Proyecto ADA

Mauricio Pinto, Jonathan Prieto, Alejandro Mamani

June 2020

1 Pregunta 1 (Voraz)

1.1 Diseño

Para obtener un matching (no necesariamente optimo), seguiremos el siguiente procedimiento:
Sea A_i un bloque de $A = A_1, A_2, ...A_n$ y B_j un bloque de $B = B_1, B_2, ..., B_m$:

- Se hará un matching del bloque A_i con B_j empezando con $i = 0$ y $j = 0$, e iremos incrementando a i y j en cada iteración hasta que lleguen a su máximo (el máximo de i es n y el máximo de j es m). Entonces, nuestro algoritmo tendría un tiempo de ejecución $O(max\{m, n\})$.

Como ejemplo, pongamos a los siguientes bloques:

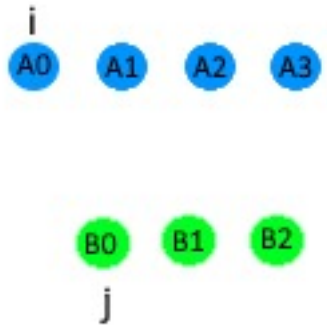
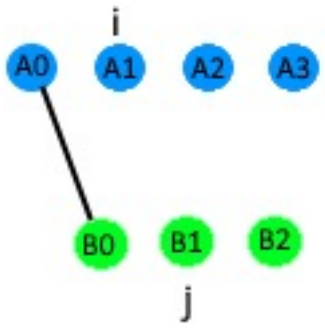
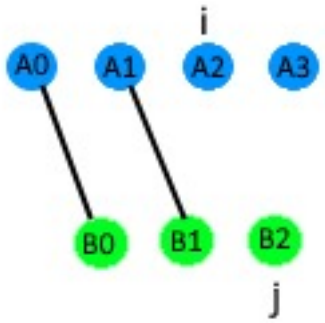


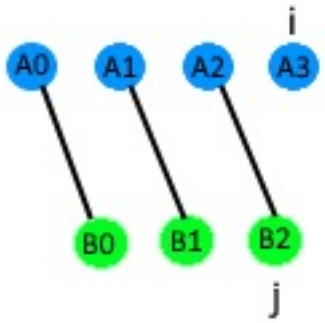
Figure 1: El matching se hará entre A_i y B_j, y se incrementarán sus valores.



(a) Paso 1: Matching e incrementación de los índices.



(b) Paso 2: Matching e incrementación de los índices.



(c) Paso 3: Matching e incrementación de los índices.

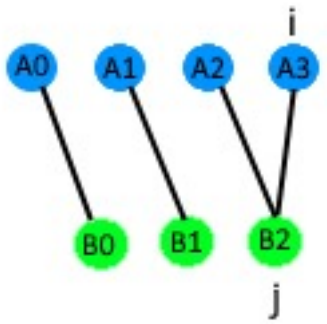


Figure 3: Paso 4: notar que ya que j alcanzó su valor máximo, no se incrementa. Se procede a hacer el matching de igual manera.

Este procedimiento nos dejaría con el siguiente matching:

$$M = (i_0, j_0)(i_1, j_1)(i_2, j_2)(i_3, j_2)$$

1.2 Implementación

El pseudocodigo para nuestro algoritmo Greedy es el siguiente:

$result \leftarrow 0$
$blocks\_A \leftarrow get\_blocks(A)$
$blocks\_B \leftarrow get\_blocks(B)$
$i \leftarrow 0$
$j \leftarrow 0$
$max \leftarrow 0$
$n \leftarrow blocks\_A.size()$
$m \leftarrow blocks\_B.size()$
if n > m **then**
  $max \leftarrow m$
  $m \leftarrow m - 1$
else
  $max \leftarrow m$
  $n \leftarrow n - 1$
end if
$match \leftarrow --$
$weight \leftarrow 0$
while i < max and j < max **do**
  $match.i \leftarrow blocks\_A[i].index$
  $match.j \leftarrow blocks\_A[j].index$
  $weight \leftarrow weight + ((blocks\_A[i].j - blocks\_A[i].i + 1)/(blocks\_B[j].i + 1)$
  $result.matching[-] \leftarrow match$
  if i < n **then**
    $i \leftarrow i + 1$
  end if
  if j < m **then**
    $j \leftarrow j + 1$
  end if
end while
$result.weight \leftarrow weight$
return result

La notación [-], representa un vector y [-]<- representa un push en un vector.

1.3 Análisis

Para el análisis, omitiremos el tiempo de ejecución de los procedimientos *get_blocks*. Ya que todas las líneas previas y posteriores a la declaración while se ejecutan sólamente una vez, aportan un tiempo constante al tiempo total de ejecución. Sumaremos sus tiempos constantes en una constante c. Luego se itera incrementando a i y j, hasta que alguno alcance el valor de max. Entonces, la línea

que evalúa la condición del while aportaría al tiempo total de ejecución con:

$$T_w = c_w max$$

donde c_w representa al tiempo constante de ejecución de la instrucción. Luego, todas las k líneas dentro de la declaración while se ejecutan $max - 1$ veces, por lo que aportan al tiempo total con:

$$T_k = \sum_{i=1}^{k} c_i(max - 1)$$

donde c_i representa la constante de tiempo que corresponde a la línea i entre las k líneas dentro de la expresión while. Por tanto, tenemos que el tiempo total de ejecución está dado por:

$$T_{total} = c_w max + \sum_{i=1}^{k} c_i(max - 1) + c$$

lo cual puede ser reducido a:

$$T_{total} = c_1 max + d$$

donde c_1 es la suma entre c_w y $\sum_{i=1}^{k} c_i$, y d es la suma entre c y $-\sum_{i=1}^{k} c_i$. Podemos asegurar que existen dos valores max_0 y c_2 tal que para todo valor de $max \geq max_0$ se cumpla que:

$$c_1 max + d \leq c_2 max$$

ya que si tomamos, por ejemplo, el caso $c_2 = 2c_1$, eventualmente $2c_1 max$ será mayor o igual que $c_1 max + d$ cuando $d \leq c_2 max$, ya que d es constante. Por tanto, podemos decir que:

$$T_{total} = O(max) = O(max((m, n))$$

2 Pregunta 2 (Recurrencia)

2.1

$$OPT(i,j) = \begin{cases} \frac{A}{B} \rightarrow \text{cuando i = j = 1} \\ \\ \frac{A_1}{\sum_{i=1}^{n} B_i} \rightarrow \text{cuando i = 1 y j > 1} \\ \\ \frac{\sum_{i=1}^{n} A_i}{B_1} \rightarrow \text{cuando j = 1 y i > 1} \\ \\ min(M_A(i,j), M_B(i,j)) \rightarrow \text{otros casos} \end{cases}$$

$M_A(i,j) \leftarrow min_{k=i-1}^{1}(OPT(K, j-1) + \frac{\sum_{k+1}^{i} A_i}{B_j})$

$M_B(i,j) \leftarrow min_{k=j-1}^{1}(OPT(i-1, K) + \frac{A_i}{\sum_{k+1}^{j} B_j})$

3 Pregunta 3 (Recursivo)

3.1 Diseño

Para resolver el problema del matching mínimo de manera recursiva, se planteó el siguiente procedimiento:

Sean i, j, k índices de los grupos de bloques A y B, donde A_i es un bloque de A en la posición $i = A.size()$, B_i un bloque de B en la posición $j = B.size()$ y A_k, B_K bloques de B o A en la posición k: La búsqueda del matching mínimo se dividirá en dos sub-procedimientos:

- El primero donde k itera sobre los bloques de A, desde $i-1$ hasta 1. En este caso (llamemosle (a)), se partirá el problema en dos subproblemas, uno con los bloques entre A_{k+1} hasta A_i y el bloque B_j, y el otro con los bloques restantes. Se decrementa a k y se repite este proceso hasta $k = 1$. Se hará una llamada recursiva a cada subproblema del lado izquierdo, que contendrá bloques de A_1 hasta A_k y bloques de B_1 hasta B_{j-1}, y al lado derecho que contendrá bloques de A_{k+1} hasta A_i y el bloque B_j. Se encontrará el peso minimo de la suma de los pesos de los matchings de ambos subproblemas. Nótese que para el subproblema del lado derecho siempre se tendrá un caso base, definido cuando $i = 1$, $j = 1$ o $i = j = 1$.

- El segundo donde k itera sobre los bloques de B, desde $j-1$ hasta 1. En este caso (llamemosle (b)), se partirá el problema en dos subproblemas, uno con los bloques entre B_{k+1} hasta B_j y el bloque A_i, y el otro con los bloques restantes. Se decrementa a k y se repite este proceso hasta $k = 1$. Se hará una llamada recursiva a cada subproblema del lado izquierdo, que contendrá bloques de B_1 hasta B_k y bloques de A_1 hasta A_{i-1}, y al lado derecho que contendrá bloques de B_{k+1} hasta B_j y el bloque A_i. Se encontrará el peso minimo de la suma de los pesos de los matchings de ambos subproblemas. Nótese que para el subproblema del lado derecho siempre se tendrá un caso base, definido cuando $i = 1$, $j = 1$ o $i = j = 1$.

Luego, se compara el peso de ambos matchings encontrados y se retorna el de menor peso.

Como ejemplo para visualizar la partición, pongamos a los siguientes bloques:

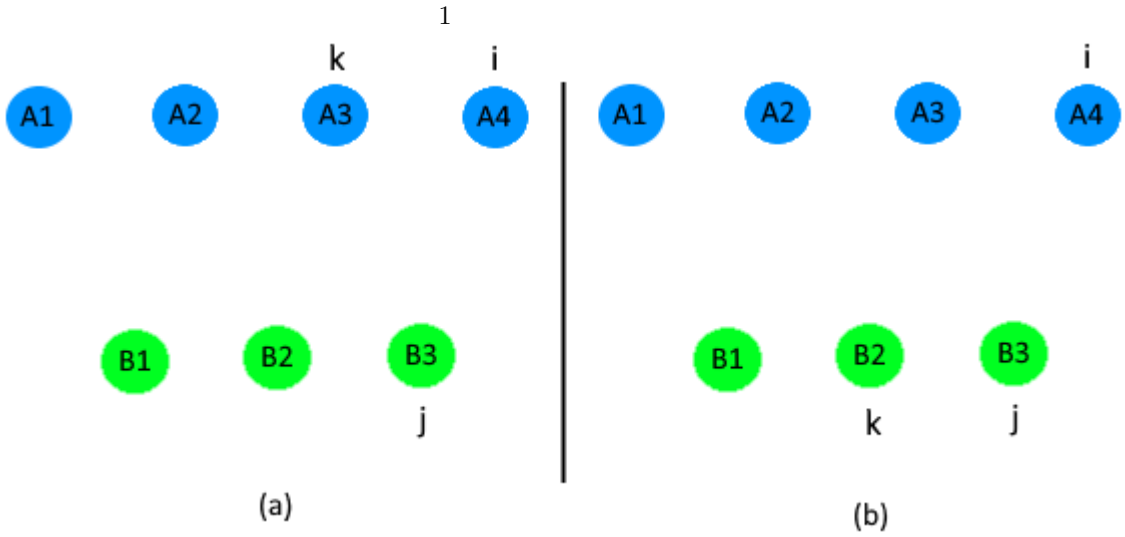


Figure 4: Para el sub-procedimiento (a), k toma el valor de $i-1$ y para el sub-procedimiento (b) toma el valor de $j-1$.

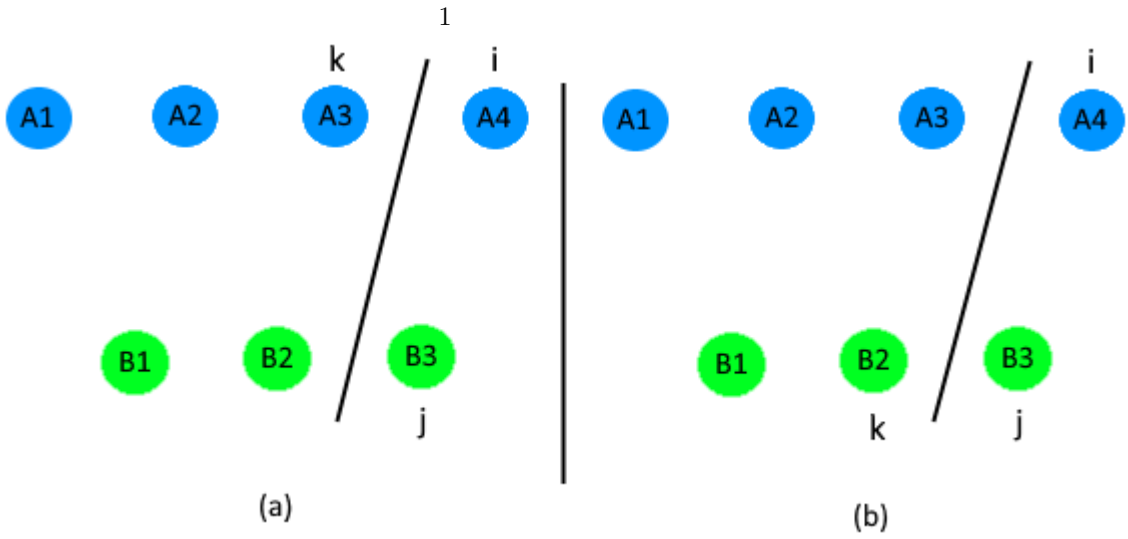


Figure 5: Se hace la división del problema en dos subproblemas como se mencionó anteriormente. La línea negra representa la división.

1

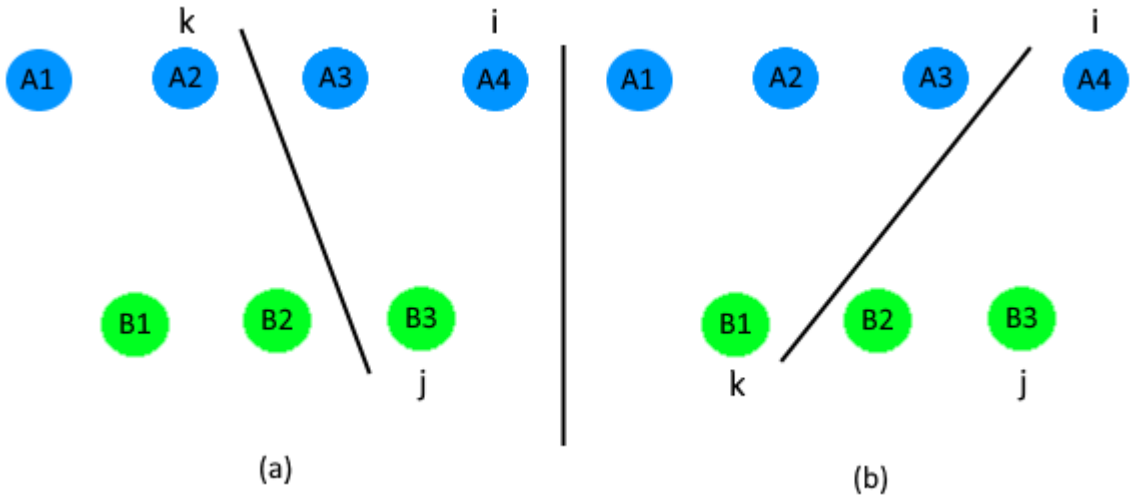
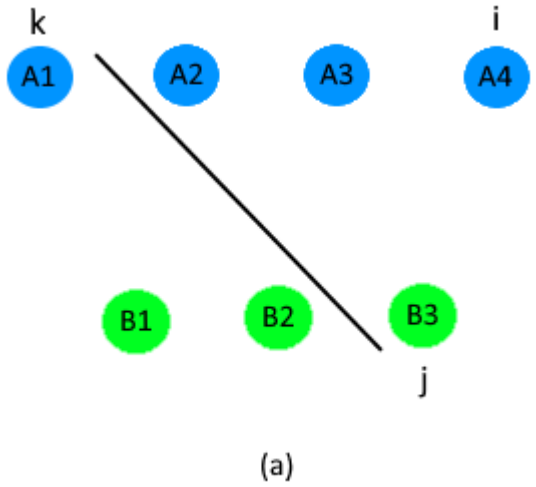


Figure 6: A medida que los valores de k disminuyen, se van haciendo distintas particiones. En caso de (b), k llega a su valor mínimo.



1

Figure 7: Se hace la última partición en (a) y k llega a su valor mínimo.

3.2 Implementación

El pseudocodigo para nuestro algoritmo Recursivo es el siguiente:

$min\_match \leftarrow 0$
$i \leftarrow A.size() - 1$
$j \leftarrow B.size() - 1$
$min\_agrupación$
$min\_división$
$min\_agrupación.weight \leftarrow INT\_MAX$
$min\_división.weight \leftarrow INT\_MAX$
if A.size() == 1 and B.size() == 1 **then**
  $match \leftarrow --$
  $match.i \leftarrow A[0].index$
  $match.j \leftarrow B[0].index$
  $min\_match.matching[-] \leftarrow match$
  $min\_match.weight \leftarrow (A[0].j - A[0].i + 1)/(B[0].j - B[0].i + 1)$
  return min_match
end if
if A.size() == 1 and B.size() > 1 **then**
  $sum \leftarrow 0$
  $match \leftarrow --$
  $match.i \leftarrow A[0].index$
  for it < B.size() **do**
    $match.j \leftarrow B[it].index$
    $min\_match.matching[-] \leftarrow match$
    $sum \leftarrow sum + (B[it].j - B[it].i + 1)$
  end for
  $min\_match.weight \leftarrow (A[0].j - A[0].i + 1)/sum$
  return min_match
end if
if B.size() == 1 and A.size() > 1 **then**
  $sum \leftarrow 0$
  $match \leftarrow --$
  $match.j \leftarrow B[0].index$
  for it = 0; it < A.size() **do**
    $match.i \leftarrow A[it].index$
    $min\_match.matching[-] \leftarrow match$
    $sum \leftarrow sum + (A[it].j - A[it].i + 1)$
  end for
  $min\_match.weight \leftarrow sum/(B[0].j - B[0].i + 1)$
  return min_match
end if
for k = i - 1; k $\geq$ 0 **do**
  $left\_A[-](A.begin(), A.begin() + k + 1)$
  $right\_A[-](A.begin() + k + 1, A.end())$

$left\_B[-](B.begin(), B.end() - 1)$
$right\_B$
$right\_B[-] \leftarrow (B[B.size() - 1])$
$min\_left \leftarrow opt\_solución(left\_A, left\_B)$
$min\_right \leftarrow opt\_solución(right\_A, right\_B)$
$merge$
$merge.matching \leftarrow merge\_matchings(min\_left.matching, min\_right.matching)$
$merge.weight \leftarrow min\_left.weight + min\_right.weight$
if merge.weight < min_agrupación.weight **then**
$min\_agrupación \leftarrow merge$
end if
end for
for k = j - 1; k ≥ 0 **do**
$left\_A[-](A.begin(), A.end() - 1)$
$right\_A[-] \leftarrow (A[A.size() - 1])$
$left\_B[-](B.begin(), B.begin() + k + 1)$
$right\_B[-](B.begin() + k + 1, B.end())$
$min\_left \leftarrow opt\_solución(left\_A, left\_B)$
$min\_right \leftarrow opt\_solución(right\_A, right\_B)$
$merge$
$merge.matching \leftarrow merge\_matchings(min\_left.matching, min\_right.matching)$
$merge.weight \leftarrow min\_left.weight + min\_right.weight$
if merge.weight < min_división.weight **then**
$min\_división \leftarrow merge$
end if
end for
return min_agrupación.weight < min_división.weight ? min_agrupación : min_división

La notación [-], representa un vector y [-]<- representa un push en un vector.

3.3 Demostración del tiempo de ejecución

T(m,n)=Ω(2max(m,n))
T(m,n)≥c × $2^{max(m,n)}$
$T(m,n) = \sum_{k=0}^{m-1} T(k, n-1) + \sum_{k=0}^{n-1} T(m-1, k)$
Por inducción →T(m,n) ≥ $c \times 2^{max(m,n)}$
Caso Base → $m = 0, n = 0$
Paso inductivo:
$T(m,n) = 2(0,0)=1$
T(m,n)≥ $\sum_{k=0}^{n} 2^{max(m,k)}$
T(m,n)≥ $2^{max(m,k)}$ T(m,n)> Ω($2^{max(m,n)}$)

4 Pregunta 4 (Memoizado)

4.1 Diseño

Para el algoritmo memoizado se usó el mismo diseño que el recursivo, teniendo como única diferencia la adición de una matriz donde se almacenan las respuestas de los subproblemas hallados. La posición M[i][j] almacena el matching para un subproblema cuyos arreglos de entrada A y B tienen tamaños i y j, respectivamente. Dado que pueden existir soluciones diferentes para dos subproblemas con arreglos de bloques de tamaños iguales (ya que los subproblemas se definen por el tamaño de los arreglos de bloques y no los tamaños de los bloques en ellos, los cuales son determinantes para hallar el peso del matching), sólamente almacenaremos las soluciones de las llamadas recursivas a los subproblemas de la izquierda (ya que los de la derecha siempre formarán un caso base).
El pseudocodigo para nuestro algoritmo Memorizado es el siguiente:

4.2 Implementación

```
min_match ← 0
i ← A.size() − 1
j ← B.size() − 1
min_agrupación
min_división
min_agrupación.weight ← INT_MAX
min_división.weight ← INT_MAX
if left then
  if A.size() <= mem.size() - 1 and B.size() <= mem[0].size() - 1 then
    if mem[A.size()][B.size()].weight ≠ 0 then
      return mem[A.size()][B.size()]
    end if
  end if
end if
if A.size() == 1 and B.size() == 1 then
  match ← −−
  match.i ← A[0].index
  match.j ← B[0].index
  min_match.matching[−] ← match
  min_match.weight ← (A[0].j − A[0].i + 1)/(B[0].j − B[0].i + 1)
  return min_match
end if
if A.size() == 1 and B.size() > 1 then
  sum ← 0
  match ← −−
  match.i ← A[0].index
  for it = 0; it < B.size() do
    match.j ← B[it].index
```

```
      min_match.matching[−] ← match
      sum ← sum + (B[it].j − B[it].i + 1)
    end for
    min_match.weight ← (A[0].j − A[0].i + 1)/sum
    return min_match
  end if
  if B.size() == 1 and A.size() > 1 then
    sum ← 0
    match ← −−
    match.j ← B[0].index
    for it = 0; it < A.size() do
      match.i ← A[it].index
      min_match.matching[−] ← match
      sum ← sum + (A[it].j − A[it].i + 1)
    end for
    min_match.weight ← sum/(B[0].j − B[0].i + 1)
    return min_match
  end if
  for k = i - 1; k ≥ 0 do
    left_A[−](A.begin(), A.begin() + k + 1)
    right_A[−](A.begin() + k + 1, A.end())
    left_B[−](B.begin(), B.end() − 1)
    right_B
    right_B[−] ← (B[B.size() − 1])
    min_left ← opt_solución(left_A, left_B)
    min_right ← opt_solución(right_A, right_B)
    merge
    merge.matching ← merge_matchings(min_left.matching, min_right.matching)
    merge.weight ← min_left.weight + min_right.weight
    if merge.weight < min_agrupación.weight then
      min_agrupación ← merge
    end if
    mem[left_A.size()][left_B.size()] ← min_left
  end for
  for k = j - 1; k ≥ 0 do
    left_A[−](A.begin(), A.end() − 1)
    right_A[−] ← (A[A.size() − 1])
    left_B[−](B.begin(), B.begin() + k + 1)
    right_B[−](B.begin() + k + 1, B.end())
    min_left ← opt_solución(left_A, left_B)
    min_right ← opt_solución(right_A, right_B)
    merge
    merge.matching ← merge_matchings(min_left.matching, min_right.matching)
    merge.weight ← min_left.weight + min_right.weight
    if merge.weight < min_división.weight then
      min_división ← merge
```

end if
$mem[left\_A.size()][left\_B.size()] \leftarrow min\_left$
end for
return min_agrupación.weight < min_división.weight ? min_agrupación : min_división

La notación [-], representa un vector y [-]<- representa un push en un vector.

4.3 Demostración del tiempo de ejecución

T(m,n)=O(m,n)
C(m,n)≥T(m,n)≥ 0
Tenemos →T(m,n) =$\sum_{k=0}^{m-1} T(k, n-1) + \sum_{k=0}^{n-1} T(m-1, k)$
Tomaremos que el tiempo para resolver los subproblemas es constante
K= número de problemas × tiempo
Tendriamos: →k×$T(m-1, n-1) + k \times T(m-1, n-1)$
$= K \times 2(T(m-1, n-1))$
$= K \times 2C(m, n)$
$C(m, n) \geq O(m, n) = T(m, n)$

5 Anexos

5.1 Código en C++ del archivo min_matching.cpp

```
#include <iostream>
#include <vector>
#include <climits>
#include <cstdlib>

using namespace std;

struct Pair {
    float i;
    float j;
    int index;
};

struct Matching {
    vector<Pair> matching;
    float weight;
};

/* Prototipos */
vector<Pair> get_blocks (vector<int>);
Matching greedy_matching (vector<int>, vector<int>);
Matching min_matching_recursive (vector<int>, vector<int>);
Matching min_matching_memoized (vector<int>, vector<int>);
Matching opt_solution (vector<Pair>, vector<Pair>);
Matchin opt_solution_mem (vector<Pair>, vector<Pair>, bool left);
vector<Pair> merge_matchings (vector<Pair>, vector<Pair>);
vector<int> string_to_vector (string);
void print_matching (Matching);

/* Matriz donde se almacenan soluciones a subproblemas */
vector<vector<Matching>> mem;

/* Funcion que genera los bloques para un vector. Recorre todo el vector
 * Generando bloques, por lo que tiene un tiempo de ejecucion de O(n) */
vector<Pair> get_blocks (vector<int> A) {
    vector<Pair> blocks;
    bool flag = false;
    Pair block;
    for (int i = 0; i < A.size (); i++) {
        block.index = blocks.size ();
        if (A[i] == 1) {
            if (!flag) {
                flag = true;
                block.i = i;
            }
            if (i == A.size () - 1) {
                    block.j = i;
                    blocks.push_back (block);
            }

        }
        else if (A[i] == 0) {
            if (flag) {
                flag = false;
                block.j = i - 1;
                blocks.push_back (block);
            }
        }
    }
    return blocks;
}

/*Genera un matching agrupando elementos de A y B uno por uno, y haciendo
 * agrupaciones y divisiones cuando uno de estos llega a su limite. Ya que
 * siempre se recorrera el vector de bloques de mayor cantidad de elementos,
 * tiene un tiempo de ejecucion de O(max{m, n})*/
Matching greedy_matching (vector<int> A, vector<int> B) {
    Matching result;

    vector<Pair> blocks_A = get_blocks (A);
    vector<Pair> blocks_B = get_blocks (B);

    int i = 0, j = 0;
    int n = blocks_A.size ();
    int m = blocks_B.size ();
    if (n == 0 or m == 0) {
        cout << "Error: No hay bloques en uno de los vectores" << endl;
        exit (1);
    }

    int max;
    if (n > m) {
        max = n;
```

```
        m -=1;
    }
    else {
        max = m;
        n -=1;
    }
    Pair match;
    float weight = 0;
    while (i < max && j < max) {
        match.i = blocks_A[i].index;
        match.j = blocks_B[j].index;
        weight += (blocks_A[i].j - blocks_A[i].i + 1) / (blocks_B[j].j - blocks_B[j].i + 1);
        result.matching.push_back (match);

        if (i < n)
            i++;
        if (j < m)
            j++;
    }
    result.weight = weight;
    return result;
}

/* Genera bloques para los vectores */
Matching min_matching_recursive (vector<int> A, vector<int> B) {
    vector<Pair> result;

    vector<Pair> blocks_A = get_blocks (A);
    vector<Pair> blocks_B = get_blocks (B);

    Matching min_matching = opt_solution (blocks_A, blocks_B);

    return min_matching;
}

Matching opt_solution (vector<Pair> A, vector<Pair> B) {
    Matching min_match;
    int i = A.size () - 1;
    int j = B.size () - 1;
    Matching min_agrupacion;
    Matching min_division;

    min_agrupacion.weight = INT_MAX;
    min_division.weight = INT_MAX;

    if (A.size () == 1 and B.size () == 1) {
        Pair match;
        match.i = A[0].index;
        match.j = B[0].index;
        min_match.matching.push_back (match);
        min_match.weight = (A[0].j - A[0].i + 1) / (B[0].j - B[0].i + 1);
        return min_match;
    }

    if (A.size () == 1 and B.size () > 1) {
        Pair match;
        float sum = 0;
        match.i = A[0].index;
        for (int it = 0; it < B.size (); it++) {
            match.j = B[it].index;
            min_match.matching.push_back (match);
            sum += B[it].j - B[it].i + 1;
        }
        min_match.weight = (A[0].j - A[0].i + 1) / sum;
        return min_match;
    }
    if (B.size () == 1 and A.size () > 1){
        Pair match;
        float sum = 0;
        match.j = B[0].index;
        for (int it = 0; it < A.size (); it++){
            match.i = A[it].index;
            min_match.matching.push_back (match);
            sum += A[it].j - A[it].i + 1;
        }
        min_match.weight = sum / (B[0].j - B[0].i + 1);
        return min_match;
    }

    for (int k = i - 1; k >= 0; k--) {
        vector<Pair> left_A (A.begin (), A.begin () + k + 1);
        vector<Pair> right_A (A.begin () + k + 1, A.end ());

        vector<Pair> left_B (B.begin (), B.end () - 1);
        vector<Pair> right_B; right_B.push_back (B[B.size () - 1]);

        Matching min_left = opt_solution (left_A, left_B);
        Matching min_right = opt_solution (right_A, right_B);

        Matching merge;
        merge.matching = merge_matchings (min_left.matching, min_right.matching);
```

```
        merge.weight = min_left.weight + min_right.weight;
        if (merge.weight < min_agrupacion.weight)
            min_agrupacion = merge;
        print_matching (merge);
    }
    for (int k = j - 1; k >= 0; k--) {
        vector<Pair> left_A (A.begin (), A.end () - 1);
        vector<Pair> right_A; right_A.push_back (A[A.size () - 1]);

        vector<Pair> left_B (B.begin (), B.begin () + k + 1);
        vector<Pair> right_B (B.begin () + k + 1, B.end ());

        Matching min_left = opt_solution (left_A, left_B);
        Matching min_right = opt_solution (right_A, right_B);

        Matching merge;
        merge.matching = merge_matchings (min_left.matching, min_right.matching);
        merge.weight = min_left.weight + min_right.weight;
        if (merge.weight < min_division.weight)
            min_division = merge;
        print_matching (merge);
    }

    return min_agrupacion.weight < min_division.weight ? min_agrupacion : min_division;
}

vector<Pair> merge_matchings (vector<Pair> left, vector<Pair> right) {
    for (int i = 0; i < right.size (); i++)
        left.push_back (right[i]);
    return left;
}

Matching min_matching_memoized (vector<int> A, vector<int> B) {
    vector<Pair> result;

    vector<Pair> blocks_A = get_blocks (A);
    vector<Pair> blocks_B = get_blocks (B);

    if (blocks_A.size () == 0 or blocks_B.size () == 0) {
        cerr << "Vector sin bloques introducido." << endl;
    }

    mem.resize (blocks_A.size ());
    for (int i = 0; i < blocks_A.size (); i++)
                mem[i].resize (blocks_B.size ());

    Matching min_matching = opt_solution_mem (blocks_A, blocks_B, false);

    return min_matching;
}

Matching opt_solution_mem (vector<Pair> A, vector<Pair> B, bool left) {
    Matching min_match;
    int i = A.size () - 1;
    int j = B.size () - 1;
    Matching min_agrupacion;
    Matching min_division;

    min_agrupacion.weight = INT_MAX;
    min_division.weight = INT_MAX;
    if (left) {
        if (A.size () <= mem.size () - 1 and B.size () <= mem[0].size () - 1) {
            if (mem[A.size ()][B.size ()].weight != 0) {
                return mem[A.size ()][B.size ()];
            }
        }
    }
    if (A.size () == 1 and B.size () == 1) {
        Pair match;
                match.i = A[0].index;
                match.j = B[0].index;
                min_match.matching.push_back (match);
                min_match.weight = (A[0].j - A[0].i + 1) / (B[0].j - B[0].i + 1);
                return min_match;
    }

    if (A.size () == 1 and B.size () > 1) {
        Pair match;
                float sum = 0;
                match.i = A[0].index;
                for (int it = 0; it < B.size (); it++) {
                        match.j = B[it].index;
                        min_match.matching.push_back (match);
                        sum += B[it].j - B[it].i + 1;
                }
                min_match.weight = (A[0].j - A[0].i + 1) / sum;
                return min_match;
    }

    if (B.size () == 1 and A.size () > 1){
        Pair match;
```

```
                float sum = 0;
                match.j = B[0].index;
                for (int it = 0; it < A.size (); it++){
                        match.i = A[it].index;
                        min_match.matching.push_back (match);
                        sum += A[it].j - A[it].i + 1;
                }
                min_match.weight = sum / (B[0].j - B[0].i + 1);
                return min_match;
    }

    for (int k = i - 1; k >= 0; k--) {
        vector<Pair> left_A (A.begin (), A.begin () + k + 1);
        vector<Pair> right_A (A.begin () + k + 1, A.end ());

        vector<Pair> left_B (B.begin (), B.end () - 1);
        vector<Pair> right_B; right_B.push_back (B[B.size () - 1]);

        Matching min_left = opt_solution_mem (left_A, left_B, true);
        Matching min_right = opt_solution_mem (right_A, right_B, false);

        Matching merge;
        merge.matching = merge_matchings (min_left.matching, min_right.matching);
        merge.weight = min_left.weight + min_right.weight;
        if (merge.weight < min_agrupacion.weight)
            min_agrupacion = merge;
        /*for (int h = 0; h < merge.matching.size (); h++)
            {
                cout << "(" << merge.matching[h].i << "," << merge.matching[h].j << ")" << " ";
            }
        cout << merge.weight << endl;*/
        print_matching (merge);
       mem[left_A.size ()][left_B.size ()] = min_left;
    }
    for (int k = j - 1; k >= 0; k--) {
        vector<Pair> left_A (A.begin (), A.end () - 1);
        vector<Pair> right_A; right_A.push_back (A[A.size () - 1]);

        vector<Pair> left_B (B.begin (), B.begin () + k + 1);
        vector<Pair> right_B (B.begin () + k + 1, B.end ());

        Matching min_left = opt_solution_mem (left_A, left_B, true);
        Matching min_right = opt_solution_mem (right_A, right_B, false);

        Matching merge;
        merge.matching = merge_matchings (min_left.matching, min_right.matching);
        merge.weight = min_left.weight + min_right.weight;
        if (merge.weight < min_division.weight)
            min_division = merge;
        /*for (int h = 0; h < merge.matching.size (); h++)
            {
                cout << "(" << merge.matching[h].i << "," << merge.matching[h].j << ")" << " ";
            }
        cout << merge.weight << endl;*/
        print_matching (merge);
       mem[left_A.size ()][left_B.size ()] = min_left;
    }

    return min_agrupacion.weight < min_division.weight ? min_agrupacion : min_division;
}

void print_matching (Matching result) {
    cout << "Matching: ";
        for (int i = 0; i < result.matching.size (); i++)
        {
                cout << "(" << result.matching[i].i << "," << result.matching[i].j << ")" << " ";
        }
        cout << endl << "Weight: " << result.weight << endl;
}

vector<int> string_to_vector (string str) {
    vector<int> result;
    for (int i = 0; i < str.length (); i++) {
        int value = str[i] - '0';
        if (value != 0 and value != 1){
            cout << "Error: Arreglo con valores no validos." << endl;
            vector<int> vacio;
            return vacio;
        }
        result.push_back (value);
    }
    return result;
}

int main () {
    vector<int> A;
    vector<int> B;
    while (true) {
        while (true) {
            string str_A, str_B;
            do {
```

```
            cout << "Ingrese arreglo A: ";
            cin >> str_A;
            if (str_A.length () == 0)
                cout << "Arreglo vacio" << endl;
        } while (str_A.length () == 0);

        do {
            cout << "Ingrese Arreglo B: ";
            cin >> str_B;
            if (str_B.length () == 0)
                cout << "Arreglo vacio" << endl;
        } while (str_B.length () == 0);
        if (str_A.length() == str_B.length ()) {
            A = string_to_vector (str_A);
                B = string_to_vector (str_B);
            if (A.size () == 0 or  B.size () == 0)
                cout << "Se ha ingresado un arreglo vacio" << endl;
            else
                break;
        }
        else {
            cout << "Arreglos de tamanos distintos. Ingrese nuevamente" << endl;
        }

    }

    Matching result;
     int option;
     bool new_vector = false;
     cout << "[1] Matching Voraz" << endl;
     cout << "[2] Matching Minimo Recursivo" << endl;
     cout << "[3] Matching Minimo Memoizado" << endl;
     cout << "[4] Ingresar Nuevo Vector" << endl;
     cout << "[0] Salir" << endl;
     while (!new_vector) {
         cout << "Ingrese el algoritmo a utilizar: ";
         cin >> option;
         switch (option) {
             case 0:
                 return 0;
             case 1: ;
                 result = greedy_matching (A, B);
                 cout << "resultado" << endl;
                 print_matching (result);
                 break;
             case 2: ;
                 result = min_matching_recursive (A, B);
                 cout << "resultado" << endl;
                 print_matching (result);
                 break;
             case 3: ;
                 result = min_matching_memoized (A, B);
                 cout << "resultado" << endl;
                 print_matching (result);
                 break;
             case 4: ;
                 new_vector = true;
                 break;
             default:
                 cout << "No existe la opcion " << option << endl;

         }
     }
 }
}
```