ANÁLISIS

El problema propuesto es el problema del cambio de monedas estudiado en teoría. Se asume que hay que devolver un cambio de cantidad N, con n tipos de monedas {1...n} cuyo valor es $\{d_1, d_2, \ldots, d_n\}$. Adicionalmente, se pone la restricción de que exista un número finito de monedas de cada tipo $\{m_1, m_2, \ldots, m_n\}$. También se asume que $d_1 = 1$.

DISEÑO DEL ALGORITMO

- **Resolución del problema por etapas:** El problema se puede resolver por etapas. En cada etapa se seleccionaría echar o no echar una moneda de un tipo dado. Supondremos que las monedas están ordenadas de menor a mayor valor d_i para asegurar una solución óptima factible desde las etapas más tempranas.
- **Ecuación recurrente**:

La solución depende de las etapas (tipos de monedas a considerar devolver, que notaremos como *i*) y del cambio que quede por devolver, que notaremos como *j*. Llamaremos al valor T(i,j) al mínimo número de monedas a devolver para un cambio *j*, supuesto que consideramos echar o no echar monedas de tipo 1, echar o no echar monedas de tipo 2, echar o no echar monedas de tipo 3, … hasta echar o no echar monedas de tipo i.

En la etapa i consideramos echar monedas de tipo *i*. Caben dos posibles decisiones:

- Devolver una moneda de tipo *i*: En tal caso, el mínimo número de monedas a devolver sería $1+T(i, j-d_i)$; es decir, 1 (por la moneda que estamos echando, y el mínimo número de monedas a devolver para un cambio que resta el valor de la moneda que estamos devolviendo, supuesto que podemos seguir echando más monedas de tipo i.
- No devolver una moneda de tipo i: En tal caso, el mínimo número de monedas a devolver será el mismo que cuando ni siquiera habíamos considerado echar una moneda de tipo i.

Con estas dos decisiones posibles, y dado que el problema es de minimización, tendríamos que podríamos expresar la ecuación recurrente como:

$$T(i,j)= \min\{T(i-1, j), 1+T(i, j-d_i)\}$$

Los casos base para esta ecuación serían:

- Sin cambio a devolver, j=0: T(i,0)= 0 . Da igual el tipo de moneda a devolver que estemos considerando; si no hay que devolver cambio, el mínimo número de monedas a devolver es de 0.
- Considerando echar monedas de sólo $d_1 = 1$ unidad. En tal caso, T(1, j)= j ; es decir, para devolver un cambio j con monedas de 1 céntimo necesitamos j monedas.

- **Valor objetivo:** Se desea conocer el valor T(n,N), el mínimo número de monedas a devolver para un cambio N, suponiendo que consideramos echar cualquier tipo de monedas desde el {1...n}.

- **Verificación del cumplimiento del P.O.B:** (ver teoría)
- **Diseño de la memoria:**
    - Para resolver el problema, T(i,j) se representará como una matriz.
    - Esta matriz tendrá n filas. Cada fila i estará asociado a un tipo de moneda i.
    - Esta matriz tendrá N+1 columnas. Cada columna estará asociada a cada posible cambio a devolver entre {0..N}.
    - Cada celda de la matriz T(i,j) contendrá el mínimo número de monedas a devolver para un cambio j, suponiendo que estamos considerando devolver monedas entre todos los tipos desde el 1 hasta el i.
    - La memoria se rellenará de la siguiente forma:
        - En primer lugar, se rellenan las celdas correspondientes a los casos base.
        - En segundo lugar, se rellenarán las filas {2, 3, …., n}, en orden secuencial.
        - Cada fila se rellenará en orden creciente de columnas {1...N}.

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

ALGORITMO T, V= Monedas(N, n, {d_1, d_2, …, d_n})

T ← matriz de n filas indexadas {1...n} y N columnas indexadas {0..N}
Para cada fila i en {1...n}, hacer:
    T(i,0)= 0
Para cada columna j en {1...n}, hacer:
    T(1, j)= j

Para cada fila i en {2..n}, hacer:
    Para cada columna j en {1..N}, hacer:
        T(i,j)= min{T(i-1, j), 1+T(i, j-d_i)}

V ← T(n, N)
Devolver T, V

- **Recuperación de la solución:** La solución se recuperará desde el valor objetivo de la matriz calculada previamente, recorriendo hacia atrás la recurrencia. El siguiente algoritmo devuelve la solución S (el conjunto de monedas concreto a devolver), suponiendo un número de monedas de cada tipo infinito:

ALGORITMO S= RecuperaMonedas(T(1..n, 0..N) : Tabla resultante del algoritmo Monedas)

S ← ∅
i ← n, j ← N
Mientras j <> 0, hacer:
    Si i>1 y T(i,j)= T(i-1, j), hacer:
        i ← i-1
    En otro caso, hacer:
        j ← j-d_i
        Añadir moneda de tipo i a S
Devolver S

- **Recuperación de la solución en el caso de que haya un número finito $\{m_1, m_2, \ldots, m_n\}$ de cada tipo de moneda**

Para resolver este problema de forma óptima completamente, habría que reformular la ecuación inicial levemente e incluir restricciones. El enunciado pide que modifiquemos sólo la función de recuperación de la solución, por lo que podría darse el caso de que, en algún sistema monetario, el algoritmo completo (construcción de la solución y recuperación de la misma) no fuese óptimo.

La idea general del algoritmo de recuperación podría ser la siguiente:

- Deberíamos llevar un contador de cada tipo de monedas que se ha devuelto $\{c_1, c_2, \ldots, c_n\}$. Cada valor c_i estaría inicialmente inicializado a 0 (porque no se ha decidido echar ninguna moneda).
- Cuando estamos evaluando T(i,j), se podría considerar echar una moneda de tipo i sólo si se cumple que $c_i < m_i$. En caso contrario, sólo se debería considerar la decisión T(i-1, j) para recorrer hacia atrás la solución hasta un caso base.
- Si se decide devolver una moneda de tipo i, se debe actualizar el contador $c_i \leftarrow c_i + 1$
- Si se consigue llegar al caso base T(i,0), se devuelven las monedas seleccionadas.
- Si se llega al caso base T(1, j), con j>0, y no quedan monedas de tipo 1, entonces se decide que no hay solución (en este caso el algoritmo no es óptimo porque podría haber alguna decisión en algún T(i,j), distinta a la tomada, que sí devolviese algún conjunto de monedas).

El algoritmo de recuperación de la solución modificado es como sigue:

ALGORITMO S= RecuperaMonedas2(T(1..n, 0..N) : Tabla resultante del algoritmo Monedas)

C ← $\{c_1, c_2, \ldots, c_n\}$
Para cada i en {1..n}, hacer:
    $c_i \leftarrow 0$
S ← ∅
i ← n, j ← N
Mientras j <> 0, hacer:

    Si T(i,j) <> T(i, j-d_i) ó $c_i >= m_i$, hacer:
        i ← i-1
    En otro caso, hacer:
        j ← j-d_i
        Añadir moneda de tipo i a S
    Si i < 1, hacer:
        Devolver “No hay solución”
Devolver S

DETALLES DE IMPLEMENTACIÓN

En la solución planteada en el código fuente hay varios ficheros:

- **Problema.h / problema.cpp:** Contienen una clase Problema que modela una instancia del problema a resolver. En particular:
  - Se mantiene en memoria en un array ValoresTipos el conjunto de valores de cada tipo de moneda, supuesto ordenados en orden creciente y con el primer valor igual a 1.

    - Se mantiene en memoria un array CantidadTipo con el número de monedas que la máquina tiene de cada tipo.
    - Se mantiene en memoria una variable CantidadDevolver con el cambio a devolver.
    - La variable N indica el número de tipos de monedas diferentes que se tiene.
    - Adicionalmente, se implementan varios getters para conocer el valor de un tipo de moneda i (getValorTipo(i)), el número de tipos de monedas (getN()) y la cantidad a devolver (getCantidadDevolver())
    - Una función para saber si hay monedas en la máquina de un tipo dado i (HayMonedasTipo(i)), así como para quitar una moneda de un tipo i dado en la máquina (QuitarMonedaTipo(i)).
    - Funciones auxiliares para leer una instancia del problema desde fichero (LeerFichero(fichero)) y para ordenar las monedas por valor (Ordenar).
- **Solucion.h / solucion.cpp:** Contienen una clase Solucion para representar la solución a una instancia del problema (conjunto de valores de monedas concretas a devolver). En particular:
    - El campo ValorMonedas implementa un array de valores de monedas devueltas como la solución, de tamaño N (campo N).
    - El campo coste contiene el número mínimo de monedas a devolver (solución al problema original, valor V del algoritmo Monedas previo).
    - Método para conover cuántas monedas hay ahora en la solución (getN()), e insertar una moneda de valor v en el array anterior (setValorMoneda(idx, v).
    - Sobrecarga del operador += para añadir una nueva moneda a la solución, dado su valor v.
- **ProgDinCambioMonedas.h / ProgDinCambioMonedas.cpp:** Implementa en una única función Solución=AlgoritmoProgDinCambioMonedas(Problema) los algoritmos previos dados en Monedas y RecuperaMonedas2

Una instancia del problema se proporciona al programa a través de fichero, con el siguiente formato:

Linea 1: Valor N, número de tipos de monedas.
Linea 2: Valor con el cambio a devolver
Linea 3→ N+2 : par de valores di mi con el valor de cada tipo de moneda y la cantidad existente en la máquima
Lineas N+2→ fin de fichero: Cada línea proporciona información de una calle (plazas que unen, precio y nombre de la calle).

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***
3. Se nos mostrará por consola la solución al problema dado en el fichero **problema.dat**, junto con su coste.