Análisis y Diseño de Algoritmos

Proyecto Parcial

Benjamín Díaz García
benjamin.diaz@utec.edu.pe

Gabriel Spranger Rojas
gabriel.spranger@utec.edu.pe

Rodrigo Céspedes Velásquez
rodrigo.cespedes@utec.edu.pe

Profesor:
Ph.D. Juan Gabriel Gutiérrez Alva

June 16, 2020

Introducción

En el presente trabajo se presentarán, explicarán y demostraŕan algoritmos ideados por nosotros para resolver diferentes problemas propuestos.
Cada problema será resuelto a través de distintos tipos de algoritmos (Voraz, Recursivo, Memoizado y de Programación Dinámica.)

A lo largo del trabajo se seguirá una estructura predefinida.
Primero, se presentará el problema, tipo de algoritmo y lo que es requerido del algoritmo, así como sus entradas y salidas. Después se exhibirá el pseudocódigo correspondiente a nuestra solución. Finalmente, se expondrá el análisis de tiempo para dicha respuesta, además de su demostración. En algunos casos también se pondrá la recurrencia correspondiente (si es pertinente).

Puede que para algunas preguntas existan formatos diferentes, pues en algunos casos se recurre a distintos métodos para evidenciar o demostrar alguna característica de nuestra respuesta, pero en general casi todas las preguntas siguen la misma estructura.

1. Algoritmo Voraz

Analice, diseñe e implemente un algoritmo voraz con complejidad lineal para el problema MIN-MATCHING. Su algoritmo no necesariamente debe encontrar el matching de peso mínimo.

Require: Dos arreglos A y B de ceros y unos de tamaño p con n y m bloques, respectivamente.

Ensure: Un matching entre A y B, no necesariamente óptimo, y su peso.

MIN-MATCHING-GREEDY(A', B')**:**

```
Sea A' el arreglo con los bloques de A con n bloques.
Sea B' el arreglo con los bloques de B con m bloques
while i < m − 1 and j < n − 1 do
  if A'[i] == B'[j] then
    auxA'.pushback(i)
    auxB'.pushback(j)
    R.pushback( pair(auxA', auxB') )
    sumavalor += A'[i] / B'[j]
    i++
    j++
  end if
else if  A'[i] > B'[j]  then
  auxA .pushback(i)
  valA' = A'[i]
  while (valA' > valB') and (j < n − 1)  do
    auxB'.pushback(j)
    valB' += B'[j]
    j++
  end while
  R.pushback( pair(auxA', auxB') )
  sumavalor += valA / valB'
  i++
else
  auxB'.pushback(j)
  valB' = B'[j]

  while (valA' < valB') and (i < m − 1)  do
```

28: $auxA'$.pushback(i)
29: $valA' \mathrel{+}= A'[i]$
30: $i++$
31: **end while**
32: R.pushback($pair(auxA', auxB')$)
33: $sumavalor \mathrel{+}= valA'$ / $valB'$
34: $j++$
35: $valA' = 0$
36: $valB' = 0$
37: $auxA'$.clear()
38: $auxB'$.clear()
39: **end while**
40:
41: $auxA'$.pushback$(i, (i+1) \dots (m-1))$
42: $valA' \mathrel{+}= (A'[i], A'[i+1] \dots A'[m-1])$
43:
44: $auxB'$.pushback$(j, (j+1) \dots (n-1))$
45: $valB' \mathrel{+}= (B'[j], B'[j+1] \dots B[n-1])$
46:
47: R.pushback($pair(auxA', auxB')$)
48: $sumavalor \mathrel{+}= valA'$ / $valB'$
49:
50: **return** $pair(sumavalor, R)$

Análisis de Tiempo

Las veces seran analizadas en el peor de los casos. Por temas de practicidad utilizaremos la letra c como la constante de mayor valor posible en el contexto donde se utiliza.

Linea 3 a 11:
Tiempo: c
Veces: min(m-x, n-y), ($0 \leq$ x,y $\leq$ m-1,n-1)

Linea 12 a 14:
Tiempo: c
Veces: min(m-x, n-y)

Linea 15 a 19:
Tiempo: c
Veces: x

Linea 20 a 22:
Tiempo: c
Veces: min(m-x, n-y)

Linea 23 a 26:
Tiempo: c
Veces: min(m-x, n-y)

Linea 27 a 31:
Tiempo: c
Veces: y

Linea 32 a 39:
Tiempo: c
Veces: min(m-x, n-y)

Linea 41:
Tiempo: m-(m-x)
Veces: 1

Linea 42:
Tiempo: m-(m-x)
Veces: 1

Linea 44:
Tiempo: n-(n-y)
Veces: 1

Linea 45:
Tiempo: n-(n-y)
Veces: 1

Linea 47 a 50:
Tiempo: c

Veces: 1

Entonces tenemos que:

c*min(m-x,n-y) + c*min(m-x,n-y) + c*x + c*min(m-x,n-y) + c*min(m-x,n-y) +c*y + c*min(m-x,n-y) + c*(m-(m-x)) + c*(m-(m-x)) + c*(n-(n-y)) + c*(n-(n-y)) + c

Es equivalente a:

c*min(m-x,n-y) + c*x + c*y + c*(m-(m-x)) + c*(n-(n-y)) + c

Es equivalente a:

c*min(m-x,n-y) + c*x + c*y + c*m - c*(m-x) + c*n - c*(n-y) + c

Es equivalente a:

c*min(m-x,n-y) + c*x + c*y + c*m - c*m + c*x + c*n - c*n + c*y + c

Es equivalente a:

c*min(m-x,n-y) + c*x + c*y + c*x + c*y + c

Se sabe que min(m-x,n-y) $\leq$ max(m,n), entonces esta expresión es mayor o igual

c*max(m,n) + c*m + c*n + c*m + c*n + c

Es equivalente a:

c*max(m,n) + c*m + c*n + c

Sabemos que m $\leq$ max(m,n) y que n $\leq$ max(m,n), entonces la siguiente expresión es mayor o igual

c*max(m,n) + c*max(m,n) + c*max(m,n) + c*max(m,n)

Lo cual es equivalente a

c*max(m,n)

Y sabemos que c*max(m,n) $\leq$ O(max(m,n))

2. Recurrencia

Plantee una recurrencia para $OPT(i,j)$.

$$
OPT(i,j) = \begin{cases}
\frac{A_1}{B_1} & \text{si } i = 1 \text{ y } j = 1 \\
\\
\frac{A_1+A_2+...+A_i}{B_1} & \text{si } j = 1 \text{ y } i > 1 \\
\\
\frac{A_1}{B_1+B_2+...+B_j} & \text{si } i = 1 \text{ y } j > 1 \\
\\
\min \left\{ \min_{k=j-1}^{1}\{ \; OPT(i-1,k) + \frac{A_i}{B_{k+1}+...+B_j} \; \}, \right. \\
\\
\left. \min_{k=1}^{i-1}\{ \; OPT(k,j-1) + \frac{A_{k+1}+...+A_i}{B_j} \; \} \quad \right\} & \text{caso contrario} \\
\end{cases}
$$

3. Recursivo

Analice, diseñe e implemente un algoritmo recursivo con complejidad exponencial para el problema MIN-MATCHING. Su algoritmo deberá encontrar el matching del peso mínimo.

Require: Dos arreglos A y B de ceros y unos, de tamaño p con n y m bloques, respectivamente.
Ensure: Un matching entre A y B, de peso mínimo, y su peso.

MIN-MATCHING(A, B)**:**

Sea A' el arreglo con los bloques de A con n bloques.
Sea B' el arreglo con los bloques de B con m bloques
Sea X el vector de las conexiones que se van a realizar entre A' y B'.
Sea *Respuesta* un $pair < vector, float >$ que contiene X y el peso.
if $n == 1$ **and** $m == 1$ **then**
  Respuesta.first.emplace_back(A'_1, B'_1)
  Respuesta.second $= \dfrac{A'_1}{B'_1}$
  return Respuesta
end if

if $m == 1$ **then**
  Respuesta.first.emplace_back$(A'_1 + A'_2 + \ldots + A'_n, B'_1)$
  Respuesta.second $= \dfrac{A'_1 + A'_2 + \ldots + A'_n}{B'_1}$
  return Respuesta
end if

if $n == 1$ **then**
  Respuesta.first.emplace_back$(A'_1, B'_1 + B'_2 + \ldots + B'_m)$
  Respuesta.second $= \dfrac{A'_1}{B'_1 + B'_2 + \ldots + B'_m}$
  return Respuesta
end if

$min \leftarrow \infty$
$A_* = A \setminus A'_n$ {Le quitamos el último bloque a A}

```
for k = m - 1 downto 1 do
    $B_* = B'_{1 \ldots k}$ {Agarramos los primeros $k$ bloques}
    Respuesta' ← MIN-MATCHING($A_*, B_*$)
    Respuesta'.second ← Respuesta'.second + $\frac{A'_n}{B_{k+1} + \ldots + B_m}$
    if $min$ > Respuesta'.second then
        Respuesta.first.clear()
        Respuesta.first.emplace_back($A'_n, B_{k+1} + \ldots + B_m$)
        Respuesta.first = Respuesta.first ∪ Respuesta'.first
        Respuesta.second = Respuesta'.second
    end if
end for

$B_* = B \setminus B'_m$ {Le quitamos el último bloque a $B$}
for k = 1 to n - 1 do
    $A_* = A'_{1 \ldots k}$ {Agarramos los primeros $k$ bloques}
    Respuesta' ← MIN-MATCHING($A_*, B_*$)
    Respuesta'.second ← Respuesta'.second + $\frac{A'_{k+1} + \ldots + A'_n}{B'_m}$
    if $min$ > Respuesta'.second then
        Respuesta.first.clear()
        Respuesta.first.emplace_back($A'_{k+1} + \ldots + A'_n, B'_m$)
        Respuesta.first = Respuesta.first ∪ Respuesta'.first
        Respuesta.second = Respuesta'.second
    end if
end for

return Respuesta
```

Análisis de Tiempo

El tiempo del algoritmo viene dado por la siguiente relación de recurrencia:

$$T(n,m) = \begin{cases} m + k_1 & \text{si } n = 1 \text{ y } m > 1 \\ \\ n + k_2 & \text{si } m = 1 \text{ y } n > 1 \\ \\ k_3 & \text{si } m = 1 \text{ y } n = 1 \\ \\ \sum_{k=2}^{m-1} T(n-1,k) + \sum_{k=2}^{n-1} T(k,m-1) & \text{caso contrario} \\ \end{cases}$$

Ahora demostremos que $T(n,m) = \Omega(2^{\max\{n,m\}})$.

En el caso base tenemos que $n = 1$ y $m = 1$:

$$k_3 \geq c\ 2^{\max\{1,1\}}$$

Si $c = \dfrac{k_3}{2}$ tenemos que:

$$k_3 \geq k_3$$

Por lo tanto, el caso base cumple. Ahora veamos el caso inductivo:

$$T(n,m) \geq c\ 2^{\max\{n,m\}}$$

T(n,m) es:

$$T(n,m) = \sum_{k=2}^{m-1} T(n-1,k) + \sum_{k=2}^{n-1} T(k,m-1)$$

Le quitamos una sumatoria, la desigualdad se mantiene:

$$T(n,m) \geq \sum_{k=2}^{m-1} T(n-1,k)$$

Agarramos el último término de la sumatoria:

$$T(n,m) \geq T(n-1,m-1)$$

Por hipótesis inductiva tenemos que:

$$T(n-1, m-1) \geq c \ 2^{\max\{n-1, m-1\}}$$

Le sumamos 1 a n y m:

$$T(n, m) \geq c \ 2^{\max\{n, m\}-1}$$

Por lo tanto, si $c = \dfrac{1}{2}$:

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

4. Memoizado

Analice, diseñe e implemente un algoritmo memoizado para el problema MIN-MATCHING. Su algoritmo deberá encontrar el matching de peso minimo.

Require: Dos arreglos A y B de ceros y unos, de tamaño p con n y m bloques, respectivamente.
Ensure: Un matching entre A y B, de peso mínimo, y su peso.

MIN-MATCHING(A, B)**:**

Sea A' el arreglo con los bloques de A con n bloques.
Sea B' el arreglo con los bloques de B con m bloques
Sea X el vector de las conexiones que se van a realizar entre A' y B'.
Sea *Respuesta* un $pair < vector, float >$ que contiene X y el peso.
Sea M una matriz declarada globalmente que contiene las *Respuestas*.
if $M[n][m]$!= ∞ **then**
  return $M[n][m]$
end if

if $n == 1$ **and** $m == 1$ **then**
  if $M[1][1] == \infty$ **then**
    Respuesta'.first.emplace_back(0, 0)
    Respuesta'.second = $\frac{A'_1}{B'_1}$
    $M[1][1]$ = Respuesta'
  end if
  return $M[1][1]$
end if

if $m == 1$ **then**
  if $M[n][1] == \infty$ **then**
    Respuesta'.first.emplace_back($A'_1 + A'_2 + \ldots + A'_n, B'_1$)
    Respuesta'.second = $\frac{A'_1 + A'_2 + \ldots + A'_n}{B'_1}$
    $M[n][1]$ = Respuesta'
  end if

    return $M[n][1]$
end if

if $n == 1$ **then**
    if $M[1][n] == \infty$ **then**
        Respuesta'.first.emplace_back$(A'_1, B'_1 + B'_2 + \ldots + B'_m)$
        Respuesta'.second $= \dfrac{A'_1}{B'_1 + B'_2 + \ldots + B'_m}$
        $M[1][n] =$ Respuesta'
    end if
    return $M[1][n]$
end if

$min \leftarrow \infty$
$A_* = A \setminus A'_n$ {Le quitamos el último bloque a A}
for k = m - 1 **downto** 1 **do**
    $B_* = B'_{1\ldots k}$ {Agarramos los primeros k bloques}
    Sea n_* y m_* el número de bloques en A_* y B_*, respectivamente.
    if $M[n_*][m_*]$!= ∞ **then**
        Respuesta' $= M[n_*][m_*]$
        Respuesta'.first = Respuesta'.first $\cup$ $(A_n, B_{k+1} + \ldots + B_m)$
        Respuesta'.second += $\frac{A_n}{B_{k+1}+\ldots+B_m}$
    else
        Respuesta' = MIN-MATCHING(A_*, B_*)
        Respuesta'.first = Respuesta'.first $\cup$ $(A_n, B_{k+1} + \ldots + B_m)$
        Respuesta'.second += $\frac{A_n}{B_{k+1}+\ldots+B_m}$
    end if
    if $min >$ Respuesta'.second **then**
        Respuesta = Respuesta'
    end if
end for

$B_* = B \setminus B'_m$ {Le quitamos el último bloque a B}
for k = 1 **to** n - 1 **do**
    $A_* = A'_{1\ldots k}$ {Agarramos los primeros k bloques}
    Sea n_* y m_* el número de bloques en A_* y B_*, respectivamente.
    if $M[n_*][m_*]$!= ∞ **then**

Respuesta' = $M[n_*][m_*]$
Respuesta'.first = Respuesta'.first $\cup$ $(A_{k+1} + \ldots + A_n, B_m)$
Respuesta'.second += $\frac{A_n}{B_{k+1}+\ldots+B_m}$
else
Respuesta' = MIN-MATCHING(A_*, B_*)
Respuesta'.first = Respuesta'.first $\cup$ $(A_{k+1} + \ldots + A_n, B_m)$
Respuesta'.second += $\frac{A_{k+1}+\ldots+A_n}{B_m}$
end if
if $min >$ Respuesta'.second **then**
Respuesta = Respuesta'
end if
end for

return Respuesta

Análisis de Tiempo

En este caso, cada vez que calculamos el peso mínimo de una combinación en especial, la guardamos en su lugar correspondiente en la matriz de memoización para que siguientes llamadas recursivas con los mismos parámetros solo agarren el valor de la matriz, de tal manera, tenemos un acceso en $O(1)$ para resultados memoizados. Como cada llamada recursiva siempre estará dentro de los límites n y m de la matriz de memoización, cada celda será calculada como máximo, una vez. Por lo tanto, tenemos que en el peor caso, llenaremos todas las celdas de la matriz, teniendo un total de n x m accesos a la matriz. Por ello, podemos concluir que el algoritmo memoizado corre en $O(mn)$.

Implementación

La implementación del algoritmo Greedy, Recursivo y Memoizado, se encuentra en el siguiente repositorio:

- `https://github.com/MenuMarino/Proyecto-ADA`