# Practica-4-ALGORITMICA.pdf

Nashor

Algorítmica

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

**Práctica 4 ALGORÍTMICA**
**IGNACIO CARVAJAL HERRERA**
**GRUPO D2**

# 1. Enunciado del problema

**Ejercicio 4**
En el departamento de una empresa de traducciones, se desea hacer traducciones de textos entre varios idiomas. Se dispone de algunos diccionarios D(i,j) para traducir del idioma i al idioma j. Asumimos que para cada diccionario D(i,j) también tenemos el diccionario D(j,i). En el caso más general, no se dispone de diccionarios para cada par de idiomas i,j, por lo que es preciso realizar varias traducciones consecutivas i ->$I_1$ ->$I_2$ ->... $I_k$ ->j de idioma en idioma para llegar a la traducción deseada. Dados N idiomas y M diccionarios, determina si es posible realizar la traducción entre dos idiomas i y j dados como entrada y, en caso de ser posible, determina la secuencia de traducciones i ->$I_1$ ->$I_2$ ->... $I_k$ ->j a realizar que implique hacer el menor número de traducciones posible.

# 2. Análisis y notación a utilizar

- Se dispone de un total de N idiomas.
- Se dispone de un total de M diccionarios, cada diccionario D(i, j) permite traducir del idioma i al idioma j y viceversa.
- Con los idiomas y diccionarios dados, se debe poder traducir de un idioma i a un idioma j minimizando el número de traducciones realizadas.
- Utilizaremos una matriz D[N][N] donde cada casilla indicará el mínimo número de traducciones necesarias (diccionarios utilizados) para traducir los idiomas i y j.
- Con una matriz P[N][N] almacenaremos el índice del nodo intermedio (idioma) al que hay que realizar una traducción para poder llegar al idioma objetivo.
- Las variables origen y objetivo indican la traducción que se desea realizar en el menor número de pasos.

El objetivo es minimizar los valores que contendrá la matriz de traducciones D [N][N] permitiéndonos traducir con el menor número de diccionarios del idioma i al idioma j.

En otras palabras, el propósito es minimizar la expresión:

$$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} D_{i\,j}$$

# 3. Diseño de componentes

Veamos si el problema se puede resolver con Programación Dinámica:

- **El problema a resolver es de minimización.** Se debe encontrar un subconjunto de diccionarios y minimizar el número de traducciones que se realizan.

- **El problema debe de poder modelarse mediante una ecuación recurrente.** Definiendo el término T[i][j] como la secuencia mínima de diccionarios utilizados considerando que queremos llegar al idioma j, y consideramos los i primeros diccionarios. Lo que se busca en el problema es T[M][N], es decir, la mínima secuencia de traducciones después de escoger (o no) entre los M diccionarios para traducir al idioma N.

- Los idiomas se interpretarán como nodos, de manera que están enumerados desde 0..N.
- Asumimos que D es la matriz de adyacencia del grafo, D[i][j] es la distancia o número de traducciones para ir directos desde el nodo i al nodo j.
- D[i][j] = +INF si no hay arco (diccionario) entre i y j.
- **Función objetivo:** minimizar $D_N[i][j]$, con N el número de nodos del grafo, para todo i y para todo j.
- **El problema es resoluble por etapas:** En cada etapa se considera pasar por un nodo intermedio **k** para cada par de nodos **i**, **j** de origen y destino.

❖ **Caso base** :

$D_0[i][j] = D[i][j]$
Camino de i a j sin pasar por ningún nodo intermedio = Matriz de adyacencia

❖ **Caso general**

$D_k[i][j] = \min \{ D_{k-1}[i][j], D_{k-1}[i][k] + D_{k-1}[k][j] \}$

Interpretando la expresión anterior vemos como el caso general supone que para ir desde i hasta j, pudiendo pasar (o no) por los nodos 1 a k como intermedios, tiene dos posibilidades:

➢ Que el camino de i a j no pase por k. Lo cual se corresponde con la primera parte de la expresión: $D_{k-1}[i][j]$
➢ Que el camino de i a j pase por k. Segunda parte: $D_{k-1}[i][k] + D_{k-1}[k][j]$

- ★ **Verificación del Principio de optimalidad de Bellman.**

$D_0$[i][j] es óptimo: el mejor camino para traducir del idioma i al idioma j sin pasar por ningún otro idioma intermedio es D[i][j]. $D_1$[i][j] también es óptimo, ya que se selecciona el mejor camino entre ir directos del nodo i al nodo j o pasando por el nodo 1.

...$D_k$[i][j] es óptimo: En caso contrario, habría otros nodos en el camino de i a j pasando por 1...k-1 tal que el número de traducciones o nodos intermedios fuese menor que el considerado. Esta situación es imposible que suceda debido a que la ecuación recurrente siempre selecciona el menor número de traducciones.

# ● Algoritmo diseñado

Una vez vistas las componentes de diseño, queda por verificar que el problema es resoluble por Programación Dinámica

```
Algoritmo Traducciones(MatrizAdy[1..N][1..N], P[1..N][1..N] )

Para i = 0..N, hacer:
        Para j = 0..N, hacer:
                D[i][j] = MatrizAdy[i][j] // Longitud del camino mínimo conocido inicial
                P[i][j] = 0 // Para ir desde i a j inicialmente no se pasa por ningún nodo.

Para k = 0..N, hacer:
        Para i = 0..N, hacer:
                Para j = 0..N, hacer:
                        Si D[i][j] > D[i][k] + D[k][j], entonces:
                                D[i][j] = D[i][k] + D[k][j]
                                P[i][j] = k
                        Fin-Si

Devolver D, P
```

# 4. Eficiencia

La eficiencia del algoritmo anterior depende del tamaño de la matriz de adyacencia que se va a ir rellenando.

En primer lugar tenemos las sentencias repetitivas de inicialización. Son dos bucles anidados, y el bloque de sentencias que ejecutan tan solo está formado por operaciones elementales, de manera que la eficiencia del primer bucle es $O(n^2)$.

La segunda parte del algoritmo es la que realiza las comprobaciones y respectivas actualizaciones de datos. Consiste en tres bucles for. Si analizamos la estructura interna observamos que se realiza una sentencia condicional y como la comprobación de la condición y las sentencias que ejecuta cuando ésta se dá son operaciones elementales entonces la eficiencia de este if es O(1).
Eso era todo el bloque de sentencias que se realiza dentro de los bucles, tan solo queda conocer la eficiencia del más externo. El bucle que depende de la variable j (el más interno) realiza n iteraciones, al igual que los otros dos. Por lo que aplicando la expresión para calcular la eficiencia de sentencias repetitivas obtenemos que la eficiencia del bucle que depende de la variable k (el más externo) tiene eficiencia $O(n^3)$ y aplicando la regla de la suma con la primera mitad del algoritmo (la inicialización) concluimos que la eficiencia del algoritmo es **$O(n^2) + O(n^3) = O(n^3)$**

# 5. Detalles de implementación

Instrucciones para compilar:

g++ main.cpp -o practica

- Las matrices se inicializan conforme al valor de la variable V que indica el número de idiomas y diccionarios.

- La matriz de adyacencia (graph[ ][ ]) que indica que diccionarios existen .Es la que se encuentra ya inicializada en el main.

- Se piden los dos idiomas a los que se desea realizar la traducción.

- Se realiza una llamada a la función traducciones() que sigue la implementación correspondiente al pseudocódigo que hay a continuación.

- Copiamos la matriz de adyacencia en la matriz D, que indicará el número mínimo de traducciones para traducir del idioma i al idioma j. **D[i][j] contiene la distancia mínima entre i y j.**

- Inicializamos a cero la matriz P que indica el idioma intermedio al que ha sido necesario realizar una traducción. **P(i,j) = k**, donde k es un idioma intermedio en el camino entre i y j. **Para calcular la secuencia tendremos que calcular P(i,k) y P(k,j).**

- A través de una serie de bucles se busca el camino mínimo para llegar desde el nodo i al nodo j, se actualizan estados de las matrices D y P en caso de encontrar un camino mínimo.

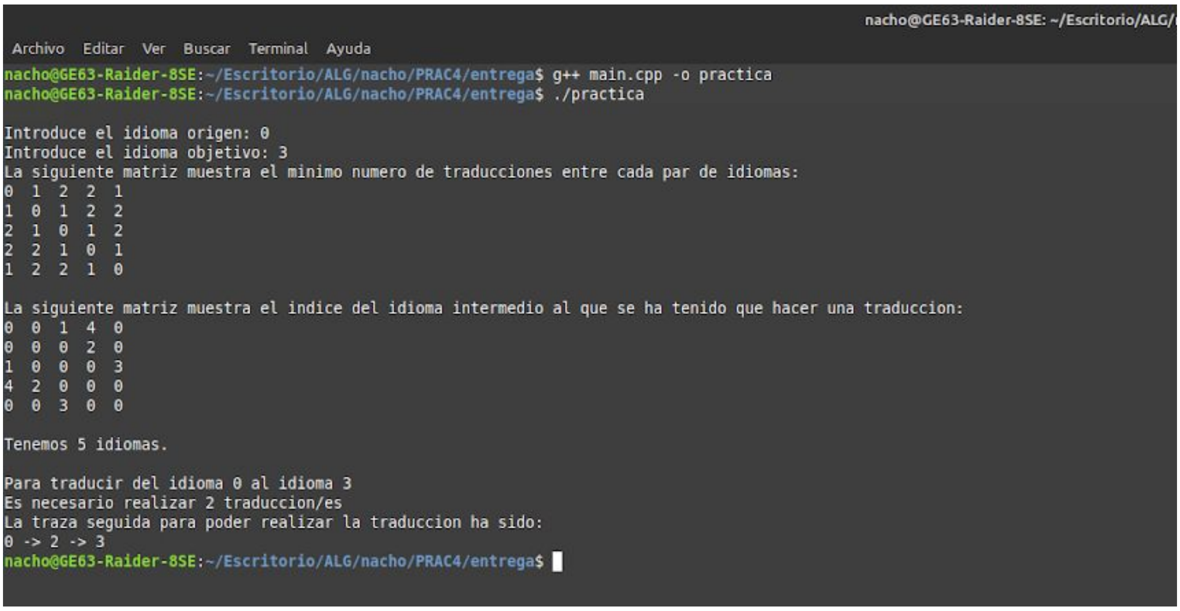
- Se llama a la función print. Ésta se encarga de imprimir los valores de las matrices D y P.

# 6. Pruebas

El ejemplo está ya implementado en el propio .cpp de manera que con tan solo compilarlo y ejecutarlo se nos muestra un ejemplo.

Tener en cuenta que el número de diccionarios e idiomas utilizados para este ejemplo es 5. Si proporcionamos las entradas:
- i = 0
- j = 3

El programa nos va a indicar el número de traducciones necesarias para traducir del idioma con índice 0 al idioma con índice 3.



```
nacho@GE63-Raider-8SE:~/Escritorio/ALG/nacho/PRAC4/entrega$ g++ main.cpp -o practica
nacho@GE63-Raider-8SE:~/Escritorio/ALG/nacho/PRAC4/entrega$ ./practica

Introduce el idioma origen: 0
Introduce el idioma objetivo: 3
La siguiente matriz muestra el minimo numero de traducciones entre cada par de idiomas:
0  1  2  2  1
1  0  1  2  2
2  1  0  1  2
2  2  1  0  1
1  2  2  1  0

La siguiente matriz muestra el indice del idioma intermedio al que se ha tenido que hacer una traduccion:
0  0  1  4  0
0  0  0  2  0
1  0  0  0  3
4  2  0  0  0
0  0  3  0  0

Tenemos 5 idiomas.

Para traducir del idioma 0 al idioma 3
Es necesario realizar 2 traduccion/es
La traza seguida para poder realizar la traduccion ha sido:
0 -> 2 -> 3
nacho@GE63-Raider-8SE:~/Escritorio/ALG/nacho/PRAC4/entrega$
```