ANÁLISIS

Se dispone de un total de M diccionarios, cada diccionario para poder traducir entre dos idiomas indistintamente. El problema puede plantearse como un grafo en el que cada vértice sería un idioma, y existirían aristas entre nodos v y w siempre que exista un diccionario de v a w. El grafo sería no dirigido, dado que el mismo diccionario permite traducir de v a w o de w a v. Podríamos también asociar el peso de coste 1 a cada arista, indicando de esta forma que se hace uso de 1 diccionario para pasar de un idioma a otro.

Con este planteamiento, el problema se puede plantear como un caso particular de problema de caminos mínimos, en el que podríamos hallar la secuencia de traducciones entre cualquier par de idiomas i,j. Trabajaramos con la matriz de adyacencia del grafo L(i,j), que tendrá valor L(i,j)=1 si existe un diccionario de i a j (y viceversa), o valor 0 si no existe tal diccionario.

DISEÑO DEL ALGORITMO

- **El problema propuesto es un problema de optimización**, dado que se debe minimizar el número de diccionarios a usar para realizar una traducción entre idiomas.
- **Se puede resolver por etapas**: En cada etapa comprobaremos si una secuencia de traducciones de i a j, T(i,j), es mejor como está o considerando traducir por un idioma intermedio k. Numeraremos los diccionarios desde 1..M para plantear la ecuación.
- **Ecuación recurrente**:

    - **Variables:**
        - i: Idioma original desde el que se desea traducir.
        - j: idioma final al que se desea traducir.
        - k: Etapas. Diccionario intermedio a considerar en la traducción de i a j.
        - $T_k(i,j)$= Número mínimo de diccionarios a usar para traducir desde el idioma i hasta el idioma j, habiendo considerado usar (o no usar) diccionarios desde el 1 hasta el k para realizar traducciones intermedias.
    - **Valor objetivo:**
        - Se desea conocer $T_M(i,j)$ para todo i,j; es decir, la secuencia de traducciones mínima desde cada idioma i a cada idioma j, suponiendo que consideramos usar cualquier subconjunto de los diccionarios disponibles M para realizar la traducción.
    - **Caso/s base:**
        - El caso base se dará desde la matriz de adyacencia, cuando no consideramos pasar por ningún diccionario intermedio; es decir: $T_0(i,j)= L(i,j)$
    - **Caso general:**
        - En $T_k(i,j)$, ya hemos resuelto la secuencia de traducciones mínima entre i y j usando (o no) diccionarios desde el 1 hasta el k-1, y tratamos de decidir si es mejor hacer uso del diccionario k. Tenemos dos opciones:
            - Es mejor usar el diccionario k: $T_k(i,j)= T_{k-1}(i,k) + T_{k-1}(k,j)$
            - No es mejor usar k: $T_k(i,j)= T_{k-1}(i,j)$

        De este modo, obtenemos la ecuación:

        $$T_k(i,j)= \min\{ T_{k-1}(i,j), T_{k-1}(i,k)+T_{k-1}(k,j) \}$$

- **Verificación del cumplimiento del P.O.B:** Para el caso base $T_0(i,j)$ en el se decide no pasar por ningún idioma intermedio, la solución $T_0(i,j)=L(i,j)$ es óptima, dado que no existe una secuencia de traducción mejor entre cualquier par de idiomas. Cuando llegamos a resolver $T_k(i,j)$, no sería posible obtener una solución mejor que la dada por el algoritmo, dado que ésta depende de $T_{k-1}(i,j)$ para todo i,j, que es óptima. Si existiese una solución mejor que la dada, $T_{k-1}(i,j)$ debería no ser óptimo, y querría decir que existe al menos una decisión errónea en las tomadas hasta llegar a $T_k(i,j)$. Esto no es posible, dado que la ecuación asegura que siempre se va escogiendo el valor mínimo.

- **Diseño de la memoria:**

  - Para resolver el problema, $T_k(i,j)$ se representará como una matriz.
  - La matriz tendrá n filas, indexadas de 1..n. Cada fila i estará asociada a un idioma origen.
  - La matriz tendrá n columnas, indexadas de 1..n. Cada columna j estará asociada a un idioma destino.
  - Cada celda de la matriz $T_k(i,j)$ contendrá el mínimo número de diccionarios a usar para traducir desde i hasta j, haciendo uso de un subconjunto de diccionarios entre 1 y k.
  - La memoria se rellenará de la siguiente forma:
    - En primer lugar, se rellenan las celdas correspondientes a los casos base $T_0(i,j)=L(i,j)$.
    - En segundo lugar, se pasará por todas las etapas, comprobando si cada diccionario k aporta algo a la traducción mínima de i a j.
    - En cada etapa, se comprobará para cada par (i,j) si es mejor traducir a través del diccionario k, o no hacerlo (ecuación recurrente).
  - Necesitaremos una memoria auxiliar P(i,j) para reconstruir la solución. P(i,j)=k indica que para traducir de i a j, primero hay que traducir de P(i,k) y luego de P(k,j).

Con este diseño, construimos el algoritmo de Programación Dinámica como sigue:

ALGORITMO T, P= Diccionarios($\{I_1, I_2, \ldots, I_n\}$: Conjunto de idiomas, origen, destino : Idiomas origen y destino, $\{d_1, d_2, \ldots, d_M\}$: Conjunto de diccionarios)

```
Construir matriz de adyacencia L(1..M, 1..M), inicializada a L(i,j)= infinito para todo i,j
Para cada idioma i en 1..n, hacer:
        Para cada idioma j en 1..n, hacer:
                Si i=j, hacer L(i,j)= 0
                En otro caso, si existe k tal que i,j están en d_k, hacer:
                        L(i,j)= L(j,i)= 1
        Fin-Para
Fin-Para

Crear tabla T[1..n, 1..n] inicializada a L
Crear tabla P(1..n, 1..n)
Para cada idioma i en 1..n, hacer:
        Para cada idioma j en 1..n, hacer:
                Si L(i,j)<>infinito, hacer:
                        P(i,j)= i
                En otro caso, hacer
                        P(i,j)= vacío
```

```
Para cada diccionario k en 1..M, hacer:
        Para cada i en 1..n, hacer:
                Para cada j en 1..n, hacer:
                        Si T(i,j) > T(i,k)+T(k,j), hacer:
                                T(i,j) = T(i,k)+T(k,j)
                                P(i,j)= k

Devolver T, P
```

- **Recuperación de la solución:** La solución se recuperará desde el valor objetivo de la matriz calculada previamente, recorriendo hacia atrás la recurrencia guardada en la matriz P. El siguiente algoritmo devuelve la solución S :

```
ALGORITMO S= RecuperaSolución( P, origen, destino )

    S ← ∅
    Si P(origen, destino) = vacío,
            devolver S=”No hay solución”
    En otro caso, Si origen=destino o P(origen, destino)= origen
            devolver S
    En otro caso,
            Calcular S1= RecuperaSubsecuencia(P, origen, P(origen, destino))
            Calcular S2= RecuperaSubsecuencia(P, P(origen, destino), destino)
            Si S1=”No hay solución” o S2=”No hay solución” ,
                    devolver S=”No hay solución”
            En otro caso, hacer
                    devolver S=[{origen}, S1, {P(i,j)}, S2, {destino}]
```

DETALLES DE IMPLEMENTACIÓN

En la solución planteada en el código fuente hay un único fichero **main.cpp** conteniendo:

- Función **Diccionarios**, que implementa el algoritmo *Diccionario* previo.
- Función ***RecuperaSolución***, que implementa el algoritmo *RecuperaSolución* previo. Esta función es llamada desde **Diccionarios** para calcular la solución final.

El funcionamiento del programa es el siguiente:

1. Se lee desde fichero los idiomas existentes y los diccionarios
2. Se llama al algoritmo PD, que devuelve el coste y la recuperación de la solución.
3. Se muestran los resultados.

Adicionalmente, se ha construido un fichero makefile para construir el programa de forma simple. Se compila y ejecuta de la siguiente forma:

1. Nos situamos en la carpeta donde está el código fuente y, desde la terminal, escribimos: ***make***
2. Ejecutamos el programa desde la terminal escribiendo: ***./Solucion < problema.dat***