Proyecto ADA

Cesar Madera[1], Enrique Sobrados[2], Johan Tanta[3]

[1−3]Ciencia de la Computación
Universidad de Ingeniería y Tecnología, Barranco
{[1]cesar.madera,[2]enrique.sobrados,[3]johan.tanta}@utec.edu.pe

Junio 16, 2020

Índice

1. Algoritmo Voraz

1.1. Eleccion voraz

Sea i, j punteros en los vectores de los bloques A y B respectivamente que empiezan desde el inicio. **Eleccion Voraz**:

- Empezar a realizar un subconjunto con índice i, un $i - división$ con la condición: $B_j > B_{j+1}$
- Al no cumplir la anterior condición, empezar a realizar un subconjunto con índice j, un $j - agrupamiento$ con la condición: $A_i < A_{i+1}$

1.2. Pseudocódigo

Algorithm 1 Devuelve el Peso

```
procedure WEIGHT(matchings)
    suma_total = 0
    tempa = A[matchings[1].first].longitud
    tempb = B[matchings[1].second].longitud
    t_a = matchings[0].first
    t_b = matchings[0].second
    for i = 2 to size(matchings) do
        if matchings[i].first == t_a then
            tempb += B[matchings[i].second].longitud
        else if matchings[i].second == t_b then
            tempa += A[matchings[i].first].longitud
        else
            suma_total += tempa/tempb
            t_a = matchings[i].first
            t_b = matchings[i].second
            tempa = A[matchings[i].first].longitud
            tempb = B[matchings[i].second].longitud
    suma_total += tempa/tempb
    return suma_total
```

Algorithm 2 Devuelve un Match entre A y B

```
procedure GREEDY_MIN(A, B)
    i = 1
    j = 1
    cont_B = 0
    cont_A = 0
    while  i < size(A) -1 and j < size(B) - 1 do
        if B[j + 1].longitud < B[j].longitud then
            if cont_B then
                j++
                i++
                cont_B = 0
            else
                matchings.push(i,j)
                j++
                cont_A++
        else if A[i].longitud < A[i + 1].longitud then
            matchings.push(i,j)
            if cont_A then
                i++
                j++
                cont_A = 0
                cont_B = 0
            else
                i++
                cont_B = 0
        else
            matchings.push(i, j));
            i++
            j++
            cont_B = 0
    if i == size(A) - 1 then
        while j < size(B) do
            matchings.push(i, j)
            j++
    else
        while i < size(A) do
            matchings.push(i, j)
            i++
    return matchings, Weight(matchings)
```

1.3. Tiempo de ejecucion

El tiempo de ejecución para este algoritmo en el peor de los casos es: La linea 6 se ejecuta m+n-1 veces sin contar las constantes y en la linea 31 se ejecutaa ese 1 faltante. La funcion Weight tiene un tiempo de ejecucion de max{m.n} ya que va a iterar en el maximo número de tuplas de matchings el cual es max{m,n}. Por lo tanto el tiempo de ejecucion del algoritmo: $T(m, n) = m+n+max\{m, n\}$

1.4. Demostración

Demostrar: $T(m, n) = O(max\{m, n\})$

$$\begin{aligned} m+n &\leq 2 \times max\{m, n\} \\ m+n+max\{m, n\} &\leq 2 \times max\{m, n\} + max\{m, n\} \\ m+n+max\{m, n\} &\leq 3 \times max\{m, n\}, C_1 = 3 \\ T(m, n) &= O(max\{m, n\}) \end{aligned}$$

2. Opt(i, j)

2.1. Solución óptima

Sea X la solución óptima del problema. Asimismo, sea P y Q subconjuntos.

- Donde P y el índice j, es un $j-agrupamiento$ definido:

$$P = (k, j), (k+1, j), \ldots, (i-1, j), (i, j) \qquad 2 \leq k \leq i$$

- Donde Q y el índice i, es una $i-division$ definida:

$$Q = (i, l), (i, l+1), \ldots, (i, j-1), (i, j) \qquad 2 \leq l \leq j$$

X debe incluir una solución óptima entre los subconjuntos P y Q, debido a ello se observan los siguientes escenarios:

- Si $P \in X$: Luego X debe incluir una solución óptima del subproblema que está dado por los bloques de $A = \{A_1, A_2, \cdots, A_{k-1}\}$ y de $B = \{B_1, B_2, \cdots, B_{j-1}\}$.
- Si $Q \in X$: Luego X debe incluir una solución óptima del subproblema que está dado por los bloques de $A = \{A_1, A_2, \cdots, A_{i-1}\}$ y de $B = \{B_1, B_2, \cdots, B_{l-1}\}$.

2.2. Planteamiento de la Recurrencia

Para calcular el peso asociado a una agrupación es calculada por:

$$W_A(r,s,j) \quad = \quad \frac{A_r + A_{r+1} + \cdots + A_s}{B_j}$$

Para calcular el peso asociado a una división es calculada por:

$$W_D(i,m,n) \quad = \quad \frac{A_i}{B_m + B_{m+1} + \cdots + B_n}$$

Asimismo, para cada (i,j) se define:

$$\begin{aligned} M_A(i,j) &= min_{k=i}^{2}\,(W_A(k,i,j) + Opt(k-1,j-1)) \\ M_D(i,j) &= min_{l=j}^{2}\,(W_D(i,l,j) + Opt(i-1,l-1)) \end{aligned}$$

Se plantea la siguiente recurrencia

$$Opt(i,j) = \begin{cases} W_A(1,i,1) & \text{j == 1} \\ W_D(1,1,j) & \text{i == 1} \\ min(M_A(i,j), M_D(i,j)) & \text{Caso contrario} \end{cases}$$

3. Algoritmo Recursivo

3.1. Pseudocódigo

Se definen las siguientes funciones:

Algorithm 3 Devuelve las tuplas y el peso de un i-division

```
procedure MatchDivision(i, m, n)
    Tuplas MatchD
    for p = m to n do
        MatchD.push( i , p )
    return MatchD
```

Algorithm 4 Devuelve las tuplas y el peso de un j-agrupamiento

```
procedure MatchGroup(r, s, j)
    Tuplas MatchG
    for p = r to s do
        MatchG.push( p , j )
    return MatchG
```

Luego se define la función Opt(i,j):

Algorithm 5 Devuelve el min matching

```
1: procedure Opt(i, j)
2:     if i == 1 then
3:         return MatchDivision(i, 0, j)
4:     else if j == 1 then
5:         return MatchGroup(0, i, j)
6:     else
7:         Tuplas min_resultk.weight = ∞
8:         Tuplas min_resultl.weight = ∞
9:         for k = i down to 2 do
10:            Match = MATCHGROUP( k, i, j )
11:            SubProblem = Opt( k − 1, j − 1 )
12:            result = SubProblem + Match
13:            if min_resultk.weight > result.weight then
14:                min_resultk = result
15:        for l = j down to 2 do
16:            Match = MATCHDIVISION( i, l, j )
17:            SubProblem = Opt( i − 1, l − 1 )
18:            result = SubProblem + Match
19:            if min_resultl.weight > result.weight then
20:                min_resultl = result
21:        return min( min_resultl, min_resultk )
```

3.2. Tiempo de Ejecución

Al analizar el tiempo de ejecución del Algorithm 5, se obtiene lo siguiente:

$$Opt(i,j) = \begin{cases} c & \text{i == 1} \vee \text{j == 1} \\ T(i,j) = \sum_{k=i}^{2} T(k-1, j-1) + \sum_{k=j}^{2} T(i-1, k-1) & \text{Caso contrario} \end{cases}$$

Probaremos por induccion que $T(i,j) = \Omega(2^{max(i,j)})$, con $c = \frac{1}{2}$:

- Como caso base, donde i = 1, j = 1 y $c_1 = 1$, se ejecuta la línea 2 y 3.

$$\begin{aligned} T(i,j) &\geq c_1 2^{max(1,1)} \\ T(i,j) &\geq 2^1 \end{aligned}$$

- Paso inductivo:

$$\Omega(2^{max(i,j)}) = T(i,j) \qquad 1 \leq i \leq m-1 \quad \wedge \quad 1 \leq j \leq n-1$$

Se sabe que:

$$
\begin{aligned}
T(m,n) &= \sum_{k=m}^{2} T(k-1,n-1) + \sum_{k=n}^{2} T(m-1,k-1) \\
T(m,n) &= \sum_{k=m}^{2} T(k-1,n-1) + \sum_{k=n}^{2} T(m-1,k-1) \geq \sum_{k=m}^{2} T(k-1,n-1) \\
T(m,n) &\geq \sum_{k=m}^{2} T(k-1,n-1) \\
T(m,n) &\geq T(m-1,n-1) \\
T(m,n) &\geq 2^{max(m-1,n-1)} \\
T(m,n) &\geq 2^{max(m,n)-1} \\
T(m,n) &\geq \frac{1}{2}2^{max(m,n)}
\end{aligned}
$$

Se concluye que $T(m,n) = \Omega(2^{max(m,n)})$

$$
T(m,n) \geq \frac{1}{2}2^{max(m,n)} \qquad \text{m} \geq 1 \wedge \text{n} \geq 1
$$

4. Algoritmo Memoizado

4.1. Pseudocódigo

Se reutiliza las funciones MatchGroup y MatchDivision definidos en la anterior sección. Antes de implementar el algoritmo, se inicializa toda la matriz en cero.

Algorithm 6 Devuelve el min matching utilizando una matriz como apoyo

```
1: procedure Opt(i, j)
2:     if Matrix[i][j] != 0 then
3:         return Matrix[i][j]
4:     if i == 1 then
5:         Matrix[i][j] = MatchDivision(i, 0, j)
6:         return Matrix[i][j]
7:     else if j == 1 then
8:         Matrix[i][j] = MatchGroup(0, i, j)
9:         return Matrix[i][j]
10:    else
11:        Tuplas min_resultk.weight = ∞
12:        Tuplas min_resultl.weight = ∞
13:        for k = i down to 2 do
14:            Match = MATCHGROUP( k, i, j )
15:            SubProblem = Opt( k − 1, j − 1 )
16:            result = SubProblem + Match
17:            if min_resultk.weight > result.weight then
18:                min_resultk = result
19:        for l = j down to 2 do
20:            Match = MATCHDIVISION( i, l, j )
21:            SubProblem = Opt( i − 1, l − 1 )
22:            result = SubProblem + Match
23:            if min_resultl.weight > result.weight then
24:                min_resultl = result
25:        Matrix[i][j] = min( min_resultl, min_resultk )
26:        return Matrix[i][j]
27:
```

4.2. Tiempo de ejecución

El tiempo de ejecucion de este algoritmo está dado por:

$$T(i,j) = (\ \#\ \text{SubProblemas}\ ) * (\text{Tiempo por SubProblema}) \tag{1}$$

Se esta tomando en consideracion que las funciones MatchDivision y MatchGroup se ejecutan en tiempo constante, por lo que solo se necesita contabilizar el numero de subproblemas existentes[1]. Se replantea la ecuación (1):

$$T(i,j) = (\ \#\ \text{SubProblemas}\ ) * c \tag{2}$$

Asimismo, debido a las llamadas recursivas de los subproblemas $T(k-1, j-1)$ y $T(i-1, l-1)$ en las lineas 15 y 21 del algoritmo memoizado la cantidad de subproblemas que se resuelven son:

$$(\text{\# SubProblemas}) = (i-1)*(j-1) + \underbrace{1}_{\text{Problema original: } T(i,j)}$$

Entonces, para $T(m,n)$ se obtiene:

$$\begin{aligned} T(m,n) &= (m-1)*(n-1)+1 \\ m*n &\geq (m-1)*(n-1)+1 = T(m,n) \\ m*n &\geq T(m,n) \end{aligned}$$

Por lo que se demuestra que $T(m,n) = O(mn)$

5. Algoritmo Dinámico

5.1. Pseudocódigo

En primer lugar, se define la función OPT_Result(i,j).

Algorithm 7 Devuelve el min matching utilizando una matriz como apoyo

```
procedure OPT_Result(i, j)
    if i == 1 then
        Matrix[i][j] = GetMatchDivision(i, 1, j)
    else if j == 1 then
        Matrix[i][j] = GetMatchGroup(1, i, j)
    else
        min_resultk = math.inf
        min_resultl = math.inf
        indexMinGroup = 1
        indexMinDivision = 1
        for k = i down to 1 do
            Match = GetMatchGroup(k, i, j)
            SubProblem = Matrix[k-1][j-1]
            result = SubProblem + Match
            if min_resultk > result then
                min_resultk = result
                indexMinGroup = k
        for l = j down to 1 do
            Match = GetMatchDivision(i, l, j)
            SubProblem = Matrix[i-1][l-1]
            result = SubProblem + Match
            if min_resultl > result then
                min_resultl = result
                indexMinDivision = l
        if min_resultl > min_resultk then
            Matrix[i][j] = min_resultk
            minSubProblem[i, j] = (indexMinGroup-1, j-1)
        else
            Matrix[i][j] = min_resultl
            minSubProblem[i, j] = (i-1, indexMinDivision-1)
```

Finalmente, se diseña el algoritmo de programacion dinámica.

Algorithm 8 Devuelve min matchings usando DP

```
procedure DynamicProgramming(x, y)
    for i = 1 to x do
        for j = 1 to y do
            OPT_Result(i, j)
    OPT_Result(x, y)
    return Matrix[x][y]
```

5.2. Tiempo de Ejecución

Para demostrar la complejidad del algoritmo, se observa lo siguiente:

- En la función OPT_RESULT(m, n), se observan dos bucles independientes:

$$\begin{aligned} T(m,n) &= m+n \\ T(m,n) &\leq max\{m,n\} \\ O(max\{m,n\}) &= T(m,n) \end{aligned}$$

- En la función DynamicProgramming(x, y), se observan dos bucles anidados llamando a la función OPT_RESULT(m, n), por lo que:

$$\begin{aligned} T(m,n) &= (m-1)(n-1)O(max\{m,n\}) \\ T(m,n) &\leq (m)(n)O(max\{m,n\}) \\ T(m,n) &\leq max\{m,n\}^2 O(max\{m,n\}) \\ T(m,n) &\leq max\{m,n\}^3 \\ O(max\{m,n\}^3) &= T(m,n) \end{aligned}$$

Por lo tanto el tiempo de ejecución del algoritmo de programación dinámica es $O(max\{m,n\}^3)$

6. Algoritmo Dinamico Mejorado

6.1. Pseudocódigo

Se implementa la siguiente función para calcular el u

Algorithm 9 Inicializar la constante u

```
procedure InicializarU
    u = sumaBloquesA[len(A)-1]/sumaBloquesB[len(B)-1]
```

Las listas **sumaBloquesA** y **sumaBloquesB** obtienes las sumas acumuladas de los pesos de los bloques de **A** y **B**.

Luego, se realizan los siguientes cambios en las funciones de GetMatchDivision(i, m, n) y GetMatchGroup(r, s, j). Estas funciones devuelven los pesos de los matchs.

Algorithm 10 Obtener el peso de un match de división

```
procedure GETMATCHDIVISION(i, m, n)
    if m == n then
        return abs(A[i].longitud/B[m].longitud - u)
    if m == 0 then
        return abs(A[i].longitud/sumaBloquesB[n] - u)
    suma = sumaBloquesB[n] - sumaBloquesB[m-1]
    return abs(A[i].longitud/suma - u)
```

Algorithm 11 Obtener el peso de un match de agrupacion

```
procedure GETMATCHGROUP(r, s, j)
    if r == s then
        return abs(A[r].longitud/B[j].longitud - u)
    if r == 0 then
        return abs(sumaBloquesA[s]/B[j].longitud - u)
    suma = sumaBloquesA[s] - sumaBloquesA[r-1]
    return abs(suma/B[j].longitud - u)
```

Se realizan esos cambios para el correcto funcionamiento del algoritmo de programación dinámica mejorada.

6.2. Tiempo de Ejecución

Como los cambios realizados no influyen en la notación de O-grande, el tiempo de ejecución es el mismo que el algoritmo de programación dinámica.

$$O(max\{m,n\}^3) \quad = \quad T(m,n)$$

7. Algoritmos de Transformaciones de Matrices

Para la transformación de matrices se desarrollo un algoritmo general para los tres métodos de transformación de matrices (Greedy , Dinamica, Dinamica-Mejorada). Dentro de este algoritmo se encuentra la función MIN_MATCHING representa los algoritmos anteriormente mencionados, que podran ser greedy, dinamica y dinamica mejorada.

7.1. Pseudocódigo

Algorithm 12 Devuelve un conjunto de matches

```
procedure TRANSFORMACION_MIN(matrixA, matrixB, GetSubmtachings = False)
    if GetSubmatching then
        for i = 1 to size(matrixA) do
            result = MIN_MATCHING(matrixA[i] , matrixB[i] , GetSubmatching)
            MatrixMatchings.insert(result)
    else
        sumatoria = 0.0
        for i = 1 to size(matrixA) do
            restult = Greedy.MIN_MATCHING(matrixA[i],matrixB[i])
            sumatoria = sumatoria + result
            MatrixMatchings.insert(matchings)
        return sumatoria
```

7.2. Tiempo de Ejecucion

A cada algoritmo descrito anteriormente se debe agregar el número de filas

- **Transformacion Greedy**, dos matrices A y B de ceros y unos de tamaño p x q.

$$O(pq) \quad = \quad Tr(m,n)$$

- **Transformacion Programcion Dinamica**, dos matrices A y B de ceros y unos de tamaño p x q.

$$O(pq^3) \quad = \quad Tr(m,n)$$

- **Transformacion Programcion Dinamica Mejorada**, dos matrices A y B de ceros y unos de tamaño p x q.

$$O(pq^3) \quad = \quad Tr(m,n)$$

8. Procesamiento de imagenes

Para esta sección del proyecto que fue realizado en python, se utilizó la libreria Pillow, para facilitar la manipulación de imagenes.

8.1. Escala de Grises

Para convertir la matriz de pixeles RGB a Escala de grises, se han implementado 4 funciones, LUM_601, LUM_709,LUM_240 y LUM_imput. Este ultimo permite setear los factores de conversion a los que el usuario le pasa. Todas las anteriores funciones llaman a la siguiente subrutina convert:

```
def convert(image, R, G, B):
    ConvertedMatrix01 = []

    for y in range(height):
        array01 = []
        for x in range(width):

            RGB = image.getpixel((x,y))
            Gris = int(RGB[0]*R + RGB[1]*G + RGB[2]*B)
            if(Gris > 122):
                array01.append(1)
            else:
                array01.append(0)
        ConvertedMatrix01.append(array01)
    return ConvertedMatrix01
```

El umbral utilizado para generar los arreglso de bloques ha sido 122. El cual si el Valor de gris es menor que ese valor se agregará como 1, caso contrario, será 0.

9. Animación de Imagenes

9.1. Terminos importantes

Para la animacion de imagenes, estamos identificando 3 conjuntos de pixeles:

- Submatchings: relacion entre un bloque a varios (division o agrupacion)
- Antimatchings: Todos los bloques de 0's a los lados de los Submatchings, los cuales no son afectados por estos mismos
- pixelesLibres: Todos los pixeles que no sean afectados por los dos anteriores(podria ser que existan en el medio de dos bloques divididos o que el matching este justo al final, por lo que los bloques superiores o inferiores deberian desaparecer o aparecer en la transformacion.

9.2. Pseudocodigo

Pasando con el algoritmo:

Algorithm 13 Genera una lista de K imagenes intermedias

```
procedure GenerarImagenesIntermedias(MatrixMatchings,img1,img2,directorio):
    MEGA_MATRIX = []
    for i = NUM_IMG+1 down to 1: do
        listaVacia =[]
        Mega_Matrix.append(listaVacia)
    for i in range(len(MatrixMatchings) ) do
        row11 = GetRow(img1,i
        row12 = GetRow(img2,i
        matchings = MatrixMatchings[i]
        antimatchigs = us.GetAntiMatching(matchings)
        for submatching in matchings: do
            submatching.getProporcionalidad()
        for submatching in antimatchings: do
            submatching.getProporcionalidad()
        matrix = us.generarMatrizPorlinea(matchings, antimatchings,row11,row12
        for j in range(len(MEGA_MATRIX)): do
            MEGA_MATRIX[j].append(matrix[j])
        for  i in range(NUM_IMG+1): do
             pil.ArmarImagen(img2,MEGA_MATRIX,i,directorio)
```

La funcion us.generarMatrizPorLinea, recibe los matchings y antimatchings, y las filas, para poder ejecutar la logica de la animacion y devuelve una matriz de NUM_IMG+1 * ancho de la imagen.

Por otra parte, la función ArmarImagen recibe todas las matrices anteriormente generadas y distribuye los pixeles para poder generar las N

10. Repositorio

Enlace al repositorio git: https://github.com/cesar214567/ProyectoADA

11. Bibliography

1. Demaine,Erik.(2011).Lecture 19: Dynamic Programming I: Fibonacci, Shortest Paths.Dynamic Programing. United States. MIT