

Algoritmos y Estructuras de Datos II



Grafos Dirigidos

1

Resultados esperados del aprendizaje



- Analizar la conveniencia de representar algunos problemas reales mediante el modelo “grafos dirigido”.
- Discutir alternativas de implementación de
- Implementar algoritmos para resolver problemas cotidianos de búsqueda de caminos mínimos, centro del grafo y listado de caminos alternativos,
- Continuar desarrollando las habilidades de construcción de software sofisticadas

Algoritmos y Estructuras de Datos II

2

2

Grafos



- 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 $V = \{v_1, v_2, \dots, v_n\}$ 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.

Algoritmos y Estructuras de Datos II

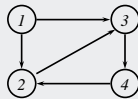
3

3

Grafos



- Existe como máximo una arista conectando cualesquiera dos vértices.
- Dos vértices se llaman **adyacentes** si existe una arista que los conecta.
- Se dice que un grafo está **conectado** si existe un camino entre cualquier par de vértices.
- La figura muestra estas definiciones: un grafo con cuatro vértices y cinco aristas. El vértice 4 es adyacente al 3 pero no al 1. El sub-grafo compuesto por los vértices 2, 3 y 4 está **conectado**.

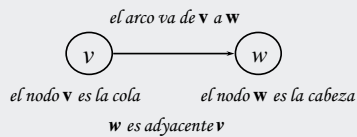


Algoritmos y Estructuras de Datos II

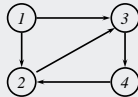
4

4

Grafos Dirigidos



Grafo dirigido de cuatro vértices y cinco aristas.



Algoritmos y Estructuras de Datos II

5

5

Grafos: Ejemplos de uso



- Los vértices pueden representar ciudades y los arcos la distancia entre ellas.
- Los vértices pueden representar bloques de un programa de computador y los arcos posibles transferencias de flujo de control.
- Los vértices pueden representar las asignaturas de una carrera universitaria y los arcos la relación de previaturas entre ellas.
- Los vértices pueden representar los estados, por ejemplo de un autómata, y los arcos la transición entre ellos.
- Los vértices pueden representar los eventos de principio y fin de una tarea, y las aristas las tareas necesarias para la ejecución de un proyecto.

Algoritmos y Estructuras de Datos II

6

6

Grafos dirigidos: camino



- Un camino en un grafo dirigido es una secuencia de vértices v_1, v_2, \dots, v_n , tal que
 - $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$ son arcos.
- Este camino va del vértice v_1 al vértice v_n , pasando por todos los vértices intermedios.
- La longitud de un camino es el número de arcos del camino. Un vértice por sí mismo implica un camino de largo 0.
- Un camino es simple si todos sus vértices, excepto tal vez el primero y el último, son distintos.
- Un ciclo es un camino simple de longitud por lo menos dos, que empieza y termina en el mismo vértice.

Algoritmos y Estructuras de Datos II

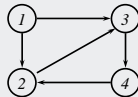
7

7

Representaciones de grafos dirigidos



- **Matriz de adyacencias:** requiere un espacio mínimo del orden de n^2 , siendo n la cantidad de vértices.
- **Lista de adyacencias:** requiere una cantidad de espacio proporcional a la suma de la cantidad de arcos más la cantidad de vértices.
- Las listas pueden implementarse en forma estática o dinámica.

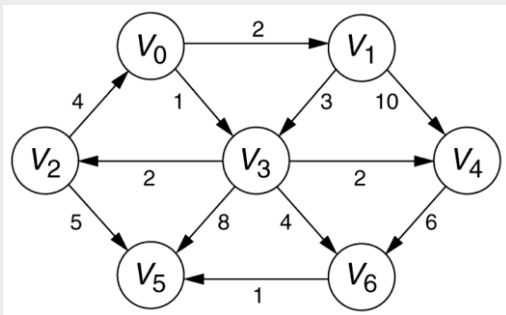


Algoritmos y Estructuras de Datos II

8

8

Grafo dirigido

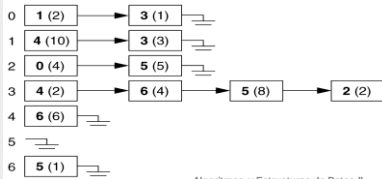
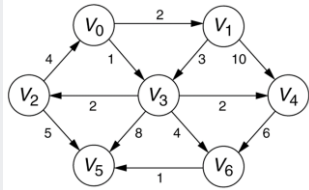


Algoritmos y Estructuras de Datos II

9

9

Lista de adyacencias



Algoritmos y Estructuras de Datos II

10

10

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

Algoritmos y Estructuras de Datos II

11

11

Problema de los caminos más cortos con un origen: el algoritmo de Dijkstra


- Sea un **Grafo dirigido** $G = (V, A)$ en que cada arco tiene una etiqueta no negativa, y donde un vértice se especifica como **origen**
- **Problema:** determinar el **costo del camino más corto** desde el **origen** a **cada uno** de los demás vértices de V
- La longitud de un camino es la suma de los costos de los arcos del camino
- **Técnica "ávida"**
- S conjunto de vértices cuya distancia más corta al origen es conocida. Al principio S sólo contiene el origen
- En cada paso se agrega algún vértice v restante a S , cuya distancia desde el origen es la más corta posible
- D vector que registra la longitud de camino especial más corta a cada vértice

Algoritmos y Estructuras de Datos II

12

12

Algoritmos “ávidos” (“avaros”, “voraces” – “greedy”)




- Dado C (entradas) , el algoritmo ávido devuelve en cada iteración un conjunto S tal que $S \subseteq C$
- S “prometedor”
- **elementos de la técnica:**
 - C – conjunto de candidatos (entradas)
 - Función **solución**
 - Función de **selección**
 - Función de **factibilidad**
 - Función **objetivo**

Algoritmos y Estructuras de Datos II13

13

Algoritmos “ávidos” Funcionamiento básico




1. Elegir el mejor elemento de C posible (elemento *más prometedor*)
2. Retirarlo del conjunto C de candidatos
3. Comprobar si produce una solución factible, y si es así, incluirlo en S
4. Si no es factible, descartar
5. Repetir 1-4 hasta alcanzar la función objetivo o agotar los elementos de C

Algoritmos y Estructuras de Datos II14

14

Problema de los caminos más cortos con un origen: el algoritmo de Dijkstra



- Dado un grafo dirigido $G=(V,A)$ en que cada arco tiene una etiqueta no negativa, y donde un vértice se especifica como **origen**
- **Problema:** determinar el costo del camino más corto desde el **origen** a **todos** los demás vértices de V
- La longitud de un camino es la suma de los costos de los arcos del camino
- **Técnica “ávida”**
- S conjunto de vértices cuya distancia más corta al origen es conocida. Al principio S sólo contiene el origen
- En cada paso se agrega algún vértice v restante a S , cuya distancia desde el origen es la más corta posible
- D vector que registra la longitud de camino especial más corta a cada vértice

Algoritmos y Estructuras de Datos II15

15

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

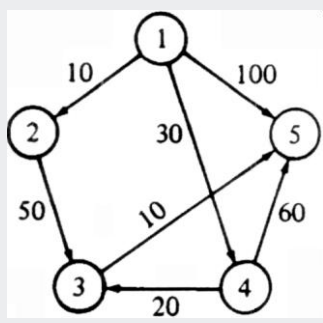
ParaCada v perteneciente a V-S hacer

$$D[v] = \min (D[v], D[w] + \text{costo}(w,v))$$

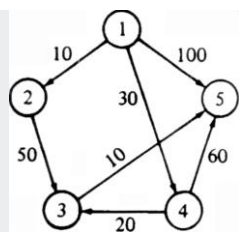
FinMientras;

FIN {Dijkstra}

16



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

18

Dijkstra: recuperación de caminos



- Usar otro array P de vértices, tal que P[v] contiene el vértice inmediato anterior a v en el camino menor. Inicialmente todos los P[v] = 1.

procedure Dijkstra (con caminos)

COM

Inicializar S, D, P. 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

Para cada v perteneciente a V-S hacer

si D[w] + costo(w,v) < D[v] hacer

D[v] = D[w] + costo(w,v) y P[v] = w

fin si

FinMientras;

FIN {Dijkstra}

Algoritmos y Estructuras de Datos II

19

19

Dijkstra: recuperación de caminos



- Para el grafo del ejercicio anterior, el vector P al final del algoritmo tendrá los valores:

$P[2] = 1, P[3] = 4, P[4] = 1, \text{ y } P[5] = 3.$

- Para encontrar el camino más corto desde el vértice 1 al vértice 5, recorreremos los predecesores en orden inverso, comenzando por el vértice 5
- Vemos que 3 es el predecesor de 5, 4 el de 3, y 1 el de 4
- Entonces el camino más corto de 1 a 5 es:

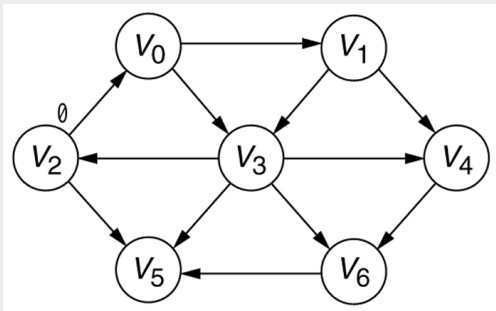
1, 4, 3, 5

Algoritmos y Estructuras de Datos II

20

20

El grafo luego de marcar los vértices alcanzables desde el origen (V2) con costo 0

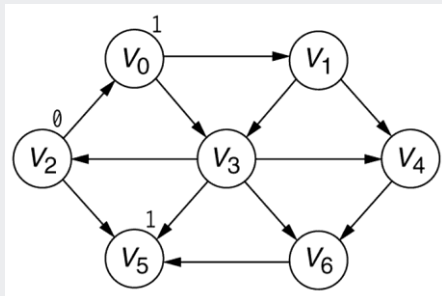


Algoritmos y Estructuras de Datos II

21

21

El grafo luego de marcar los vértices alcanzables desde el origen con costo 1

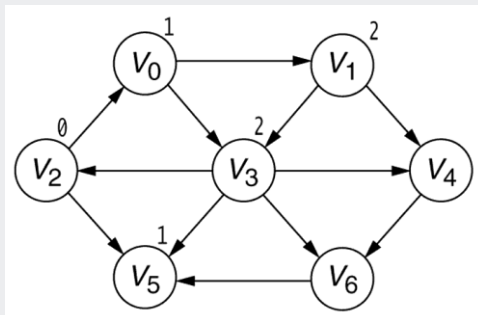


Algoritmos y Estructuras de Datos II

22

22

El grafo luego de marcar los vértices alcanzables desde el origen (V_2) con costo 2

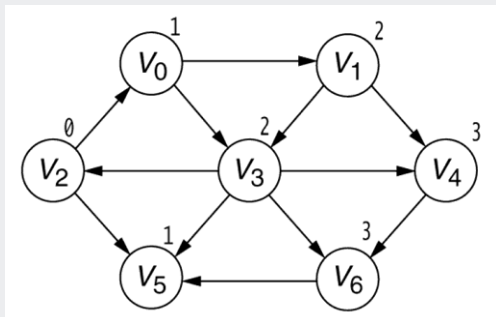


Algoritmos y Estructuras de Datos II

23

23

Los caminos más cortos finales



Algoritmos y Estructuras de Datos II

24

24

Dibuje el Grafo y Resuelva por Dijkstra




	A	B	C	D	E
A	-	1	2	7	-
B	7	-	1	2	-
C	-	-	-	3	-
D	6	-	4	-	4
E	-	2	-	8	-

Algoritmos y Estructuras de Datos II

25

25

Caminos más cortos entre todos los pares




- Problema: obtener una tabla que indique el menor camino entre todos los pares de vértices
 - Ejemplo: tiempos de vuelos entre ciudades
- Dado un grafo dirigido $G = (V, A)$ en que cada arco tiene un costo no negativo $C[v, w]$
- Encontrar el camino de longitud más corta para cada par ordenado de vértices (v, w)
- Se podría utilizar Dijkstra, tomando por turno cada vértice como origen...
- Más directo: algoritmo de Floyd.

Algoritmos y Estructuras de Datos II

26

26

El algoritmo de Floyd



- Usa una matriz A de $n \times n$ en la que se calculan las longitudes de los caminos más cortos.
- Inicialmente $A[i, j] = C[i, j]$ para todo $i \neq j$, y si no hay arco de i a j se pone ∞ . Los elementos de la diagonal se hacen 0 .
- n iteraciones en la matriz A . Al final de la k -ésima iteración $A[i, j]$ tendrá por valor la longitud menor de ir de i hasta j y que no pase por un vértice mayor que k

$$A_k[i, j] = \min \begin{cases} A_{k-1}[i, j] \\ A_{k-1}[i, k] + A_{k-1}[k, j] \end{cases}$$

Algoritmos y Estructuras de Datos II

27

27

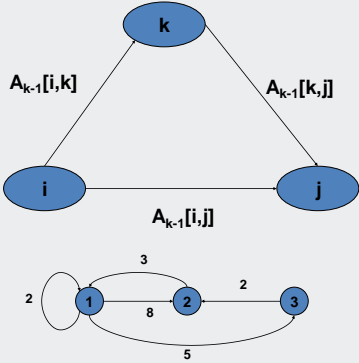
El algoritmo de Floyd

Problema de los caminos más cortos entre todos los pares:

```
Función Floyd (A : array[1..n,1..n] of real;  
                C : array[1..n,1..n] of real);  
var i, j, k : integer;  
begin  
  for i:= 1 to n do  
    for j:= 1 to n do  
      A[i,j]:= C[i,j];  
  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];  
end;
```

28

Algoritmo de Floyd



29

Floyd: recuperación de caminos

- Agregamos una matriz P en donde $P[i, j]$ contiene aquél vértice k que determinó que Floyd encontrara el menor valor para $A[i, j]$.
- Si $P[i, j]=0$, entonces el camino más corto desde i a j es directo, siguiendo el arco de i a j .
- Para el grafo del ejemplo anterior, la matriz P al final de Floyd contiene:

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

P

30

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;
```

31

Floyd: recuperación de caminos



```
procedure camino ( i, j: integer );
var k: integer;
begin
  k := P[i, j];
  if k = 0 then salir;
  camino(i, k);
  imprimir(k);
  camino(k, j)
end; { camino }
```

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

P

32

Localización del centro de un grafo:
excentricidad



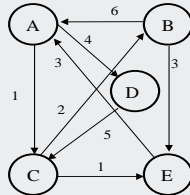
- Dado $G=(V,A)$, la **excentricidad** de un nodo v se define como la **máxima** de todas las longitudes mínimas de los caminos entre cada uno de los otros nodos y el nodo v .
- El centro de G es un vértice de mínima excentricidad.
- Para obtener el centro de un grafo hacer:
 - aplicar Floyd para obtener el largo de los caminos,
 - encontrar el **máximo** valor en cada columna i , y con ello se obtiene la excentricidad de i ,
 - encontrar el vértice con **excentricidad mínima**: el **centro** de G .

33

Ejercicio Floyd y excentricidad



- Utilizando Floyd calcule los caminos mínimos
- ¿cuáles son las excentricidades de todos los vértices?
- ¿cuál es el Centro del Grafo, y cual la excentricidad?



Algoritmos y Estructuras de Datos II

34

34

Cerradura transitiva: algoritmo de Warshall.



- Puede ser interesante saber sólo si existe un camino que vaya del vértice i al j
- En este caso, la matriz de costos indicará 1 si hay arco, o 0 si no lo hay
- Se desea obtener la matriz A tal que $A[i,j]=1$ si hay camino de i a j , o 0 si no lo hay
- Se conoce como cerradura transitiva de la matriz de adyacencia

Algoritmos y Estructuras de Datos II

35

35

Cerradura transitiva: algoritmo de Warshall.



procedure Warshall (A : **array**[1..n,1..n] **of** boolean;
 C : **array**[1..n,1..n] **of** boolean);

i, j, k : integer;

COM

for $i := 1$ **to** n **do**

for $j := 1$ **to** n **do**

$A[i,j] := C[i,j]$;

for $k := 1$ **to** n **do**

for $i := 1$ **to** n **do**

for $j := 1$ **to** n **do**

if $A[i,j] = \text{false}$

then $A[i,j] := A[i,k] \text{ and } A[k,j]$;

FIN;

Algoritmos y Estructuras de Datos II

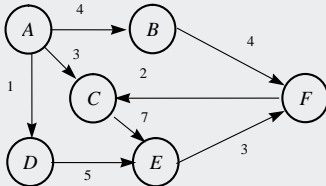
36

36

Ejercicios



- Aplique el algoritmo de Warshall para calcular la cerradura transitiva del siguiente grafo



- Calcule los caminos mínimos, excentricidad y centro del grafo

Algoritmos y Estructuras de Datos II

37

37

Recorridos de grafos dirigidos



- Es necesario visitar los vértices y los arcos de forma sistemática.
- Búsqueda en profundidad, generalización del recorrido en preorden de árboles.
- Dado un grafo G , en el cual inicialmente todos los vértices están marcados como no visitados
- Se selecciona un vértice v como vértice de partida y se marca como visitado.
- Luego se recorre cada vértice no visitado adyacente a v usando recursivamente la búsqueda en profundidad.
- Una vez visitados todos los vértices que se pueden alcanzar desde v , la búsqueda está completa. Si quedan vértices sin visitar, se selecciona otro como partida y se repite el procedimiento

Algoritmos y Estructuras de Datos II

38

38

Búsqueda en profundidad

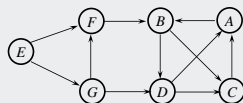


método Tvertice.bpf ();

```

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

¿De qué orden es el tiempo de ejecución es este algoritmo?



Algoritmos y Estructuras de Datos II

39

39

Búsqueda en profundidad, análisis



método Tvertice.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 $n \leq a$ 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.

Algoritmos y Estructuras de Datos II

40

40

Búsqueda en profundidad



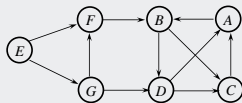
- Como se ve, el algoritmo anterior no tiene ninguna salida, “solamente” realiza la visita de los vértices en el orden indicado.
- Es el algoritmo base para la obtención de caminos, verificación de ciclos, etcétera.
- Un camino desde un vértice Origen a otro vértice Destino, puede ser obtenido a partir del siguiente algoritmo.

Algoritmos y Estructuras de Datos II

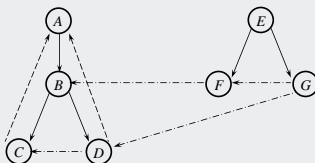
41

41

Recorrido en profundidad



- Bosque abarcador en profundidad.



Grafo recorrido inicialmente en A y luego en E

Algoritmos y Estructuras de Datos II

42

42

Bosque abarcador en profundidad.



- Los arcos que llevan a vértices nuevos se conocen como “**arcos de árbol**” y forman un “**bosque abarcador en profundidad**”.
- Existen otros tres tipos de arcos:
 - **Arco de retroceso.** Va de un vértice a uno de sus antecesores en el árbol.
 - **Arco de avance.** Va de un vértice a un descendiente propio.
 - **Arco cruzado.** Va de un vértice a otro que no es ni antecesor ni descendiente.

Algoritmos y Estructuras de Datos II

43

43

Identificación de los tipos de arco



- Numerar en profundidad.
- Si el arco es de avance, va de un vértice de baja numeración a uno de alta numeración (que ya fue visitado).
- Si es de retroceso, a la inversa (y el destino es un ancestro)
- Los arcos cruzados van de alta numeración a baja numeración (pero el destino no es un ancestro)
- w es un descendiente de v si y sólo si,

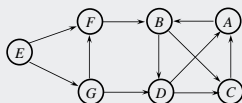
$$Num(v) < Num(w) \Leftrightarrow Num(v) + \text{Cantidad de descendientes de } v$$

Algoritmos y Estructuras de Datos II

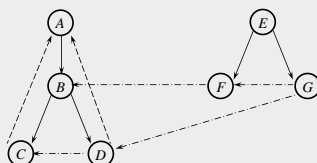
44

44

Recorrido en profundidad – identificación de arcos



- Bosque abarcador en profundidad.



Grafo recorrido inicialmente en A y luego en E

Algoritmos y Estructuras de Datos II

45

45

Obtención de un camino



```
método Camino (Origen, Destino : Tvértice;  
                ElCamino: TCamino);  
w : Tvértice;  
COM  
  Visitar(Origen); Agregar(Origen, ElCamino)  
  Para cada adyacente w  
    Si w = Destino entonces Guardar(ElCamino+Destino)  
    Sino  
      Si no(visitado(w)) entonces Camino(w, Destino, ElCamino)  
      Fin si  
    Fin si  
  Fin para cada  
  Quitar(Origen, ElCamino)  
FIN
```

Algoritmos y Estructuras de Datos II

46

46

Formas de recuperar caminos



- El algoritmo presentado es una propuesta que admite variantes, por ejemplo en qué lugar verificar si se llega al Destino. Continúa realizando una "bpf" luego de haber encontrado un camino.
- "El Camino" que se define, en realidad es una colección de vértices que se maneja con disciplina LIFO, y que contiene todos los vértices que todavía están pendientes en la recursión.
- ¿Cómo puede usarse para recuperar todos los caminos posibles entre un par de vértices? Reflexionar sobre el hecho de estar marcado como visitado.
- ¿Cómo puede usarse para ayudar a clasificar los tipos de arcos de una recorrida? Notar la diferencia entre estar en el camino y haber sido visitado.

Algoritmos y Estructuras de Datos II

47

47

Grafos dirigidos acíclicos



- El GDA es un grafo dirigido sin ciclos.
 - Son más generales que los árboles, pero menos que los grafos dirigidos arbitrarios.
 - Útiles para representar expresiones aritméticas con subexpresiones comunes.
 - También son apropiados para representar órdenes parciales.
- Ejemplo de expresión aritmética:

$$((a + b) * c + ((a + b) + e) * (e + f)) * ((a + b) * c)$$

Algoritmos y Estructuras de Datos II

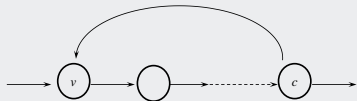
48

48

Prueba de Acicilidad.



- Se realiza búsqueda en profundidad y **si se encuentra un arco de retroceso, el grafo tiene un ciclo.**
- Si un grafo dirigido tiene un ciclo, siempre habrá un arco de retroceso en la búsqueda en profundidad.



Algoritmos y Estructuras de Datos II

49

49

Clasificación topológica



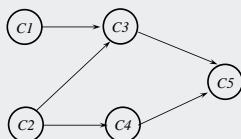
- Es un proceso de asignación de un orden lineal a los vértices de un grafo dirigido acíclico tal que, si existe un arco del vértice i al vértice j , i aparece antes que j en el ordenamiento lineal de todos los vértices.
- Ejemplos.:
 - Proyecto que se divide en tareas, en donde pueden apreciarse relaciones de órdenes específicos de ejecución
 - Previaturas de cursos
- Los GDA pueden usarse para modelar de forma natural estas situaciones.

Algoritmos y Estructuras de Datos II

50

50

Clasificación topológica



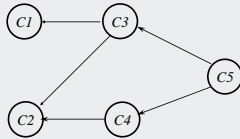
- Ej: estructura de previaturas de 5 cursos
- $C1, C2, C3, C4, C5$ es una clasificación topológica de este grafo

Algoritmos y Estructuras de Datos II

51

51

Clasificación Topológica



- Invertir los arcos, indicando entonces las dependencias
 - ej, C5 depende de C3 y de C4
- Ejecutar una búsqueda en profundidad, con procesamiento en la salida recursiva

Algoritmos y Estructuras de Datos II

52

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
```

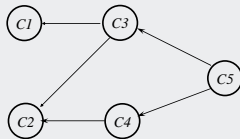
```
  imprimir (); //agregar "this" al principio de la lista de  
  previas....
```

```
  FIN; {ClasificacionTopologica}
```

Algoritmos y Estructuras de Datos II

53

Clasificación Topológica



- Invertir los arcos, indicando entonces las dependencias
 - ej, C5 depende de C3 y de C4
- Ejecutar el algoritmo anterior, para el vértice destino (o para cada vértice que no tenga arcos incidentes, o sea, que son vértices "finales")

Algoritmos y Estructuras de Datos II

54

52

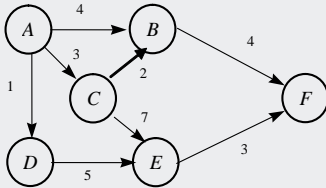
53

54

Ejercicios



- Dado el siguiente grafo, halle uno o más órdenes topológicos para la tarea "F"



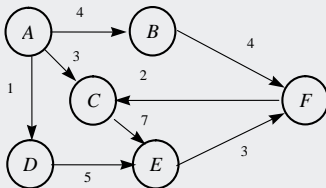
Algoritmos y Estructuras de Datos II

55

Ejercicios



- Dado el siguiente grafo
 - Determine si contiene ciclos
 - encuentre los componentes fuertes y el grafo reducido



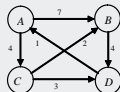
Algoritmos y Estructuras de Datos II

56

Ejercicios de grafos



- Aplique el algoritmo de Floyd al siguiente grafo, mostrando cada iteración.



- ¿Cuál es el Centro del Grafo? ¿Por qué?
- Escriba un algoritmo que implemente una BÚSQUEDA EN AMPLITUD de un grafo, y exprese el orden del tiempo de ejecución del mismo

Algoritmos y Estructuras de Datos II

57
