# Semillero de Programación

## Algoritmo de Floyd-Warshall

Ana Echavarría Juan Francisco Cardona

Universidad EAFIT

12 de abril de 2013

# Contenido

# Contenido

# Problema 1 - Cut Ribbon

Hay que hallar el mínimo número de cortes que hay que hacerle a una cinta de longitud $n$ para que queden únicamente pedazos de tamaño $a$, $b$ o $c$.
Este problema es un caso del problema coin change que consiste en hallar el mínimo número de monedas/billetes que hay que usar para tener una cantidad $v$ de dinero.

# Problema 1 - Cut Ribbon

Sea $f(n)$ el mínimo número de cortes que hay que hacerle a una cinta de tamaño $n$ para que queden solo cintas de tamaño $a$, $b$ o $c$.

$f(0) = 0$

$$f(i) = \min \left\{ \begin{array}{ll} \infty & \\ 1 + f(i-a) & \text{si } i \geq a \\ 1 + f(i-b) & \text{si } i \geq b \\ 1 + f(i-c) & \text{si } i \geq c \end{array} \right\} \text{ para } 1 \leq i \leq n$$

# Implementación I

```
1  const int MAXN = 4005;
2  const int INF = 1 << 30;
3  int dp[MAXN];
4
5  int main(){
6      int n, a, b, c;
7      while (cin >> n >> a >> b >> c){
8          dp[0] = 0;
9          for (int i = 1; i <= n; ++i){
10             dp[i] = INF;
11             if (i >= a) dp[i] = max(dp[i], 1 + dp[i-a]);
12             if (i >= b) dp[i] = max(dp[i], 1 + dp[i-b]);
13             if (i >= c) dp[i] = max(dp[i], 1 + dp[i-c]);
14         }
15         cout << dp[n] << endl;
16     }
17     return 0;
18 }
```

# Problema 2 - Scuba diver

Este problema es parecido al problema de la mochila pero con las siguientes dos modificaciones:

- El problema es de minimización por lo que no se restringe a una capacidad de máximo $j$ sino una capacidad de mínimo $j$.
- Se tienen dos "mochilas" para llenar y se deben cumplir las restricciones de ambas.

# Problema 2 - Scuba diver

Sea $dp(i, j, k)$ el mínimo peso que se puede llevar si se usan los tanques $1 \ldots i$ y se quieren llevar mínimo $j$ litros de oxígeno y $k$ litros de nitrógeno.

$dp(0, j, k) = \infty \quad$ para $1 \leq j \leq minOxy$ y $1 \leq k \leq minNitro$
$dp(0, 0, 0) = 0$
$dp(i, j, k) =$

$$\min \left\{ \begin{array}{l} dp(i-1, j, k) \\ dp(i-1, j - oxy[i], k - nitro[i]) \end{array} \right\} \text{ para } \begin{array}{l} 1 \leq i \leq n \\ 0 \leq j \leq minOxy \\ 0 \leq k \leq minNitro \end{array}$$

Nota: Si $j - oxy[i] < 0$ o $k - nitro[i] < 0$ se tomará el valor de 0 ya que se está cumpliendo con la restricción de la mínima cantidad de oxígeno / nitrógeno.

# Implementación I

```
1  const int MAXN = 1005;
2  const int MAXOXY = 25, MAXNITRO = 85;
3  const int INF = 1 << 30;
4  int oxy[MAXN], nitro[MAXN], weight[MAXN];
5  int dp[MAXN][MAXOXY][MAXNITRO];
6
7  int main(){
8     int cases; cin >> cases;
9     while (cases--){
10       int min_oxy, min_nitro;
11       cin >> min_oxy >> min_nitro;
12       int n; cin >> n;
13       for (int i = 1; i <= n; ++i)
14          cin >> oxy[i] >> nitro[i] >> weight[i];
15
16       for (int j = 0; j <= min_oxy; ++j){
17          for (int k = 0; k <= min_nitro; ++k){
18             dp[0][j][k] = INF;
```

# Implementación II

```
19              }
20          }
21          dp[0][0][0] = 0;
22
23          for (int i = 1; i <= n; ++i){
24              for (int j = 0; j <= min_oxy; ++j){
25                  for (int k = 0; k <= min_nitro; ++k){
26                      int dont_take = dp[i-1][j][k];
27                      int take = weight[i] +
            dp[i-1][max(0, j-oxy[i])][max(0, k-nitro[i])];
28                      dp[i][j][k] = min(dont_take, take);
29                  }
30              }
31          }
32          cout << dp[n][min_oxy][min_nitro] << endl;
33      }
34      return 0;
35  }
```

# Problema 3 - Knapsack

Implementar directamente el algoritmo para el problema de la mochila mostrado en la sesión anterior

# Problema 4 - History Grading

Hay que hallar la longitud de la subsecuencia común más larga entre la secuencia base y las demás secuencias.
Hay que tener cuidado ya que la entrada `s` no es la secuencia en sí sino que `s[i]` es la posición del número `i` en la secuencia.

# Implementación I

```
1  const int MAXN = 25;
2  int order[MAXN];    // El orden de las secuencias
3  int pattern[MAXN];  // La secuencia base
4  int query[MAXN];    // La secuencia por la que se pregunta
5  int dp[MAXN][MAXN]; // La dp para LCS
6
7  int main(){
8     int n; cin >> n;
9     for (int i = 1; i <= n; ++i){
10       cin >> order[i];
11       pattern[order[i]] = i;
12    }
13
14    while (cin >> order[1]){
15       query[order[1]] = 1;
16       for (int i = 2; i <= n; ++i){
17          cin >> order[i];
18          query[order[i]] = i;
```

# Implementación II

```
19          }
20
21          for (int j = 0; j <= n; ++j) dp[0][j] = 0;
22          for (int i = 0; i <= n; ++i) dp[i][0] = 0;
23
24          for (int i = 1; i <= n; ++i){
25              for (int j = 1; j <= n; ++j){
26                  dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
27                  if (pattern[i] == query[j]){
28                      dp[i][j] = max(dp[i][j], 1 + dp[i-1][j-1]);
29                  }
30              }
31          }
32          cout << dp[n][n] << endl;
33      }
34       return 0;
35  }
```

# Contenido

# All-Pair Shortest Paths Problem (APSP)

## Entrada

Un grafo con pesos $G = (V, E)$

## Objetivo

Hallar la distancia más corta entre todos los pares de nodos o decir que hay un ciclo de peso negativo en el grafo.

# All-Pair Shortest Paths Problem (APSP)

**Preguntas**

Con los elementos que se han enseñado en el semillero ¿se puede hallar el camino más corto entre todas las parejas de nodos?

# All-Pair Shortest Paths Problem (APSP)

## Preguntas

Con los elementos que se han enseñado en el semillero ¿se puede hallar el camino más corto entre todas las parejas de nodos?

## Respuesta

Sí, se pueden utilizar los algoritmo de Dijkstra / Bellman-Ford que halla la distancia más corta entre un nodo y todos los demás. Estos algoritmos se corren desde cada uno de los nodos y con eso se tiene la distancia más corta cualquier nodo y todos los demás.

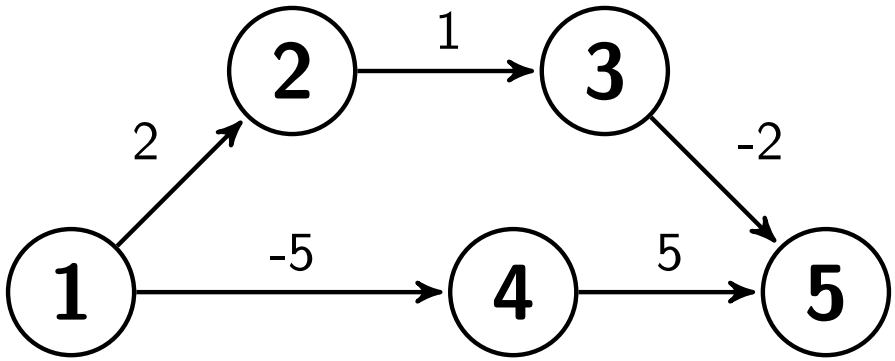
# Contenido

# Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall es un algoritmo que usa programación dinámica para hallar la distancia más corta entre todos los pares de nodos de un grafo con pesos.
Para propósitos del problema consideremos que el grafo tiene $n$ nodos numerados desde 1 hasta $n$.

# Algoritmo de Floyd-Warshall

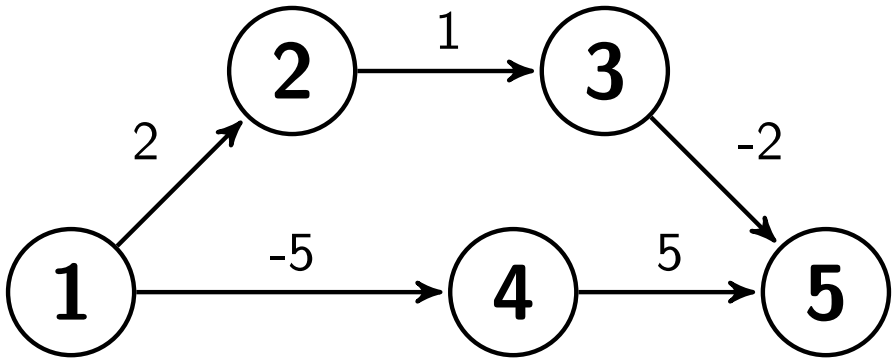
Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.




- ¿Cuál sería el valor de $f(1, 2, 0)$ ?

# Algoritmo de Floyd-Warshall

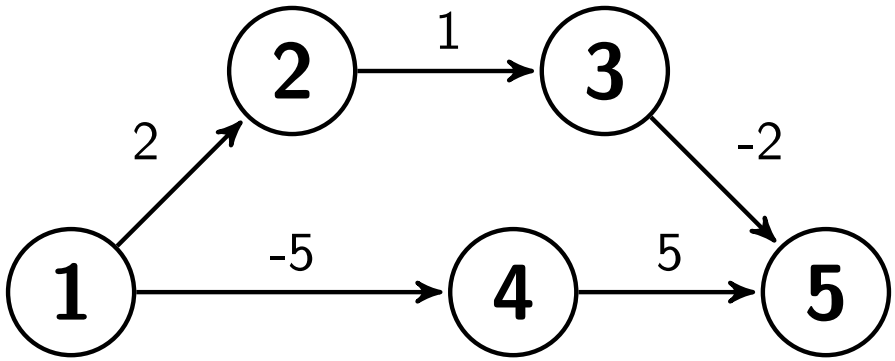
Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.




- ¿Cuál sería el valor de $f(1, 2, 0)$ ? 2
- ¿Cuál sería el valor de $f(1, 5, 2)$ ?

# Algoritmo de Floyd-Warshall

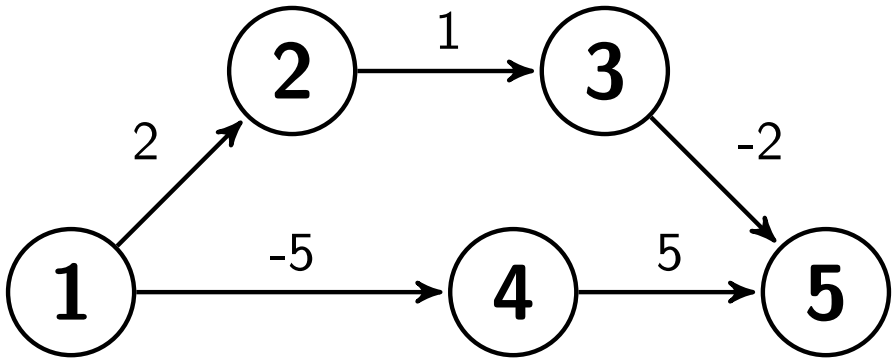
Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.




- ¿Cuál sería el valor de $f(1, 2, 0)$ ? 2
- ¿Cuál sería el valor de $f(1, 5, 2)$ ? $+\infty$
- ¿Cuál sería el valor de $f(1, 5, 3)$ ?

# Algoritmo de Floyd-Warshall

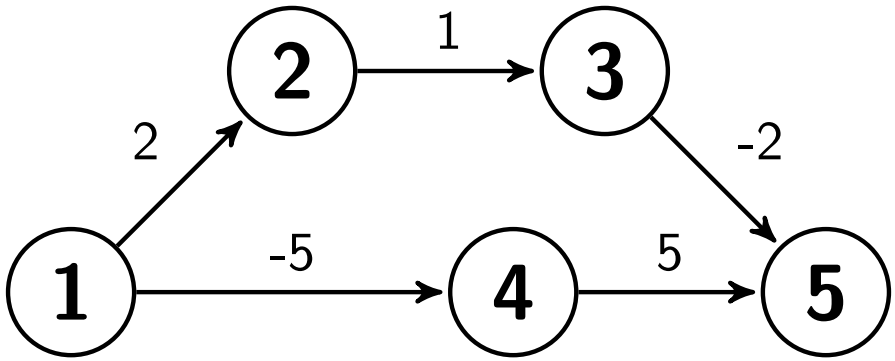
Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.




- ¿Cuál sería el valor de $f(1, 2, 0)$ ? 2
- ¿Cuál sería el valor de $f(1, 5, 2)$ ? $+\infty$
- ¿Cuál sería el valor de $f(1, 5, 3)$ ? 1
- ¿Cuál sería el valor de $f(1, 5, 4)$ ?

# Algoritmo de Floyd-Warshall

Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.




- ¿Cuál sería el valor de $f(1, 2, 0)$ ? $\quad 2$
- ¿Cuál sería el valor de $f(1, 5, 2)$ ? $\quad +\infty$
- ¿Cuál sería el valor de $f(1, 5, 3)$ ? $\quad 1$
- ¿Cuál sería el valor de $f(1, 5, 4)$ ? $\quad 0$

# Algoritmo de Floyd-Warshall

Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.

$$d(i,j,0) = \left\{ \begin{array}{ll} 0 & \text{si } i = j \\ C_{i,j} & \text{si } (i,j) \in E \\ +\infty & \text{en otro caso} \end{array} \right\} \text{ para } \begin{array}{l} 1 \leq i \leq n \\ 1 \leq j \leq n \end{array}$$

Donde $C_{i,j}$ es el costo que tiene asociado la arista $(i, j)$

# Algoritmo de Floyd-Warshall

Sea $d(i, j, k)$ la distancia más corta entre el nodo $i$ el nodo $j$ si como nodos intermedios solo se pueden usar los nodos $1 \dots k$.

$$d(i,j,k) = \min \left\{ \begin{array}{ll} d(i,j,k-1) & (\text{no usar nodo } k) \\ d(i,k,k-1) + d(k,j,k-1) & (\text{usar } k) \end{array} \right\}$$

Para $1 \leq i, j, k \leq n$

# Implementación

```
1  for (int i = 1; i <= n; ++i) {
2     for (int j = 1; j <= n; ++j){
3        d[i][j][0] = INF;
4     }
5     d[i][i][0] = 0;
6  }
7
8  for (int edge = 0; edge < m; ++edge){
9     int u, v, c; cin >> u >> v >> c;
10    d[u][v][0] = min(d[u][v], c); //Por si hay un loop de peso > 0
11 }
12
13 for (int k = 1; k <= n; ++k){
14    for (int i = 1; i <= n; ++i){
15       for (int j = 1; j <= n; ++j){
16          d[i][j][k] = min(d[i][j][k-1],
17                           d[i][k][k-1] + d[k][j][k-1]);
18       }
19    }
20 }
```

# Complejidad

## Preguntas

- ¿Cuál es la complejidad **en tiempo** del algoritmo de Floyd-Warshall?
- ¿Cuál es la complejidad **en memoria** del algoritmo de Floyd-Warshall?

# Complejidad

## Preguntas

- ¿Cuál es la complejidad **en tiempo** del algoritmo de Floyd-Warshall?
- ¿Cuál es la complejidad **en memoria** del algoritmo de Floyd-Warshall?

## Respuestas

- $O(n^3)$
- $O(n^3)$

# Optimización del espacio

El espacio del algoritmo de Floyd-Warshall se puede reducir a $O(n^2)$ (no almacenar la tercera dimensión) si se tiene en cuenta lo siguiente:

- si `d[i][j][k] = d[i][j][k-1]` no importa que no se guarden los dos en el mismo lugar ya que son el mismo valor
- para `d[i][j][k]` solo se usan los valores de `d[i][k][k-1]` y `d[k][j][k-1]`.
  Se puede ver que `d[i][k][k-1]` = `d[i][k][k]` y que `d[k][j][k-1]` = `d[k][j][k]` por lo que ninguno de los valores se sobreescribe y el algoritmo sigue siendo correcto.

# Implementación con optimización del espacio

```
1  for (int i = 1; i <= n; ++i) {
2     for (int j = 1; j <= n; ++j){
3        d[i][j] = INF;
4     }
5     d[i][i] = 0;
6  }
7
8  for (int edge = 0; edge < m; ++edge){
9     int u, v, c; cin >> u >> v >> c;
10    d[u][v] = min(d[u][v], c); // Por si hay un loop de peso > 0
11 }
12
13 for (int k = 1; k <= n; ++k){
14    for (int i = 1; i <= n; ++i){
15       for (int j = 1; j <= n; ++j){
16          d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
17       }
18    }
19 }
```

# Detectando un ciclo de peso negativo

El grafo tiene un **ciclo de peso negativo**, si y solo si luego de la ejecución del algoritmo de Floyd-Warshall **algún elemento de la diagonal es negativo**.

# Demostración

$\rightarrow$

Supongamos que hay un ciclo de peso negativo que empieza en el nodo $i$. Sea $j$ el nodo más grande que pertenece a ese ciclo. En la ejecución del algoritmo cuando $k = j$

`d[i][i] = min(d[i][i], d[i][j] + d[j][i])`

Pero `d[i][j] + d[j][i]` es el costo del ciclo ($< 0$) ya que el mayor nodo del ciclo es el nodo $k$ luego `d[i][i]` $< 0$.

$\leftarrow$

Supongamos que `d[i][i]` $< 0$ para algún $i$ luego de la ejecución del algoritmo. Claramente hay un ciclo de peso negativo que contiene el nodo $i$.

# Contenido

# Clausura transitiva

## Entrada

Un grafo cualquiera $G = (V, E)$

## Objetivo

Para cada pareja de nodos $(i, j)$ , hallar si existe un camino desde $i$ hasta $j$.

# Clausura transitiva

## Idea

Hay un camino desde el nodo $i$ hasta el nodo $j$ si y solo si pasa al menos una de las siguientes:

- El nodo $i$ es igual al nodo $j$ (caso base).
- Hay una arista del nodo $i$ al nodo $j$ (caso base).
- Existe un nodo $k$ tal que haya un camino del nodo $i$ al nodo $k$ y del nodo $k$ al nodo $j$ (caso recursivo).

Pensar en si hay un camino de $i$ hasta $j$ que como aristas intermedias las aristas $1 \ldots k$

# Implementación

```
1  for (int i = 1; i <= n; ++i) {
2     for (int j = 1; j <= n; ++j){
3        d[i][j] = false;
4     }
5     d[i][i] = true;
6  }
7
8  for (int edge = 0; edge < m; ++edge){
9     int u, v; cin >> u >> v;
10    d[u][v] = true;
11 }
12
13 for (int k = 1; k <= n; ++k){
14    for (int i = 1; i <= n; ++i){
15       for (int j = 1; j <= n; ++j){
16          d[i][j] = d[i][j] or (d[i][k] and d[k][j]);
17       }
18    }
19 }
```

# Minimax

## Entrada

Un grafo con pesos $G = (V, E)$

## Objetivo

Para cada pareja de nodos $(i, j)$ , hallar el camino de $i$ hasta $j$ donde la arista más grande del camino sea lo más pequeña posible.

## Ejemplos

Que el peaje más caro sea lo más barato posible.
Que la autopista más larga sea lo más corta posible.

# Minimax

Sea $d(i,j,k)$ el costo de la arista más pequeña entre las aristas más grandes de todos los caminos de $i$ hasta $j$ si como nodos intermedios solo se pueden usar los nodos $1 \ldots k$.

$$d(i,j,0) = \left\{ \begin{array}{ll} 0 & \text{si } i = j \\ C_{i,j} & \text{si } (i,j) \in E \\ +\infty & \text{en otro caso} \end{array} \right\} \text{ para } \begin{array}{l} 1 \leq i \leq n \\ 1 \leq j \leq n \end{array}$$

$$d(i,j,k) = \min \left\{ \begin{array}{l} d(i,j,k-1) \\ \max \ \{d(i,k,k-1) + d(k,j,k-1)\} \end{array} \right\}$$

$$\text{Para } 1 \leq i,j,k \leq n$$

# Implementación

```
1  for (int i = 1; i <= n; ++i) {
2     for (int j = 1; j <= n; ++j){
3        d[i][j] = INF;
4     }
5     d[i][i] = 0;
6  }
7
8  for (int edge = 0; edge < m; ++edge){
9     int u, v, c; cin >> u >> v >> c;
10    d[u][v] = c;
11 }
12
13 for (int k = 1; k <= n; ++k){
14    for (int i = 1; i <= n; ++i){
15       for (int j = 1; j <= n; ++j){
16          d[i][j] = min( d[i][j] , max(d[i][k], d[k][j]) );
17       }
18    }
19 }
```

# Maximin

## Entrada

Un grafo con pesos $G = (V, E)$

## Objetivo

Para cada pareja de nodos $(i, j)$ , hallar el camino de $i$ hasta $j$ donde la arista más pequeña del camino sea lo más grande posible.

## Ejemplos

Que el trayecto menos seguro sea lo más seguro posible.
Que la autopista de menos carriles tenga la mayor cantidad de carriles.

# Implementación

```
1  for (int i = 1; i <= n; ++i) {
2     for (int j = 1; j <= n; ++j){
3        d[i][j] = -INF;
4     }
5     d[i][i] = INF;
6  }
7
8  for (int edge = 0; edge < m; ++edge){
9     int u, v, c; cin >> u >> v >> c;
10    d[u][v] = c;
11 }
12
13 for (int k = 1; k <= n; ++k){
14    for (int i = 1; i <= n; ++i){
15       for (int j = 1; j <= n; ++j){
16          d[i][j] = max( d[i][j] , min(d[i][k], d[k][j]) );
17       }
18    }
19 }
```

# Contenido

# Tarea

## Tarea

Resolver los problemas de

`http://contests.factorcomun.org/contests/55`

# Ayudas

## Problema A

## Problema B

## Problema C