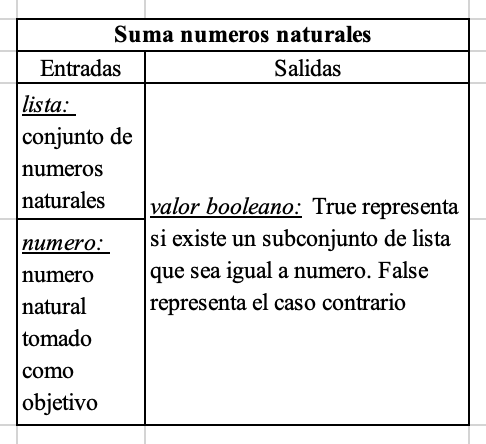
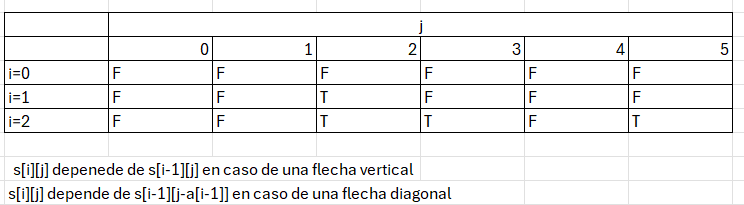
1. **Suma de numeros naturales**
2. Entradas y salidas del problema



1. Función que representa optimización del problema.
2. Ecuación de recurrencia
3. Grafo de necesidades



1. Algoritmo desarrollado en GCL

***fun*** *sumaNaturales (lista, numero):*

***var*** *tamanoLista:* ***nat***

***var*** *memo: Array [0,n) of* ***nat***

***var*** *x :* ***nat***

*v****ar*** *i :* ***nat***

***var*** *j :* ***nat***

*tamanoLista :=* ***len****(lista)*

*memo := []*

***do******x in range*** *(tamanoLista+1)*

*memo.****append*** *([False] \* (numero + 1))*

***od***

*memo [0][0] := True*

***do*** *i* ***in range*** *(1, tamanoLista + 1)*

***do*** *j* ***in range*** *(numero + 1)*

*memo [i][j] := memo [i-1][j]*

***if*** *j >= lista [i-1]* ***and*** *memo [i-1][j-numero[i-1]]*

*memo [i][j]:= True*

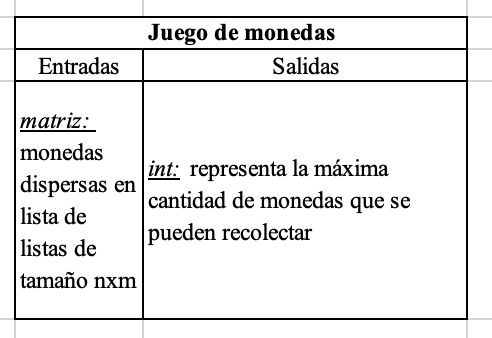
*fi*

*od*

*od*

***ret*** *memo [tamanoLista][numero]*

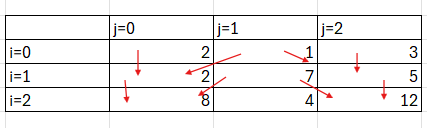
1. **El juego de las monedas**
2. Entradas y salidas del problema



1. Función que representa optimización del problema.
2. Ecuación de recurrencia

* *Siendo* *S el conjunto de valores posibles que se pueden tomar desde la* ***fila anterior***

1. Grafo de necesidades



1. Algoritmo desarrollado en GCL

***fun*** *monedas(matrizMonedas):*

***var*** *nFilas:* ***nat***

***var*** *nColumnas:* ***nat***

***var*** *memo: Array [0, n) of* ***nat***

***var*** *i:* ***nat***

***var*** *j:* ***nat***

***var*** *k:* ***int***

***var*** *max:* ***int***

*nFilas :=* ***len****(matrizMonedas)*

*nColumnas :=* ***len****(matrizMonedas[0])*

*memo := []*

***do*** *i* ***in range****(nFilas) →*

*memo.****append****([0] \* nColumnas)*

***od***

***do*** *j* ***in range****(nColumnas) →*

*memo[0][j] := matrizMonedas[0][j]*

***od***

***do*** *i* ***in range****(1, nFilas) →*

***do*** *j* ***in range****(nColumnas) →*

*max := 0*

***do*** *k* ***in*** *[-1, 0, 1] →*

***if*** *0 ≤ j + k < nColumnas →*

*max :=* ***max****(max, memo[i-1][j + k])*

***fi***

***od***

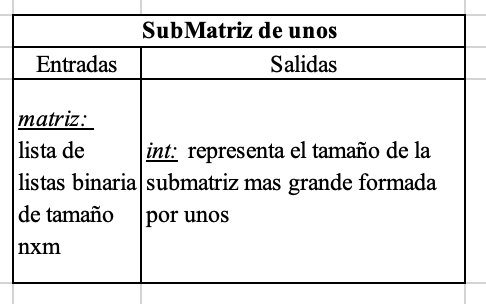
*memo[i][j] := matrizMonedas[i][j] + max*

***od***

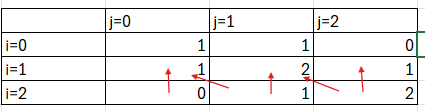
***od***

***ret max****(memo[nFilas-1])*

1. **Submatriz de unos**
2. Entradas y salidas del problema.



1. Función que representa optimización del problema.
2. Ecuación de recurrencia
3. Grafo de necesidades



*NOTA: El número 2 en la matriz f(i,j) indica que hay una submatriz cuadrada de unos de tamaño 2 \* 2 terminando en esa celda.*

1. Algoritmo desarrollado en GCL

***fun*** *subMaxima(matriz):*

***var*** *num\_filas:* ***na****t*

***var*** *num\_columnas:* ***nat***

***var*** *memo: Array [0,n) of* ***nat***

***var*** *subMatrizMax: nat*

***var*** *i:* ***nat***

***var*** *j:* ***nat***

*num\_filas :=* ***len****(matriz)*

*num\_columnas :=* ***len****(matriz[0])*

*memo := []*

***do*** *i* ***in range****(num\_filas) →*

*memo.****append****([0] \* num\_columnas)*

***od***

*subMatrizMax := 0*

***do*** *i* ***in range****(num\_filas) →*

*memo[i][0] := matriz[i][0]*

*subMatrizMax :=* ***max****(subMatrizMax, memo[i][0])*

***od***

***do*** *j* ***in range****(num\_columnas) →*

*memo[0][j] := matriz[0][j]*

*subMatrizMax :=* ***max****(subMatrizMax, memo[0][j])*

***od***

***do*** *i* ***in range****(1, num\_filas) →*

***do*** *j* ***in range****(1, num\_columnas) →*

***if*** *matriz[i][j] = 1 →*

*memo[i][j] := 1 +* ***min****(memo[i-1][j], memo[i][j-1], memo[i-1][j-1])*

*subMatrizMax :=* ***max****(subMatrizMax, memo[i][j])*

***fi***

***od***

***od***

***ret*** *subMatrizMax*