

Técnica de diseño #2: Programación dinámica



Análisis y Diseño de Algoritmos
Ing. Román Martínez M.

Programación dinámica



- Técnica para enfrentar la solución de problemas.
- Consiste en dividir un problema en instancias del problema más pequeñas, y resolver primero las instancias más pequeñas, almacenar los resultados y contar con estos para no recalcularlos cuando la solución de instancias de mayor nivel lo necesiten.
- Utiliza un enfoque de tipo ***bottom-up***.
- Transforma a una solución natural de tipo **recursivo, en una solución iterativa,** almacenando resultados.

EJEMPLO: Coeficiente binomial

- El coeficiente binomial es un término que sirve para encontrar la cantidad de combinaciones de tamaño 'k' entre 'n' elementos, y se obtiene así:

$$\binom{n}{k} = \frac{n!}{k! (n-k)!}$$

- Evitando el comportamiento de la función factorial, el coeficiente binomial también se puede definir así:

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k = 0 \text{ o } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{si } 0 < k < n \end{cases}$$

Coeficiente binomial

- La definición recursiva, permite plantear un algoritmo de solución con la técnica de divide y vencerás:

function binomial(int n, k): int;

if (k=0) or (k=n) then return 1

else return binomial(n-1, k-1) + binomial(n-1,k);

$$O\left(\begin{matrix} n \\ k \end{matrix}\right)$$

- ¿Cuál es el inconveniente de este algoritmo?*
- Se recalculan los mismos términos varias veces..
- Ejemplo: ***binomial(n-1, k-1)*** y ***binomial(n-1,k)*** requieren ambos de ***binomial(n-2,k-1)***.

Coeficiente binomial

- Enfoque de solución con PROGRAMACIÓN DINÁMICA:
 - Considerar que se tiene un arreglo B en donde se guardan los coeficientes binomiales (i,j).
 - Dada la definición recursiva del problema se tiene que:
$$B[i,j] = 1 \text{ si } j = 0 \text{ o } j = i$$
$$B[i,j] = B[i-1,j-1] + B[i-1,j] \text{ si } i > 0 \text{ y } i < j$$
 - Resolver las instancias de B comenzando por el primer renglón (problema más pequeño), y continuando secuencialmente.

Coeficiente binomial



EJEMPLO: Encontrar $B[4,2]$

- $B[0,0] = 1$
- $B[1,0] = 1$
- $B[1,1] = 1$
- $B[2,0] = 1$
- $B[2,1] = B[1,0] + B[1,1] = 2$
- $B[2,2] = 1$
- $B[3,0] = 1$
- $B[3,1] = B[2,0] + B[2,1] = 3$
- $B[3,2] = B[2,1] + B[2,2] = 3$
- $B[4,0] = 1$
- $B[4,1] = B[3,0] + B[3,1] = 4$
- $B[4,2] = B[3,1] + B[3,2] = 6$

Algoritmo para el coeficiente binomial

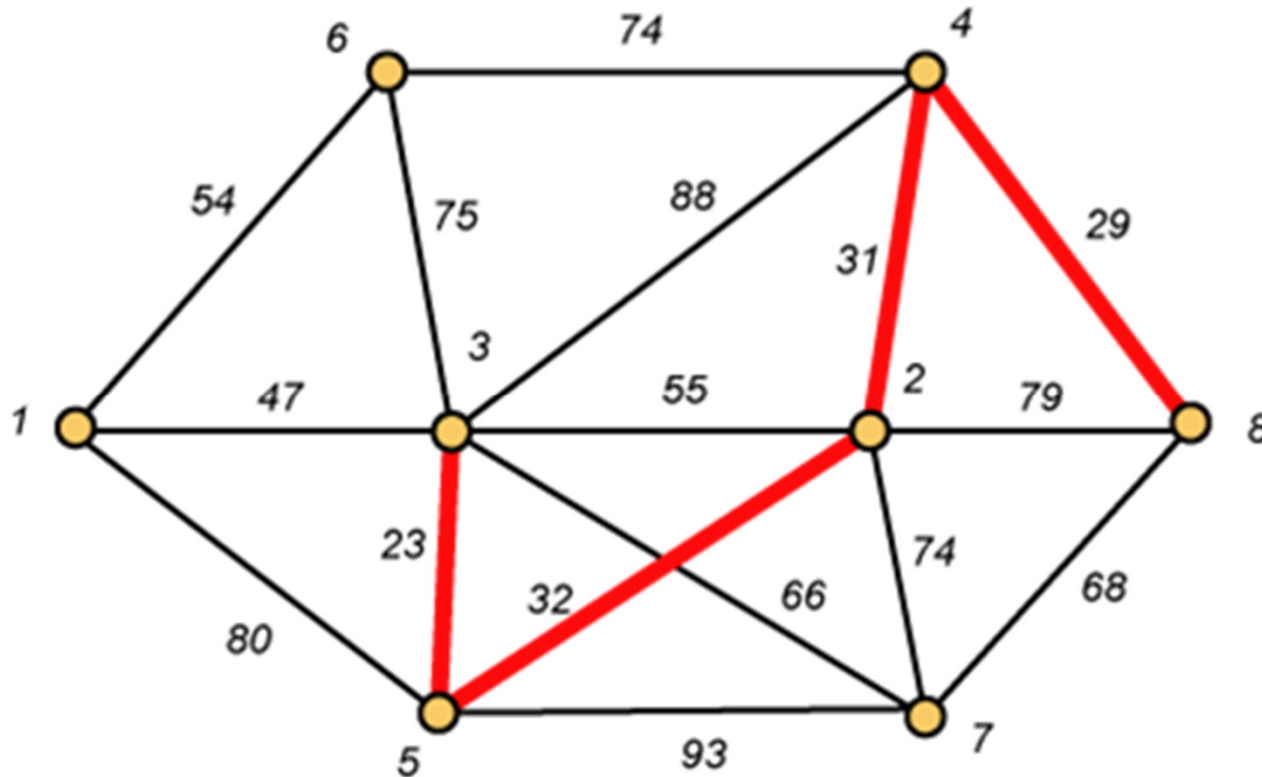


```
Function binomial (int n, k) : int;  
for i = 0 to n do  
    for j = 0 to minimo(i, k) do  
        if (j=0) or (j=i) then B[i, j] = 1  
        else  
            B[i, j] = B[i-1, j-1] + B[i-1, j];  
    return B[n, k];
```

$O(nk)$

EJEMPLO: Camino más corto

- ¿Cuál es el camino o ruta óptima para ir de un nodo a otro en un grafo ponderado, sin importar por cuántos nodos se pasa?



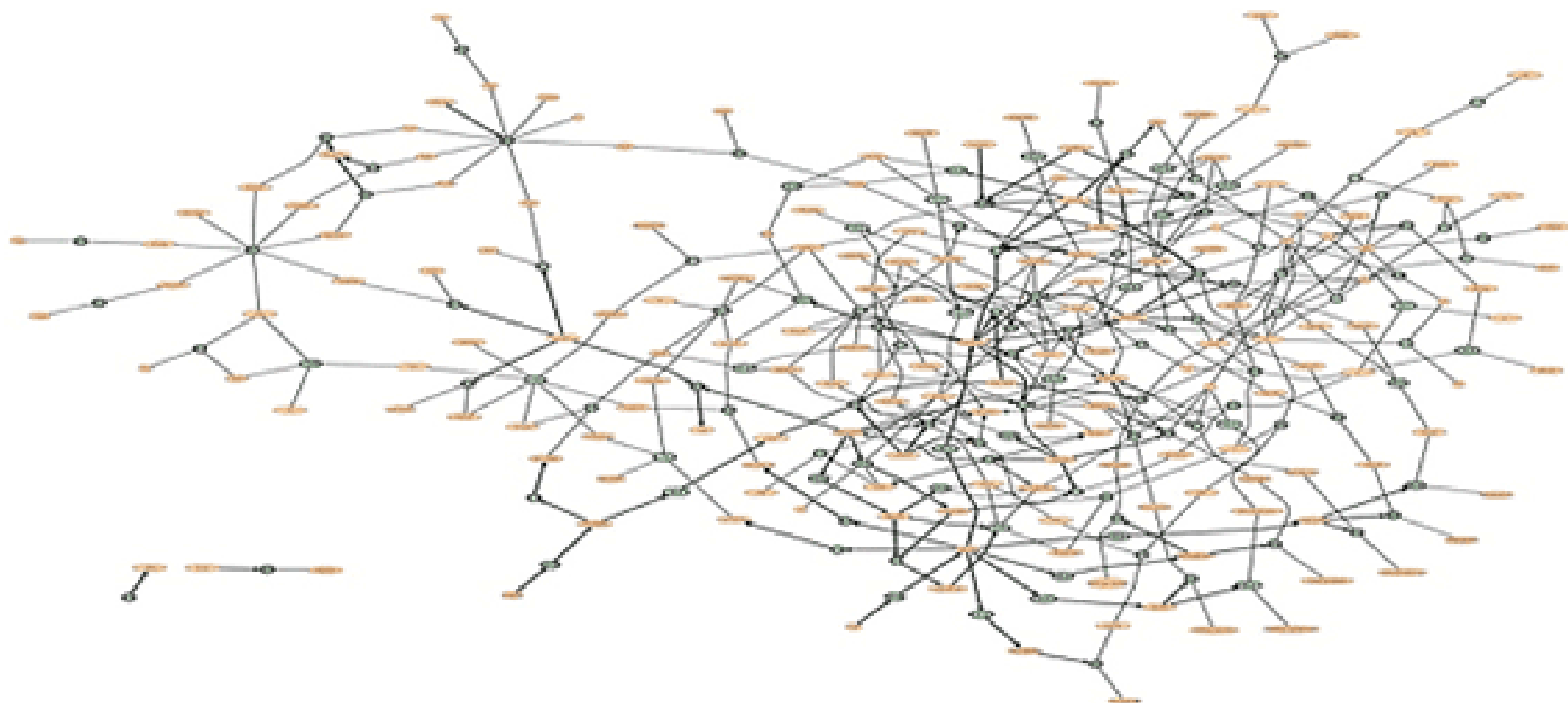
Breve repaso de terminología de **GRAFOS**

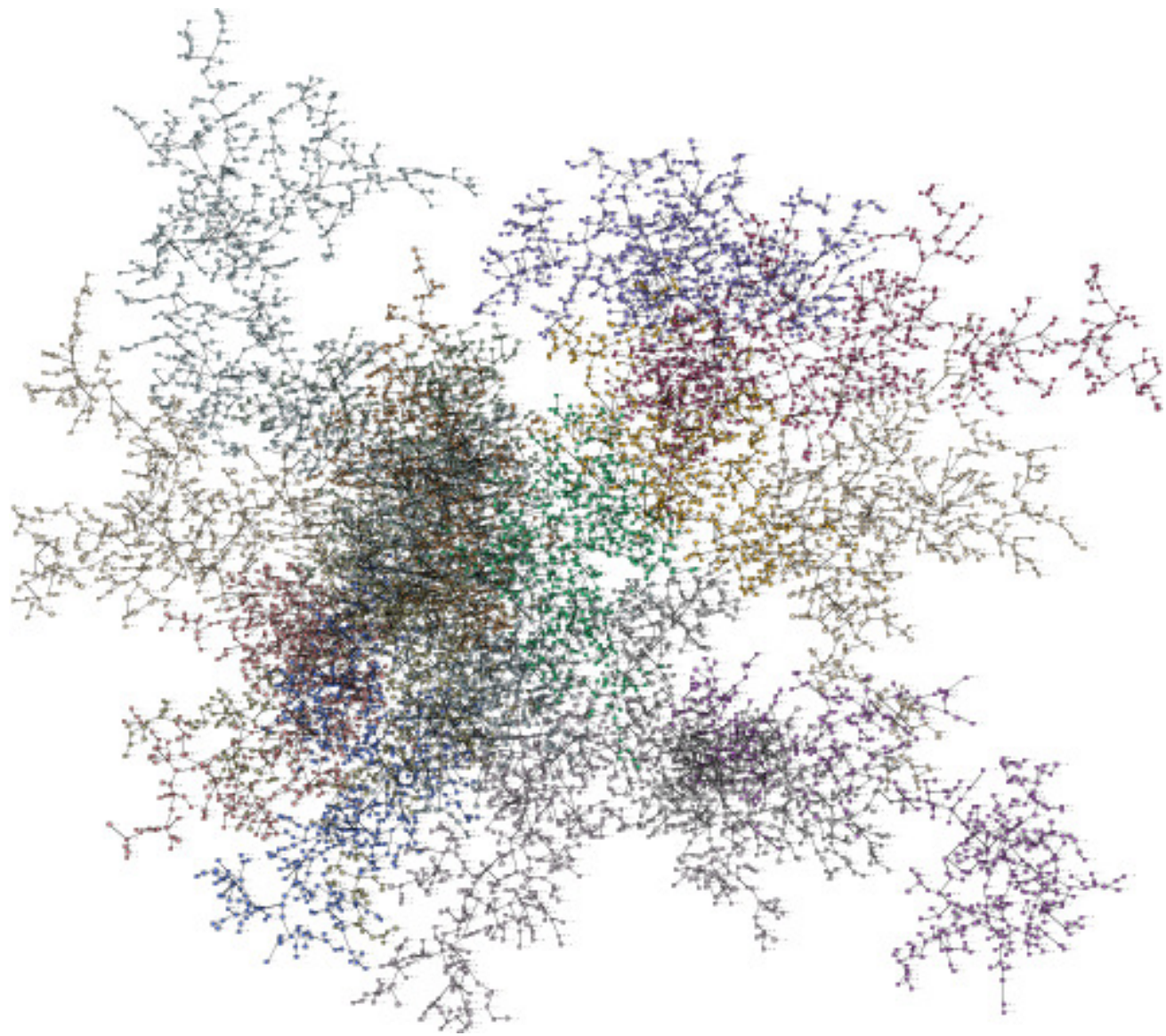


- Estructura de datos con relación entre los elementos de tipo RED, es decir, de “muchos a muchos”...
- Los datos se guardan en **NODOS** o **VÉRTICES** y los nodos se relacionan con **ARCOS**.
- Si los arcos tienen valor, el grafo es **PONDERADO**.
- Dos nodos unidos por un arco son **ADYACENTES**.
- Los **VECINOS** de un nodo son todos sus nodos adyacentes.
- Un **CAMINO** es una secuencia de nodos adyacentes.
- Un **CICLO** es un camino en el que el nodo inicial es igual al nodo final.
- Cuando los arcos tienen dirección, se tiene un grafo **DIRIGIDO** o **DIGRAFO**.

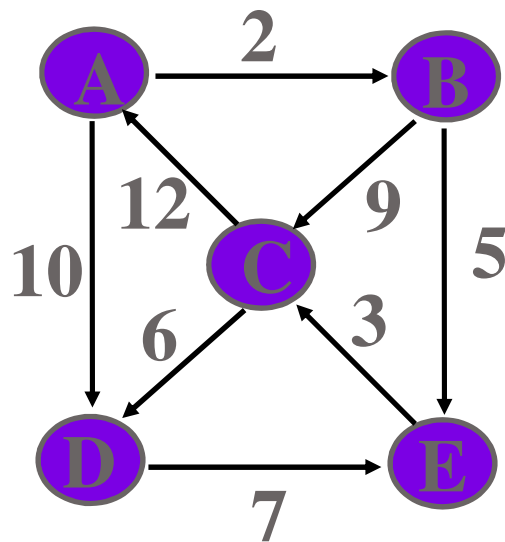
Representación física y Aplicaciones de un Grafo

- Formas más comunes de representación en memoria: *Matriz de Adyacencias, Lista de Adyacencias, Lista de Arcos.*
- Aplicaciones principales:
 - Conectividad, Redes de Transporte
 - ✓ ¿Existe un camino entre dos Nodos?
 - ✓ ¿Cuál es el costo mínimo de conexión para todos los Nodos?
 - ✓ *¿Cuál es la ruta óptima para ir de un Nodo a otro?*
 - Autómatas o Diagramas de Estado





El problema del camino más corto



Camino más corto de A a C:

- A-B-C = 11
- **A-B-E-C = 10**
- A-D-E-C = 20

- Es un problema de optimización...
- Se busca obtener, para un vértice origen, el costo mínimo para llegar a cada uno del resto de los nodos, sin importar por cuántos nodos se tenga que pasar...

Propuestas de solución



- **Algoritmo obvio:** Obtener todos los caminos posibles para llegar de un nodo a otro, y elegir al menor... Este proceso se repite para todos los nodos...
- El análisis de este algoritmo lleva a concluir que tendría un comportamiento de orden factorial...
- **Algoritmo con colas priorizadas:** Conceptualmente válido, pero complejo de implementar (requiere de HEAPS).

Algoritmo de Floyd



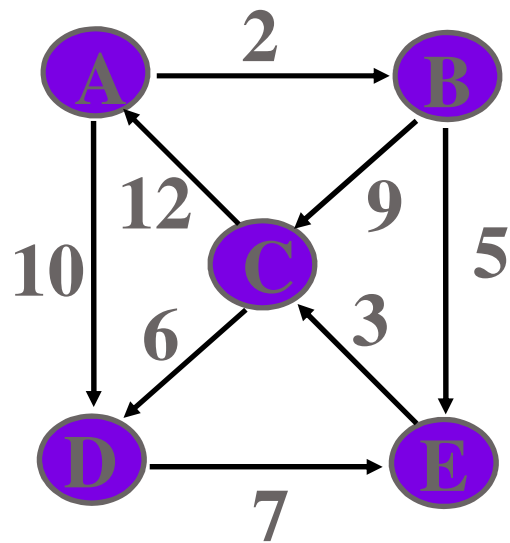
- Propuesta basada en la técnica de la programación dinámica...
- Se basa en la representación del grafo en una matriz de adyacencias bajo las siguientes condiciones:
 - $M[i,j]$ guarda el peso del arco del nodo i al nodo j .
 - $M[i,j]$ guarda el valor 0 cuando $i = j$
 - $M[i,j]$ guarda el valor ∞ cuando no hay arco entre el nodo i y el nodo j .

Algoritmo de Floyd



- Dada la matriz de adyacencias, obtener la matriz del camino más corto en donde el elemento $[i,j]$ es el valor del camino más corto para ir del nodo i al nodo j ...
- El algoritmo también se puede utilizar para mostrar el camino más corto (no sólo el valor)...
- Este algoritmo tiene un comportamiento de orden **$O(n^3)$** .

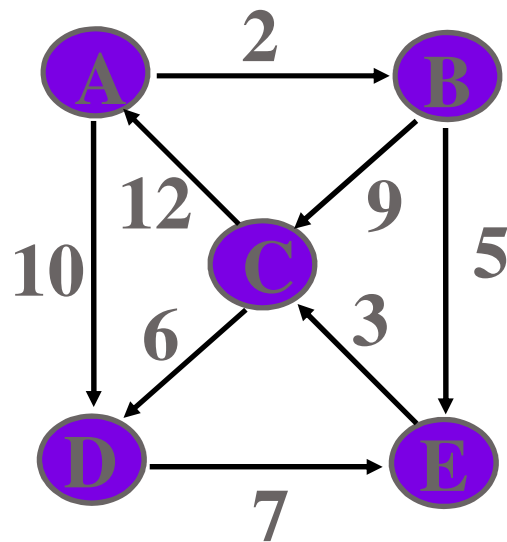
Ejemplo



MATRIZ DE ADYACENCIAS

	A	B	C	D	E
A	0	2	∞	10	∞
B	∞	0	9	∞	5
C	12	∞	0	6	∞
D	∞	∞	∞	0	7
E	∞	∞	3	∞	0

Ejemplo



MATRIZ DEL CAMINO MÁS CORTO

	A	B	C	D	E
A	0	2	10	10	7
B	20	0	8	14	5
C	12	14	0	6	13
D	22	24	10	0	7
E	15	17	3	9	0

Algoritmo de Floyd



- Sea D^k una matriz del camino más corto, en donde cada elemento $[i,j]$ indica cuál es el camino más corto desde el nodo i hasta el nodo j , pasando sólo por los nodos desde 1 hasta k ...
- D^0 es la matriz con los caminos más cortos sin pasar por otros vertices... por lo tanto, es la matriz de adyacencias original...
- D^n será la matriz con los caminos más cortos pasando por TODOS los nodos posibles... ***ESTA ES LA MATRIZ QUE SE BUSCA...***

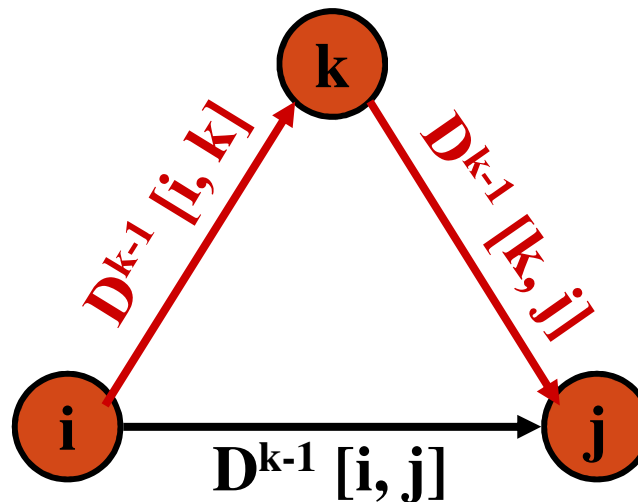
Algoritmo de Floyd



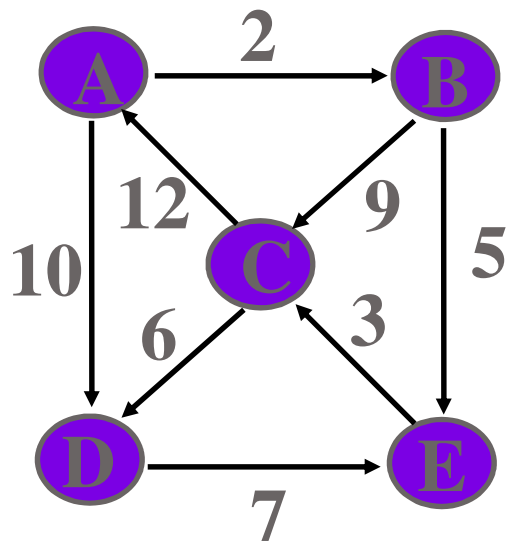
- ¿Cómo obtener a la matriz D^n dada la matriz D^0 ?
- El enfoque de la PROGRAMACIÓN DINÁMICA preguntaría si a partir de la matriz D^0 se puede obtener la matriz D^1 ... obteniendo a D^1 , si se puede obtener a D^2 , y así sucesivamente hasta obtener a D^n , que es lo que buscamos...
- Analicemos nuestro ejemplo...

Algoritmo de Floyd

- Se debe escoger el mínimo de 2 caminos:
 - El que **no** incluye al nodo k, o
 - El que **sí** incluye al nodo k.



Ejemplo



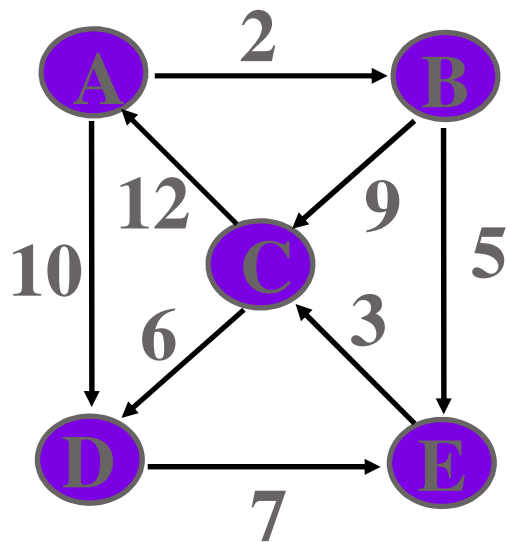
$$C-A + A-B = 12 + 2$$

Mínimo entre:
C-D y C-A + A-D

D^0	A 1	B 2	C 3	D 4	E 5
A 1	0	2	∞	10	∞
B 2	∞	0	9	∞	5
C 3	12	∞	0	6	∞
D 4	∞	∞	∞	0	7
E 5	∞	∞	3	∞	0

D^1	A 1	B 2	C 3	D 4	E 5
A 1	0	2	∞	10	∞
B 2	∞	0	9	∞	5
C 3	12	14	0	6	∞
D 4	∞	∞	∞	0	7
E 5	∞	∞	3	∞	0

Ejemplo



D¹	A 1	B 2	C 3	D 4	E 5
A 1	0	2	∞	10	∞
B 2	∞	0	9	∞	5
C 3	12	14	0	6	∞
D 4	∞	∞	∞	0	7
E 5	∞	∞	3	∞	0

D²	A 1	B 2	C 3	D 4	E 5
A 1	0	2	11	10	7
B 2	∞	0	9	∞	5
C 3	12	14	0	6	19
D 4	∞	∞	∞	0	7
E 5	∞	∞	3	∞	0

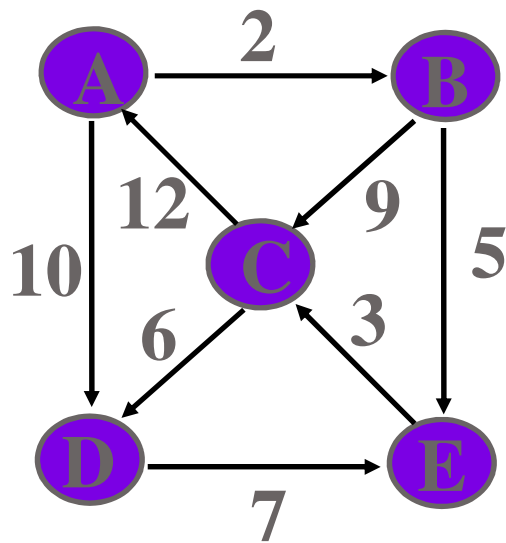
$$A-B + B-C = 2 + 9$$

$$A-B + B-E = 2 + 5$$

Mínimo entre: $C-D$ y $C-B + B-D$

$$C-B + B-E = 14 + 5$$

Ejemplo



D²	A 1	B 2	C 3	D 4	E 5
A 1	0	2	11	10	7
B 2	∞	0	9	∞	5
C 3	12	14	0	6	19
D 4	∞	∞	∞	0	7
E 5	∞	∞	3	∞	0

D³	A 1	B 2	C 3	D 4	E 5
A 1	0	2	11	10	7
B 2	21	0	9	15	5
C 3	12	14	0	6	19
D 4	∞	∞	∞	0	7
E 5	15	17	3	9	0

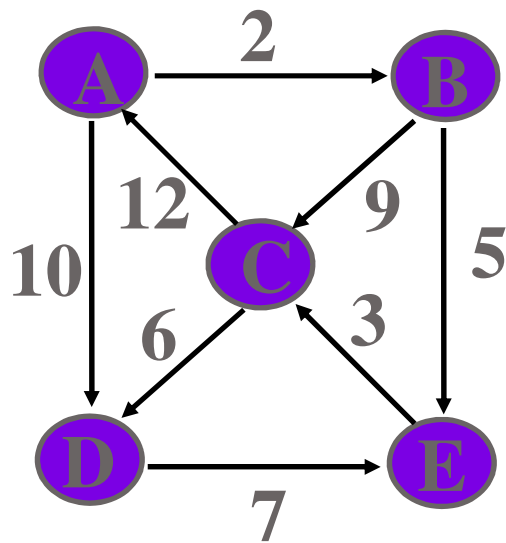
$$B-C + C-A = 9 + 12$$

Mínimo entre: A-D y A-C + C-D

Mínimo entre: A-E y A-C + C-E

$$E-C + C-A = 3 + 12$$

Ejemplo



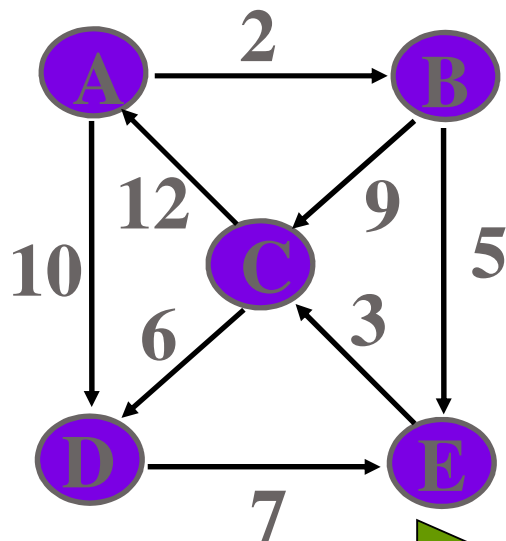
D³	A 1	B 2	C 3	D 4	E 5
A 1	0	2	11	10	7
B 2	21	0	9	15	5
C 3	12	14	0	6	19
D 4	∞	∞	∞	0	7
E 5	15	17	3	9	0

D⁴	A 1	B 2	C 3	D 4	E 5
A 1	0	2	11	10	7
B 2	21	0	9	15	5
C 3	12	14	0	6	13
D 4	∞	∞	∞	0	7
E 5	15	17	3	9	0

Mínimo entre: A-E y A-D + D-E

Mínimo entre: C-E y C-D + D-E

Ejemplo



MATRIZ META

Mínimo entre: A-C y A-E + E-C

Mínimo entre: B-A y B-E + E-A

$D-E + E-A = 7 + 15$

D⁴	A 1	B 2	C 3	D 4	E 5
A 1	0	2	11	10	7
B 2	21	0	9	15	5
C 3	12	14	0	6	13
D 4	∞	∞	∞	0	7
E 5	15	17	3	9	0

D⁵	A 1	B 2	C 3	D 4	E 5
A 1	0	2	10	10	7
B 2	20	0	8	14	5
C 3	12	14	0	6	13
D 4	22	24	10	0	7
E 5	15	17	3	9	0

Algoritmo de Floyd



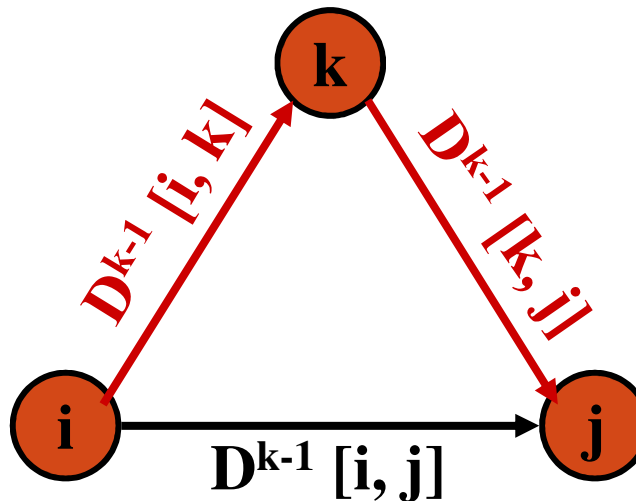
- Generalizando...

$$D^k[i,j] = \text{minimo}(D^{k-1}[i,j] , D^{k-1}[i,k] + D^{k-1}[k,j])$$

- Esta relación recursiva, es el planteamiento de solución al problema...
- La técnica de la programación dinámica, sugiere que se obtenga el caso más pequeño y a partir de ese, se construyan los casos superiores, almacenando resultados si se requieren...

Algoritmo de Floyd

- Se debe escoger el mínimo de 2 caminos:
 - El que **no** incluye al nodo k, o
 - El que **sí** incluye al nodo k.



$$D^k[i, j] = \min(D^{k-1}[i, j], D^{k-1}[i, k] + D^{k-1}[k, j])$$

Algoritmo de Floyd



- Por lo tanto, el algoritmo se puede plantear de la siguiente manera...

D = matriz de adyacencias

for k = 1 to n do

for i = 1 to n do

for j = 1 to n do

D[i,j] = minimo(D[i,j], D[i,k]+D[k,j]);

Algoritmo de Floyd

- ¿Porqué se puede usar una sólo matriz y NO se requieren 'n' matrices intermedias?

Los valores de la k-ésima columna y el k-ésimo renglón NO cambian en la k-ésima iteración...

$$D[i,k] = \text{minimo}(D[i,k], D[i,k]+D[k,k]) = D[i,k]$$

$$D[k,j] = \text{minimo}(D[k,j], D[k,j]+D[k,k]) = D[k,j]$$

Puesto que $D[i,j]$ en la k-ésima iteración se calcula con **su propio valor previo y los de la k-ésima columna y el k-ésimo renglón que se mantienen de la iteración anterior, NO HAY PROBLEMA en usar la propia matriz...**

Algoritmo de Floyd



- **¿Cómo encontrar los nodos que conforman el camino más corto?**
- Almacenar en una matriz auxiliar, el índice más grande del nodo intermedio por el que se pasa en el camino más corto del nodo i al nodo j .
- Utilizando esa matriz, si para ir del nodo i al nodo j se pasa por el nodo k , entonces se pasa también por el nodo con el índice más grande del camino más corto del nodo i al nodo k , y del nodo k al nodo j ...

Algoritmo de Floyd



- Para obtener la matriz con el último nodo visitado en el camino más corto...

D = matriz de adyacencias

P = matriz de último nodo inicializada en ceros.

for k = 1 to n do

for i = 1 to n do

for j = 1 to n do

if ($D[i,k] + D[k,j] < D[i,j]$) then

$P[i,j] = k;$

$D[i,j] = D[i,k] + D[k,j];$

Algoritmo de Floyd

- Para desplegar el camino más corto...

Modulo camino (inicio, fin)

if ($P[inicio, fin] \neq 0$) then

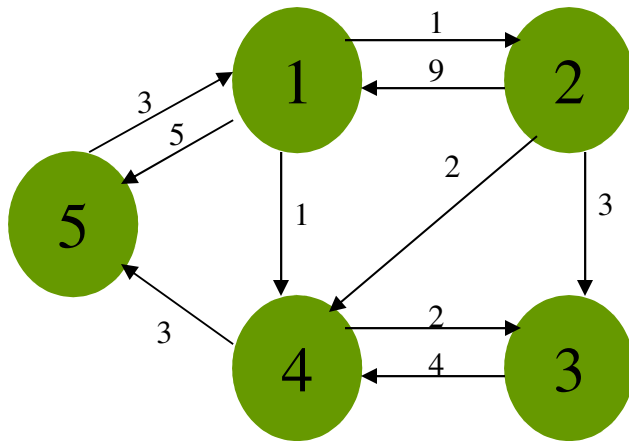
camino(inicio, $P[inicio, fin]$);

write($P[inicio, fin]$);

camino($P[inicio, fin]$, fin);

- *Recordar que **$P[inicio, fin]$** es el nodo intermedio con el índice más grande en el camino más corto de inicio a fin.*

Ejemplo



P	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

D	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

Modulo camino (inicio, fin)
if (P[inicio, fin] <> 0) then
 camino(inicio, P[inicio, fin]);
 write(P[inicio, fin]);
 camino(P[inicio, fin], fin);

El principio de optimalidad



- Todo problema de optimización, se puede resolver con la técnica de la **programación dinámica**, si la solución cumple con el ***principio de optimalidad...***
- **PRINCIPIO DE OPTIMALIDAD:** *La solución óptima de la instancia de un problema siempre contiene las soluciones óptimas de todas las subinstancias que conforman la solución...*
- Ejemplos: Camino más corto vs. más largo...

Un ejemplo más...

Multiplicación encadenada de matrices



- A diferencia del problema resuelto por el algoritmo de Strassen, en que se hace más eficiente la multiplicación de matrices MUY grandes, ahora se buscará efficientizar el proceso de multiplicar una secuencia de matrices de tamaño relativamente pequeño...
- ***¿Qué condición se debe cumplir para que dos matrices se puedan multiplicar?***
- La cantidad de columnas de la primer matriz debe de ser igual a la cantidad de renglones de la segunda matriz...

Multiplicación encadenada de matrices

- Sea la matriz M1 una matriz de orden $n \times m$ y la matriz M2 de orden $m \times p$...
- Si se multiplican ambas matrices, la matriz resultante será de orden $n \times p$...
- *¿Cuántas multiplicaciones escalares requiere el cálculo de un elemento de la matriz resultante?*
 m multiplicaciones...
- Por lo tanto, TODA la matriz resultante requiere de $m \times n \times p$ multiplicaciones...

Multiplicación encadenada de matrices



- **¿Cómo influye el orden en que se ejecuten las multiplicaciones de matrices en una secuencia encadenada de multiplicaciones $M_1 \times M_2 \times \dots \times M_n$?**
- Puesto que la cantidad de multiplicaciones escalares a realizar está determinado por el tamaño de la matrices a multiplicar, y...
- puesto que la multiplicación de matrices es ASOCIATIVA...
- Existe un orden ÓPTIMO para multiplicar las matrices y obtener los resultados más eficientemente...

EJEMPLO:

Multiplicación encadenada de matrices



- Se desea obtener la multiplicación de **A X B X C X D**, donde **A** es una matriz de 20 X 2, **B** una matriz de 2 X 30, **C** una matriz de 30 X 12 y **D** una matriz de 12 X 8...
- *¿De qué tamaño será la matriz resultante?*
20 X 8
- *¿Cuál será la forma más eficiente de multiplicar las matrices?*
La que requiera menos multiplicaciones escalares...

EJEMPLO:

Multiplicación encadenada de matrices

- Se desea obtener la multiplicación de **A X B X C X D**, donde **A** es una matriz de 20 X 2, **B** una matriz de 2 X 30, **C** una matriz de 30 X 12 y **D** una matriz de 12 X 8...

- *¿Cuántas multiplicaciones escalares habría en la secuencia normal de multiplicación?*

$$\mathbf{A} \times \mathbf{B} = 20 \times 2 \times 30 = 1200 + \dots$$

$$\text{resultado anterior} \times \mathbf{C} = 20 \times 30 \times 12 = 7200 +$$

$$\text{resultado anterior} \times \mathbf{D} = 20 \times 12 \times 8 = 1920$$

$$= \mathbf{10,320}$$

- ¿Habría una forma más OPTIMA?

SI !!! ...

EJEMPLO:

Multiplicación encadenada de matrices

- Se desea obtener la multiplicación de **A X B X C X D**, donde **A** es una matriz de 20 X 2, **B** una matriz de 2 X 30, **C** una matriz de 30 X 12 y **D** una matriz de 12 X 8...
- *¿Cuál es el orden de multiplicación que MINIMIZA las multiplicaciones escalares?*
 - $\mathbf{B} \times \mathbf{C} = 2 \times 30 \times 12 = 720 + \dots$
 - resultado anterior $\times \mathbf{D} = 2 \times 12 \times 8 = 192 +$
 - $\mathbf{A} \times \text{resultado anterior} = 20 \times 2 \times 8 = 320$
 - = 1,232**
- **Encontrar cómo se obtiene el orden más eficiente de multiplicación ES NUESTRO PROBLEMA...**

Multiplicación encadenada de matrices



- Propuestas de solución:
 - **FUERZA BRUTA:** Encontrar TODAS las posibles combinaciones de orden para las multiplicaciones, calcular la cantidad de multiplicaciones escalares, y seleccionar la mínima. ***Comportamiento de orden exponencial.***
 - **PROGRAMACIÓN DINÁMICA** si se aplica el principio de optimalidad...
 - ✓ **Algoritmo propuesto por Godbole (1973) con un comportamiento de orden cúbico.**

Diseño del Algoritmo para la Multiplicación encadenada de matrices

- El problema de tamaño n significa que se tienen n matrices a multiplicar: $\mathbf{M}_1 \times \mathbf{M}_2 \times \dots \times \mathbf{M}_n \dots$
- Cada matriz tiene dimensiones $d_{i-1} \times d_i \dots$
- Por lo tanto... M_1 es de tamaño $d_0 \times d_1 \dots M_2 : d_1 \times d_2 \dots M_3 : d_2 \times d_3 \dots$ y así hasta $M_n : d_{n-1} \times d_n \dots$
- Los datos d_i son la entrada para el algoritmo del problema, y NO los datos de las matrices...
- La cantidad de multiplicaciones escalares al multiplicar $M_k \times M_{k+1}$ es igual a $d_{k-1} \times d_k \times d_{k+1} \dots$

Diseño del Algoritmo para la Multiplicación encadenada de matrices

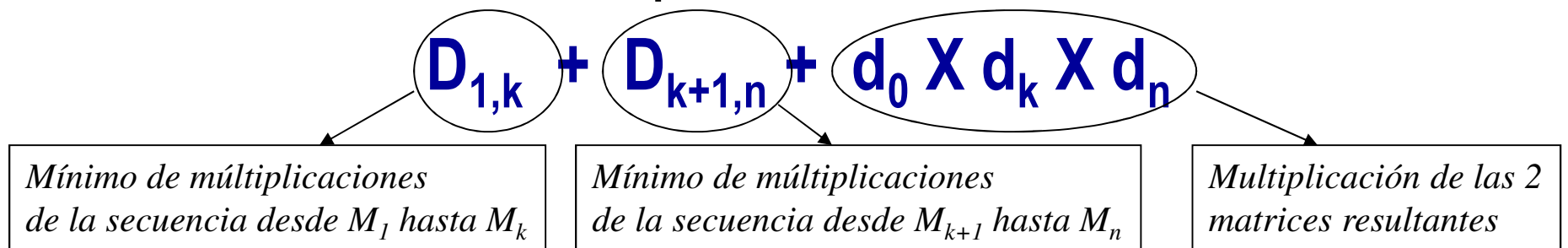
- Sea $D_{i,j}$ la cantidad mínima de multiplicaciones escalares al multiplicar la secuencia de matrices desde \mathbf{M}_i hasta \mathbf{M}_j ...
- *¿Cómo se puede obtener $D_{i,j}$?*
 - $D_{i,j} = 0$ cuando $i = j$...
 - $D_{i,j} = d_{i-1} \times d_i \times d_{i+1}$ cuando $i+1 = j$...
- El resto de los casos se tienen que generalizar "**pensando recursivamente**", y aplicando el **principio de optimalidad** que corresponde a las diferentes formas de agrupamiento de las matrices...

Diseño del Algoritmo para la Multiplicación encadenada de matrices

- Sea una posible agrupación:

$$(M_1 \times M_2 \times \dots \times M_k) \times (M_{k+1} \times M_{k+2} \times \dots \times M_n)$$

- La cantidad de multiplicaciones escalares sería:



- Puesto que se pueden hacer diversas agrupaciones...
- $D_{1,n}$ sería el mínimo de $D_{1,k} + D_{k+1,n} + d_0 \times d_k \times d_n$ para los valores de k desde 1 hasta $n-1$...

Diseño del Algoritmo para la Multiplicación encadenada de matrices

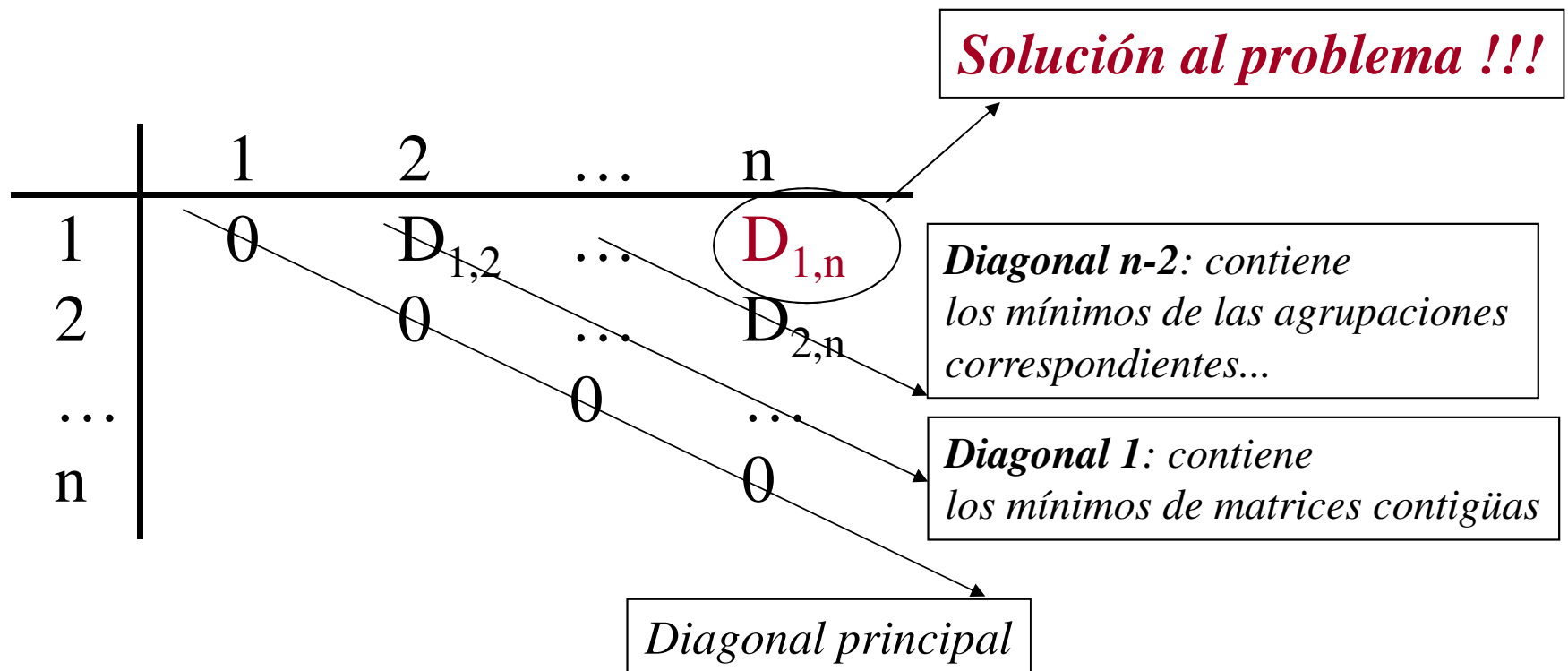
- Por lo tanto, generalizando...

$$D_{i,j} = \text{mínimo} (D_{i,k} + D_{k+1,j} + d_{i-1} \times d_k \times d_j) \\ \text{para } i \leq k \leq j-1$$

- El valor que resuelve el problema es $D_{1,n}$...
- Y según la técnica de la programación dinámica, éste se puede obtener a partir de los valores de base que serían las $D_{i,i}$...
- Dado que $D_{i,j}$ es un espacio matricial... algorítmicamente hablando se trabajará con una matriz D ...

Diseño del Algoritmo para la Multiplicación encadenada de matrices

$$D_{i,j} = \text{mínimo} (D_{i,k} + D_{k+1,j} + d_{i-1} \times d_k \times d_j) \\ \text{para } i \leq k \leq j-1$$



Algoritmo para la Multiplicación encadenada de matrices

for i = 1 to n do $D[i,i] = 0$;

for diag=1 to n-1 do

for i = 1 to n-diag do

j = i + diag;

*$D[i,j] = \text{minimo}(i, j, D, d)$; /*Esta función calcula
el valor mínimo entre los diversos valores de:*

*$D[i,k] + D[k+1, j] + d[i-1]*d[k]*d[j]$*

*para k desde i hasta j-1 */*

return $D[1,n]$;

$O(n^3)$

Algoritmo para la Multiplicación encadenada de matrices

- *¿Cómo obtener la agrupación más eficiente además del valor de las multiplicaciones escalares mínimas?*
- Utilizar una matriz auxiliar que guarde la última matriz utilizada en la agrupación más óptima (similar al caso del camino más corto)...
- Con esta matriz, una rutina recursiva puede desplegar la agrupación más eficiente...
- *Ver capítulo 3, sección 3.4 para los detalles...*

Un ejemplo más... ABB óptimo



Recordando...

- Un Arbol Binario de Búsqueda (ABB) es una estructura de datos útil para realizar la búsqueda de datos...
- Los nodos tienen 0, 1 ó 2 hijos...
- Los descendientes por la izquierda son menores, los de la derecha son mayores...
- La altura del árbol determina el peor caso en la búsqueda...

ABB óptimo

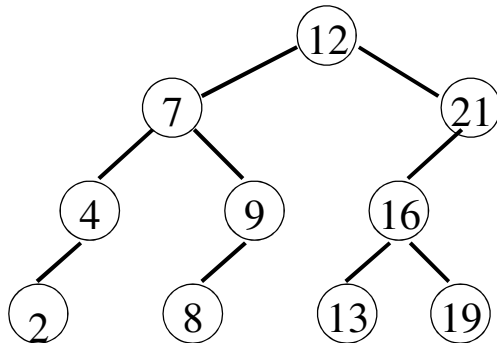


PROBLEMA:

- dada una secuencia ordenada de llaves a insertar en un ABB... d_1, d_2, \dots, d_n ($d_1 < d_2 < \dots < d_n$)
- de las cuales se conoce la probabilidad de que sean buscadas en el ABB... p_1, p_2, \dots, p_n
- ¿cuál es la forma del ABB que **minimiza** el tiempo promedio de búsqueda de las llaves?

¿Cómo se calcula el tiempo promedio de búsqueda en un ABB?

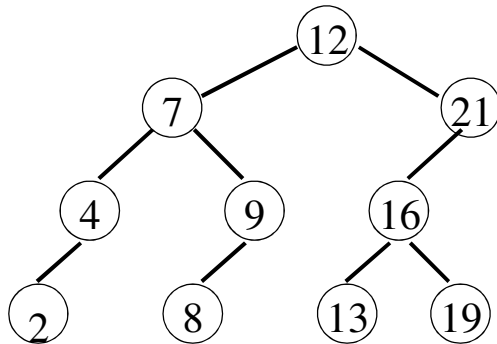
- Si todas las llaves tienen la misma probabilidad de ser buscadas...
- El tiempo promedio de búsqueda sería...
- La sumatoria de la cantidad de comparaciones que se requieren para encontrar cada llave...
- entre la cantidad de llaves...



$$\begin{aligned} &1 + \\ &2 + 2 + \\ &3 + 3 + 3 + \\ &4 + 4 + 4 + 4 = 30/10 = \mathbf{3.0} \end{aligned}$$

¿Cómo se calcula el tiempo promedio de búsqueda en un ABB?

- Si cada una de las llaves tiene una probabilidad de ser buscada...
- El tiempo promedio de búsqueda sería...
- La sumatoria de multiplicar la cantidad de comparaciones que se requieren para encontrar cada llave por su probabilidad de ser buscada...



$$\begin{aligned} &1(p_6) + \\ &2(p_3) + 2(p_{10}) + \\ &3(p_2) + 3(p_5) + 3(p_8) + \\ &4(p_1) + 4(p_4) + 4(p_7) + 4(p_9) \end{aligned}$$

ABB óptimo

¿Qué se busca optimizar?

- Para una secuencia de llaves, con sus probabilidades de ser buscadas, existen diversos ABB que las pueden guardar...
- *¿Cuál es el ABB que minimiza el tiempo promedio de búsqueda?*
- **Ejemplo:** Para un ABB con 3 llaves, ¿Cuáles son los posibles árboles a formar?

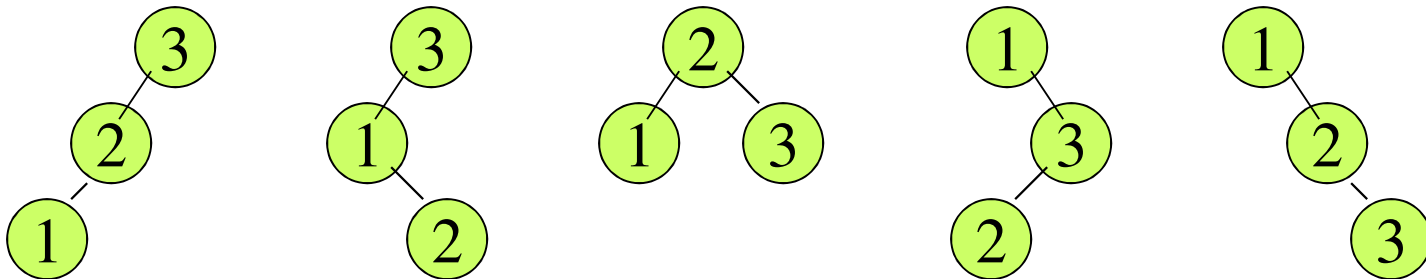
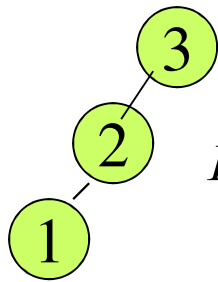


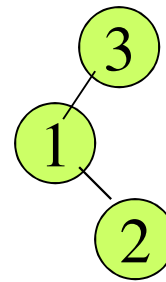
ABB óptimo

¿Qué se busca optimizar?

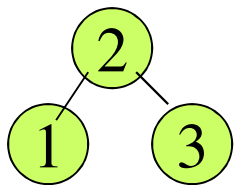
- Si las probabilidades de que se busque cada una de las llaves son: $p_1 = 0.7$, $p_2 = 0.2$ y $p_3 = 0.1$...
- *¿Cuál es el tiempo promedio de búsqueda en cada árbol?*



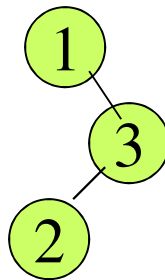
$$1(0.1) + 2(0.2) + 3(0.7) = 2.6$$



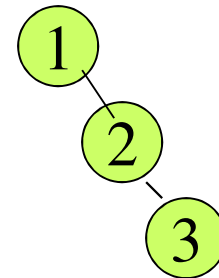
$$1(0.1) + 2(0.7) + 3(0.2) = 2.1$$



$$1(0.2) + 2(0.7) + 2(0.1) = 1.8$$



$$1(0.7) + 2(0.1) + 3(0.2) = 1.5$$



$$1(0.7) + 2(0.2) + 3(0.1) = \mathbf{1.4}$$

¿Cómo obtener el ABB óptimo?



- Calcular todas las posibles opciones de formas de ABB para una secuencia de llaves, y después encontrar la del tiempo promedio mínimo... tiene un comportamiento exponencial...
- *¿Se podrá utilizar la PROGRAMACIÓN DINÁMICA?*
- Es un problema de optimización...
- Y se aplica el principio de optimalidad, pues para el ABB del tiempo mínimo promedio, se tienen subárboles con el tiempo mínimo promedio...

ABB óptimo

Aplicando la programación dinámica

- Sea $A_{i,j}$ el tiempo mínimo promedio para la búsqueda de cualquier llave de la secuencia de d_i a d_j ...
- Se desea encontrar $A_{1,n}$ que es la solución al problema...
- $A_{i,i} = p_i$ ya que estamos hablando de ABB's de un sólo nodo, y por lo tanto se requieren en promedio $1 \times p_i$ comparaciones...
- Busquemos ahora la solución general...

ABB óptimo: Caso general

- Sea el ABB óptimo 'k' aquel cuya raíz es la llave d_k ...
- entonces, los subárboles del nodo raíz d_k , son ABB óptimos que tienen el tiempo mínimo promedio en sus búsquedas...
 - el subárbol izquierdo tiene a las llaves d_1 a d_{k-1} ...
 - y por lo tanto, el valor del tiempo mínimo promedio para el subárbol izquierdo es $A_{1,k-1}$...
 - el subárbol derecho tiene a las llaves d_{k+1} a d_n ...
 - y por lo tanto, el valor del tiempo mínimo promedio para el subárbol derecho es $A_{k+1,n}$...

ABB óptimo: Caso general

- *¿Cuál es el tiempo mínimo óptimo para el ABB óptimo 'k' cuya raíz es la llave d_k ?*
 - La raíz del árbol, requiere una comparación $X p_k$... más...
 - El tiempo mínimo promedio de los subárboles...
 - Pero dado que los subárboles están en un nivel más abajo que la raíz...
 - buscar a cualquier llave que no sea la raíz del árbol, involucra una comparación más...
 - y esto indica que debemos acumular la probabilidad correspondiente...

ABB óptimo: Caso general

Por lo tanto y generalizando...

- *¿Cuál es el tiempo mínimo óptimo para el ABB óptimo 'k' cuya raíz es la llave d_k ?*

$$A_{1,k-1} + p_1 + p_2 + \dots + p_{k-1} + p_k + A_{k+1,n} + p_{k+1} + \dots + p_n$$

- que equivale a: $A_{1,k-1} + A_{k+1,n} + \sum_{m=1}^n p_m$

- y para $A_{1,n} = \underset{1 \leq k \leq n}{\text{mínimo}} (A_{1,k-1} + A_{k+1,n}) + \sum_{m=1}^n p_m$

- donde $A_{i,i-1}$ y $A_{j+1,j}$ valen 0

Algoritmo para obtener el ABB óptimo

- Propuesto por Gilbert and Moore (1959).
- **ENTRADAS:**
 - cantidad de llaves n
 - probabilidades de que sean buscadas cada una de las llaves p_i
- **SALIDAS:**
 - Tiempo mínimo promedio para la búsqueda de llaves (en el ABB óptimo).
 - Matriz **R** ($1..n+1 \times 0..n$) donde **R[i,j]** contiene el índice de la llave que es la raíz en el ABB óptimo que contiene a las llaves desde i hasta j . Esta matriz servirá para construir el ABB óptimo si se requiere.

Algoritmo para obtener el ABB óptimo

for i = 1 to n do // inicialización de matrices de resultados

{ A[i,i-1] = 0; A[i,i] = p[i];

R[i,i] = i; R[i,i-1] = 0; }

A[n+1,n] = 0; R[n+1,n] = 0;

for diag=1 to n-1 do

for i = 1 to n-diag do

{ j = i + diag;

A[i,j] = minimo(i, j, A) + sumatoria(i, j, p); }

*/*La función mínimo calcula el valor mínimo entre los diversos valores de:*

A[i,k-1] + A[k+1, j] para k desde i hasta j . La función sumatoria calcula la suma de las probabilidades de la llave i hasta la llave j ./ }*

return A[1,n];

$O(n^3)$

Construcción del ABB óptimo

- Dada la matriz R...

función ABB (i,j) : apuntador;

k = R[i,j];

if (k = 0) return NULL;

else

{ q = new nodo(llave[k]);

q->izq = ABB(i,k-1);

q->der = ABB(k+1,j);

return q; }



El problema del viajero

Problema...



- Imagina que tú y otra persona, han pedido trabajo en cierta empresa...
- Quién los va a contratar, les dice que el contrato será para quien le indique cuál es la ruta más óptima para visitar 20 ciudades distintas del país, pasando sólomente una vez por cada una de ellas, en el menor tiempo posible (y al menor costo)...
- Las 20 ciudades, están concetadas por vías de transporte con todas las restantes...
- Este es el **problema del viajero**... ¿cómo solucionarlo?

Problema...



- La otra persona, tiene experiencia programando, y ha decidido poner a su computadora a trabajar para encontrar esta ruta óptima...
- Implementa un programa que encontrará todos los posibles caminos desde la ciudad inicial, pasando por todas las ciudades, hasta llegar de nuevo a la ciudad inicial y calculará para cada uno de ellos sus costos... de ahí obtendrá el menor, para dar su respuesta...
- ¿Cuánto se tardará la computadora en darle el resultado?...

Problema...



- ¿Cuántos caminos posibles hay en las 20 ciudades?
- **19!** = 121,645,100,408,832,000
- Si el cálculo de la longitud de cada camino se tarda 1 microsegundo en la computadora....
- El resultado tardaría **3,857 años!!!**
- Pero tú, que conoces algunas técnicas de diseño de algoritmos, al observar que es un problema de optimización, decides explorar la posibilidad de utilizar a la **PROGRAMACIÓN DINÁMICA** para resolverlo...

Propuesta de solución...



- El problema se puede modelar con un grafo ponderado en el que cada vértice será una ciudad, y los arcos tendrán los costos que implica viajar de una ciudad a otra...
- El problema generará como resultado la ruta más óptima que incluye a todos los vértices del grafo (Ciclo Hamiltoniano)...
- Puesto que en un ciclo, lo importante es empezar y terminar en el mismo vértice, para el análisis de este problema, es irrelevante el vértice inicial, por lo que se tomará el primer vértice como base (v_1)...

Propuesta de solución...



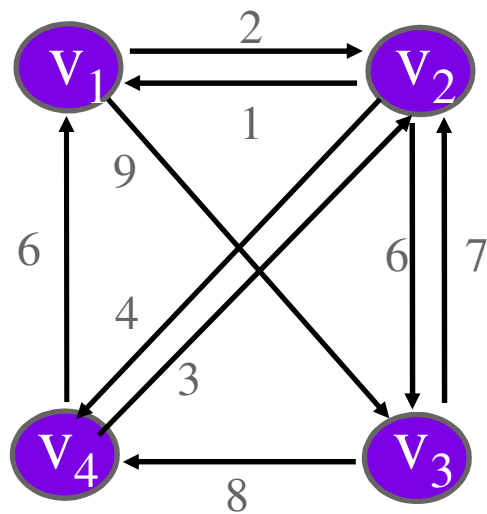
- Sea v_k el siguiente vértice a visitar después de v_1 en la ruta más óptima...
- el camino de v_k a v_1 será el más corto que pase una vez por todos los vértices restantes... y por lo tanto el más óptimo...
- Esto significa, que para resolver el ciclo completo, se tiene un subproblema, que cumplirá la condición de ser el más óptimo para poder solucionar el caso general...
- Por lo tanto, **SI** se puede aplicar la **PROGRAMACIÓN DINÁMICA....**

Propuesta de solución...



- Sea W la matriz de adyacencias del grafo correspondiente...
- Sea V el conjunto de todos los vértices del grafo...
- Sea A un subconjunto de V ...
- Entonces $D[v_i, A]$ es la longitud del camino más corto de v_i hasta v_1 pasando una vez por todos los vértices de A ...

Ejemplo...



- $V = \{v_1, v_2, v_3, v_4\}$
 - Si $A = \{v_3\} \dots$
 - $D[v_2, A] = \text{longitud}[v_2, v_3, v_1]$
 $= \infty$
 - Si $A = \{v_3, v_4\} \dots$
-
- $D[v_2, A] = \text{mínimo}(\text{longitud}[v_2, v_3, v_4, v_1], \text{longitud}[v_2, v_4, v_3, v_1])$
 $= \text{mínimo}(20, \infty) = 20$

Continuando con el análisis...

- Para un grafo de n vértices...
- $D[v_i, \emptyset]$ es la longitud del camino de v_i a v_1 sin pasar por algún vértice...
- Por lo tanto: $D[v_i, \emptyset] = W[i, 1]$
- Por otro lado, el resultado que se desea obtener es: $\text{mínimo}(W[1, j] + D[v_j, V - \{v_1, v_j\}])$ para j desde 2 hasta n ...

*Costo mínimo para
ir del nodo 1 al j*

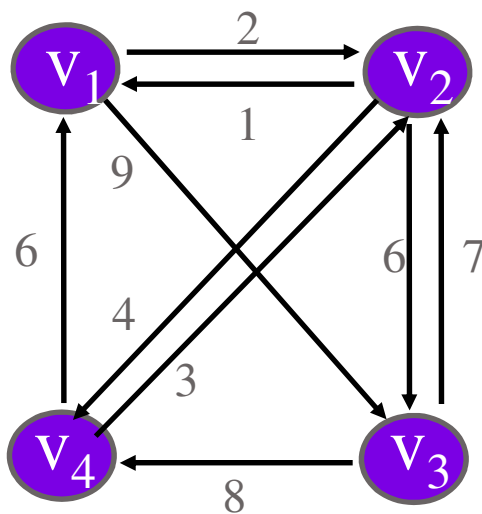
*Costo mínimo para ir del
nodo j al nodo 1 pasando por todos
los nodos menor el 1 y el j*

Continuando con el análisis...



- Y generalizando...
- Para todos los vértices excepto el 1, y cuando el vértice_i no se encuentre en A...
- $D[v_i, A] = \text{mínimo}(W[i, j] + D[v_j, A - \{v_j\}])$
para v_j que se encuentre en A...
- Esta es la clave del algoritmo...

En el ejemplo...

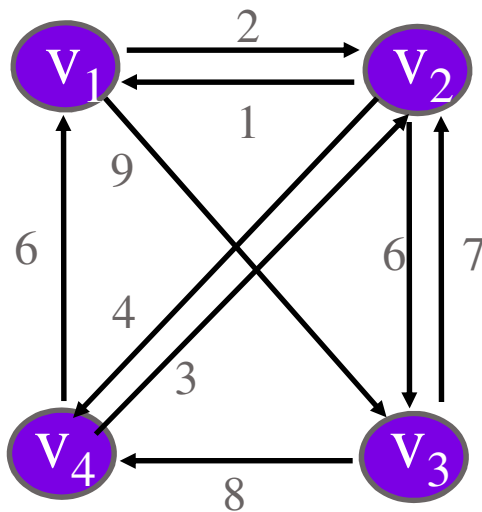


- $D[v_2, \emptyset] = 1$, $D[v_3, \emptyset] = \infty$,
 $D[v_4, \emptyset] = 6$
- Para A de un elemento:
 - $D[v_3, \{v_2\}] = \min(W[3,j] + D[v_j, \{v_2\} - \{v_j\}])$
 para v_j en $\{v_2\}$
 - $W[3,2] + D[v_2, \emptyset] = 7 + 1 = 8$

- $D[v_4, \{v_2\}] = 3 + 1 = 4$
- $D[v_2, \{v_3\}] = 6 + \infty = \infty$
- $D[v_4, \{v_3\}] = \infty + \infty = \infty$

- $D[v_2, \{v_4\}] = 4 + 6 = 10$
- $D[v_3, \{v_4\}] = 8 + 6 = 14$

En el ejemplo...

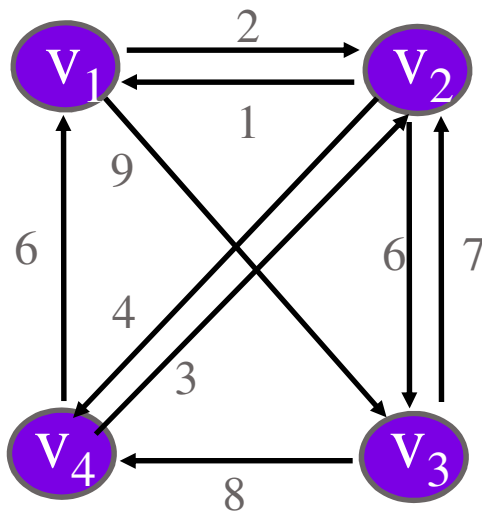


- Para A de dos elementos:

$$\begin{aligned} D[v_4, \{v_2, v_3\}] &= \text{minimo}(W[4, j] + D[v_j, \{v_2, v_3\} - \{v_j\}]) \\ &\quad \text{para } v_j \text{ en } \{v_2, v_3\} \\ &= \text{mínimo} (W[4, 2] + D[v_2, \{v_3\}] , W[4, 3] + D[v_3, \{v_2\}]) \\ &= \text{minimo}(3 + \infty, \infty + 8) = \infty \end{aligned}$$

- $D[v_3, \{v_2, v_4\}] = \text{minimo}(7 + 10, 8 + 4) = 12$
- $D[v_2, \{v_3, v_4\}] = \text{minimo}(6 + 14, 4 + \infty) = 20$

En el ejemplo...



- Finalmente, para A de tres elementos:

$$D[v_1, \{v_2, v_3, v_4\}] = \text{minimo}(W[1, j] + D[v_j, \{v_2, v_3, v_4\} - \{v_j\}]) \text{ para } v_j \text{ en } \{v_2, v_3, v_4\}$$

$$= \text{mínimo} (W[1, 2] + D[v_2, \{v_3, v_4\}] ,$$

$$W[1, 3] + D[v_3, \{v_2, v_4\}] ,$$

$$W[1, 4] + D[v_4, \{v_2, v_3\}])$$

$$= \text{minimo}(2 + 20 , 9 + 12, \infty + \infty) = 21$$

CAMINO ÓPTIMO: v_1, v_3, v_4, v_2, v_1

Algoritmo del problema del viajero...

for $i=2$ to n do $D[i, \emptyset] = W[i, 1];$

for $k=1$ to $n-2$ do

*for (todos los subconjuntos A que contengan k
vértices desde v_2 hasta v_n)*

for (todas las $i \neq 1$ y en que v_i no esté en A)

$D[i, A] = \text{minimo}(W[i, j] + D[j, A - \{v_j\}]);$

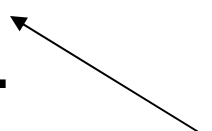
$D[1, V - \{v_1\}] = \text{minimo}(W[1, j] + D[j, V - \{v_1, v_j\}]);$

return $D[1, V - \{v_1\}];$

Regresando al inicio...



- Complejidad de tiempo del algoritmo con programación dinámica: $T(n) = (n-1)(n-2)2^{n-3}$
- Que corresponde a un orden: **$O(n^2 2^n)$**
- Para el problema de las 20 ciudades...
- $19 \times 18 \times 2^{17} = 44,826,624 \dots$
- Que la misma computadora tardaría a 1 microsegundo por cálculo...
- **45 segundos!!!**



*Es un problema
al que no se
le ha encontrado
una solución
más óptima*