Problema 2.

Se quiere obtener la suma máxima de los elementos de un arreglo circular (donde el último elemento es adyacente al primero…) tal que ningún par de elementos adyacentes sea parte de la suma (si el i-ésimo elemento es parte de la suma, entonces no podrán serlo en elemento (i-1) ni el elemento (i+1)).

El arreglo circular será dado por la entrada como una lista de enteros no negativos separados por espacio donde el primer y el último elemento de la misma son adyacentes.

A) **Definición de un estado**:

MEMO[i][k]: Representa la máxima suma a partir del elemento i (inclusive) hasta el último elemento respetando la restricción sobre los elementos adyacentes descrita arriba donde la variable k indica si se está usando o no el último elemento, es decir, MEMO[i][0] almacena la mayor suma desde i hasta el último elemento SIN utilizar el último elemento, y MEMO[i][1] guarda la mayor suma desde i hasta el último elemento forzosamente utilizándolo.

Se utilizó esta aproximación debido a que, en el proceso de cómputo de la solución final, muchos sino todos los valores ya calculados serían solo válidos para una situación especial, por esto se utiliza la segunda dimensión de la matriz **k**, con ella se almacenan valores calculados correctos para los dos casos más grandes que se pueden ver en cualquier instancia de este problema: si tomar el primer elemento o no tomarlo repercute en la decisión de tomar el último o no tomarlo y viceversa.

También se contempló una solución que constara tomar uno de los valores extremos y calcular la máxima suma posible con los valores del subarreglo circular interno que quedaría al remover el elemento tomado y sus adyacentes del problema original, sin embargo no se consideró más allá dado que, además de diferenciar los casos donde se utiliza el primero o el último de los elementos de cada subarreglo, también sería necesario poder direccionar de manera no ambigua a un subarreglo interno para almacenar en la matriz de memorización, esto se haría utilizando una matriz de dos dimensiones donde la primera indique el primer índice el subarreglo y la segunda indicaría el último índice del mismo subarreglo (además de otra dimensión para identificar el uso del primer o el último elemento), pero esa aproximación tomaría grandes cantidades de memoria en casos relativamente pequeños.

B) **Relación de recurrencia**: DP(i, k)

$$dp(i, 0) = max(N[i] + dp(i+2, 0) , dp(i+1, 0));$$
$$dp(i, 1) = \{$$
$$\quad i = 0, max (dp(i+2, 1) , dp(i+1, 1));$$
$$\quad i \neq 0, max (N[i] + dp(i+2, 1) , dp(i+1, 1));$$
$$\}$$

*** Nótese que inicialmente los valores en la matriz memo para el último elemento deben de estar calculados antes de llamar al algoritmo, esto se ve con claridad en el código fuente de la solución a este problema proporcionado con este documento bajo el nombre de **2.c** ***

Esta relación de recurrencia fue la deseada por ser relativamente natural a la hora de pensar en la solución del problema. Véase de esta manera: *Si no se tomara en cuenta la cualidad circular del arreglo, es decir, si se considerase al mismo como un arreglo común y lineal entonces, se podría utilizar una solución DP idéntica a la usada por el algoritmo **01 knapsack**, donde todos los elementos del arreglo tendrían el mismo peso y la capacidad original es virtualmente infinita, en este caso la única restricción que se puede ver es la no utilización de elementos adyacentes en la solución final. Dicho esto, lo único restante es tomar en cuenta que los elementos extremos del arreglo no sean utilizados.*