Problema 4: Particionando y ordenando

Amanda Marrero Santos

Loraine Monteagudo García

Carlos Rafael Ortega Lezcano

Universidad de La Habana

2018-2019

Resumen

En el siguiente documento se exponen distintas soluciones de un problema cuyo objetivo es buscar la partición definida sobre un conjunto A que tenga costo mínimo, que depende a su vez de los costos de los subconjuntos contenidos en ella. Para el cálculo del costo de los subconjuntos cobra importancia su cardinalidad y su posición en la partición. Primero, se realiza un algoritmo de fuerza bruta que genera todas las posibles particiones, luego se calculan todas las ordenaciones que tendrán los subjuntos y se elije aquella que tenga menor costo, obteniéndose un costo de $O(B_n * n! * n)$. A continuación, este algoritmo se mejora determinando la mejor ordenación para los subconjuntos, mejorando el costo a $O(B_n * n^2)$. Por último, se intenta con una solución dinámica que a pesar de ser fallida, sirve como preliminar para otra cuyo costo será $O(n^3)$.

I. Análisis del Problema

El algoritmo recibirá como entrada:

- Conjunto A de n enteros:

A = | a_1 | a_2 | a_3 | ... | a_{n-1} | a_n |

Se define como el costo de un conjunto el producto de su cardinalidad por la suma de los máximos de los conjuntos que se crearon antes, es decir, aquellos que tenían índices menores o iguales al suyo. Es decir, dada una ordenación $\{S_1, S_2, ..., S_m\}$ el costo del conjunto S_i, denotado como c_{s_i} será:

$$c_{s_i} = |S_i| * \sum_{k=0}^{i} max(S_k)$$

En total, el costo de una partición $P = \{S_1, S_2, ..., S_m\}$ será la suma de los costos de todos sus subconjuntos, es decir, será: $C = \sum_{k=0}^{m} c_{S_k}$

Teniendo en cuenta esto se quiere generar como salida:

- La partición de A en subconjuntos y el orden en que han de darse estos para que la suma de los costos sea mínima. Por lo que se retornará una partición P:

$$\text{P} = \boxed{S_1 \mid S_2 \mid S_3 \mid ... \mid S_{m-1} \mid S_m}$$

  Donde los S_i, con $0 < i \leq m$ serán subconjuntos de A que pertenecerán a la partición formada de ella en la que el orden de estos subconjuntos importa por la definición de costo dada.

Considerando las características del problema se llevaron a cabo distintos algoritmos para obtener la solución del ejercicio que se expondrán a continuación.

II. Fuerza Bruta

Como solución del problema se debe dar la partición de un conjunto inicial A y el orden en que los elementos de dicha partición se deben añadir para que el costo de dicha ordenación sea mínimo. Por lo tanto, para comprobar el comportamiento de las soluciones, se realizó como primera aproximación del problema un algoritmo de búsqueda exhaustiva.

Primero se generaron todas las posibles particiones de un conjunto inicial A. Para ello, se determina un elemento i, y luego se calculan todas las posibles permutaciones del conjunto sin ese elemento recursivamente. La única partición posible para un conjunto de tamaño 1 es la partición que lo contiene así mismo como subconjunto, por lo que este será el caso base. Luego, existen distintas maneras de añadir el elemento i que se excluyó, esto es, incluyéndolo a cada uno de los subconjuntos que se formaron en las particiones.

Luego se determinan todas las posibles ordenaciones que podían existir entre los elementos de dichas particiones. Es decir, se calculan las permutaciones de cada partición de A.

A continuación, para calcular el costo de cada ordenación se recorre cada subconjunto de la partición. Como dicho costo se define como la suma de los máximos conjuntos que tienen índice menor o igual al actual, entonces para calcular el costo del subconjunto j se lleva una variable $c_j = \sum_{i=0}^{j} m_i$, donde m_i es el máximo elemento del subconjunto i. Nótese que por esta definición:

$$c_{j+1} = \sum_{i=0}^{j+1} m_i = \sum_{i=0}^{j} m_i + m_{j+1} = c_j + m_{j+1}$$

Por lo que se lleva una suma acumulativa de los máximos anteriores y esta se va actualizando por cada subconjunto nuevo que se analiza.

El costo total de la ordenación de una partición es la suma de los máximos de los subconjuntos que se encuentran antes del actual por la cardinalidad de dicho conjunto, así que el costo final cf se define como: $cf = \sum_{i=0}^{k} c_i * |p_i|$, donde $|p_i|$ es la cardinalidad del subconjunto i y k la cantidad total de elementos de la partición. Igualmente para calcular este costo se puede llevar una suma acumulativa, ya que el costo total de la partición hasta el momento de analizar un subconjunto j es: $cf_j = \sum_{i=0}^{j} c_i * |p_i|$, y para calcular dicho costo hasta el conjunto $j+1$, se tiene que:

$$cf_{j+1} = \sum_{i=0}^{j+1} c_i * |p_i| = \sum_{i=0}^{j} c_i * |p_i| + c_{j+1} * |p_{j+1}| = cf_j + c_{j+1} * |p_{j+1}|$$

Por lo que dichas variables, c y cf se calculan simultáneamente recorriendo linealmente la ordenación de la partición.

La complejidad de generar todas las particiones de un conjunto A está dada por la cantidad de particiones de un conjunto de tamaño n, que está relacionada con los números de Bell, que satisfacen la siguiente fórmula de recurrencia:

$$B_n = \sum_{k=0}^{n} \binom{n}{k} B_k$$

Por lo tanto, la complejidad de calcular el costo de todas las particiones ordenadas de A, siendo A un conjunto de n elementos es: B_n. A continuación, se calculan las permutaciones entre los elementos de las particiones, una partición tendrá a lo sumo n elementos, por lo que la cantidad de permutaciones será menor o igual a $n!$. Después, se recorren linealmente las permutaciones para calcular el costo de los subconjuntos formados y, como se mencionó anteriormente, se tiene un máximo de n elementos, por lo que $O(n)$ será la complejidad. Finalmente, el costo total del algoritmo será $O(B_n * n! * n)$

III. Fuerza Bruta Mejorado

Una de las posibles mejoras que podría tener el algoritmo anterior es intentar determinar el orden en que los subconjuntos se pueden ordenar dada una partición. En esto consiste el próximo enfoque para la resolución del problema.

Al igual que en el algoritmo anterior se buscarán todas las particiones del conjunto, pero esta vez se seguirá el siguiente enfoque para determinar las posiciones en las que se pondrán los subconjuntos: estos se ordenarán según el mayor elemento de cada subconjunto, de menor a mayor. Para demostrar la correctitud del razonamiento anterior demostremos el siguiente lema:

Lema 1

Sea $P = \{S_1, S_2, ..., S_m\}$ una partición óptima, entonces se cumple que los subconjuntos S_i, con $i = \overline{1, n}$ están ordenados de menor a mayor según su máximo elemento.

Si P es óptimo, entonces es la partición que tiene menor costo. Supongamos que P no cumple lo planteado, es decir, no estará ordenado, por lo que existirá una inversión: $P = \{S_1, S_2, ..., S_i, S_{i+1}, ..., S_m\}$, con $max(S_i) > max(S_{i+1})$. Formemos entonces la partición P' que tendrá los mismos subconjuntos que P, pero crearemos otro subconjunto S'_i que tendrá la misma cantidad de elementos que S_i pero con los mínimos elementos que pertenecían a S_i y S_{i+1}, en S'_{i+1} pondremos los máximos. Por lo tanto, $|S_i| = |S'_i|$ y $|S_{i+1}| = |S'_{i+1}|$, además, como el máximo elemento de ambos estaba en S_i, y ahora estará en S_{i+1}, entonces $max(S'_{i+1}) = max(S_i)$

Por la definición de costo, el costo de la partición P' es $C'_P = \sum_{k=1}^{m} c'_{S_k}$, donde c'_{S_k} es el costo del subconjunto S_k definido como: $c'_{S_k} = |S_k| \sum_{l=0}^{k} max(S_l)$. Se tendrá entonces que:

$$C'_P = \sum_{k=1}^{m} c_{S_k} = \sum_{k=1}^{i-1} c_{S'_k} + c_{S'_{i+1}} + c_{S_i} + \sum_{k=i+2}^{m} c_{S_k}$$

Además, el costo de la partición óptima P será:

$$C_P = \sum_{k=1}^{m} c_{S_k} = \sum_{k=1}^{i-1} c_{S_k} + c_{S_{i+1}} + c_{S_i} + \sum_{k=i+2}^{m} c_{S_k}$$

Observemos que por la definición de costo, el costo de todos los subconjuntos antes de S_i y después de S_{i+1} no se verán afectados, por lo que:

$$\begin{aligned} C_P - C'_P &= (\sum_{k=1}^{i-1} c_{S_k} + c_{S_{i+1}} + c_{S_i} + \sum_{k=i+2}^{m} c_{S_k}) - (\sum_{k=1}^{i-1} c_{S_k} + c_{S'_{i+1}} + c_{S'_i} + \sum_{k=i+2}^{m} c_{S_k}) \\ &= (c_{S_{i+1}} + c_{S_i}) - (c_{S'_{i+1}} + c_{S'_i}) \end{aligned}$$

Para demostrar que el costo de P' es menor que el costo de P, entonces se debe demostrar que:

$$C_P - C'_P > 0 \iff (c_{S_{i+1}} + c_{S_i}) - (c_{S'_{i+1}} + c_{S'_i}) > 0 \iff c_{S_{i+1}} + c_{S_i} > c_{S'_{i+1}} + c_{S'_i}$$

Por lo tanto, demostremos que $c_{S_{i+1}} + c_{S_i} > c_{S'_{i+1}} + c_{S'_i}$
El costo $c_{S'_{i+1}}$ del subconjunto S'_{i+1} y el costo S'_i denotado como $c_{S'_{i+1}}$ en P' cumplirá que:

$$c_{S'_i} = |S'_i|(\sum_{k=0}^{i-1} max(S_k) + max(S'_i)$$

$$c_{S'_{i+1}} = |S'_{i+1}|(\sum_{k=0}^{i-1} max(S_k) + max(S'_i) + max(S'_{i+1}))$$

Por lo que:

$$c_{S'_i} + c_{S'_{i+1}} = (|S'_{i+1}| + |S'_i|) * \sum_{k=0}^{i-1} max(S_k) + |S'_{i+1}| * max(S'_{i+1}) + (|S'_{i+1}| + |S'_i|) * max(S'_i)$$

Mientras que en P el costo de los subconjuntos S_i y S_{i+1}, c_{S_i} y $c_{S_{i+1}}$ cumplirán que:

$$c_{S_i} = |S_i|(\sum_{k=0}^{i-1} max(S_k) + max(S_i))$$

$$c_{S_{i+1}} = |S_{i+1}|(\sum_{k=0}^{i-1} max(S_k) + max(S_i) + max(S_{i+1}))$$

Luego, las suma de ambos costos será:

$$c_{S_i} + c_{S_{i+1}} = (|S_{i+1}| + |S_i|) * \sum_{k=0}^{i-1} max(S_k) + |S_{i+1}| * max(S_{i+1}) + (|S_{i+1}| + |S_i|) * max(S_i)$$

Luego, $c_{S_{i+1}} + c_{S_i} > c'_{S_{i+1}} + c'_{S_i}$, ssí:

$$|S_{i+1}| * max(S_{i+1}) + (|S_{i+1}| + |S_i|) * max(S_i) > |S'_{i+1}| * max(S'_{i+1}) + (|S'_{i+1}| + |S'_i|) * max(S'_i)$$
$$|S_{i+1}| * max(S_{i+1}) + |S_{i+1}| * max(S_i) + |S_i| * max(S_i) > |S'_{i+1}| * max(S'_{i+1}) + |S'_{i+1}| * max(S'_i) + |S'_i| * max(S'_i)$$

Se tendrá que la cardinalidad de ambos subconjuntos se mantendrá igual, es decir, $|S_{i+1}| = |S'_{i+1}|$ y $|S_i| = |S'_i|$. Además, tendremos que $max(S'_{i+1}) = max(S_i)$ ya que los máximos entre S_{i+1} y S_i se intercambiaron y, como $max(S_{i+1}) < max(S_i)$, entonces en S_{i+1} todos los elementos serán menores que $max(S_i)$, por lo que $|S'_{i+1}| * max(S'_{i+1}) = |S_{i+1}| * max(S_i)$, y al simplificar este término en la expresión e igualar las cardinalidades nos queda:

$$|S_{i+1}| * max(S_{i+1}) + |S_i| * max(S_i) > |S_{i+1}| * max(S'_i) + |S_i| * max(S'_i)$$

Por lo que esta es la expresión simplificada que se debe demostrar. Notemos que se tiene que $|S_{i+1}| * max(S_{i+1}) \geq |S_{i+1}| * max(S'_i)$, ya que S'_i contiene los menores elementos, y $max(S_i) > max(S'_i)$ ya que el máximo del conjunto S_i, que era el máximo entre todos los elementos entre S_i y S_{i+1} ya no estará en S'_i, por lo tanto:

$$|S_{i+1}| * max(S_{i+1}) + |S_i| * max(S_i) > |S_{i+1}| * max(S'_i) + |S_i| * max(S'_i)$$

Lo que demuestra que $C_p > C'_p$ y contradice que la partición P sea óptima. Por lo tanto, lo supuesto es falso y P es una permutación que cumple que los subconjuntos S_i, con los $i = \overline{1, n}$ están ordenados de menor a mayor según su máximo elemento.

Luego, se calculará el costo de cada una de estas particiones con el algoritmo explicado en la solución de fuerza bruta.

Generar todas las particiones, es, como en el caso expuesto previamente, $O(B_n)$, para un conjunto de n elementos. A diferencia del algoritmo anterior se ordenan los elementos de las particiones según el máximo elemento de estas. Determinar cuál es el máximo elemento de los subconjuntos requiere recorrer todos los elementos del conjunto A, ya que dichos subconjuntos serán disjuntos, por lo que es $O(n)$. Ordenar una partición de m elementos será $O(m * log(m))$, a lo sumo las particiones tendrán n elementos, así que el costo será $O(n * log(n))$. Por lo tanto, en total la complejidad de la ordenación serán $O(n * log(n) + n) = O(n * log(n))$. Luego, se calculará el costo de cada una de las particiones, y esto es $O(n)$, por lo que la complejidad total del algoritmo será $O(B_n * nlog(n) * n) = O(B_n * n^2 log(n))$

A pesar de que se disminuye la complejidad del algoritmo anterior, el costo sigue siendo alto, por lo que ya después de analizar bien el comportamiento de las soluciones y las características del ejercicio desarrollamos el siguiente algoritmo.

IV. Primera Solución Dinámica

El segundo intento de resolución de nuestro problema fue a través de una dinámica que tuvo como idea principal calcular el costo de los subconjuntos de nuestro conjunto principal calculando primero el costo de los subconjuntos anteriores.

Primero, añadir que este enfoque se basa en el siguiente teorema:

Teorema

Sea A un conjunto de enteros, la partición de A de menor costo en ordenación por el máximo elemento de cada subconjunto (condición que debe cumplir por **Lema 1**), de aquellas que solamente difieren por la distribución de los elementos de A en los subconjuntos, es: $P \mid \forall i : 0 \leq i \leq n-1 :$ máx(P_{i+1}) − máx(P_i) es máxima, donde P_i representa el i-ésimo subconjunto de la partición P.

No es posible aumentar la diferencia entre los máximos de cada subconjunto sin incumplir el orden de estos, demostrado en el **Lema 1**

Supongamos que existe P tal que $|P| = p$ y para cada subconjunto de P, P_i, se cumple: $|P_i| = p_i$, donde $\sum_{i=0}^{p-1} |P_i| = |P|$, donde el costo de P es mínimo para valores fijos de $|P|$ y de $|P_i|$, existe un par P_i, P_{i+1} tal que $\text{máx}(P_{i+1}) - \text{máx}(P_i)$ no es máxima, en P_{i+1} existe un elemento a tal que $a < \text{máx}(P_i)$, si se intercambian estos valores, entonces el máximo de P_i disminuirá, resultando que la partición resultante es de menor costo para P y las cardinalidades de los subconjuntos.

Basados en el razonamiento previo, para los posibles valores de $|P|$ y $|P_i|$, la partición de menor costo es aquella que cumple con el lema anterior. Ahora probemos que esta partición constituye una ordenación de los elementos de A. Para ello, dada la cardinalidad de la partición y la de cada uno de los subconjuntos que están en esta, pasemos a formar una partición que cumpla con el lema anterior; coloquemos el $\text{máx}(A)$ junto a los $|P_{p-1}|$ elementos consecutivos del conjunto A ordenado de forma decreciente. Luego analicemos el máximo del conjunto $A - P_{p-1}$ y repitamos el proceso que se siguió para la partición anterior, si continuamos llenando los subconjuntos de P, resulta que la partición cumple con el lema, ya que no es posible disminuir la diferencia entre los máximos de subconjuntos consecutivos, acorde a la distribución de los elementos de A, la partición estará formada por secuencias consecutivas de elementos de una ordenación creciente de A

A continuación, nuestro algoritmo lo que hace es, dado un conjunto ordenado $A = \{a_1, a_2, ..., a_n\}$ determinar cuál es la mejor manera de particionar este subconjunto cogiendo elementos consecutivos, es decir, cuál es el menor costo posible. Para esto, en nuestra dinámica cada elemento en la posición i, j cumple la siguiente definición:

$dp[i, j]$: menor costo del subconjunto que comienza en a_j y termina en a_{i+j} (por lo que tendrá tamaño $i + 1$)

Notemos que el costo de los n primeros subconjuntos no depende del costo de los subconjuntos anteriores, por lo que nuestra solución es, dado el conjunto $A = \{a_1, a_2, ..., a_n\}$ determinar primero cuál es la mejor manera de particionar los subconjuntos $\{a_1\}$, $\{a_1, a_2\}$, ..., $\{a_1, a_2, ..., a_{n-1}\}$, $\{a_1, a_2, ..., a_{n-1}, a_n\}$ para que estos subconjuntos obtengan menor costo.

Por ejemplo, dado el subconjunto $A = \{1, 2, 3, 4\}$ las posiciones de la matriz dp representarán los siguientes subconjuntos:

$$\begin{bmatrix} \{1\} & \{2\} & \{3\} & \{4\} \\ \{1,2\} & \{2,3\} & \{3,4\} & 0 \\ \{1,2,3\} & \{2,3,4\} & 0 & 0 \\ \{1,2,3,4\} & 0 & 0 & 0 \end{bmatrix}$$

El costo de los subconjuntos en la columna j dependerán de subconjuntos que se formaron anteriormente, exactamente, notemos que solo tiene sentido considerar aquellos que se encuentran en la diagonal $j - 1$ para formar subconjuntos que sean disjuntos.

Se intenta minimizar el costo de los subconjuntos, así que se debe considerar el costo de todos los elementos de la diagonal $j - 1$ y elegir aquel cuyo costo sea menor. Por lo tanto, la forma de obtener $dp_{i,j}$ es a partir de la definición de costo, que recordemos que es para un subconjunto S_i:

$$c_{s_i} = |S_i| * \sum_{k=0}^{i} max(S_k) \tag{1}$$

Por lo tanto, en la dinámica se guardará un expresión similar, que es:

$$dp[i,j] = (i+1) * (min(\frac{dp[k,m]}{k+1}) + A[i+j])$$

Donde k, m están corriendo por la diagonal $j-1$
Observemos las similitudes con la definición de costo:

1. En $dp_{i,j}$ estarán los subconjunto de tamaño $i+1$, por lo que $i+1 = |S_i|$
2. Se debe calcular luego $\sum_{k=0}^{i} max(S_k)$, pero despejando este término en (1) se tiene que:

$$\sum_{k=0}^{i} max(S_k) = \frac{c_{s_i}}{|S_i|}$$

Observemos además que la sumatoria buscada es igual a la sumatoria del subconjunto anterior a él más el máximo del subconjunto actual y recordemos que como el conjunto A se encuentra ordenado, entonces el máximo elemento del subconjunto será el último, y este se encuentra en la posición A_{i+j}. Por lo que si el subconjunto anterior en la partición de A es el que está representado en la posición $dp[k,m]$, entonces:

$$\sum_{k=0}^{i} max(S_k) = \frac{dp[k,m]}{k+1} + A[i+j]$$

Luego, se quiere elegir el conjunto que tenga menor costo, por lo que de todas las sumatorias de los máximos se elige aquella cuya sumatoria sea menor, es decir, con mínimo $\frac{dp[k,m]}{k+1}$

Todos los términos en un estado i, j dependen de la diagonal $j-1$ que se calculó anteriormente, y la matriz dp fue pensada para que se llenara diagonalmente, pero resulta más fácil llenarla verticalmente (de arriba hacia abajo), ya que todos los elementos en una misma columna dependen de la misma diagonal, que empieza en la columna anterior, por lo que este fue el método llevado a cabo para llenar la diagonal. Además, notemos que en una diagonal, el elemento mínimo siempre será el mismo, por lo que solo es necesario calcularlo una sola vez.

A continuación, el resultado final estará en la última diagonal, que tendrá el costo de los subconjuntos que contienen el último elemento del conjunto A, por lo que todos los números se encontrarán representados en los subconjuntos disjuntos seleccionados.

Después para hallar el costo de la partición y quedarnos con aquellos subconjuntos que tengan menor costo se deberá recorrer la diagonal, buscar el costo de los subconjunto elegidos y quedarnos con la partición que menor costo tenga. Para facilitarnos este cálculo, se lleva una lista que tiene según cada columna cuál es la posición en la que está el menor costo elegido de la diagonal que le corresponde a esa columna. Con dicha lista, se recorren las posiciones hacia atrás para calcular el costo de una partición, este recordemos que es:

$$C = \sum_{k=0}^{m} c_{S_k}$$

Lo que se traduce fácilmente en la sumatoria del valor de $dp[i, j]$ para todos los subconjuntos elegidos en la partición.

El problema con este procedimiento es que para minimizar el costo de la suma de los subconjunto no basta con minimizar el costo de cada uno de los subconjuntos. Es decir, la definición de costo es una sumatoria, y para minimizarla no siempre es factible minimizar el costo de cada uno de sus componentes. Lo que pasa con el algoritmo es que hay decisiones que toma para minimizar el costo de un conjunto que luego influyen en la elección de otro subconjunto cuyo costo es mayor.

A pesar del fracaso de está dinámica, la transformación para una solución que resolviese este problema no fue muy difícil, esta solución será expuesta a continuación.

V. Segunda Solución Dinámica

El fallo en la solución anterior es que estábamos basando nuestra selección en el costo individual de los subconjuntos sin tener en cuenta el costo que la suma de ellos ocuparía en la solución final, es decir, sin considerar el costo de la partición que se estaba formando. Pero una vez se tiene el costo de los subconjuntos resulta fácil llevar otra matriz que acumule el costo de la partición que se lleva hasta ahora con los conjuntos ordenados. La diferencia es que, esta vez, la decisión de cuál es el mejor subconjunto a elegir según los subconjuntos que se tienen hasta el momento se basa en el costo de la partición.

Llamaremos a la primera matriz acumulativa, que tendrá pocas diferencias con la matriz del ejercicio anterior, $dpAcc$. Esta seguirá llevando el costo del subconjunto elegido en la partición, es decir, tendrá como expresión:

$$dpAcc[i, j] = (i + 1) * (\frac{dp[k, m]}{k + 1} + A[i + j])$$

Donde $dp[k, m]$ correrá por la diagonal $j - 1$ y será la matriz que optimice el costo de la partición. La elección del subconjunto representado por la posición k, m será la misma que se seleccionará por otra matriz acumulativa, representada por dp, esta tendrá como expresión:

$$dp[i, j] = min(dp[k, m] + (i + 1) * (\frac{dpAcc[k, m]}{k + 1} + A[i + j]))$$

Con k, m que seguirá corriendo por la diagonal $j - 1$. Analicemos lo que se plantea en la definición de dp: se apoya en $dp[k, m]$, que estará previamente calculado y representará el costo de la partición hasta el subconjunto representado por la posición k, m; que no es más que el subconjunto que contiene los elementos que van desde la posición m hasta la posición $k + m$. Como la definición de costo no es más que una suma acumulativa, reusar este cálculo es válido. Después, le añadimos el costo que tendría el subconjunto actual de usar este subconjunto y sus subconjuntos anteriores en la partición que se quiere formar. En resumen, se tendrá el costo de la partición formada hasta el subconjunto representado por la posición i, j, que es dada una partición de m elementos:

$$C = \sum_{k=0}^{m} c_{S_k} = \sum_{k=0}^{m-1} c_{S_k} + c_{S_m}$$

De todas estas posibles particiones se elegirá aquella que tenga el menor costo de la partición, ya que es esto lo que se busca minimizar.

Para buscar después la partición generada se siguen guardando las posiciones de las elecciones que se hicieron, a continuación estas se reconstruyen para retornar los subconjuntos seleccionados. La diferencia con el algoritmo anterior es que ya no se deberá calcular el costo de las particiones y determinar cuál es la óptima es más fácil: el costo de las particiones completas se encuentran en la diagonal, por lo que el menor elemento de la diagonal será la que formará parte de la partición óptima. Además, solo se reconstruirá la partición que fue óptima.

Recorrer las matrices dp y $dpACC$ para llenarlas requerirá $\frac{n*(n-1)}{2}$ operaciones, ya que estas matrices son triangulares superiores izquierda. Luego, para completar cada una de estas posiciones se deberá iterar por una diagonal que tendrá a lo sumo n elementos para determinar el costo mínimo. Por lo tanto, llenar las matrices tendrá un costo de $\frac{n^2*(n-1)}{2}$, lo cual da una complejidad a este algoritmo de $O(n^3)$. Después, para la obtención de la partición óptima se itera por la diagonal de dp recorriendo n elementos, por lo que se realizarán n operaciones. La misma cantidad de operaciones se harán para reconstruir la partición ya que la partición tendrá a lo sumo n subconjuntos. Por lo que en total, la obtención y reconstrucción de la partición requerirá $O(n)$ operaciones, y la complejidad del algoritmo será $O(n^3 + n) = O(n^3)$

Demostración de la correctitud

Para demostrar la correctitud del algoritmo, demostremos por doble inducción fuerte que $dp[i, j]$ tiene el menor costo de la partición formada por el subconjunto que empieza en a_j y termina en a_{i+j} y por los otros subconjuntos disjuntos que minimizan el costo de la partición formada a partir de conjunto A hasta la posición $i + j$.

Inducción en j:

Como caso base, tenemos $j = 0$: en ese caso debemos de demostrar que las particiones formadas por los subconjuntos que terminan en la posición i tienen costo mínimo, pero no es posible formar estos subconjuntos con otros de tal manera que sean disjuntos y cumplan con la propiedad del **Lema 2**, así que el costo de la partición es el costo de los subconjuntos.

Supongamos que $dp[i, m]$, con $m < j$ tiene el costo de la partición formada por el subconjunto que empieza en a_m y termina en a_{i+m}, entonces debemos demostrar que $dp[i, j]$ contiene el costo de la partición formada por el subconjunto que empieza en a_j y termina en a_{i+j}, esto lo demostraremos haciendo inducción en i.

Inducción en i: