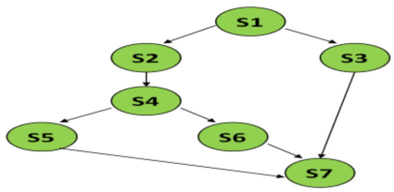


GRAFOS

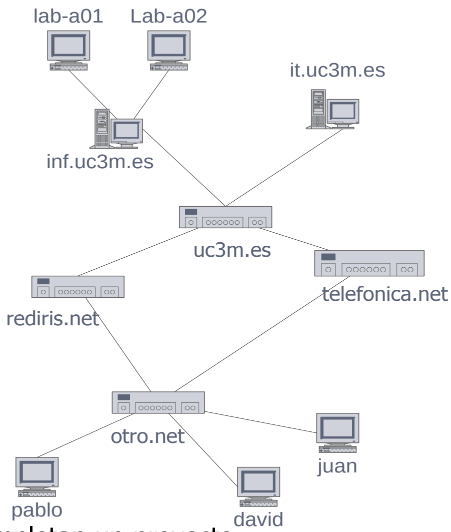
Introducción a la teoría de grafos

INTRODUCCIÓN

- Los grafos sirven para representar relaciones arbitrarias (no necesariamente jerárquicas) entre objetos de datos

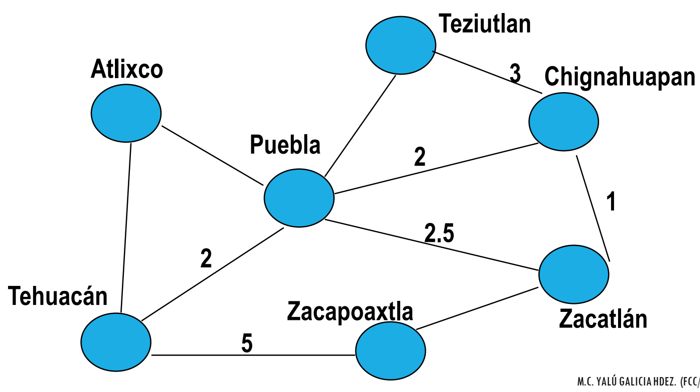
APLICACIONES

- Circuitos electrónicos
  - Tarjetas impresas
  - Circuitos integrados
- Redes de transporte
  - Autopistas
  - Vuelos
- Redes de ordenadores
  - LANs
  - Internet
  - Web
- Planeación (rutas críticas)
  - Planificación de las tareas que completan un proyecto



EJEMPLO DE USO DE GRAFOS

- En una red de carreteras las poblaciones representan los vértices del grafo y las carreteras de unión de dos poblaciones, los arcos, de modo que a cada arco se asocia información tal como la distancia entre dos ciudades, el consumo de gasolina por automóvil, etc.



DEFINICIONES

- Un grafo consiste en un conjunto de vértices o nodos (V) y un conjunto de arcos o aristas (A).
- Un grafo se representa con el par G = (V,A).
- El número de elementos de V se llama orden del grafo
- Un grafo nulo es un grafo de orden cero
- Un arco o arista está formado por un par de nodos u y v, y se representa por (u,v)
- Un grafo es dirigido (o digrafo) si los pares de nodos que forman los arcos son ordenados y se representan u $\rightarrow$ v.

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 5

DEFINICIONES

- Un grafo no dirigido es aquel que los arcos están formados por pares de nodos no ordenados, se representa u – v.
- Si (u,v) es una arista en el conjunto de aristas del grafo A(G), entonces u y v se dice que son vértices adyacentes.
  - Es decir, dos vértices son adyacentes si hay un arco que los une.
- Un arco tiene, a veces, asociado un factor de peso, en cuyo caso se dice que es un grafo valorado o ponderado.

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 6

FUNDAMENTOS: GRAFOS DIRIGIDOS

- Grafo no dirigido

V(G1) = {a,b,c,d}

A(G1) = {(a,b),(a,d),(b,c),(b,d)}

Adyacentes a a: b, d

Adyacentes a b: a, c, d

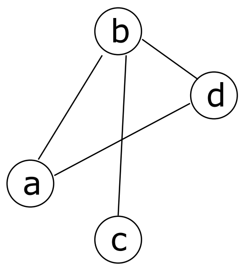
- Grafo dirigido

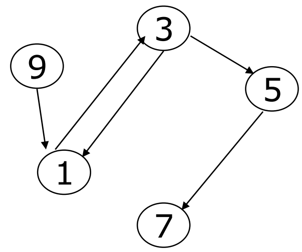
V(G2) = {1,3,5,7,9}

A(G2) = {(1,3),(3,1),(9,1),
(3,5),(5,7)}

Adyacentes a 1: 3

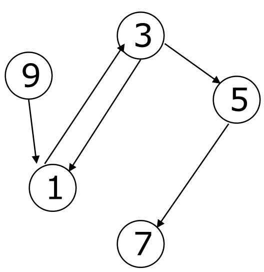
Adyacentes a 3: 1, 5





FUNDAMENTOS

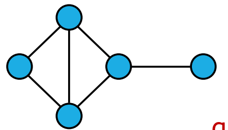
- Grado de un vértice (o nodo)
  - En un grafo no dirigido
    - Grado de un nodo u = nº de aristas que contienen a u
  - En un grafo dirigido
    - Grado de entrada de u = nº de arcos que llegan a u
    - Grado de salida de u = nº de arcos que salen de u



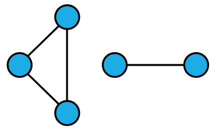
Grado entrada/salida para:
3?
9?
1?
5?

FUNDAMENTOS

- Grafos conexos
  - Un grafo no dirigido es conexo si existe un camino entre cualquier par de nodos que forman el grafo
  - Ejemplos:

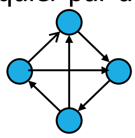


grafo conexo



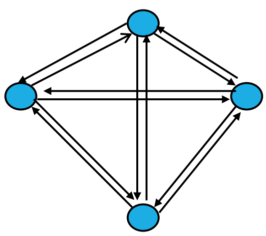
grafo no conexo con dos componentes conexas

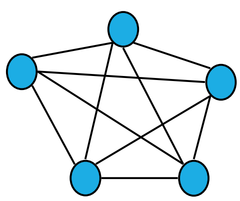
- Grafos fuertemente conexos
  - Un grafo <u>dirigido</u> es fuertemente conexo si existe un camino entre cualquier par de nodos que forman el grafo
  - Ejemplos:



FUNDAMENTOS

- Grafos Completos
  - Un grafo completo es aquel en que cada vértice está conectado con todos y cada uno de los restantes nodos.
  - Si existen n vértices, habrá (n-1) aristas en un grafo completo y dirigido, y n(n-1)/2 aristas en un grafo no dirigido completo.
  - Ejemplos:





Grafo completo no dirigido

Grafo completo dirigido

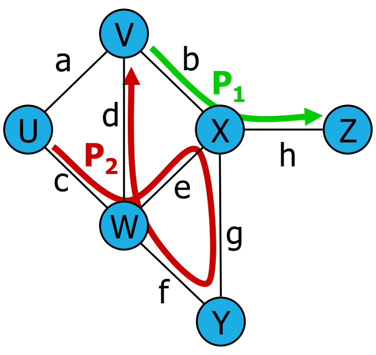
UN GRAFO CON V VÉRTICES TIENE A LO MÁS V (V – 1)/2 ARISTAS

- Prueba: El total de V2 posible pares de vértices incluyen los vértices con bucles y cuentan doble para cada par de vértice distinto, entonces el número de aristas es a lo más

$$(V2 - V)/2 = V (V - 1)/2.$$

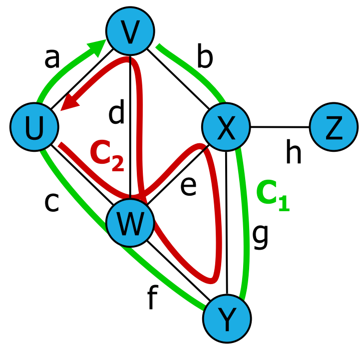
FUNDAMENTOS: CAMINO

- Un camino P de longitud n en el grafo G desde u0 a un es la secuencia de n+1 vértices P = (u0, u1, ..., un) tal que (ui,ui+1) son arcos de G para 0 ≤ i ≤ n
- Un camino es simple si todos los nodos que forman el camino son distintos, pudiendo ser iguales los extremos del camino
- Ejemplo:
  - P1 es simple
  - P2 no es simple

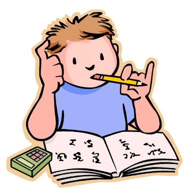


FUNDAMENTOS: CICLOS Y BUCLES

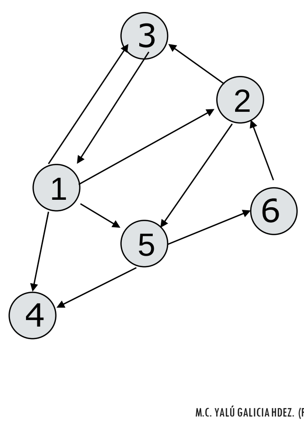
- Un ciclo es un camino simple cerrado con u0=un, compuesto al menos por tres nodos
- Un ciclo es simple si todos sus vértices y arcos son distintos
- Un arco que va desde un vértice a sí mismo (u,u) se denomina bucle o lazo
- Ejemplo
  - C1 es un ciclo simple
  - C2 es un ciclo no simple



ACTIVIDAD INDIVIDUAL



- Obtener todos los caminos de vértice 1 al 4 y del 4 a cualquier otro vértice
- Determinar cual de ellos es simple
- Hay ciclos? Cual?
- ¿Se trata de un grafo conexo?



TAD GRAFO

- Composición:
  - <grafo> :: = {<vertice>} + {<arista>}
  - <vertice> ::= <<refVertice>> + [<<info>>]
  - <arista> ::= <<refVertice>> + <<refVertice>>

  - <grafoEtiquetado> :: = {<vertice>} + {<aristaEtiquetada>}
  - <vertice> ::= <<refVertice>> + [<<info>>]
  - <aristaEtiquetada> ::= <<refVertice>> + <<refVertice>> + <<etiqueta>>

TAD GRAFO: OPERACIONES

Creación del grafo	**crearGrafo (grafo)**
Inserción de vértices	**insertarVertice(grafo, vertice)**
Eliminación de vértices	**borrarVertice(grafo, referenciaVertice)**
Inclusión de aristas	**insertarArista(grafo, vertice1, vertice2)**
Borrar aristas	**borrarArista(grafo,arista)**
Recorrido del grafo	**recorrer(grafo,tipoRecorrido)**

TAD GRAFO: OPERACIONES

Acceso a los vertices	**info(referenciaVertice) → Informacion** **grado(referenciaVertice) → Entero** ***gradoEntrante(referenciaVertice) → Entero*** ***gradoSaliente(referenciaVertice) → Entero*** **adyacentes(referenciaVertice) → {referenciaVertice}** ***incidentes{referenciaVertice) → {referenciaVertice}*** **esAdyacente(refenciaVertice1, referenciaVertice2) → Boolean**
Modificación de vertices	**asignarInfo(referenciaVertice, valorInformacion)**

Acceso a las aristas	**vertices(referenciaArista) → (refVertice, refVertice)** ***destino(referenciaArista) → refVertice*** ***origen(referenciaArista) → refVertice*** **etiqueta((referenciaArista) → etiqueta**
Modificación de aristas	**asignarEtiqueta(referenciaArista, valorEtiqueta)**

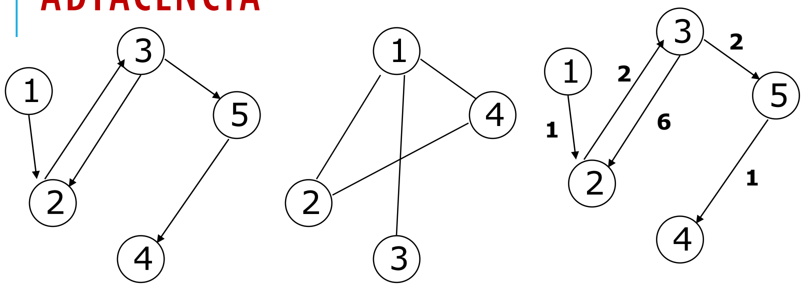
REPRESENTACIÓN DE GRAFOS

- Matriz de adyacencias
- Sea G = (V,A) un grafo de n nodos, suponemos que los nodos V = {u1,...,un} están ordenados y podemos representarlos por sus ordinales {1,2,...,n}.
- La representación de los arcos se hace con una matriz A de nxn elementos aij definida:

$$aij \begin{cases} 1 \text{ si hay arco } (ui,uj) \\ 0 \text{ si no hay arco } (ui,uj) \end{cases}$$

- En resumen, la matriz de adyacencia a es un arreglo de dos dimensiones que representa las conexiones entre pares de vértices.

REPRESENTACIÓN: MATRIZ DE ADYACENCIA



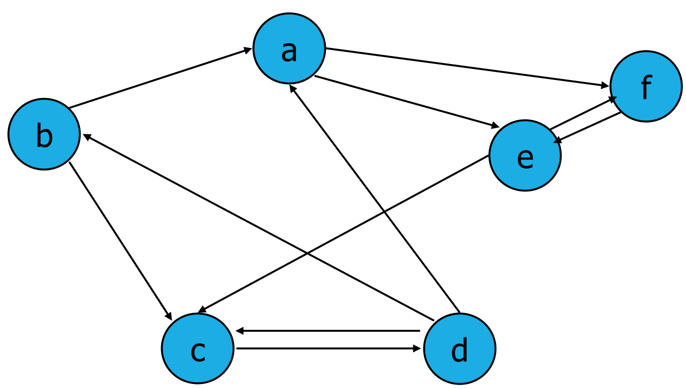
0	1	0	0	0
0	0	1	0	0
0	1	0	0	1
0	0	0	0	0
0	0	0	1	0

0	1	1	1
1	0	0	1
1	0	0	0
1	1	0	0

0	1	0	0	0
0	0	2	0	0
0	6	0	0	2
0	0	0	0	0
0	0	0	1	0

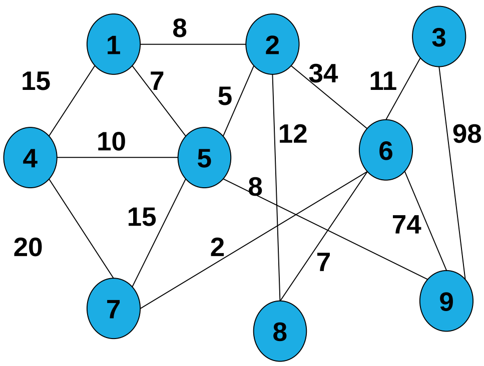
ACTIVIDAD INDIVIDUAL

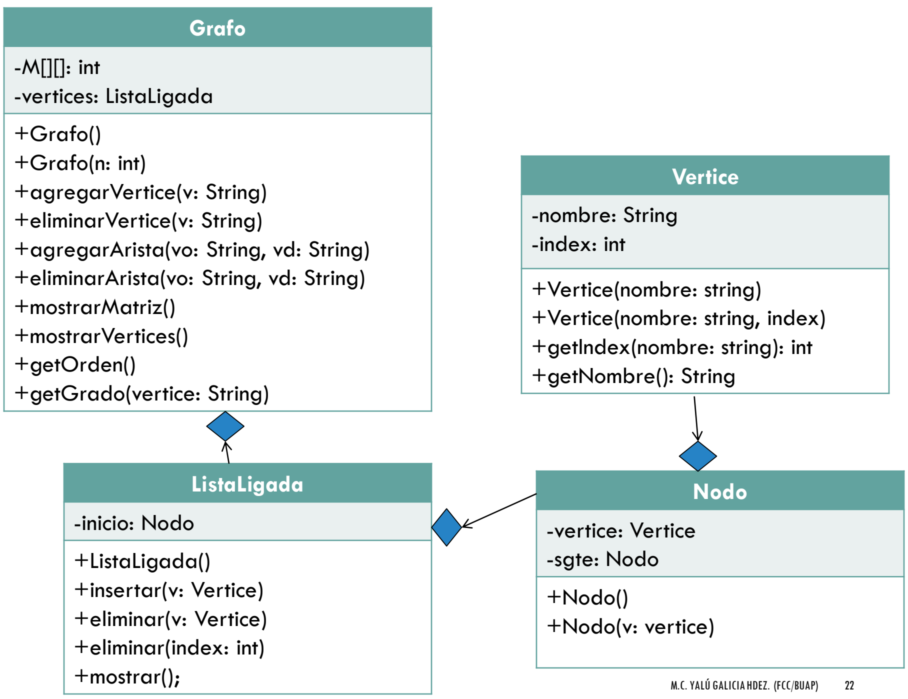
- Obtén la matriz de adyacencia para el siguiente grafo.



ACTIVIDAD COLABORATIVA

- En binas deducir la matriz de adyacencia para el siguiente grafo con aristas ponderadas.



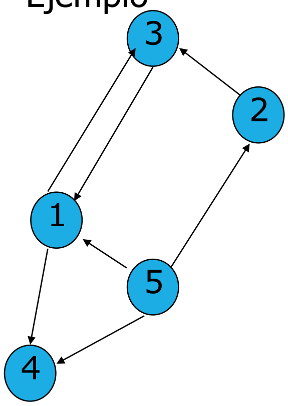


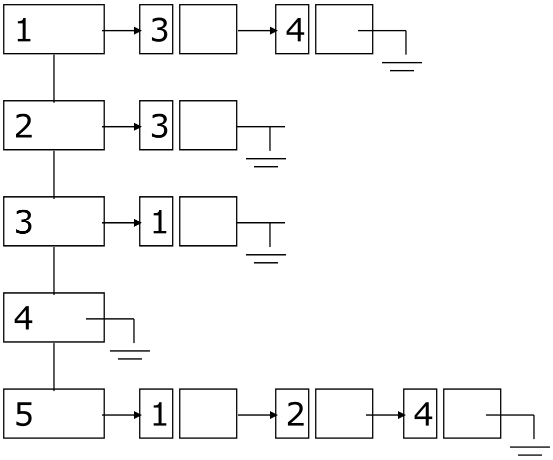
REPRESENTACIÓN

- Matriz de adyacencia
  - Poco eficiente si el nº de vértices varía a lo largo del tiempo de vida del grafo
  - Puede darse el caso de que el nº de vértices sea mayor del previsto inicialmente
  - Poco eficiente cuando el grafo tiene pocos arcos (la matriz es "dispersa")
- Listas de adyacencia
  - Representar una lista de todos los vértices
  - Cada vértice guarda una lista de adyacencia con un objeto arista para cada vértice alcanzable desde él

REPRESENTACIÓN: LISTAS DE ADYACENCIA

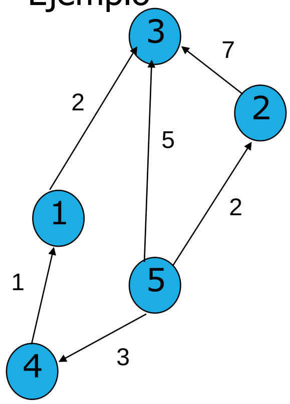
- Ejemplo

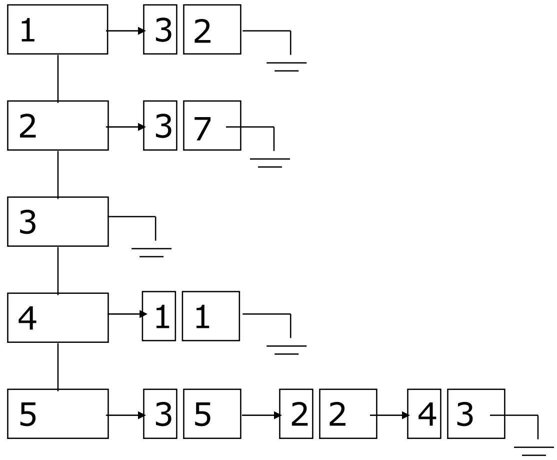




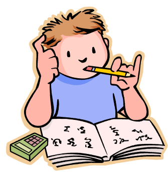
REPRESENTACIÓN: LISTAS DE ADYACENCIA

- Ejemplo

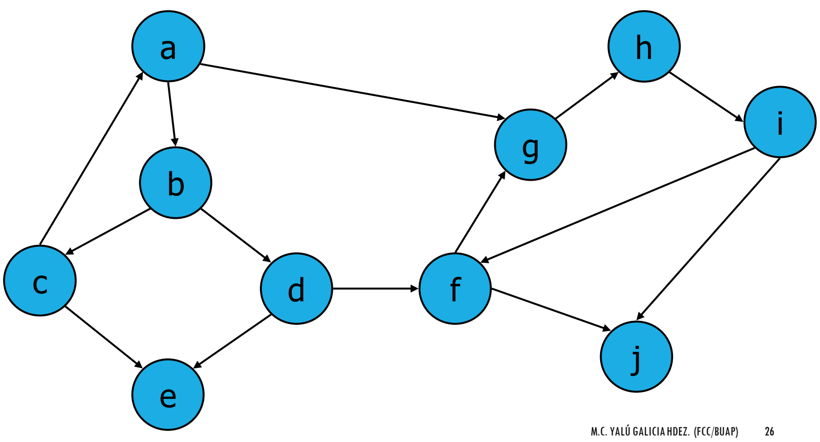




ACTIVIDAD INDIVIDUAL

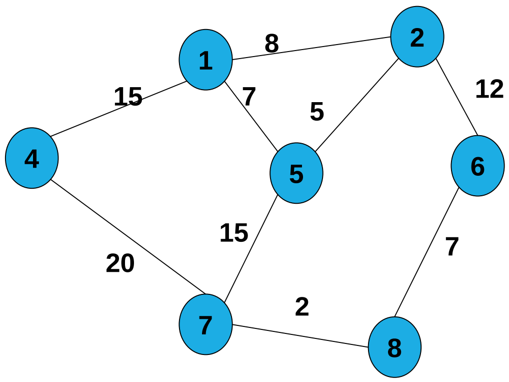


- Obtener la lista de adyacencia para el siguiente grafo.



ACTIVIDAD COLABORATIVA

- En binas obtener la lista de adyacencia para el siguiente grafo con aristas ponderadas.



¿QUE HEMOS APRENDIDO?

ALGORITMOS DE BÚSQUEDA O RECORRIDOS EN GRAFOS

Profundidad y anchura

RECORRIDOS EN GRAFOS

- Muchas de las aplicaciones computacionales naturalmente incluyen no solo un conjunto de elementos sino un conjunto de conexiones entre pares de esos elementos
- Las relaciones implicadas por esas conexiones guían inmediatamente a un conjunto de preguntas:
  - ¿Hay alguna manera de ir de un elemento a otro siguiendo las conexiones?
  - ¿Cuántos otros elementos pueden ser alcanzados desde un elemento dado?
  - ¿Cual es la mejor manera de ir desde un elemento a otro?

ALGORITMOS DE BÚSQUEDA O RECORRIDOS EN GRAFOS

- Recorrer un grafo consiste en pasar exactamente una vez por cada uno de los vértices del grafo en busca de algún vértice en particular.
- Existen varias formas de realizar este proceso dependiendo del objetivo particular:
  - Recorrido o búsqueda primero en profundidad – DFS (Depth-First Search)
  - Recorrido o búsqueda primero en achura – BFS (Breadth-First Search)

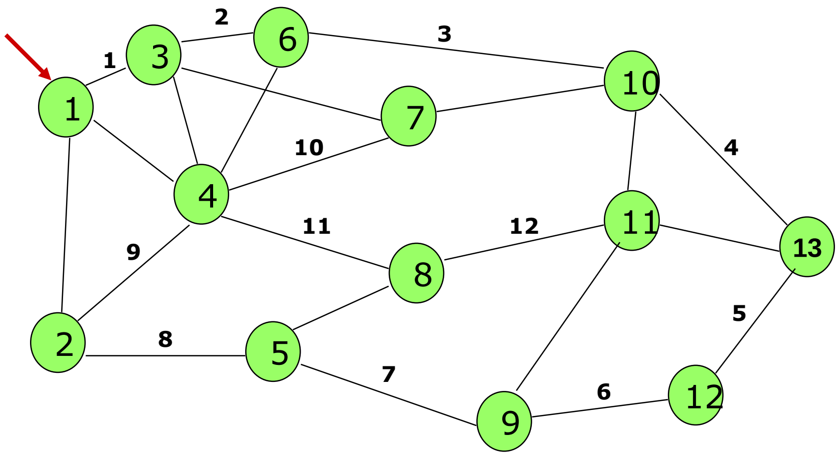
RECORRIDOS: OPERACIONES AUXILIARES

- Marcar vértice como visitado
  - Si los vértices están identificados por algún tipo ordinal, emplear un conjunto que contenga los identificadores de los vértices visitados
- Encontrar los vértices adyacentes
  - Con matrices de adyacencia: recorrer la fila correspondiente al vértice, buscando columnas igual a 1
  - Con listas de adyacencia: recorrer la lista
- Estructuras auxiliares
  - TDA Pila en Profundidad
  - TDA Cola en Anchura

RECORRIDOS

- Primero en profundidad (DFS: Depth First Search)
  - Visitar vértice inicial vi
  - Visitar vértice adyacente a vi
  - ... proceder así hasta encontrar uno ya visitado...
  - Volver atrás hasta llegar a un vértice con adyacentes sin visitar
  - El recorrido termina cuando volviendo atrás llegamos al vértice inicial vi y no quedan adyacentes por recorrer

RECORRIDO PRIMERO EN PROFUNDIDAD



1, 3, 6, 10, 13, 12, 9, 5, 2, 4, 7, 8, 11

RECORRIDO EN PROFUNDIDAD

```
DFS(vi)
{
 marcar vi como visitado
 para cada vk adyacente a vi
        si vk no visitado
        entonces DFS(vk)
}
```

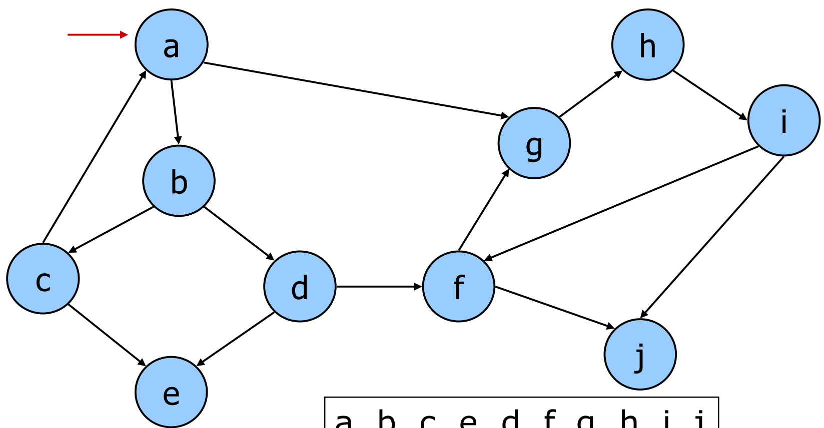

RECORRIDO PRIMERO EN PROFUNDIDAD

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0	0	0	0	0
3	1	0	0	1	0	1	1	0	0	0	0	0	0
4	1	1	1	0	0	1	1	1	0	0	0	0	0
5	0	1	0	0	0	0	0	1	1	0	0	0	0
6	0	0	1	1	0	0	0	0	0	1	0	0	0
7	0	0	1	1	0	0	0	0	0	1	0	0	0
8	0	0	0	1	1	0	0	0	0	0	1	0	0
9	0	0	0	0	1	0	0	0	0	0	1	1	0
10	0	0	0	0	0	1	1	0	0	0	1	0	1
11	0	0	0	0	0	0	0	1	1	1	0	0	1
12	0	0	0	0	0	0	0	0	1	0	0	0	1
13	0	0	0	0	0	0	0	0	0	1	1	1	0

1
2
4
3
6
10
7
11
8
5
9
12
13

ACTIVIDAD INDIVIDUAL

- Obtener la matriz de adyacencia y el recorrido DFS del siguiente grafo. Inicia en el nodo a



a, b, c, e, d, f, g, h, i, j

RECORRIDO PRIMERO EN PROFUNDIDAD

- **Solución**

	a	b	c	d	e	f	g	h	i	j
a	0	1	0	0	0	0	1	0	0	0
b	0	0	1	1	0	0	0	0	0	0
c	1	0	0	0	1	0	0	0	0	0
d	0	0	0	0	1	1	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	1	0	0	1
g	0	0	0	0	0	0	0	1	0	0
h	0	0	0	0	0	0	0	0	1	0
i	0	0	0	0	0	1	0	0	0	1
j	0	0	0	0	0	0	0	0	0	0

a
b
c
e
d
f
g
h
i
j

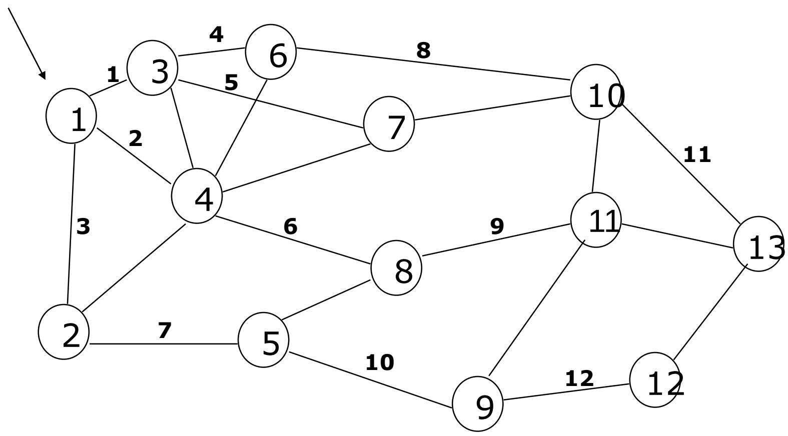
RECORRIDOS

- Primero en anchura (BFS: Breadth First Search)
  - Visitar vértice inicial vi
  - Visitar todos los vértices adyacentes a vi
  - Al terminar, comenzar a visitar los adyacentes a los adyacentes a vi
  - ... proceder así hasta que no queden vértices por visitar

RECORRIDOS: ANCHURA

```
Inicio
        marcar vi como visitado
        meter vi en cola Q
        mientras cola Q no vacía hacer
                sacar v de cola Q
                para cada vk adyacente a v hacer
                        si vk no visitado entonces
                                marcar vk visitado
                                meter vk en cola Q
                        fin_si
                fin_para
        fin_mientras
Fin
```

RECORRIDO PRIMERO EN ANCHURA



1, 3, 4, 2, 6, 7, 8, 5, 10, 11, 9, 13, 12

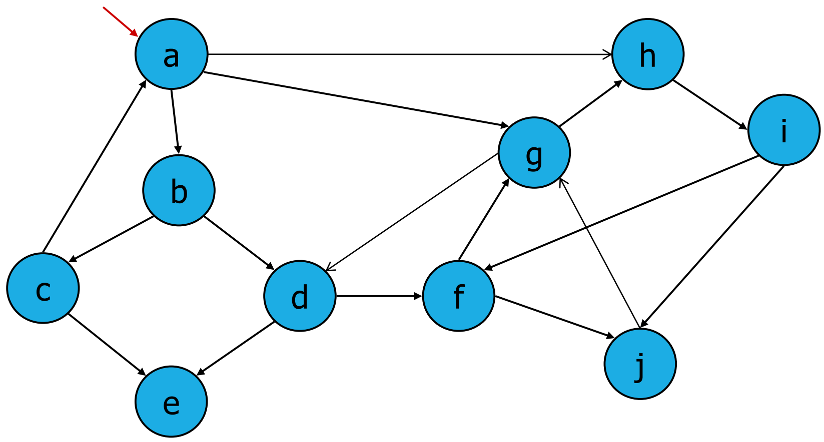
RECORRIDO PRIMERO EN ANCHURA

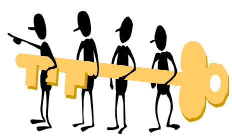
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0	0	0	0	0
3	1	0	0	1	0	1	1	0	0	0	0	0	0
4	1	1	1	0	0	1	1	1	0	0	0	0	0
5	0	1	0	0	0	0	0	1	1	0	0	0	0
6	0	0	1	1	0	0	0	0	0	1	0	0	0
7	0	0	1	1	0	0	0	0	0	1	0	0	0
8	0	0	0	1	1	0	0	0	0	0	1	0	0
9	0	0	0	0	1	0	0	0	0	0	1	1	0
10	0	0	0	0	0	1	1	0	0	0	1	0	1
11	0	0	0	0	0	0	0	1	1	1	0	0	1
12	0	0	0	0	0	0	0	0	1	0	0	0	1
13	0	0	0	0	0	0	0	0	0	1	1	1	0

1
2
3
4
5
6
7
8
9
10
11
12
13

ACTIVIDAD COLABORATIVA

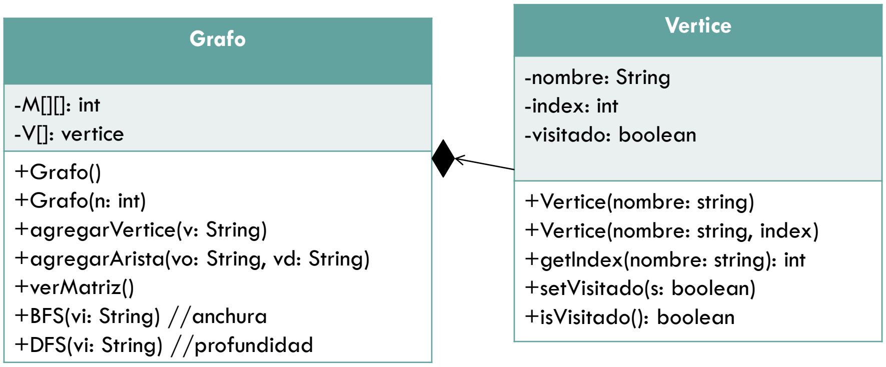
- En binas obtener la matriz de adyacencia y el recorrido del grafo en anchura





ACTIVIDAD COLABORATIVA

- En equipos de 4, suponiendo que se cuenta con la clase Grafo como se muestra.
  - Implementar los recorridos en profundidad y anchura



M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 44

¿QUE HEMOS APRENDIDO?

ALGORITMOS DEL CAMINO MÁS CORTO

Sin pesos, Dijkstra, costos negativos

ALGORITMOS DEL CAMINO MÁS CORTO

- Este tipo de aplicación es muy generalizado, pues uno de los objetivos de tener un grafo, es poder analizar los desplazamientos desde cualquier nodo a los demás que conforman el grafo, lo que se puede utilizar desde diferentes puntos de vista.
- Fundamentalmente, el planteamiento es el siguiente: a partir de un nodo i del grafo, encontrar los recorridos óptimos (caminos mínimos) para ir a cada uno de los nodos restantes, a partir de la base de que los arcos pueden representar, además de la existencia de la conexión, el costo o distancia para desplazarse de un nodo a otro.
- Existen varias formas de alcanzar la solución, algunas de ellas son: Longitud del camino sin pesos, Dijkstra, costos negativos, Floyd, etc.

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 47

ALGORITMOS DEL CAMINO MÍNIMO

- Supongamos que se cuenta con un grafo dirigido G=(V,A) en el cual cada arista (vi, vj) tiene asociado un costo Ck no negativo y donde un vértice se especifica como origen.
- El Costo de un camino {v1, v2, … vn} es

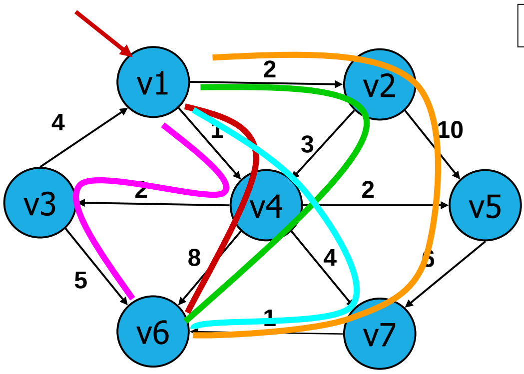
$$\sum_{k=1}^{n-1} C_k$$

- Donde Ck es la suma de los costos de las aristas del camino. A esto se le llama **longitud ponderada del camino** o longitud del camino.

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 48

ALGORITMOS DEL CAMINO MÍNIMO

- La idea es determinar el costo del camino más corto desde el origen a todos los demás vértices.



Ejemplo: v1 → v6

costo: 1+8 = 9

costo: 1+2+5 = 8

costo: 2+3+8 = 13

costo: 2+10+6+1 = 19

costo: 1+4+1 = 6

Camino más corto: { v1, v4, v7, v6 }

APLICACIONES ...

- Hay muchos ejemplo donde es interesante resolver el problema del camino más corto o longitud mínima
  - Si los vértices representan computadoras, las aristas podrían representar costos de comunicación o costos de tiempo, entonces el camino representaría la vía más económica para enviar información de una computadora a otra.
  - Si el grafo representa un mapa de vuelos, cada vértice representaría una ciudad y cada arista una ruta aérea. El camino más corto determinaría el tiempo de viaje mínimo para ir de cierta ciudad a todos los destinos.

PROBLEMA DE LA RUTA MAS CÓRTA

- El problema de la ruta más corta se puede resolver utilizando programación lineal sin embargo, debido a que el método simplex es de complejidad exponencial, se prefiere utilizar algoritmos que aprovechen la estructura en red que se tiene para estos problemas.
- Para ello, el algoritmo mantiene un conjunto S de nodos cuyos pesos finales de camino mínimo desde el nodo origen ya han sido determinados.

ALGORITMO DE DIJKSTRA

```
Dijkstra (G, s)
Inicializar
 Para cada v perteneciente a V[G] Hacer
       d[v] = infinito
       p[v] = nulo
       d[s] = 0
       S =  vacío
       Q = V[G]
       mientras Q no vacío Hacer
            u = nodo v con min d[v]
            S = S unión u //se añade al conjunto de nodos finalizados
            Para cada v perteneciente Adyacente u Hacer
               Si d[v] > d[u] + w(u,v) Entonces
                   d[v] = d[u] + w(u,v)
                   p(v) = u
Fin
```

CAMINOS DE LONGITUD MÍNIMA: DIJKSTRA

- Pasos iniciales:
  - Crear una tabla con la siguiente configuración inicial.
  - V: son los vértices
  - Visitado: indica si el vértice ha sido o no visitado.
  - dv: es la distancia temporal por cada vértice. Será la longitud del camino más corto del origen a Vk, usando como intermediarios sólo vértices conocidos.
  - pV: Vértice predecesor o anterior, es el último vértice que ocasionó un cambio de dv.

V	visitado	dv	pv
V1	F	0	0
V2	F	∞	0
V3	F	∞	0
V4	F	∞	0
V5	F	∞	0
V6	F	∞	0
V7	F	∞	0

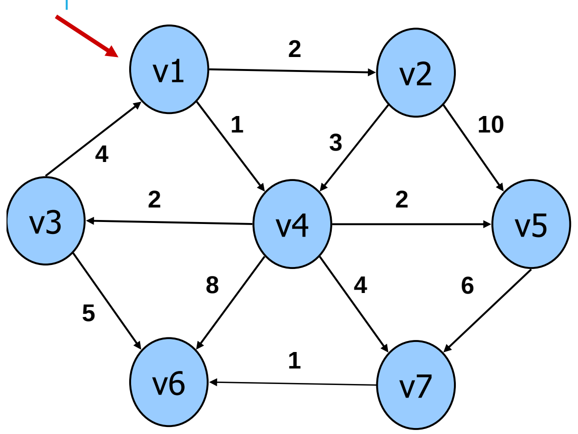
CAMINOS DE LONGITUD MÍNIMA: DIJKSTRA

- Asigna etiquetas temporales (dv) a cada vértice, que son cotas superiores de las distancias mínimas del vértice origen a cada uno de los demás
- Las etiquetas temporales se van convirtiendo en permanentes en cada iteración, representando entonces la distancia mínima del origen a cada vértice
- Comienza con
  - dv = 0 para el vértice origen
  - dv = ∞ (infinito)

CAMINOS DE LONGITUD MÍNIMA: DIJKSTRA

```
Inicio
 Mientras exista  Vertice no vistado
          vi = nodo no visitado con min dv
         poner vi como visitado
         para cada vk adyacente a vi hacer
             si vk no está visitado entonces
                   si  dv(vk) > dv(vi) + Costo(vi, vk) entonces
                             dv(vk) = dv(vi) + Costo(vi , vk)
                              pv(vk) = vi
                    fin_si
             fin_si
     fin_para
 Fin_mientras
Fin
```

EJEMPLO



V	Visitado	dv	pv
V1	F	0	-
V2	F	∞	-
V3	F	∞	-
V4	F	∞	-
V5	F	∞	-
V6	F	∞	-
V7	F	∞	-

V	visitado	dv	pv
V1	v	**0**	0
V2	F	**2**	**v1**
V3	F	∞	0
V4	F	**1**	**v1**
V5	F	∞	0
V6	F	∞	0
V7	F	∞	0

V	visitado	dv	pv
V1	v	0	0
V2	F	2	v1
V3	F	**3**	**v4**
V4	v	1	v1
V5	F	**3**	**v4**
V6	F	**9**	**v4**
V7	F	**5**	**v4**

V	visitado	dv	pv
V1	v	0	0
V2	v	2	v1
V3	F	3	v4
V4	v	1	v1
V5	F	3	v4
V6	F	9	v4
V7	F	5	v4

Ya visitado, se deja

Si $dv(v_5) > dv(v_2)+Costo(v_2, v_5)$
ajustar $dv(v_5)$
Sino
dejarlo igual

V	visitado	dv	pv
V1	V	0	0
V2	V	2	v1
V3	V	**3**	v4
V4	V	1	v1
V5	F	**3**	v4
V6	F	**9**	v4
V7	F	**5**	v4

Ya visitado, se deja

Si $dv(v_5) > dv(v_3)+Costo(v_3, v_5)$
ajustar $dv(v_5)$
Sino
dejarlo igual

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 59

V	visitado	dv	pv
V1	V	0	0
V2	V	2	v1
V3	V	**3**	v4
V4	V	1	v1
V5	V	**3**	v4
V6	F	**9**	v4
V7	F	**5**	v4

Si $dv(v_7) > dv(v_5)+Costo(v_7, v_5)$
ajustar $dv(v_7)$
Sino
dejarlo igual

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 60

V	visitado	dv	pv
V1	V	0	0
V2	V	2	v1
V3	V	3	v4
V4	V	1	v1
V5	V	3	v4
V6	F	9	v4
V7	V	5	v4

Si $dv(v_6) > dv(v_7)+Costo(v_6 , v_7)$
ajustar $dv(v_6)$
Sino
dejarlo igual

dv_7 + costo(7,6) = 5 + 1 = 6

dv_6 = 9

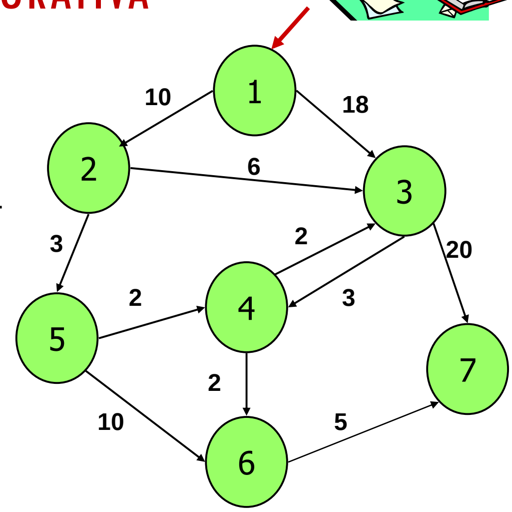
V	visitado	dv	pv
V1	T	0	0
V2	T	2	v1
V3	T	3	v4
V4	T	1	v1
V5	T	3	v4
V6	F	**6**	**v7**
V7	T	5	v4

V	visitado	dv	pv
V1	T	0	0
V2	T	2	v1
V3	T	3	v4
V4	T	1	v1
V5	T	3	v4
V6	T	6	v7
V7	T	5	v4

El camino mínimo para ir de v1 a v6 es:
v1, v4, v7, v6 costo 6

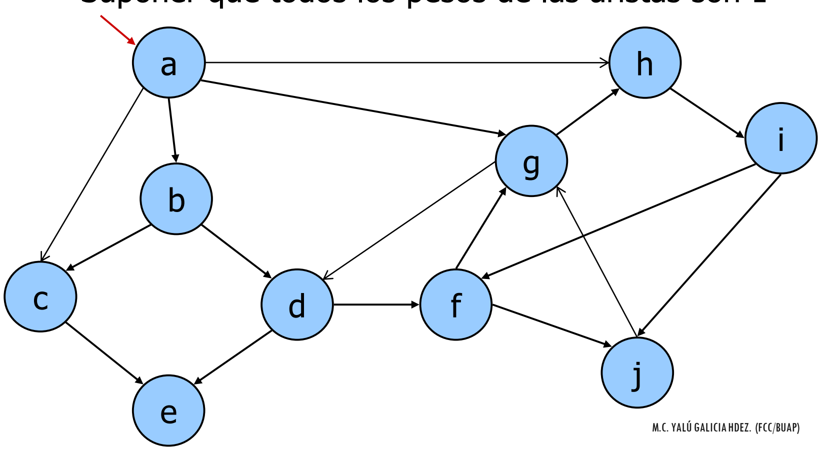
ACTIVIDAD COLABORATIVA

- En binas, aplicar el algoritmo de Dijkstra para obtener el camino de longitud mínima para llegar del vértice 1 al resto de los vértices.
- obtener el camino mínimo del vértice 1 al 6

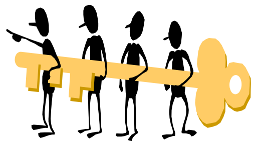


ACTIVIDAD COLABORATIVA

- En binas, aplicar el algoritmo de Dijkstra para obtener el camino de longitud mínima del vértice a al j. Suponer que todos los pesos de las aristas son 1



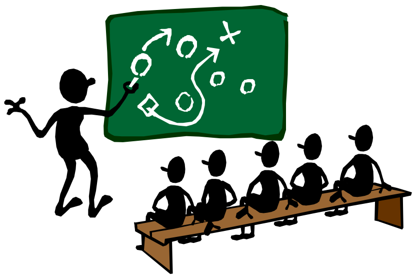
ACTIVIDAD COLABORATIVA



- En equipos de 4, suponiendo que se cuenta ya con la clase Grafo implementada usando un arreglo de Vértices y una matriz de adyacencia de enteros, agregar los siguientes métodos:
- Crear la tabla inicial con los valores iniciales correspondientes
- Del arreglo de vértices **obtener** el vértice con dv mínimo no visitado
  - Vertice getDvMinimo( )
- Obtener el costo de la arista
  - int getCosto(String vi, String vk)
  - int getCosto(int vi, int vk)
  - int getCosto(Vertice vi, Vertice vk)

ACTIVIDAD PLENARIA

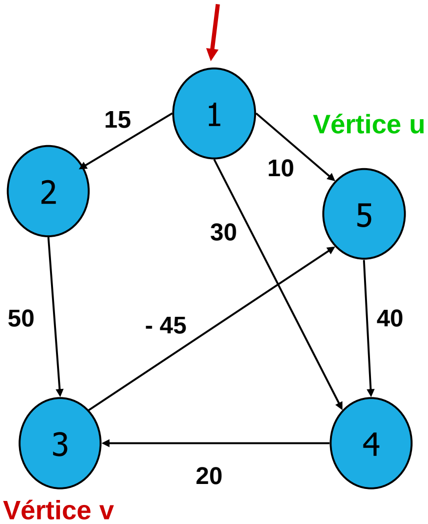
- Compartiendo la solución con todos



¿QUE HEMOS APRENDIDO?

CAMINOS DE LONGITUD MÍNIMA: COSTOS NEGATIVOS

- Si el grafo tiene aristas de costo negativo, el algoritmo de Dijkstra no funciona.
- El problema es que una vez que un vértice u se declara conocido es posible que desde algún otro vértice v desconocido haya un camino de regreso a u que sea muy negativo.
- En tal caso, tomar un camino del origen a v con regreso a u será mejor que hacerlo del origen a u sin usar v.



COSTOS NEGATIVOS O BELLMAN-FORD (NO OPTIMIZADO)

```
BellmanFord(Grafo G, nodo_fuente s)
for v ∊ V[G] do
  distancia[v]=INFINITO;
  predecesor[v]=NIL
distancia[s]=0
for i=1 to |V[G]-1| do
 for (u,v) ∊ E[G] do
          if distancia[v]>distancia[u] + peso(u,v) then
            distancia[v] = distancia[u] + peso (u,v)
            predecesor[v] = u
 // comprobamos si hay ciclos negativo
  for (u,v) ∊ E[G] do
          if distancia[v] > distancia[u] + peso(u,v) then
                    print ("Hay ciclo negativo")
                    return FALSE
return TRUE
```

COSTOS NEGATIVOS O BELLMAN-FORD

```
BellmanFord(Grafo G, nodo_fuente s)
for v ∊ V[G] do
 distancia[v]=INFINITO; padre[v]=NIL
distancia[s]=0
encolar(s, Q)
en_cola[s]=TRUE
mientras Q!=0 then
 u = extraer(Q)
   en_cola[u]=FALSE
 for v ∊ ady[u] do
          if distancia[v]>distancia[u] + peso(u,v) then
             distancia[v] = distancia[u] + peso (u,v)
             padre[v] = u
             if en_cola[v]==FALSE then
                    encolar(v, Q)
                    en_cola[v]=TRUE
```

SOBRE ESTE ALGORITMO

- Este tipo de algoritmos eran originales de ruteo de ARPANET
  - http://neo.lcc.uma.es/evirtual/cdd/tutorial/red/bellman.html
  - http://neo.lcc.uma.es/evirtual/cdd/applets/BellmanFord/Example3.html
  - http://compprog.wordpress.com/2007/11/29/one-source-shortest-path-the-bellman-ford-algorithm/

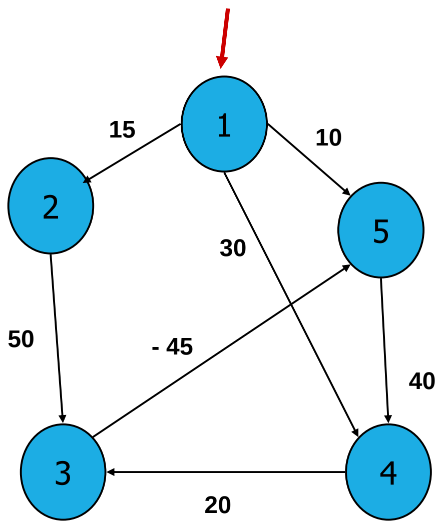
COSTOS NEGATIVOS

- Para poder resolver grafos con costos negativos olvidemos el concepto de vértices conocidos (visitados) ya que el algoritmo necesita ser capaz de cambiar de idea.
- Este algoritmo utiliza una cola e inicia con una tabla como la siguiente

V	Dv	pV
1	0	0
2	∞	0
3	∞	0
4	∞	0
5	∞	0

ALGORITMO COSTOS NEGATIVOS

```
Inicio
        encolar vi en cola Q
        mientras cola Q no vacía hacer
            sacar v de cola Q
            para cada vk adyacente a v hacer
                si Dv(vk) > Dv(v) + Costo(v, vk) entonces
                         Dv(vk) = Dv(v) + Costo(v , vk)
                         pV(vk) = v
                        si vk no está en cola Q entonces
                                meter vk en cola Q
                        fin_si
                fin_si
             fin_para
        fin_mientras
Fin
```

V	Dv	pV
1	0	0
2	oo	-
3	oo	-
4	oo	-
5	oo	-

COLA

1				

V=

Vk=

V	Dv	pV
1	0	0
2	15	1
3	oo	
4	30	1
5	10	1

V = 1, Vk = ~~2~~, ~~4~~, ~~5~~

V	Dv	pV
1	0	0
2	15	1
3	65	2
4	30	1
5	10	1

V = 2 , Vk = ~~3~~

V	Dv	pV
1	0	0
2	15	1
3	50	4
4	30	1
5	10	1

V = 4, Vk = ~~3~~

V	Dv	pV
1	0	0
2	15	1
3	50	2
4	30	1
5	10	1

V = 5, Vk = ~~4~~

~~1~~	~~2~~	~~4~~	~~5~~	3				

COLA

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 75

V	Dv	pV
1	0	0
2	15	1
3	50	4
4	30	1
5	5	3

V = 3, Vk = ~~5~~

V	Dv	pV
1	0	0
2	15	1
3	50	4
4	30	1
5	5	3

V = 5 , Vk = ~~4~~

Camino de longitud mínima de 1 → 5

{ 1, 4, 3, 5, }

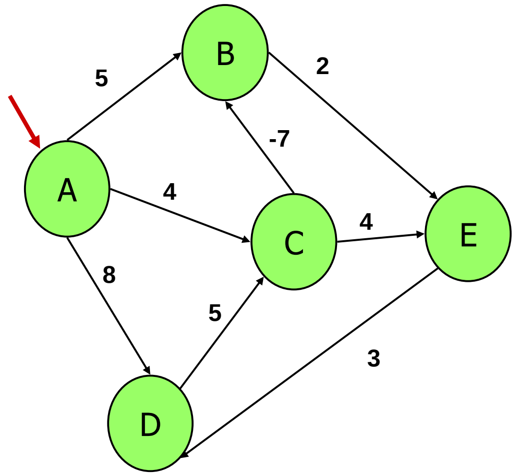
~~1~~	~~2~~	~~4~~	~~5~~	~~3~~	~~5~~			

COLA

M.C. YALÚ GALICIA HDEZ. (FCC/BUAP) 76

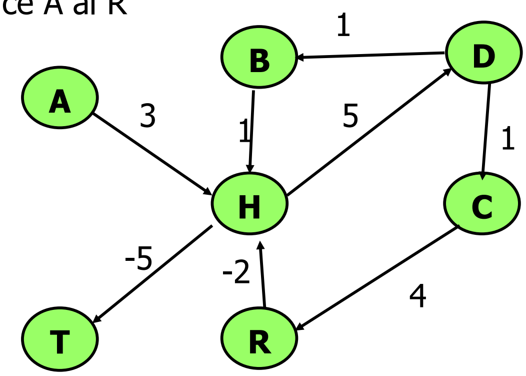
ACTIVIDAD INDIVIDUAL

- Aplicar el algoritmo de costos negativos para obtener el camino de longitud mínima para llegar del vértice A al resto de los vértices.
- obtener el camino mínimo del vértice A al E

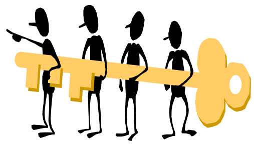


ACTIVIDAD COLABORATIVA

- En binas, aplicar el algoritmo de costos negativos para obtener el camino de longitud mínima del vértice A al R



ACTIVIDAD COLABORATIVA



- En ternas, suponiendo que se cuenta ya con la clase Grafo implementada usando un arreglo de Vértices y una matriz de adyacencia de enteros.
- Implementar en Java el método de Bellman-Ford o Costos negativos

ACTIVIDAD COLABORATIVA

- En equipos de 4
  - Analicen el escenario de la "Planta WV"
  - Identifiquen cual es el problema que puede ser resuelto utilizando los algoritmos de caminos de costo mínimo (CCM)
    - Dibujen el grafo
    - Apliquen alguno de los algoritmos de costos mínimos para solucionar el problema

QUE APRENDIMOS??