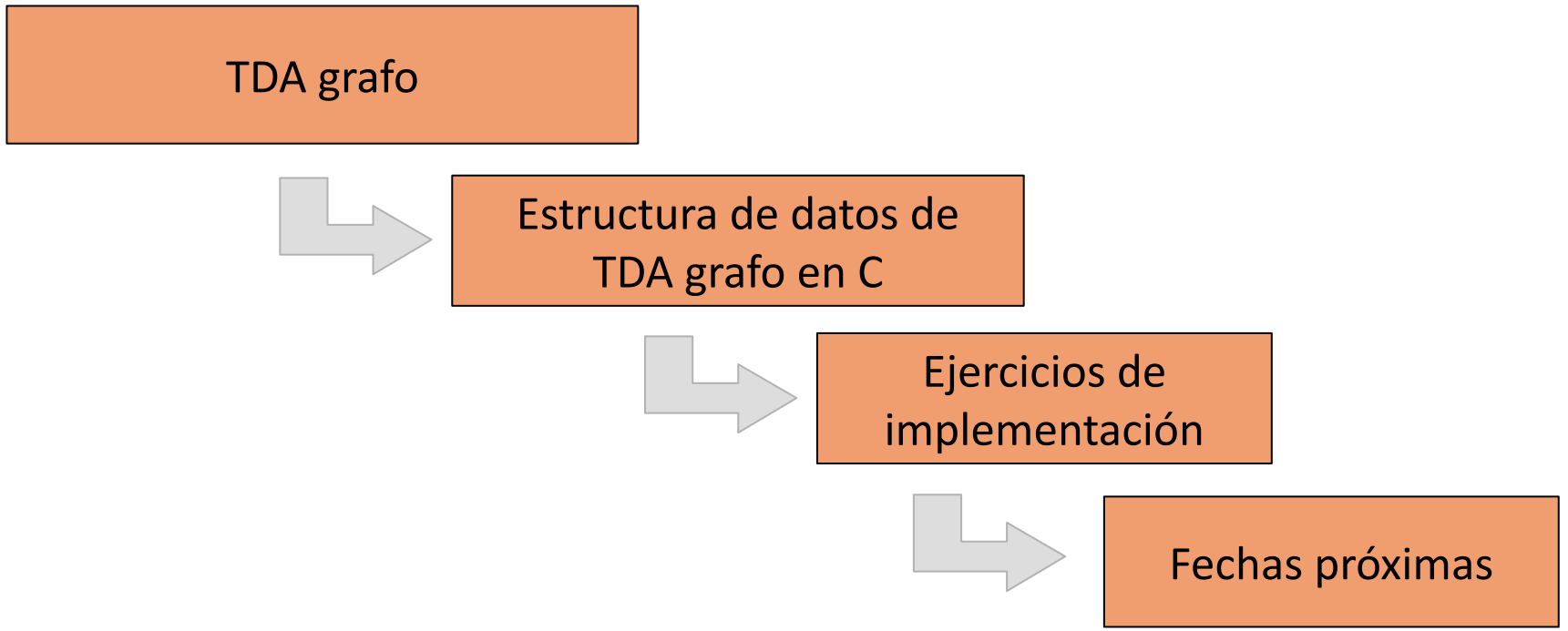
Semestre 2 – 2023

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

TDA grafo

- **Contenidos**:
  - Estructura de datos de TDA grafo
  - Operaciones de TDA grafo
- **Objetivos**:
  - Implementar recorridos DFS y BFS en TDA grafo

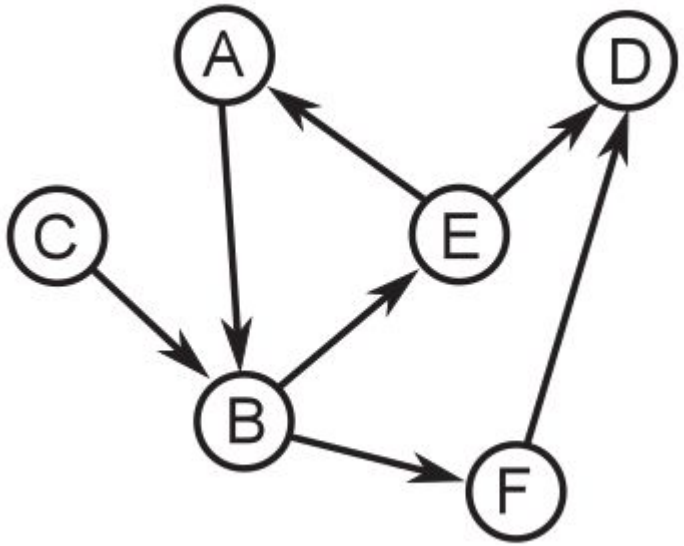
Ruta de trabajo



Especificación e implementación de TDA grafo

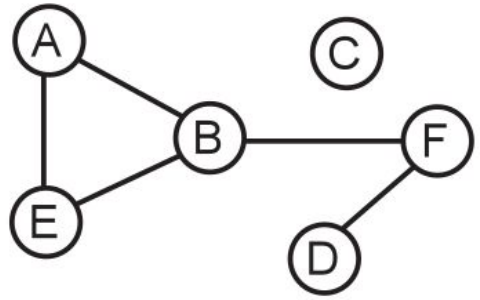
Terminología de grafos

- Un grafo G se define con:
  - **Conjunto de vértices** V={A,B,C,D,E,F}
  - **Conjunto de aristas** A={(A,B),(B,E),(B,F),(C,B),(E,D),(F,D)}



Representación de grafos

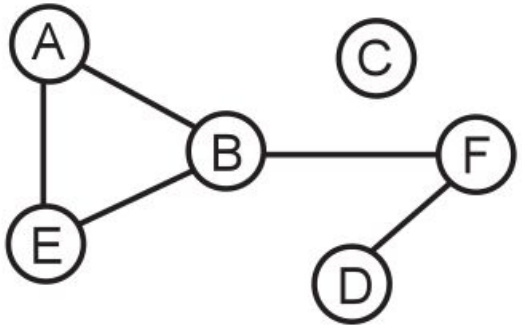
Matriz de adyacencia

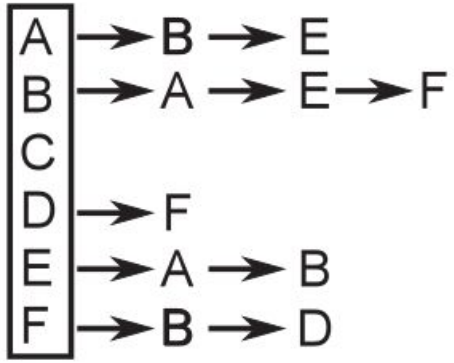


	A	B	C	D	E	F
A	0	1	0	0	1	0
B	1	0	0	0	1	1
C	0	0	0	0	0	0
D	0	0	0	0	0	1
E	1	1	0	0	0	0
F	0	1	0	1	0	0

Matriz de adyacencia de un grafo no dirigido

Lista de adyacencia





Lista de adyacencia de un grafo no dirigido

Definición de TDA grafo

- **Estructura de datos**
  - Un grafo es una colección no lineal de elementos homogéneos que se entiende como un conjunto de vértices y un conjunto de aristas que conectan dichos vértices. Las aristas pueden tener un peso asociado.
  - La estructura de datos que representa un grafo consiste de una de las siguientes alternativas:
    - **vértices + listas de adyacencia**
    - vértices + matriz de adyacencia

Implementación de estructura de datos de TDA grafo

- La estructura de datos que puede representar un grafo usando listas de adyacencia es la siguiente:

```
struct grafo
{
    TDAlista* A;
    int* V;
    int n;
    int m;
};

typedef struct grafo TDAGrafoLA;
```

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Implementación de operaciones de TDA grafo

```c
TDAGrafoLA* crearGrafoVacio(int n)
{
    int i;

    TDAGrafoLA* G=malloc(sizeof(TDAGrafoLA));
    G->A = malloc(sizeof(TDAlista)*n);
    G->V = malloc(sizeof(int)*n);

    for (i=0; i<n; i++)
    {
      G->A[i] = crearListaVacia();
      G->V[i] = i+1;
    }
    G->n = n;
    G->m = 0;

    return  G;
}
```

```c
void imprimirListaAdyacencia(TDAGrafoLA* G)
{
    int i;
    printf("Numero de vertices:%d \n",G->n);
    for (i=0; i<G->n; i++)
    {
        printf("%d  ",G->V[i]);
    }
    printf("\nNumero de aristas:%d \n",G->m);
    for (i=0; i<G->n; i++)
    {
        printf("%d  ",i+1 );
        recorrerLista(G->A[i]);
    }
}
```

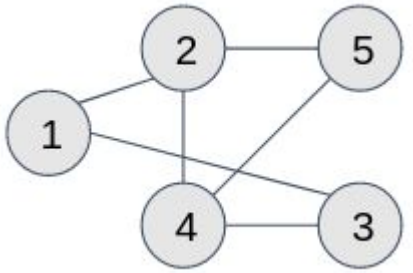
```c
void agregar_arista(TDAGrafoLA* G, int v, int w)
{
    insertarInicio(&(G->A[v]),w+1);
}
```

Implementación de operaciones de TDA grafo

```
//NoDirigido NoPonderado
TDAGrafoLA* leerGrafoNoDirigido(char *nombreArchivo)
{
    int n_vertices, m_aristas;
    int i,j,k;
    TDAGrafoLA* G;

    FILE* pf=fopen(nombreArchivo,"r");
    if (pf ==NULL)
    {
        printf("Error de archivo\n");
        return NULL;
    }
    else
    {
        //Cantidad de vértices y aristas
        fscanf(pf, "%d %d", &n_vertices,&m_aristas);
        G=crearGrafoVacio(n_vertices);
        G->n=n_vertices;
        G->m=m_aristas;

        for(k=0;k<m_aristas;k++)
        {
            fscanf(pf,"%d %d",&i, &j);
            agregar_arista(G, i, j);
            agregar_arista(G, j, i);
        }
        fclose(pf);
        return  G;
    }
}
```

grafoND-NP.in
/mnt/Dropbox/Dr...

```
5 6
1 2
1 3
2 5
2 4
3 4
4 5
```




```
Numero de vertices:5
1  2  3  4  5
Numero de aristas:6
1  3 2
2  4 5 1
3  4 1
4  5 3 2
5  4 2
```

Implementación de operaciones de TDA grafo

```
TDAlista obtenerAdyacentes(TDAGrafoLA* grafo, int vertice)
{
    TDAlista adyacentes=crearListaVacia();
    nodo* aux= grafo->A[vertice-1];
    while (aux!=NULL)
    {
        insertarInicio(&adyacentes, aux->dato);
        aux=aux->puntero;
    }
    return(adyacentes);
}
```

Actividades de implementación

Actividad 1 - individual

1. Compilar y ejecutar **lab09-grafos.c**
2. Experimentar con las funciones implementadas en **TDAgrafo.h** verificando llamadas desde función *main()* en **lab09-grafos.c**:
   - leer grafo no dirigido desde archivo “grafo-ND-NP.in”
   - imprimir listas de adyacencia
   - mostrar vértices adyacentes al vértice 2

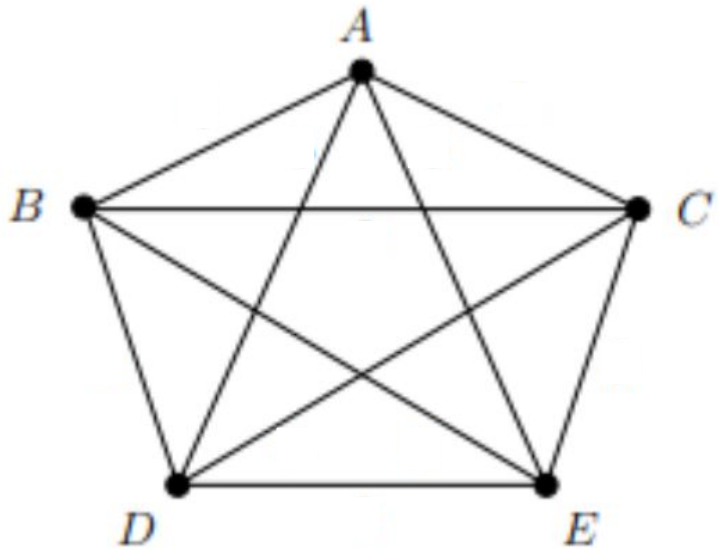
Actividad 2 - individual

1. Completar la implementación de la siguiente función en **TDAgrafoLA.h**:
   - *marcarVisitado: registra que un cierto vértice ya fue visitado (y lo muestra en pantalla)*
   - *adyacenteNoVisitado: devuelve el primer vértice adyacente que no ha sido visitado; en caso de no existir devuelve -1*
2. Realizar llamado en *main()* de **lab09-grafos.c** para mostrar recorrido DFS a partir del vértice 1.

```c
void recorridoDFS(TDAGrafoLA* grafo, int inicio)
{
    int topePila,w,i;
    TDAlista adyacentes;

    //definir tipo de dato de "visitados"

    TDApila* pila=crearPilaVacia(grafo->n);
    apilar(pila,inicio);
    //marcarVisitado(visitados,inicio);

    while (!(esPilaVacia(pila)))
    {
        topePila=tope(pila)->dato;
        adyacentes=obtenerAdyacentes(grafo,topePila);
        w=adyacenteNoVisitado(adyacentes,visitados);
        if (w!=-1)
        {
            apilar(pila,w);
            //marcarVisitado(visitados,w);
        }
        else
            desapilar(pila);
    }
    printf("\n");
}
```

Actividad 3 - individual

- El campus de una universidad congrega 5 edificios, cada uno correspondiente a una facultad. Durante los días de receso de la universidad, la seguridad del campus se ha convertido en una cuestión primordial por lo desolado del ambiente. Para dar solución a este problema, se cuenta con personal de seguridad que realiza rondas periódicas peatonales a lo largo de todo el campus, todos los días que dura el receso. El grafo mostrado representa las distancias entre los distintos edificios dentro del campus y los caminos entre ellos.
- Considerando esta información, se pide:
  - Crear archivo **grafoCampus.in** para guardar información del grafo presentado
  - Implementar función "recorridoCompleto(grafo,inicio)" en **TDAgrafoLA.h** para verificar si un vigilante puede salir desde INICIO, recorrer todos los edificios una sola vez y volver al punto de partida. Es requisito indispensable usar *recorridoDFS* implementado en Actividad 2.
  - Hacer los llamados necesarios en *main()* de **lab09-grafos.c** para dar respuesta al problema planteado.



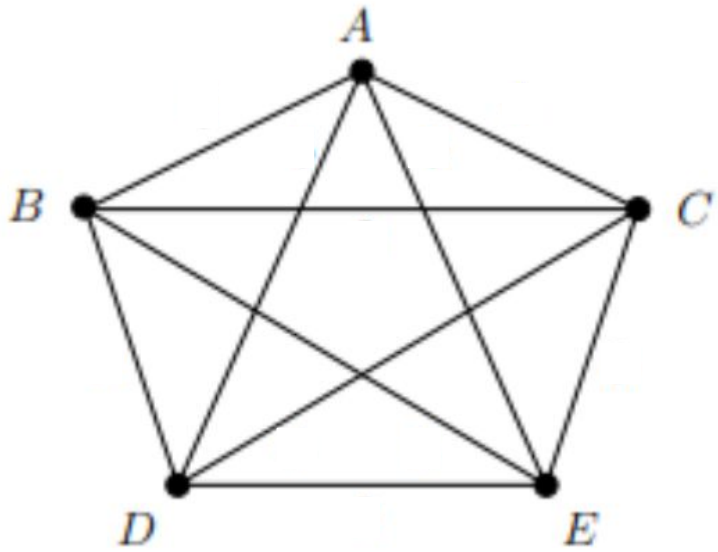
Actividad 4 - individual

1. Completar la implementación de la siguiente función en **TDAgrafoLA.h**:
   - *marcarVisitado: similar a actividad anterior*
   - *estaVisitado: dado un vértice, devuelve 1 si ya fue visitado, 0 en caso contrario*
2. Realizar llamado en *main()* de **lab09-grafos.c** para mostrar recorrido BFS a partir del vértice 1.

```
void recorridoBFS(TDAGrafoLA* grafo, int vertice)
{
    int frenteCola,w;
    TDAlista adyacentes;

    TDAcola* cola = crearColaVacia(grafo->n);
    encolar(cola, vertice);
    while(!esColaVacia(*cola))
    {
        frenteCola=frente(*cola)->dato;
        descolar(cola);
        //marcarVisitado(frenteCola);
        adyacentes=obtenerAdyacentes(grafo,frenteCola);
        while(adyacentes!=NULL)
        {
            w=adyacentes->dato;
            if (!(estaVisitado(w)))
                encolar(cola,w);
            adyacentes=adyacentes->puntero;
        }
    }
    printf("\n");
}
```

Actividad 5 - individual

- El campus de una universidad congrega 5 edificios, cada uno correspondiente a una facultad. Durante los días de receso de la universidad, la seguridad del campus se ha convertido en una cuestión primordial por lo desolado del ambiente. Para dar solución a este problema, se cuenta con personal de seguridad que realiza rondas periódicas peatonales a lo largo de todo el campus, todos los días que dura el receso. El grafo mostrado representa las distancias entre los distintos edificios dentro del campus y los caminos entre ellos.
- Considerando esta información, se pide:
  - Crear archivo **grafoCampus.in** para guardar información del grafo presentado
  - Implementar función "recorridoCompleto(grafo,inicio)" en **TDAgrafoLA.h** para verificar si un vigilante puede salir desde INICIO, recorrer todos los edificios una sola vez y volver al punto de partida. Es requisito indispensable usar *recorridoBFS* implementado en Actividad 4.
  - Hacer los llamados necesarios en *main()* de **lab09-grafos.c** para dar respuesta al problema planteado.



Entrega de actividad de laboratorio

- Entrega *obligatoria*
- Subir actividades 2 y 3 **o** 4 y 5 de esta sesión en buzón de uVirtual, en único archivo **s9_apellido_nombre.zip**
- **BONUS**: si se entregan todas las actividades, las actividades 4 y 5 se considerarán como entrega adicional, que *reemplaza la peor nota de las entregas anteriores en el semestre*.
- Se espera lab09-grafos.c y TDAgrafoLA.h (ambos modificados para responder a actividades entregadas) comprimidos en archivo .zip
- Plazo: **hoy** dentro del horario de laboratorio

Actividad de cierre

- Ir a menti.com e ingresar código 4384 3673

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Próximas fechas...

- Resumen de la semana:
  - Algoritmo de recorrido DFS
  - Algoritmo de camino mínimo Dijkstra
  - Algoritmo de flujo máximo Ford-Fulkerson

  ~~cátedra~~ – ~~refuerzo~~ – ~~laboratorio~~

- Próxima semana:
  - **PEP2**
  - **Entrega de Tarea 2**
  - Algoritmo de árboles de cobertura mínima (Prim y Kruskal)

U3 - S9