Estructuras y Tipos de Datos

Tipo Abstracto de Datos Grafos

Julio 2016

TEMA VI - T.A.D. Grafos

CONTENIDO

- Introducción
- Definiciones Básicas de Grafos
- Definición del Tipo Abstracto
- Codificación del T.A.D. Grafos

    Representación usando Matriz de Incidencia
    Representación usando Matriz de Adyacencia
    Representación usando Listas de Adyacencia

Introducción

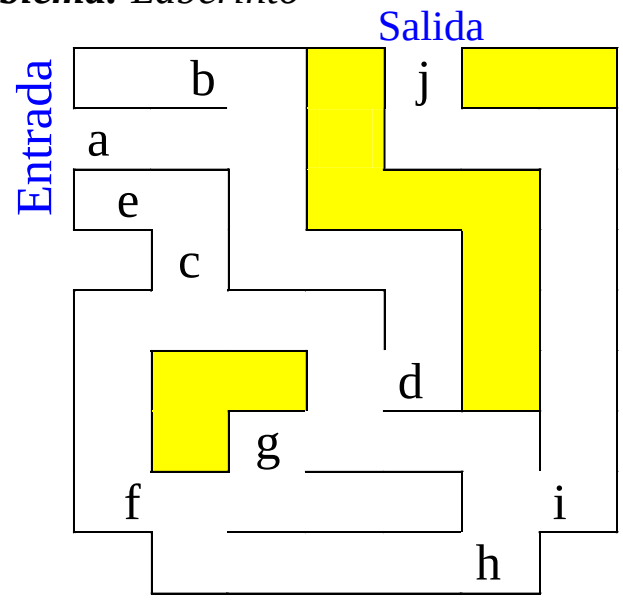
Situación

En algunos problemas, la organización de los datos mediante listas, colas, hashing o arreglos no es la más adecuada para su procesamiento.

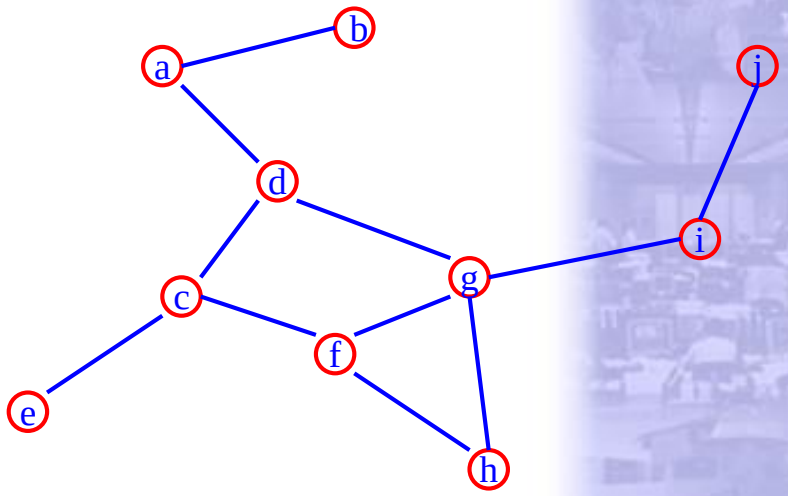
Necesitamos representar Elementos y su interrelación.

Ejemplo: *Supongamos un laberinto en donde tenemos pasillos y puertas de acceso a otro pasillo, una puerta estará relacionada con otra si existe un pasillo que las comunica, la idea es salir del laberinto, es decir, desde una puerta de entrada conseguir la puerta de salida.*

Problema: *Laberinto*



Solución Propuesta: ***grafos***



Puertas: nodos *Pasillos: arcos*

Problema: *conseguir camino desde* ***a*** *hasta* ***j***

Definiciones Básicas

Grafo

Es una tripleta $G = (V, E, \varphi)$ *donde* V *es un conjunto cuyos elementos llamaremos vértices o nodos,* E *es un conjunto cuyos elementos llamaremos arcos o uniones y* φ *es una función definida como* $\varphi: E \rightarrow V \,\&\, V$, *donde*

$$V \,\&\, V = \{ A \subset V \;\ni\; |A| = 1 \;\text{ó}\; 2 \}$$

Ejemplo

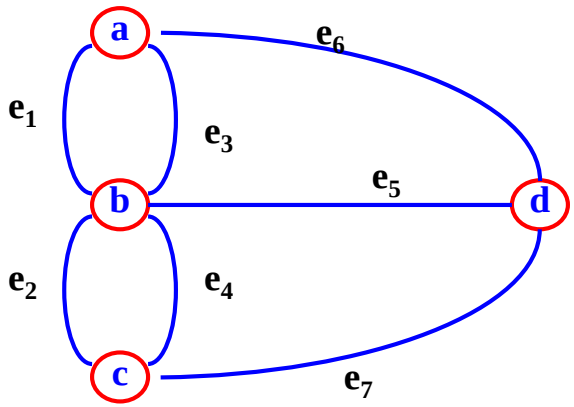
$V = \{ a, b, c, d \}$

$E = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7 \}$

$\varphi(e_1) = \{a,b\}$ $\varphi(e_2) = \{b,c\}$ $\varphi(e_3) = \{a,b\}$ $\varphi(e_4) = \{b,c\}$

$\varphi(e_5) = \{b,d\}$ $\varphi(e_6) = \{a,d\}$ $\varphi(e_7) = \{c,d\}$

Representación gráfica

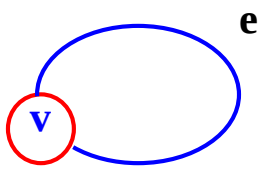


Definiciones Básicas

Lazo

Si $e \in E$ y $|\varphi(e)| = 1$ entonces e se llama lazo.

Ejemplo:



Incidencia de Arcos o Lazos en un Vértice

En un grafo tenemos arcos y lazos que inciden sobre los nodos, por lo tanto, dado un vértice $v \in V$, $e \in E$ incide en v si $v \in \varphi(e)$

Conjunto de Lazos que inciden en un vértice v

$$L(v) = \{ e \in E \ni v \in \varphi(e) \ \& \ |\varphi(e)| = 1 \}$$

Definiciones Básicas

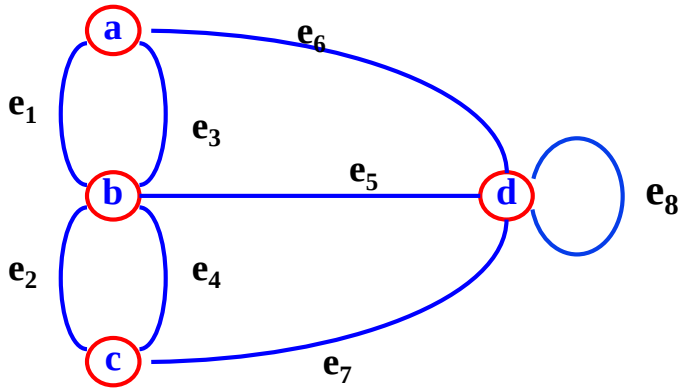
Grado de un vértice

El grado de un vértice es una función d: V → N U { 0 } definida por

$$d(v) = |U(v)| + 2|L(v)|$$

≈ *Es el número de arcos que inciden en v más dos veces el número de lazos que inciden en v.*

Ejemplo:



$d(a) = 3$
$d(b) = 5$
$d(c) = 3$
$d(d) = 5$

Camino

Un camino es una sucesión alternada de vértices y arcos que comienza y termina en un vértice.

Ejemplo:

$$P: v_{io}\ e_{io}\ v_{i1}\ e_{i1}\ \ldots\ e_{in}\ v_{in}$$

Donde v_{io} *es el vértice inicial del camino*

v_{in} *es el vértice final del camino*

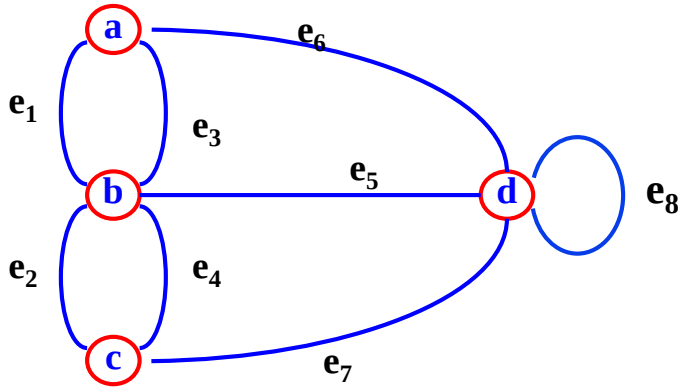
los $v_i \in V$ *y los* $e_i \in E$

Definiciones Básicas

Longitud de un camino

Es el número de arcos que tiene el camino, o el número de vértices -1 que tiene el camino.

Ejemplo:



$P(a,c) = a\ e_6\ d\ e_8\ d\ e_5\ b\ e_4\ c$

$Long(P(a,c)) = 4$

Camino Simple

Es un camino en donde no hay vértices repetidos.

Ejemplo: $P(a,c) = a\ e_6\ d\ e_5\ b\ e_4\ c$

Circuito

Es un camino tal que el vértice inicial es igual al vértice final, el camino comienza y termina en el mismo vértice.

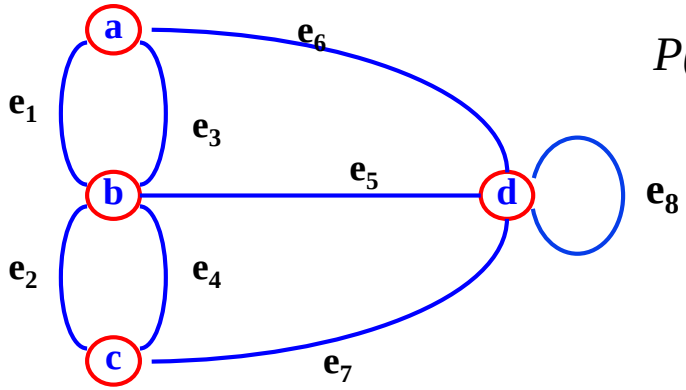
Ejemplo: $P(a,a) = a\ e_6\ d\ e_8\ d\ e_5\ b\ e_4\ c\ e_2\ b\ e_1\ a$

Definiciones Básicas

Circuito Simple

Es un circuito en donde no hay vértices repetidos, con la excepción del vértice inicial y el final.

Ejemplo:



$P(a,a) = a\ e_6\ d\ e_7\ c\ e_4\ b\ e_3\ a$

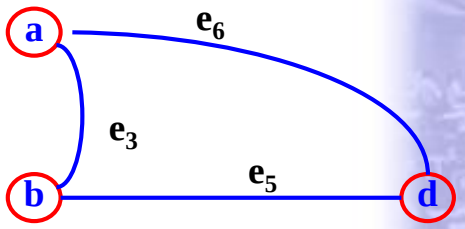
Subgrafo

Sea $G = (V, E, \varphi)$

Un grafo G' *es subgrafo de* G *si se cumple:*

1. $V' \subseteq V$
2. $E' \subseteq E$
3. $\varphi' := \varphi \mid E \equiv \forall e \in E' \quad \varphi'(e) = \varphi(e)$

Ejemplo:



Definición del T.A.D. Grafos

Datos

- *Conjunto de vértices o nodos*
- *Conjunto de arcos y*
- *Una función que establece la relación entre los vértices y los arcos*

Modelo

Un grafo G es una tripleta $G = (V, E, \varphi)$

Donde:
- V: es un conjunto de vértices (nodos o puntos).
- E: es un conjunto de arcos.
- $\varphi: E \rightarrow V \& V$ $\qquad V \& V = \{ A \subset V \ \ni \ |A| = 1 \ \text{ó} \ 2 \}$

Operaciones

- *Inicializar(G):* *inicializa el Grafo vacío.*
- *Insertar(v1,v2,G):* *agrega un arco entre el vértice v1 y el vértice v2 del grafo G.*
- *Eliminar(v1,v2,G):* *elimina un arco entre el vértice v1 y el vértice v2 del grafo G.*
- *Adyacente(v1,v2,G):* *determina si el vértice v1 es adyacente al vértice v2 en el grafo G.*
- *Grado(v,G):* *devuelve el grado del vértice v del grafo G.*
- *Longitud(v1,v2,G):* *determina cual es la longitud del camino entre el vértice v2 y el vértice v2 del grafo G, si existe.*

Representación

Requerimientos

Para implementar el Tipo de Datos Abstracto GRAFOS necesitamos

- *Representar el modelo en una estructura de datos y*
- *Convertir las operaciones en código de programas (métodos)*

Tres Posibles Representaciones

- *Matriz de Adyacencias.*
- *Matriz de Incidencia.*
- *Lista de Adyacencias.*

Representación con Matriz de Incidencias

Estrategia

La matriz de incidencia relaciona vértices con aristas.

$M_{|V|x|E|}$ = *Matriz de incidencia de* $|V|$ *filas y* $|E|$ *columnas*

Donde

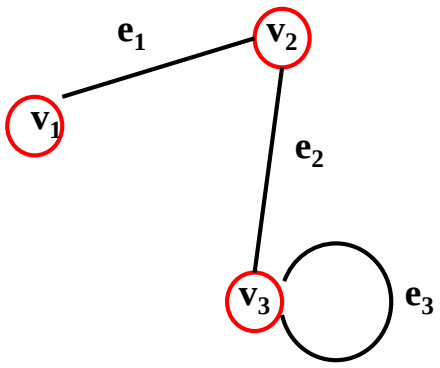
columnas = tenemos las aristas (e_j) *que pertenecen a E.*

filas = tenemos los vértices (v_i) *que pertenecen a V.*

m_{ij} = *contiene cuantas veces incide la arista* e_j *en el vértice* v_i

Ejemplo

Sea el Grafo G



Matriz de Incidencia

	e_1	e_2	e_3	
v_1	1			= 1 = $d(v_1)$
v_2	1	1		= 2 = $d(v_2)$
v_3		1	2	= 3 = $d(v_3)$

la suma de los elementos de la fila i nos da el grado del vértice i.

Representación con Matriz de Incidencias

Estructura

```
#include <iostream>
using namespace std;

#define MAXVERTICES 3
#define MAXARCOS 5

typedef int VERTICE;
typedef int ARCO;


class tgrafo {
private:
    int Grafo[MAXVERTICES][MAXARCOS];
public:
    tgrafo();
    bool insertar(VERTICE v1,VERTICE v2, ARCO a);
    bool eliminar(VERTICE v1,VERTICE v2, ARCO a);
    bool adyacente(VERTICE v1,VERTICE v2);
    int grado(VERTICE v);
    void imprimir();
};
```

Representación con Matriz de Incidencias

Métodos

```
// Constructor de la clase grafo
tgrafo::tgrafo(){
    for (VERTICE i = 0; i < MAXVERTICES; i++ )
        for (ARCO j = 0; j < MAXARCOS; j++)
            Grafo[i][j] = 0;
    }

//
// Inserta un arco entre los nodos dados
//Devuelve true si lo pudo insertar y false si el arco ya existe
bool tgrafo::insertar(VERTICE v1,VERTICE v2, ARCO a){
    int v = 0;

    if ( v1 >= MAXVERTICES || v2 >= MAXVERTICES || a >= MAXARCOS)
        return false;
    //
    // Verifica si el arco ya existe
    while(v < MAXVERTICES && Grafo[v][a] == 0)
        v++;

    if( v >= MAXVERTICES) {
        Grafo[v1][a]++;
        Grafo[v2][a]++;
        return true;
    }
    return false;
}
```

Representación con Matriz de Incidencias

Métodos, Cont…

```
//Devuelve true si o pudo eliminat y false en caso contrario
bool tgrafo::eliminar(VERTICE v1,VERTICE v2, ARCO a){

    if ( v1 >= MAXVERTICES || v2 >= MAXVERTICES || a >= MAXARCOS)
        return false;

    if( Grafo[v1][a] > 0 && Grafo[v2][a] > 0) {
        Grafo[v1][a]--;
        Grafo[v2][a]--;
        return true;
    }
    return false;
}

//
// Devuelve true si los 2 vertices son advacentes
// false en caso contrario
bool tgrafo::adyacente(VERTICE v1,VERTICE v2){
    ARCO e = 1;

    if (v1 >= MAXVERTICES || v2 >= MAXVERTICES)
        return false;

    while (e < MAXARCOS) {
        if (Grafo[v1][e] > 0 && Grafo[v2][e] > 0)
            return true;
        e++;
    }
    return false;
}
```

Representación con Matriz de Incidencias

Métodos, Cont…

```
//
// Devuelve el grado de un vertice dado
int tgrafo::grado(VERTICE v){
    int grado = 0;

    for (ARCO e=0; e < MAXARCOS; e++) {
        if (Grafo[v][e] > 0)
            grado++;
    }
    return grado;
}

void tgrafo::imprimir(){
     for (VERTICE i = 0; i < MAXVERTICES; i++ ) {
        for (ARCO j = 0; j < MAXARCOS; j++)
            cout << Grafo[i][j];
        cout << endl;
     }
}
```

Representación con Matriz de Adyacencias

Estrategia

La matriz de adyacencia relaciona vértices con vértices.

$M_{|V|x|V|}$ = *Matriz de adyacencia de* $|V|$ *filas y* $|V|$ *columnas*

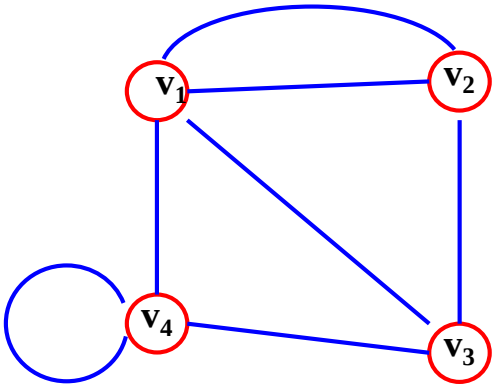
columnas = tenemos los vértices (v_j) *que pertenecen a V.*

Filas = tenemos los vértices (v_i) *que pertenecen a V.*

$m_{i\,j}$ = *contiene el número de arcos que unen al vértice* v_i *con el vértice* v_j

Ejemplo

Sea el Grafo G



Matriz de Adyacencias

	v_1	v_2	v_3	v_4	
v_1	0	2	1	1	= 4 = d(v_1)
v_2	2	0	1	0	= 3 = d(v_2)
v_3	1	1	0	1	= 3 = d(v_3)
v_4	1	0	1	2 1	= 4 = d(v_4)

La suma de la fila i nos da el grado del vértice i.

Si i = j, es un lazo, estos los multiplicamos por 2

Representación con Matriz de Adyacencias

Estructura

```
#include <iostream>
using namespace std;

#define MAXVERTICES 3

typedef int VERTICE;
typedef int ARCO;


class tgrafo {
private:
    int Grafo[MAXVERTICES][MAXVERTICES];
public:
    tgrafo();
    bool insertar(VERTICE v1,VERTICE v2);
    bool eliminar(VERTICE v1,VERTICE v2);
    bool adyacente(VERTICE v1,VERTICE v2);
    int grado(VERTICE v);
    void imprimir();
};
```

Representación con Matriz de Adyacencias

Métodos

```
//
// Constructor de la clase grafo
tgrafo::tgrafo(){
    for (VERTICE i = 0; i < MAXVERTICES; i++ )
        for (VERTICE j = 0; j < MAXVERTICES; j++)
            Grafo[i][j] = 0;
    }

//
// Inserta un arco entre los nodos dados
//Devuelve true si lo pudo insertar y false si el arco ya existe
bool tgrafo::insertar(VERTICE v1,VERTICE v2){
    int v = 0;

    if ( v1 >= MAXVERTICES || v2 >= MAXVERTICES || Grafo[v1][v2] >0)
        return false;

    Grafo[v1][v2]++;
    if (v1 != v2) // no es un lazo
        Grafo[v2][v1]++;
    return true;

}
```

VERTICE tgrafo::insertar::v2

Representación con Matriz de Adyacencias

Métodos, Cont…

```
//
//Elimina el arco entre 2 vertices dado
//Devuelve true si o pudo eliminat y false en caso contrario
bool tgrafo::eliminar(VERTICE v1,VERTICE v2){

    if ( v1 >= MAXVERTICES || v2 >= MAXVERTICES || Grafo[v1][v2] == 0 )
        return false;

    if( Grafo[v1][v2] > 0)
        Grafo[v1][v2]--;
    if ( v1 != v2 )    // No es un lazo
        Grafo[v2][v1]--;
    return true;
}

//
// Devuelve true si los 2 vertices son advacentes
// false en caso contrario
bool tgrafo::adyacente(VERTICE v1,VERTICE v2){

    if (v1 >= MAXVERTICES || v2 >= MAXVERTICES)
        return false;

    return (Grafo[v1][v2] > 0);
}
```

Representación con Matriz de Adyacencias

Métodos, Cont…

```
//
// Devuelve el grado de un vertice dado
int tgrafo::grado(VERTICE v){
    int grado = 0;

    for (VERTICE v1=0; v1 < MAXVERTICES; v1++)
        if (Grafo[v][v1] > 0)
            if (v == v1)
               grado +=2;
            else
                grado++;
    return grado;
}

void tgrafo::imprimir(){
     for (VERTICE i = 0; i < MAXVERTICES; i++ ) {
        for (VERTICE j = 0; j < MAXVERTICES; j++)
            cout << Grafo[i][j];
        cout << endl;
     }
}
```

Representación con Listas de Adyacencias

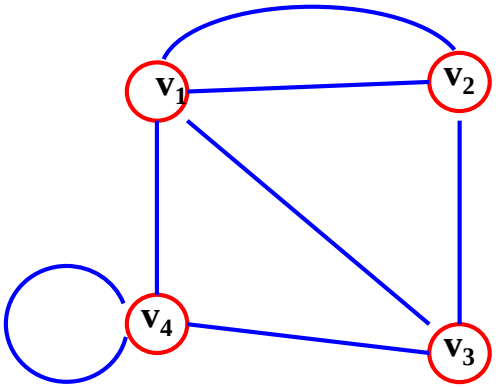
Estrategia

Es un arreglo de vértices, en donde la celda correspondiente al vértice i contiene un apuntador a la lista de los vértices adyacentes a i.

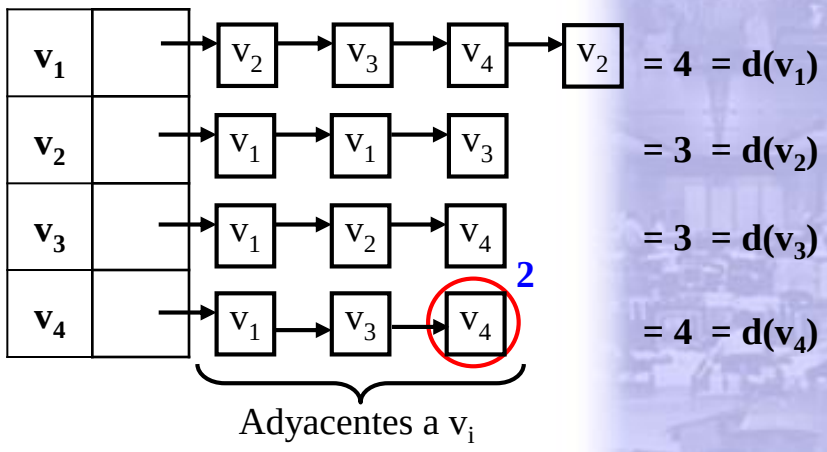
$A_{|V|}$ = *Arreglo de apuntadores de tamaño |V| (número de vértices).*
a_i = *contiene un apuntador a la lista de adyacencias del vértice* v_i

Ejemplo

Sea el Grafo G



Lista de Adyacencias



La suma de los elementos de la lista i nos da el grado del vértice i. Si $\mathbf{v_i} = \mathbf{v_j}$, *es un lazo, estos los multiplicamos por 2*

Representación con Listas de Adyacencias

Estructura

```
#include <iostream>
using namespace std;

#define MAXVERTICES 10

typedef int VERTICE;
typedef int ARCO;

typedef struct nodo{
   VERTICE v;
   struct nodo * sig;
}Nodo

class tgrafo {
private:
    Nodo *Grafo[MAXVERTICES];
public:
    tgrafo();
    bool insertar(VERTICE v1,VERTICE v2);
    bool eliminar(VERTICE v1,VERTICE v2);
    bool adyacente(VERTICE v1,VERTICE v2);
    int grado(VERTICE v);
    void imprimir();
};
```