UT7 – Grafos Dirigidos

Ejemplo 6.4. La figura 6.4 muestra una representación con lista de adyacencia para el grafo dirigido de la figura 6.1, donde se usan listas enlazadas sencillas. Si los arcos tienen etiquetas, éstas podrían incluirse en las celdas de la lista ligada.

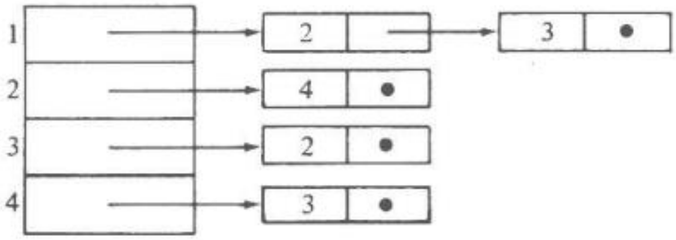


Fig. 6.4. Representación con lista de adyacencia para el grafo dirigido de la figura 6.1


```
    begin
        for i := 1 to n do
            if A[v, i] then
                return(i);
        return (0) { si se llega aquí, v no tiene vértices adyacentes }
    end; { PRIMERO }

function SIGUIENTE ( v: integer; i: integer ) : integer;
    var
        j : integer;
    begin
        for j := i+1 to n do
            if A[v, j] then
                return (j);
        return (0)
    end; { SIGUIENTE }
```

Fig. 6.6. Operaciones para recorrer vértices adyacentes.

```
i := PRIMERO(v);
while i <>Λ do begin
    w := VERTICE(v, i);
    { alguna acción en w }
    i := SIGUIENTE(v, i)
end
```

Fig. 6.7. Iteración en vértices adyacentes a *v*.

Iteración	S	w	$D[2]$	$D[3]$	$D[4]$	$D[5]$
inicial	{1}	—	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

Fig. 6.10. Cálculos de *Dijkstra* en el grafo dirigido de la figura 6.9.

Grafos

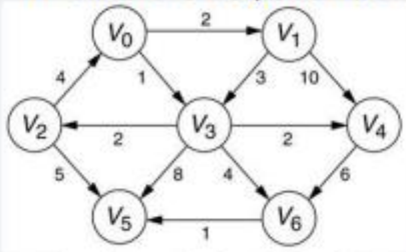
Universidad Católica del Uruguay

- Los grafos son modelos naturales para representar relaciones entre objetos de datos.
- Un grafo consiste de un conjunto finito de vértices V y de un conjunto de arcos A.

$$G = (V, A)$$

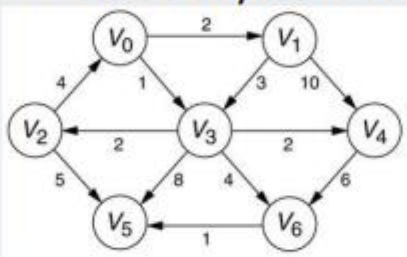
- Sea el conjunto de vértices o nodos V= $\{v_1, v_2, \ldots v_n\}$ entonces el conjunto de arcos o aristas es $A = \{(v_i, v_j)\}$. un conjunto de pares de vértices.
- Si las aristas son no dirigidas, es decir $(v_i, v_j) = (v_j, v_i)$, el grafo se llama no dirigido.
- En un grafo dirigido, la arista es un par ordenado de vértices.

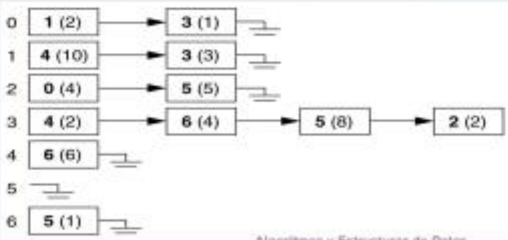
Matriz de adyacencias



	0	1	2	3	4	5	6
0		**2**		**1**			
1				**3**	**10**		
2	**4**					**5**	
3			**2**		**2**	**8**	**4**
4							**6**
5							
6						**1**	

Lista de adyacencias





TDA Grafo

- **Grafo (Vértices, Aristas)**
- Dado **un vértice origen**, indicar los **caminos mínimos** a todos los otros
- **Todos los caminos mínimos**, de todo vértice a todo otro
- **Centro de Grafo**, **excentricidad** de un vértice
- **Cerradura transitiva**
- **Búsqueda en profundidad** (recorrer sistemáticamente todo el grafo en profundidad)
- **Camino**, Caminos

12

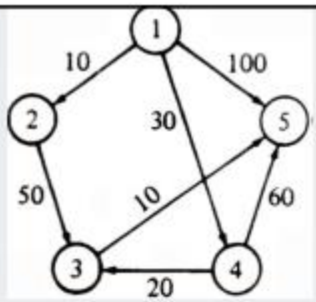
El algoritmo de Dijkstra

Función Dijkstra
COM
Inicializar *S*, *D*
S = {1};
para i = 2 a n hacer D[i] = C[1,i] *//(el valor inicial, infinito si //no hay camino directo)*
Mientras *V* <> *S* hacer
Elegir *w* perteneciente a *V-S*, tal que la distancia ***D[w])*** sea un **mínimo**
Agregar ***w*** a ***S***
ParaCada *v* perteneciente a ***V-S*** hacer
D[v] = min (D[v], D[w]+ costo(w,v)

FinMientras;

FIN {Dijkstra}

17



Iteration	S	w	D[2]	D[3]	D[4]	D[5]
initial	{1}	–	10	∞	30	100
1	{1, 2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

Floyd con recuperación de caminos

```
Función Floyd (var A : array[1..n,1..n] of real;
                          C : array[1..n,1..n] of real);
i, j, k : integer;
COM
  for i:= 1 to n do
    for j:= 1 to n do
      A[i,j]:= C[i,j]; P[i, j] := 0 ;
  for i:= 1 to n do A[i,i]:= 0;
  for k:= 1 to n do
    for i:= 1 to n do
      for j:= 1 to n do
        if (A[i,k]+A[k,j]) < A[i,j]
          then   A[i,j]:= A[i,k]+A[k,j]; P[i, j] := k ;
END;
```

Búsqueda en profundidad, análisis

```
métodoTvertice.bpf ();

 w : Tvertice;
COM
 (1)  Visitar();
 (2)     Para cada adyacente w hacer
 (3)       Si no(w.visitado()) entonces
             w.bpf()
           Fin Si
       Fin para cada
FIN {bpf}
```

- Todas las llamadas a bpf en la búsqueda en profundidad de un grafo con ***a*** arcos y ***n*** <= a vértices llevan un tiempo O(a):
  - No se llama a **bpf** en ningún vértice más de una vez
  - El tiempo consumido en las líneas (2) y (3) es proporcional a la suma de las longitudes de las listas, O(a)
- Entonces el tiempo total de la bfp de un grafo completo es O(a), o sea, el tiempo necesario para recorrer cada arco.

42

Clasificación topológica

```
procedure ClasificacionTopologica ();
  w : Tvertice;

w : Tvertice;
COM
 (1)  Visitar();
 (2)     Para cada adyacente w hacer
 (3)       Si no(w.visitado()) entonces
              w.ClasificacionTopologica()
           Fin Si
         Fin para cada
         imprimir ();  //agregar "this" al principio de la lista de
previas….
FIN; {ClasificacionTopologica}
```


56