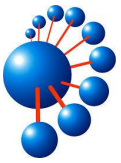


Proyecto 3: Programación dinámica y Aproximados

Análisis de Algoritmos (2022-1)

voy a entregarlo
entregado
hay una foto bug D:
cagamos entonces

Integrantes: Dazhi Enrique Feng Zong
Pablo Ignacio Zapata Schifferli
Profesora: Cecilia Hernández R.
Fecha: 12 de julio, 2022

Needleman-Wunsch

Recurrencia y descripción de subproblemas a resolver

Para resolver el problema con programación dinámica, se necesita una matriz *M*, de tamaño mínimo $n \times m$ o $m \times n$. Luego, los índices de cada fila y columna representan las secuencias Y y X (fila 0 = carácter 0 de Y, fila 1 = carácter 1 de Y, columna 0 = carácter 0 de X, etc.)

Entonces, comenzando por el primer elemento, para cada elemento se le asigna un valor (puntaje). El valor se determina con la siguiente recursión:

$$M(i, j) \;\; = \; max(M(i-1, j-1) + \; S(X_j, Y_i), M(i, j-1) \;\; - 2, M(i-1, j) \;\; - 2)$$

Entonces, el subproblema a resolver es encontrar cuál de las 3 opciones da un puntaje mayor.

1. Para la opción 1: X_j, Y_i son los caracteres que representa esa columna j y fila i. $S(X_j, Y_i)$ Es el puntaje asignado a los 2 caracteres (1 si son iguales y -1 sino). A esto se le debe sumar el valor de la diagonal anterior $M_{i-1,j-1}$, es decir, en la casilla actual se está comparando 2 caracteres sumado al puntaje anterior.
2. Para la opción 2: Se compara un carácter de X, con un gap de Y. Entonces, el puntaje asignado es de -2 + el valor del elemento a la izquierda (puntaje anterior).
3. Para la opción 3: Se compara un carácter de Y, con un gap de X. Entonces, el puntaje asignado es de -2 + el valor del elemento arriba (puntaje anterior).

Los subproblemas se tienen que resolver de esta forma, ya que necesariamente se debe comparar con valores anteriores; esto es para ver si conviene poner gaps antes o después, y ver cual es la mejor opción. Sabemos que la mejor solución para cada carácter individual de las 2 secuencias es siempre alinear 2 caracteres iguales, pero para las 2 secuencias completas, hay que ver donde colocar los gaps y donde hacer match o mismatch, para maximizar el puntaje final de toda la secuencia.

Terminando de resolver los subproblemas, el último elemento es el puntaje óptimo del problema.

Solución top-down con memoization

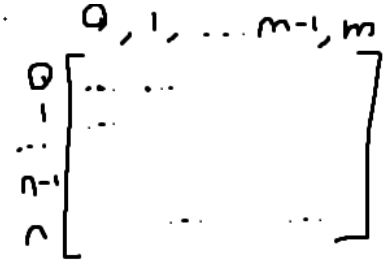
Pseudocódigo:

```
top_down(i, j, M, X, Y, B):
      if Mij == 'inf':
            if i == 0 or j ==0:  Mij = -2*(i+j)
            else:
                  if X[i-1] == Y[j-1]: r = 1
                  else: r = -1
                  m1 = top_down(i-1, j   , M, X, Y, B) - 2
                  m2 = top_down(i   , j-1, M, X, Y, B) - 2
                  m3 = top_down(i-1, j-1, M, X, Y, B) + r

                  M[i][j] = max(m1, m2, m3)
                  if M[i][j] == m1: B[i][j] = (i-1, j)
                  else if M[i][j] == m2: B[i][j] = (i, j-1)
```

```
                else: B[i][j] = (i-1, j-1)
    return Mij

solución():
    X = [0, … , n-1]      secuencia
    Y = [0, … , m-1]      secuencia
    i = n, j = m
    M = matriz n*m llena de 'inf'
    B = matriz n*m llena de (0, 0)

    top_down(i, j, M, X, Y, B)
    secuencias a, b
    while i or j:
        if B[i][j].first == i: a.push_principio('-')
        else: a.push_principio(X[i-1])
        if B[i][j].second == j: b.push_principio('-')
        else: b.push_principio(Y[j-1])
        i = B[i][j].first
        j = B[i][j].second

    retornar a, b, (Mn,m)
```

Explicación en palabras:

1. Sean las secuencias: X = [0, … , n-1], Y = [0, … , m-1].
2. Crear matrices M y B de tamaño (n+1)*(m+1). M llena de 'inf', B llena de pares (0, 0)



3. Mi,j = max((Mi-1,j)-2, (Mi,j-1)-2, (Mi-1,j-1)+r) con r = 1 si Y[i-1] == X[j-1]. r = -1 en caso contrario
4. Usar el paso 3 con Mn,m. Si algún valor no se ha calculado ('inf'), usar el paso 3 para calcularlo. Si i = 0 o j = 0, el valor es -2(i+j). Asignar a Bi,j la posición del valor máximo elegido.
5. Crear las secuencias A y B
6. Recorrer desde Bn,m hasta llegar a B1,1 usando las posiciones que apuntan *. Si en el valor que apunta hay cambio en 'i' o 'j' ** entonces agregar al principio de A, X[i-1] y en B, Y[j-1] respectivamente. En caso contrario agregar al principio '-'.
   * Ej: Si B2,2 = (1,2), entonces el siguiente valor a usar es B1,2
   ** Ej: En B2,2 = (1,2) hay cambio en 'i' pero no en 'j'
7. Retornar A, B y (Mn,m.)

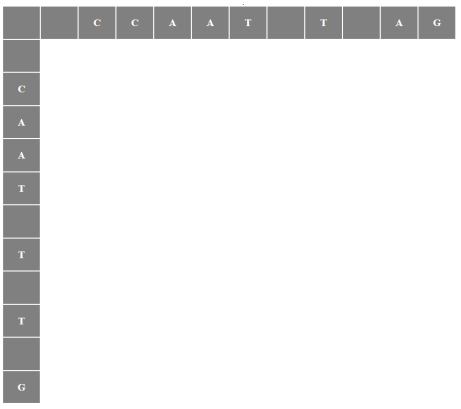
Solución bottom-up con tabulación
Pseudocódigo:

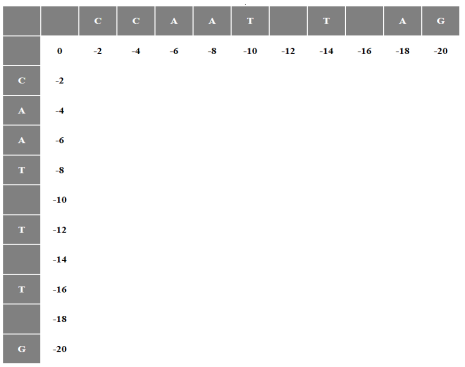
```
S(j ,i)
if character j = character i return 1
else return -1

Mijrecursion(M, i, j)
return max(M[i-1][j-1]+ S(j, i), M[i][j-1] - 2, M[i-1][j] - 2) and
[diagonal, up, left] arrow(s)
// can have more than 1 max -> return more than 1 arrow

needleman_wunsch(Y, X)
// inicialización
n, m ← length of X, Y
M ← matrix (n+1)*(m+1)
map sequence X[1, 2, 3,..., n] with columns 1, 2, 3,..., n
map sequence Y[1, 2, 3,..., m] with rows 1, 2, 3,..., m
M[0][0] ← 0
for j = 1 to n do
     M[i][j] ← Mijrecursion(M, 0, j)
for i = 1 to m do
     M[i][j] ← Mijrecursion(M, i, 0)
// aplicar recursión a cada elemento
for i = 1 to m do
     for j = 1 to n do
          M[i][j] ← Mijrecursion(M, i, j)
// traceback
max ← M[m][n]
Sequence X, Y ← traceback(m, n)
return max, X, Y
```

Paso a paso con ejemplos:
Secuencias utilizadas de ejemplo:

- Primera: **Y = {CAAT T T G}**
- Segunda: **X = {CCAAT T AG}**

1. Crear la matriz M, con tamaño (n+1)*(m+1), donde cada índice de las filas representa los caracteres de la secuencia Y, y los de las columnas los caracteres de X. Además, las filas y columnas 0 se reservarán para el carácter “-”.
2. Para cada elemento de la matriz M, Mij, elegir el máximo valor de:
    a. Mi-1,j-1 (diagonal) + S(Xj, Yi). No hay - en ninguna secuencia.
    b. Mi-1,j (arriba) + -2. Hay - en la segunda secuencia, X.
    c. Mi,j-1 (izquierda) + -2. Hay - en la primera secuencia, Y.

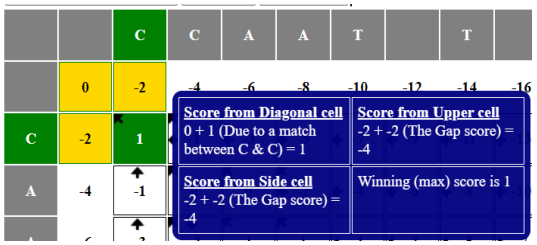




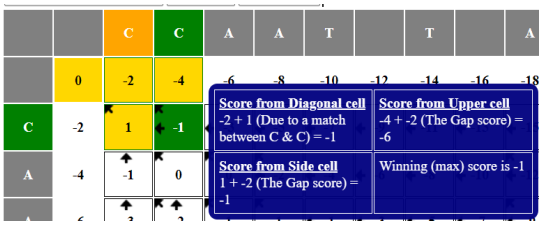
Se debe inicializar la matriz. Agregando un 0 en la casilla 0,0 y luego, para todas las filas y columnas con índice 0, se debe inicializar siguiendo la recursión mencionada.

3. Para cada elemento de la matriz M, Mij, elegir el máximo valor de:
    a. Mi-1,j-1 (diagonal) + S(Xj, Yi). No hay - en ninguna secuencia.
    b. Mi-1,j (arriba) + -2. Hay - en la segunda secuencia, X.
    c. Mi,j-1 (izquierda) + -2. Hay - en la primera secuencia, Y.

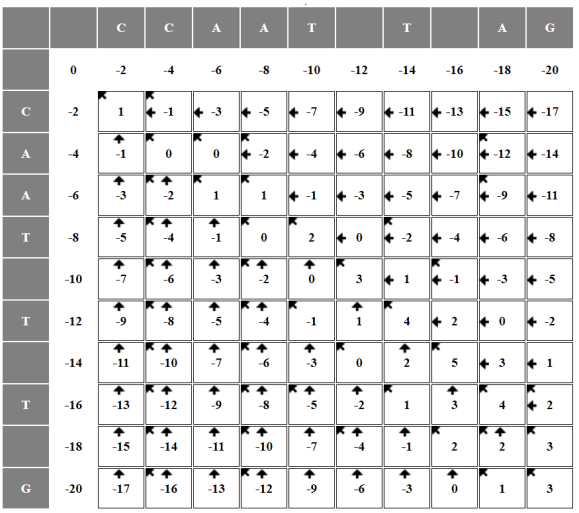
Luego de inicializar la matriz en el paso 2, se puede comenzar con los caracteres “reales”.



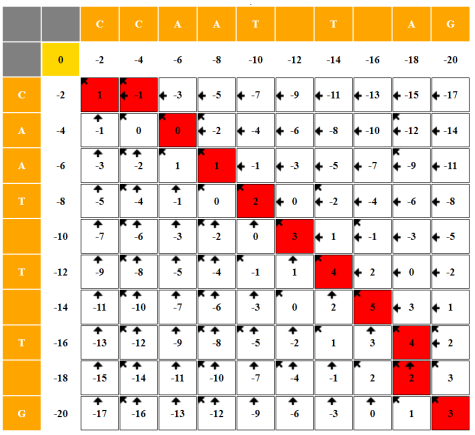
Por ejemplo para M[1][1], el mayor valor ocurre en el primer caso, donde los caracteres de la fila y columna son iguales, obteniendo un puntaje de 1 + el puntaje de Mi-1,j-1 (diagonal), 0.



Para M[1][2], el mayor valor ocurre en el primer y tercer caso, 1 (coinciden) + -2 (diagonal) = -1 y 1 (izquierda) + -2 (por el -) = -1.



4. Almacenar en la casilla de la matriz el mayor entre las 3 opciones y la(s) flecha(s) que indique cuál opción se tomó.
5. Repetir 3 y 4, con todos los elementos de la matriz, empezando por los elementos que tengan vecinos (diagonal, izquierda, arriba) con algún valor ya almacenado.
6. El puntaje es el valor del último elemento de la matriz.
7. Para el traceback, elegir el último elemento y desde ahí seguir el recorrido de las flechas, escogiendo siempre la casilla con mayor valor en caso de haya más de una opción.
8. Para alinear las secuencias , empezando por el primer elemento,
    a. Si una casilla es diagonal, colocar los caracteres de Mij.
    b. Si es arriba, colocar - en la segunda secuencia (X) y Mj en la primera (Y),
    ya que Mi ya se ocupó, mientras que Mj no.
    c. Si es izquierda, colocar Mi en la segunda secuencia (X) y - en la primera (Y),
    ya que Mj ya se ocupó, mientras que Mi no.



Por lo tanto, el ejemplo queda para **X = {CCAAT T A-G}**, tiene un - debido a que hay una flecha arriba.

Para la otra secuencia, **Y = {C−AAT T T G}** , tiene un - debido a que hay una flecha izquierda.

Complejidad algorítmica en tiempo y espacio

Tiempo: $O(mn)$

Se debe recorrer una matriz $m \times n$, donde en cada casilla se hacen asignaciones, comparaciones y sumas ($O(1)$).

Espacio: $O(mn)$

Se crea una matriz de enteros de $m \times n$ o $(m + 1) \times (n + 1)$.

**$(m + 1) \times (n + 1) = mn + m + n + 1 = O(mn)$

Minimo Vertex Cover

Ejemplo donde el algoritmo aproximado obtiene el valor óptimo y uno donde obtiene el doble del óptimo

En verde las artistas retornadas por el algoritmo

Ejemplo para valor óptimo:

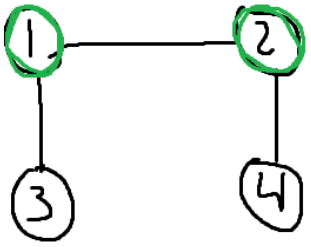
input:

4 3

1 2

1 3

2 4



Solución que entrega: 1 2

Ejemplo para valor doble del óptimo:

input:

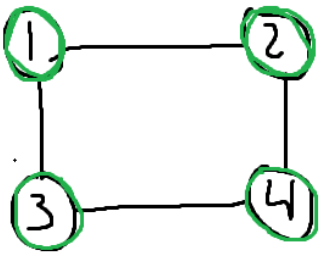
4 4

1 2

2 3

3 4

4 1



Solución que entrega: 1 2 3 4

Problema 2b

1. Agregar todos los cliques maximales a un set ‘s’.
2. Cada vértice del grafo tiene un contador ‘a’ que indica la cantidad de cliques maximales a los que pertenece.
3. Si todos los vértices de un clique ‘c’ tienen su contador ‘a’ mayor que uno, el clique ‘c’ se elimina del set ‘s’. Por cada vértice perteneciente al clique ‘c’, se reduce su contador ‘a’ en uno.
4. Repetir el paso 3 hasta que no hayan cliques que se puedan eliminar.
5. Retornar como solución el set ‘s’