Tarea 3 - Programación dinámica

Forma de trabajo: Grupos de hasta 3 personas

Integrantes:

- David Ricardo Bernal Alfonso
- Diego Mauricio Bulla

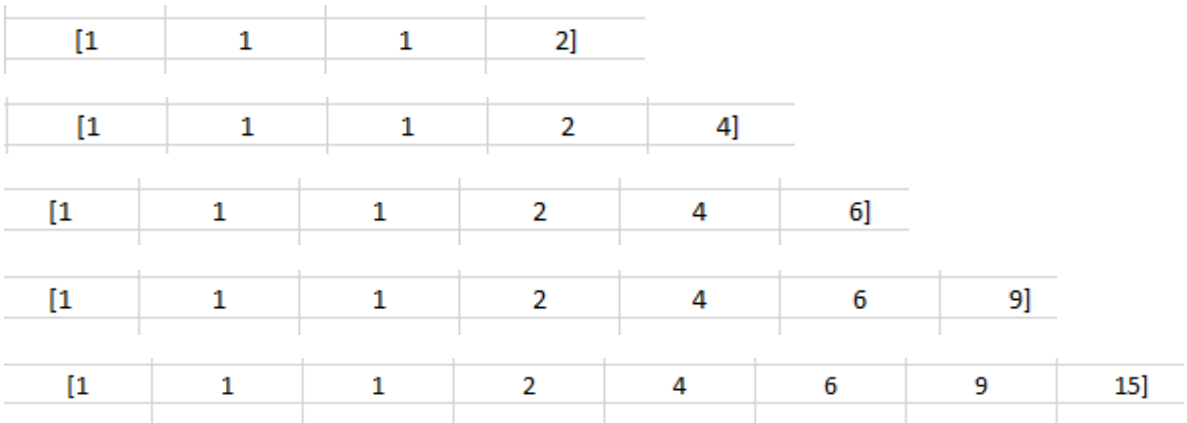
Todos los ejercicios deben desarrollarse con la técnica de programación dinámica. Para cada uno, su informe debe incluir formalización, definición del caché, recursión, diagrama de necesidades y análisis.

1. Escriba un algoritmo que determine de cuántas formas puede escribirse un número natural n como suma de 1, 3 y 4.
    a. Formalización. Se tiene un numero entero N, y se debe determinar el número el número formas de expresarlo como una suma de 1,3 y 4.
    b. Definición del cache. El cache este definido por un arreglo con tamaño n + 1 donde n es el numero entero que se debe expresar como suma de 1,3 y 4.
    c. Definición de recursión. Se divide el problema en sub-problemas para resolverlo. Sea DP [n] el número de formas de escribir N como la suma de 1,3 y 4. consideremos una posible solución $n = x_1 + x_2 + \ldots + x_n$.
        - Si el último número es 1, la suma de los números restantes es n-1. Entonces, el número que termina con 1 es igual a DP[n-1].
          ```
          DP [0] = DP [1] = DP [2] = 1
          ```
        - En cambio, si el último número es diferente de 1 como en este caso que el número final es 3 y 4 la recurrencia final seria.

          ```
          DP[i] = DP [i - 1] + DP [i - 3] + DP [i - 4]
          ```

    d. Diagrama de necesidades.

        A medida que se va recorriendo de manera recursiva el cache se va modificando como podemos verlo en la imagen de tal manera que el último número del arreglo nos indicara el número de formas como expresar el numero n que nos llega.



El número es 7 y la cantidad de formas de expresarlo como una sima de 1,3 y 4 son 15

e. Análisis.

Complejidad temporal: T(n) = O(n) porque se recorre el arreglo con un ciclo (for).

Complejidad espacial: S(n) = O(n) ya que usamos un arreglo para almacenar las respuestas que nos darán solución al problema.

2. Escriba el algoritmo del morral tal como lo vimos en clase. Imprima el caché a cada pasada del for externo.
   a. Formalización. Se tiene una secuencia S de n objetos tal que S=<S_1, S_2,…S_n> y cuyo peso está determinado por i=1,2,……,n donde p_i es el peso correspondiente para el objeto S_i. A su vez, cada objeto Si tiene una utilidad U_i para i=1,2,….,n. El peso máximo de la mochila es P y se debe determinar la mayor utilidad obtenible al cargar los objetos en dicha mochila.
   b. Definición del cache. El cache está definido por una matriz de dimensiones ixj donde i es el número de objetos o elementos de la secuencia y j es el peso máximo que puede soportar la mochila. Cada casilla representa el valor utilitario o monetario máximo de los primero objetos i que se puede conseguir con un morral con peso máximo j (donde j<=P).
   c. Definición de recursión. La recursión está dada por las siguientes condiciones:
      Si el objeto no cabe en la mochila, entonces la mejor mochila será la mochila que teníamos sin haber considerado dicho objeto.
      - *mochila[i][j] = mochila[i-1][j]*

      Si el objeto cabe en la mochila, debo decidir cuál mochila aporta mayor valor monetario entre la mochila anterior (la que teníamos sin considerar el objeto actual) y la mochila con el objeto actual insertado en ella.
      - *mochila[i][j] = max(mochila[i-1][j],valor[i-1]+mochila[i-1][j-peso[i-1]])*
   d. Diagrama de necesidades.

	0	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0	0
2	0	0	0	2	2	2	2	2
3	0	2	2	2	4	4	4	4
4	0	2	2	4	6	6	6	8
5	0	2	2	4	6	7	7	9
6	0	2	3	5	6	7	9	10

   e. Análisis. La complejidad espacial es o(n), complejidad espacial o(n)
      En el primer ciclo (for) recorremos las columnas de la matriz desde 0 hasta J donde J es el peso máximo P, y su costo será de O(P). De forma similar recorremos las filas de la matriz desde 0 hasta I donde i es número de elementos de la secuencia, es

decir, que el costo será de O(n). Por lo tanto, la complejidad temporal será de O(n*P).

Al final de la matriz, después de calcular el valor de la mochila en cada casilla de la matriz, se obtiene la respuesta que corresponde al máximo valor utilitario posible de obtener utilizando el peso máximo. El costo es O (1) ya que solo retornamos el valor en dicha posición M[n][P].

3. Ejecute varios ejemplos con el algoritmo del punto anterior y encuentre un patrón en las decisiones que toma el algoritmo vs. los valores del caché.
   En la mayoría de los casos, tienen mayor relevancia los objetos con menos peso independientemente si tienen mayor valor o no ya que el peso máximo es la condición que limita las combinaciones
4. Mejore el algoritmo del morral de dos maneras:
   a. Haga que retorne la lista de artículos escogidos.
      Se utilizó una técnica de backtracking para retornar la lista de los elementos escogidos en la mochila a partir del mejor obtenido en la mochila (la última posición de la matriz). El costo de este proceso es de O(n) ya que solo recorremos las filas de forma semi-diagonal desde el ultimo valor de la matriz M[n][P].
   b. Mejore su complejidad espacial.
5. Desarrolle y analice dos de los ejercicios 4.6, 4.8 y 4.9 de [Boh92].
   Ejercicio 4.6 – Sub-secuencia Suma
   a. Formalización. Sea S = {s_1, …, s_n} un conjunto de n números naturales. determinar si se puede partir en dos conjuntos cuyos elementos sumen la misma cantidad.
   b. Definición del cache. El cache es una matriz j x i donde j va de cero hasta la mitad de la suma total de la secuencia S, donde i va desde 0 hasta la cantidad de números que tiene la secuencia S. En otras palabras, el cache seria (suma total de s/2) * n
   c. Definición de recursión. La recursión de puede dividir de estas dos maneras
      - Sea $Matriz[i]\ [j] = True$ si el subconjunto de {arr [0], arr [1],…, arr[i-1} tiene suma igual a i se retorna verdadero.
      - Sea $Matriz[i]\ [j] = False$ si el subconjunto de {arr [0], arr [1],…, arr[i-1} tiene suma diferente de i se retorna falso.
   d. Diagrama de necesidades

	{}	{2}	{2,1}	{2,1,1}	{2,1,1,1}	{2,1,1,1,3}	{2,1,1,1,3,2}
0	true	true	true	true	true	true	true
1	false	false	true	true	true	true	true
2	false	true	true	true	true	true	true
3	false	false	true	true	true	true	true
4	false	false	false	true	true	true	true

   e. Análisis. Complejidad temporal: T(n) = O (K/2 * n) donde K es la suma de los elementos de la secuencia y n la cantidad de elementos de la secuencia.
      Complejidad espacial: S(n) = O (K/2 * n) donde K es la suma de los elementos de la secuencia y n la cantidad de elementos de la secuencia. Y K/2 representa el número de filas de la matriz y n el número de columnas.

Ejercicio 4.9 - Algoritmo log n (Fibonacci)

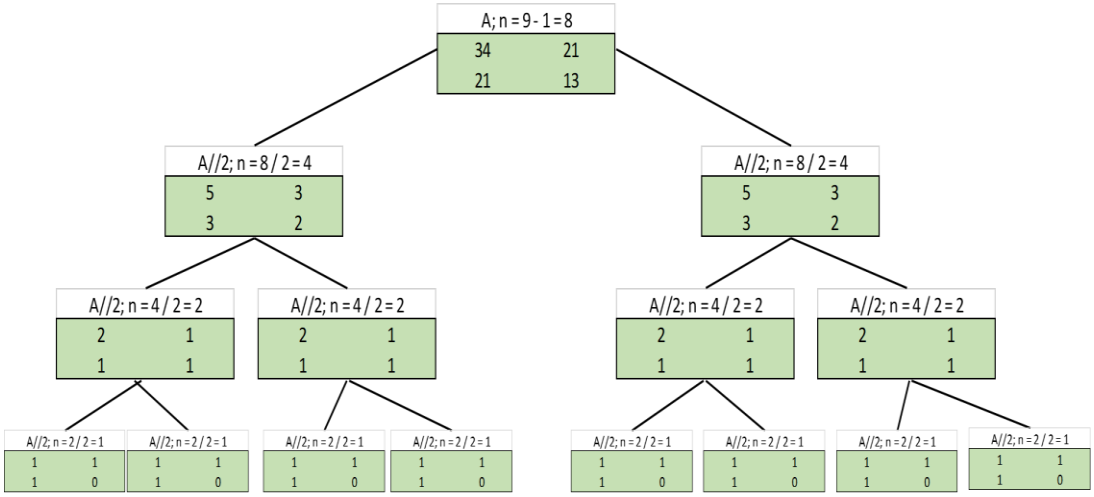
a. Formalización. Dado un numero natural n, hallar el k-esimo número de la secuencia de Fibonacci en O(log n).
b. Definición del cache. El cache en este ejercicio es la matriz A de dimensiones 2x2 ya que esta la matriz que usamos para calcular la potencia n de una matriz A dada y guardamos los productos de dicha operación (x, y, z, w) en la misma matriz A. El numero k de la seria de Fibonacci se encuentra en la primera posición de la matriz A.
c. Definición de recursión.
   Para esta implementación se utiliza una matriz para tratar la recursión simple de la serie de Fibonacci

$$F_n = F_{n-1} + F_{n-2} \; con \; F_0 = 0, F_1 = 1$$

   La representación de esta recursión en una matriz está dada de la siguiente forma

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

d. Diagrama de necesidades.



e. Análisis.

6. Desarrolle y analice tres de los ejercicios del 6.11 al 6.15 de [Das08]

Ejercicio 6.11 - LCS

a. Formalización. Dadas dos secuencias, X = (x_1,…., x_m) y Y= (y_1,…., y_n), encontrar una sub-secuencia común cuya longitud sea máxima.
b. Definición del cache. El cache se define como una matriz donde las columnas se definen como la cantidad de elemento de la secuencia X y las filas se definen como la cantidad de elementos de la secuencia Y
c. Definición de recursión. La definición de la recursión se puede dividir en estos dos casos donde vamos a comparar el elemento de la primera fila con elemento de la primera columna
   - Si ambos son iguales (i == j) Sumamos uno al elemento diagonal izquierdo de arriba y colocamos ese valor en la matriz en la ubicación en la que estamos.

$$Matriz\,[i, j] = 1 + Matriz[i-11][j-1]$$

- Si ambos elementos no son iguales: (i != j) Agregamos el valor máximo entre el elemento superior inmediato o el elemento del lado izquierdo inmediato.

$$Matriz\ [i, j] = Max\ (matriz\ [i\text{-}1, j], matriz[i,j\text{-}1])$$

d. Diagrama de necesidades

		A	B	C	D
	0	0	0	0	0
B	0	0	1	1	1
D	0	0	1	1	2

e. Análisis. Complejidad temporal: T(m,n) = O(m*n) donde m y n denota la longitud de cada cadena
Complejidad espacial: S(m,n) = O(m*n) filas y columnas de la matriz que se utiliza

Ejercicio 6.13 - Problema Cartas/Dealer

a. Formalización. Dada una secuencia S = {S_1, S_2 , …, Sn} de cartas donde cada carta Si tiene un valor Vi. Como parte de la mejor estrategia, determinar el máximo valor de cartas que se pueden obtener por el primer jugador.
b. Definición del cache. El cache es una matriz cuadrada M de dimensiones *nxn* donde n corresponde el número de cartas de la secuencia S.
c. Definición de recursión. La recursión está determinada por dos situaciones en las cuales varia el extremo (S_i, S_{i+1}, ….., S_{j-1}, S_j) de la secuencia se elige para tomar una carta:
   - Si el primer jugador escoge la carta S_i, entonces el segundo jugador debe escoger entre S_{i+1} o S_j por lo que el primer jugador queda con el mínimo entre {M(i+2, j) , M(i+2,j-1)}

$$S_i + min\ (M(i+2, j), M(i+1, j\text{-}1)\ )$$

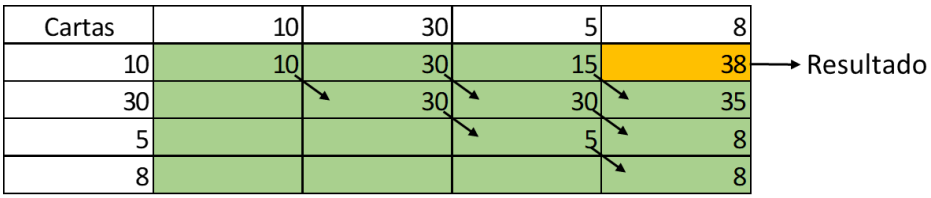
   - Si el primer jugador escoge la carta S_j, entonces el segundo jugador debe escoger entre S_i o S_{j-1} por lo que el primer jugador queda con el mínimo entre {M(i+1, j-1) , M(i,j-2)}

$$S_j + min\ (M(i+1, j\text{-}1), M(i, j\text{-}2)\ )$$

   - Entonces se debe elegir cuál de las dos situaciones otorga el mayor valor para el primer jugador:

$$M(i,j) = max\begin{cases} Si\ +\ min\ (M(i+2,j), M(i+1,j-1)\ ),\ 0 \le i \le j \le n \\ Sj\ +\ min\ (M(i+1,j-1), M(i,j-2)\ ),\ 0 \le i \le j \le n \end{cases}$$

d. Diagrama de necesidades.



e. Análisis. Complejidad temporal: T(n) = O (n^2) donde n es la cantidad de cartas existentes en la baraja (o secuencia) al iniciar la partida.
Complejidad espacial: S(n) = O (n^2). Como ambos jugadores disponen de acceso a la misma baraja, cada jugador debe recorrer la baraja, es decir, la matriz por igual por lo que el cache es nxn y se almacena el mayor valor obtenido por el primer jugador en cada jugada.

Ejercicio 6.14 - Problema Cutting Cloth

a. Formalización. Dada dos valores enteros naturales X y Y, se debe determinar un valor con la máxima suma que puede otorgar dicha matriz (pedazo de tela) con dimensiones *X*Y*.
b. Definición del cache. El cache es una matriz de dimensiones *X*Y* que representa el pedazo de tela donde se van ejecutar los cortes, en este caso X corresponde al ancho de la tela (columnas) que va desde 0 hasta j<=X, y Y corresponde al largo que posee la tela que va desde 0 hasta i<= Y.
c. Definición de recursión.