

RESUMEN-TAIS-para-tests.pdf



beaas_218



Tecnicas Algoritmicas en Ingenieria del Software



3º Grado en Ingeniería del Software



Facultad de Informática
Universidad Complutense de Madrid



Que no te escriban poemas de amor
cuando terminen la carrera ►►►►►►►

Smiley face icon

(a nosotros por
suerte nos pasa)

WUOLAH

Que no te escriban poemas de amor
cuando terminen la carrera ➡➡➡➡➡

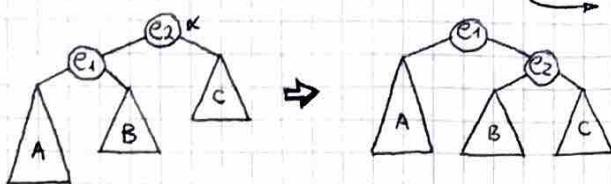


WUOLAH

(a nosotros por suerte nos pasa)

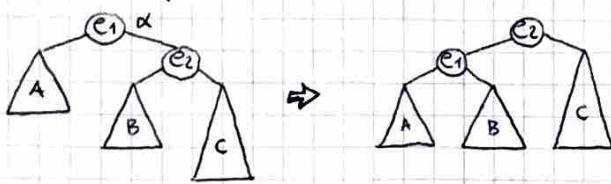
ÁRBOLES AVL (semana 1)

- Condición equilibrio: diferencia altura de los 2 hijos es a lo sumo 1
- Altura $h \in O(\log N)$ ($N = n^{\circ}$ nodos)
- Mínimo n° nodos: $S(h) = S(h-1) + S(h-2) + 1$ ($S(0)=0, S(1)=1$)
- Rotación dcha

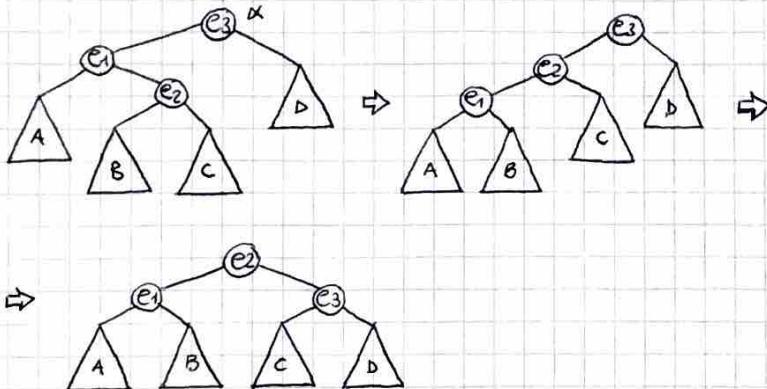


$$= \text{Fib}(h+2) - 1$$

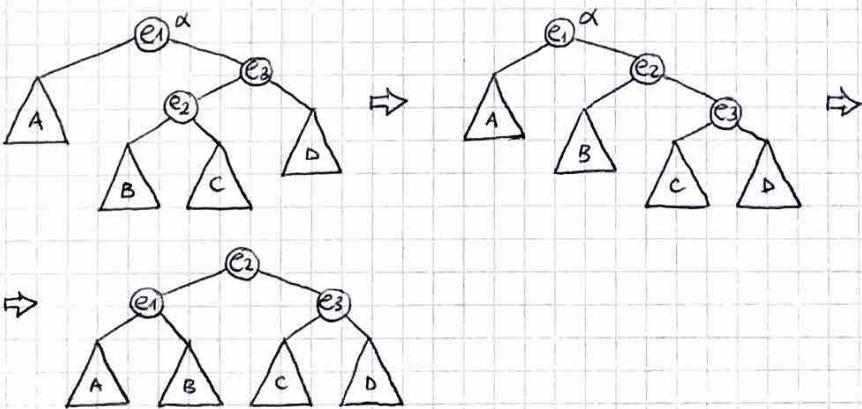
- Rotación izq



- Rotación izq-dcha



- Rotación dcha-izq



- Al rotar, la altura del árbol se mantiene
- α es el primer nodo (desde el insertado a la raíz) que no cumple la condición de equilibrio

No si antes decirte
Lo mucho que te voy a recordar

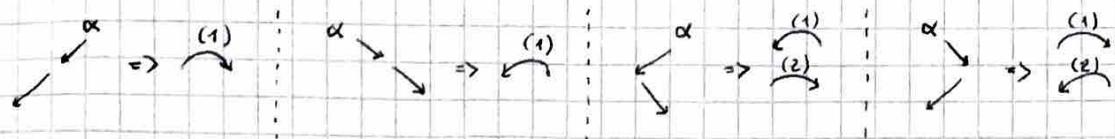
Pero me voy a graduar.
Mañana mi diploma y título he de
pagar

Llegó mi momento de despedirme
Tras años en los que has estado mi
lado.

Siempre me has ayudado
Cuando por exámenes me he
agobiado

Oh Wuolah wuolah
Tu que eres tan bonita

- Tip rotaciones: si desde α el subárbol más alto es



- Al eliminar un nodo pueden hacer falta varias rotaciones para equilibrar
- Si tenemos una secuencia creciente, para que no haya rotaciones hay que insertar la mediana, y luego recursivamente lo mismo cada mitad
- En un ABB (no AVL) el coste de insertar en el caso peor es $O(N)$
- Para recorrer de menor a mayor \rightarrow inorden (coste $O(N)$)
- La mayor diferencia entre el n^o de nodos de los hijos en un árbol AVL de altura h es $2^{h-1} - \text{Fib}(h)$
hijo izq. completo \hookleftarrow \hookrightarrow hijo dcho (lo más pequeño posible)

COLAS DE PRIORIDAD Y MONTÍCULOS (semana 2)

- priority_queue de la STL: cola de prioridad de máximos \Rightarrow top() devuelve el elemento mayor

IMPLEMENTACIÓN	PUSH	TOP	POP
vector desordenado	1	N	N
vector ordenado	N	1	1
montículo binario	$\log N$	1	$\log N$

- Un árbol binario completo ($h \geq 1$) tiene 2^{h-1} nodos en el nivel i , y tiene 2^{h-1} hojas
- Un árbol binario completo ($h \geq 0$) tiene $2^h - 1$ nodos
- La altura de un árbol binario semicompleto de n nodos es $\lfloor \log n \rfloor + 1$
- Montículo mínimo: raíz menor que sus hijos y recursivo (lo análogo con montículos de máximos)
- Montículo \rightarrow árbol binario semicompleto
- Los nodos se numeran de izq a dcha por niveles
- Flotar: al insertar (último nivel, primera pos libre desde la izq) se va intercambiando con su padre hasta que sea mayor que él (lo análogo con montículos de máximos)
- Hundir: al eliminar el más prioritario se pone en la raíz el último nodo y se va intercambiando por su hijo más pequeño hasta que ninguno de los dos sea menor que él (lo análogo con montículos de máximos)
- Un árbol binario semicompleto de k niveles tiene como mínimo 2^{k-1} nodos.
- Un montículo es un árbol binario equilibrado en altura
- Hundir y flotar en el caso peor: $O(\log N)$
- Vector de un montículo: el nodo de la posición i tiene su hijo izq en la posición $2i$ y el derecho en la $2i+1$

HEAPSORT Y COLAS PRIORIDAD VARIABLE (semana 3)

- Heapsort: ordenar un vector \rightarrow lo conviertes en montículo y luego lo inviertes sacando el elemento más prioritario cada vez
- $< a >$: montículo máximo, $> a <$: montículo mínimos (al revés)

Que no te escriban poemas de amor cuando terminen la carrera

(a nosotros por suerte nos pasa)



Ayer a las 20:20

Oh Wuolah wuolitah
Tu que eres tan bonita

Siempre me has ayudado
Cuando por exámenes me he
agobiado

Llegó mi momento de despedirte
Tras años en los que has estado mi
lado.

Pero me voy a graduar.
Mañana mi diploma y título he de
pagar

No si antes decirte
Lo mucho que te voy a recordar



Envía un mensaje...



WUOLAH

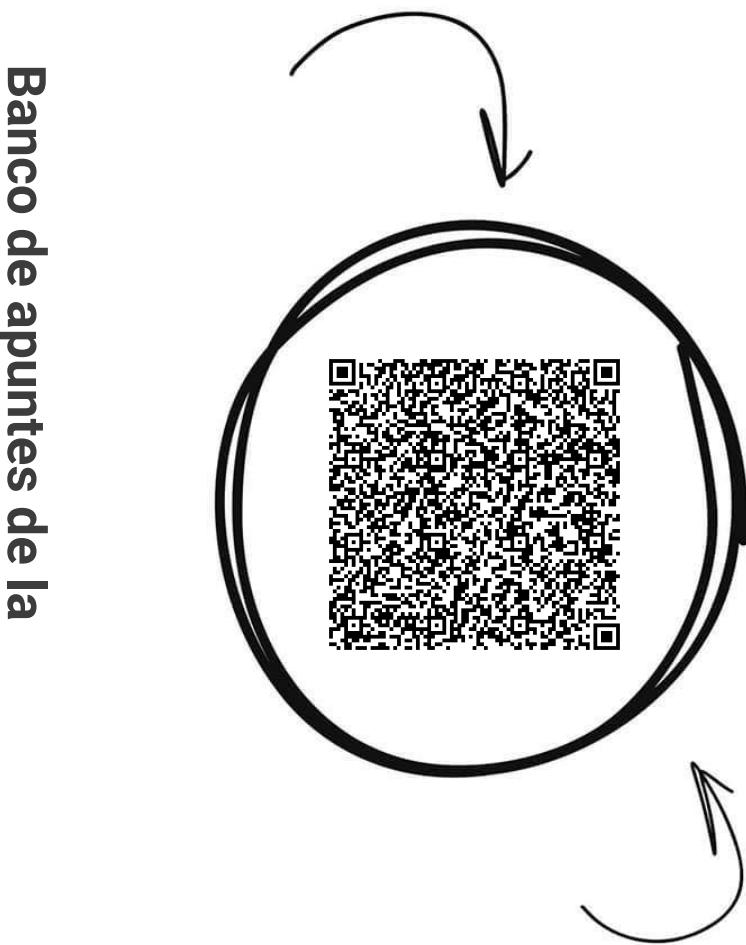


Técnicas Algorítmicas en Ing...



Comparte estos flyers en tu clase y consigue más dinero y recompensas

WUOLAH



- 1** Imprime esta hoja
- 2** Recorta por la mitad
- 3** Coloca en un lugar visible para que tus compañeros puedan escanear y acceder a apuntes
- 4** Llévate dinero por cada descarga de los documentos descargados a través de tu QR

- Convertir en montículo con hundir $\rightarrow O(N)$ (como si cada hundimiento fuera constante)
- 2^a fase ordenar : $O(N \log N)$
- Dos formas de convertir en montículo
 - ↳ Recorrer todos de izq a dcha flotando cada elemento
 - ↳ Recorrer la primera mitad de dcha a izq, hundiendo cada elemento
- Cambiar prioridad : $O(\log N)$
- Heapsort espacio adicional $O(1)$

GRAFOS NO DIRIGIDOS (semana 4)

- Matriz adyacencia
 - ↳ Grado de un vértice : $O(V)$
 - ↳ N° aristas : $O(V^2)$
- Lista adyacentes
 - ↳ Grado de un vértice : $O(\text{grado}(v))$
 - ↳ N° aristas : $O(V+A)$

Representación	Espacio	Añadir arista	Comprobar v y w adyacentes	Recorrer vértices adyacentes a v
matriz adyacencia	V^2	1	1	V
lista adyacentes	$V+A$	1	grado(v)	grado(v)
conjuntos adyacentes	$V+A$	$\log V$	$\log V$	grado(v)
lista aristas	A	1	A	A

- Recorrido en profundidad DFS : $O(V+A)$
- Recorrido en anchura BFS : $O(V+A)$
- Un grafo no dirigido con V vértices puede tener como máximo $V(V-1)/2$ aristas
- En un recorrido en anchura desde un vértice origen, cada nodo se añade como mínimo 0 veces a la cola
- Un **árbol libre** es un grafo no dirigido, **conexo** y **acíclico** (todo par de vértices está conectado exactamente por un camino)
- Ciclo euleriano : para 1 vez por cada arista
- Ciclo hamiltoniano : para 1 vez por cada vértice
- Grafo bipartito : dos conjuntos (disjuntos) de vértices tales que las aristas siempre conectan vértices en conjuntos distintos.
- Recorrido anchura : En cada momento, la cola tiene algunos vértices de un nivel (los que están a distancia d del origen) y posiblemente algunos del nivel siguiente (los que están a distancia d+1)
- Un grafo no dirigido y conexo de V vértices tiene como mínimo $V-1$ aristas
- Un bosque de V vértices y A aristas tiene $V-A$ árboles libres.
- grado(v) está acotado por V
- Recorrido en anchura con matriz adyacencia : $O(V^2)$

GRAFOS DIRIGIDOS (semana 5)

- Grafo dirigido = digrafo
- Tabla costes igual que grafos no dirigidos quitando la fila de conjuntos de adyacentes y cambiando grado(v) por grado-salida(v)
- Un digrafo puede tener como máximo $V(V-1)$ aristas



(a nosotros por suerte nos pasa)

No si antes decirte
Lo mucho que te voy a recordar

Pero me voy a graduar.
Mañana mi diploma y título he de
pagar

Llegó mi momento de despedirte
Tras años en los que has estado mi
lado.

Siempre me has ayudado
Cuando por exámenes me he
agobiado

Oh Wuolah wuolah
Tu que eres tan bonita

- Calcular digrafo inverso: $O(V+A)$
- DFS y BFS iguales que en grafo no dirigido
- Un digrafo con V vértices puede tener como mucho V componentes fuertemente conexas
- Para detectar ciclos dirigidos: vector apilado []
- Ordenación topológica: postorden inverso \rightarrow hacer dfs y cuando terminas llamas dfs recursivo con ady añades el vértice al orden. Luego eso lo inviertes (o directamente vas añadiendo por el principio en vez de por el final)
- Complementario: si el grafo original no tiene una arista, el nuevo sí (y viceversa)
- Coste de calcular complementario con matriz adyacencia: $O(V^2)$
- El postorden inverso de un digrafo acíclico es una ordenación topológica
- Coste de calcular complementario con lista adyacentes
- Un digrafo con V vértices puede tener como máximo $V!$ ordenaciones topológicas (cuando no hay aristas)

CONJUNTOS DISTINTOS, GRAFOS VALORADOS Y ÁRBOL DE RECOBERTO DE COSTE MÍNIMO (semana 6)

- [CD] Búsqueda rápida: vector con representantes id []
- [CD] Unión rápida: árbol, vector p[] para cada nodo que es su padre (él mismo si es raíz)
- [CD] Unión rápida unir(): hay que buscar las raíces de los dos elementos y poner una como hija de la otra
- [CD] Unión por tamaños: al unir el que para a ser hijo es el más pequeño \rightarrow altura logarítmica
- [CD] Unión por tamaños y compresión de caminos: poner a un nodo (y a sus padres) como hijo directo de la raíz
- [CD] $\lg^* N \sim O(1)$
- Costes M llamadas a unir y buscar sobre una partición con N elementos:

Implementación	Complejidad en el caso peor
Búsqueda rápida	$N \cdot M$
Unión rápida	$N \cdot M$
Unión rápida por tamaños	$N + M \log N$
Unión rápida con compresión de caminos	$N + M \log N$
Unión rápida por tamaños y con compresión de caminos	$N + M \lg^* N \rightarrow N + M$

- [GF] Listas adyacencia guardan punteros a aristas (para no guardar la misma arista 2 veces)
- [GF] Mismos costes paraArista $O(1)$ y DFS $O(V+A)$
- [CD] Una estructura de partición de N elementos se pueden hacer a lo sumo $N-1$ llamadas a unir que modifiquen su estructura
- [ARM] Un árbol de recubrimiento es un subgrafo T conexo y acíclico que alcanza todos los vértices del grafo original
- [ARM] Un árbol de recubrimiento tiene $V-1$ aristas. Si eliminar 1 deja de ser conexo, y si añadir 1 deja de ser acíclico.

- [ARM] Grafo conexo \Rightarrow Existe ARM
 - [ARM] Valores distintos de las aristas \Rightarrow ARM único
 - [ARM] Corte = partición de sus vértices en dos conjuntos no vacíos
 - [ARM] Una arista cruza el corte si tiene un extremo en cada conjunto
 - [ARM] Propiedad del corte: dado un corte cualquiera, la arista de menor peso que lo cruza pertenece al ARM
 - [ARM] Kruskal \rightarrow Considera las aristas en orden creciente de coste, y cada arista se selecciona si no crea ciclos
 - [ARM] Para saber si una arista crea ciclos \rightarrow Conjuntos disjuntos
 - [ARM] Kruskal {
 - Construir cola prioridad: $O(A)$ - 1 vez
 - Crear particiones unitarias: $O(V)$ - 1 vez
 - pop(): $O(\log A)$ - A veces
 - unir(): $O(\log * V) \sim O(1)$ - $V-1$ veces
 - unidos(): $O(\log * V) \sim O(1)$ - A veces
 - [ARM] Kruskal $O(A \log A)$ y espacio $O(A)$ (Prim $O(A \log V)$)
 - [ARM] Si no es conexo \rightarrow Busque recubrimiento mínimo
 - [ARM] Si hay valores repetidos no implica necesariamente que haya varios ARM
 - [CD] Detectar si un grafo es acíclico con conjuntos disjuntos (unión rápida por tamaños y compresión de caminos) $O(V + A \log * V)$ (A llamadas, V elementos)
 - [CD] Cuando se busca un elemento acaba estando a profundidad 1 o 2
 - [ARM] Kruskal se puede usar para calcular el " " " coste máximo. (cambiar signo a todas las aristas)

DIGRAFOS VALORADOS Y CAMINOS MÍNIMOS (semana 7)

- El árbol de caminos mínimos no tiene por qué ser un ARM, y los caminos de un ARM no tienen por qué ser de costo mínimo
 - Dijkstra: desde un origen a todos (os demás)
 - Permitimos ciclos, los pesos deben ser no negativos
 - Relajar arista: si $v \rightarrow w$ proporciona un camino más corto que el que tenemos para llegar a w , actualizamos (mejor vamos pasando por v)
 - Dijkstra → Considera los vértices en orden creciente de distancia desde el origen. Añade el vértice al árbol y relaja las aristas que salen de él.
 - IndexPQ para tener los vértices ordenados por distancia y la distancia puede cambiar.
 - Costes }
 - Inicializar vectores: $O(V) - 1$ vez
 - Construir cola prioridad: $O(V) - 1$ vez
 - pop(): $O(\log V)$ - V veces
 - update(): $O(\log V)$ - A veces
 - Coste total $O(A \log V)$ y espacio $O(V)$
 - Insertar y borrar se llama como máximo 1 vez por cada vértice (mientras esté en la cola su prioridad puede cambiar varias veces), y como mínimo 0 veces (los vértices no alcancables no se insertan en la cola)
 - Sólo relaja si el corte es <
 - Grafo conexo no dirigido valores distintos y positivos. Si los valores de incrementan (+) en la misma cantidad, el ARM no cambia (si su valor).

pero si pueden cambiar los caminos mínimos entre cada par de vértices

- Los grafos con aristas de valores negativos tienen camino de corte mínimo entre cada par de vértices si no contienen ciclos de corte negativo
- El camino mínimo no tiene por qué ser único, aunque los valores de las aristas sean todos distintos

ALGORITMOS VORACES (I) (semana 8)

- El método voraz construye una solución a través de una secuencia de pasos, hasta que se alcanza una solución completa al problema. Las elecciones deben ser:
 - factible (satisfacer las restricciones)
 - óptima localmente
 - irrevocable (no se puede rectificar)
- Kruskal y Dijkstra son voraces
- Voraz no funciona para cualquier sistema monetario
- Conjunto de candidatos → seleccionados o rechazados
- Función de selección: de entre los no considerados cuál es el más prometedor
- Test factibilidad: ¿el candidato es compatible con la solución parcial?
- Test solución: ¿la solución parcial ya es completa?
- Función objetivo: asocia un valor a cada solución, hay que encontrar la óptima
- Demostración por reducción de diferenciar: transformar la solución voraz obtenida en una óptima cualquiera
- Voraz sirve para mochila real (objetos fraccionables) pero no para la versión 0/1
- Coste: $O(N \log N)$, $N = \text{nº objetos}$. (Primero ordenación $N \log N$ y luego recorrer N en el caso peor) \rightarrow NO depende del peso límite
- Solución mochila real \geq solución mochila 0/1
- Mochila real: considera los objetos de mayor a menor valor/peso
- En el problema de la mochila real la solución no tiene por qué ser única
- Voraz sirve en los sistemas monetarios (con monedas ilimitadas de cada tipo) donde V_1, \dots, V_n cada V_i es múltiplo del anterior
- Mochila real: puede haber varias soluciones óptimas y no tienen por qué tener el mismo número de objetos enteros.

ALGORITMOS VORACES (II) (semana 9)

- Demostración II: Cualquier solución que no siga la voraz se puede mejorar
- Voraz funciona para tiempo mínimo en el sistema y tareas con plazo y beneficio

PROGRAMACIÓN DINÁMICA (I) (semana 10)

- Números combinatorios: $\binom{n}{r} = \begin{cases} 1 & \text{si } r=0 \vee r=n \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{si } 0 < r < n \end{cases}$
- Para calcular $\binom{n}{r}$ con ascendente se resuelven $(n+1) \cdot (r+1)$ subproblemas distintos



(a nosotros por suerte nos pasa)

- Si no usas dinámica un subproblema puede repetirse $\Theta(2^n)$ veces
- Principio optimidad Bellman: para conseguir una solución óptima basta con considerar subsoluciones óptimas
- Monedas (ilimitadas): $\text{mon}(i,j) = \begin{cases} \text{mon}(i-1,j) & \text{si } m_i > j \\ \min(\text{mon}(i-1,j), \text{mon}(i,j-m_i) + 1) & \text{si } m_i \leq j \end{cases}$
- Monedas (ilimitadas) podemos reducir espacio $O(C)$ incluso cuando se quiere reconstruir la solución
- Para calcular $\binom{n}{r}$ con descendente se resuelven $(r+1) \cdot (n-r+1) - 1$ problemas distintos
- Coste descendente: $N \cdot C$ ($N \cdot C$ llamadas distintas que se pueden hacer, cada una se invoca 2 veces como mucho y la segunda no genera más trabajo)
- Coste ascendente: $N \cdot C$ (N tipos monedas, C cantidad a pagar)
- Monedas: la solución no tiene por qué ser única

PROGRAMACIÓN DINÁMICA (II) (semana 11)

- Con dinámica podemos resolver el problema de la mochila entera
- Mochila entera: si queremos reconstruir la solución no podemos reducir espacio
- Mochila entera: si reducimos espacio recorrer de otra a 1²⁹
- Coste Floyd: $O(V^3) \rightarrow$ No depende de si el grafo es denso o disperso (e.d. del n° de aristas)
- Algoritmo Floyd: Camino mínimo entre todo par de vértices (digrafo valorado). Floyd también sirve cuando hay pesos negativos
- La matriz auxiliar para reconstruir caminos en Floyd es necesaria para poder reconstruir los caminos
- Floyd sirve para detectar ciclos con coste negativo (comprobar al final de cada iteración si algún elemento de la diagonal ppal es < 0)
- Floyd coste espacio adicional $O(1)$
- La reconstrucción de un camino en Floyd tiene coste $O(V)$ (\Rightarrow reconstruir todos los caminos entre todo par de vértices $O(V^2 \cdot V) = O(V^3)$)
- $C^k(i,j) = \text{coste mínimo para ir de } i \text{ a } j \text{ pudiendo utilizar los vértices } 1-k$
- Caminos que pasan por k y que no pasan por k
 $\hookrightarrow C^k(i,j) = \min(C^{k-1}(i,j), C^{k-1}(i,k) + C^{k-1}(k,j))$
- Cuando se rellena la matriz k-éima la diagonal principal no cambia, ni la fila k ni la columna k
- A es la matriz para reconstruir el camino: $A^k(i,j) = \text{vértice anterior a } j \text{ en el camino mínimo de } i \text{ a } j \text{ pudiendo utilizar los vértices } 1-k$

PROGRAMACIÓN DINÁMICA (III) (semana 12)

- Multiplicación encadenada N matrices: $O(N^3)$
- Multiplicación encadenada matricer: no es necesario llenar por diagonales, puede ser ↑ →
- La matriz auxiliar que guarda dónde se colocan los paréntesis permite reconstruir la solución con un coste $O(N)$ en vez de $O(N^2)$
- matricer(i,j) = $\min_{1 \leq k \leq j-1} \{ \text{matrices}(i,k) + \text{matrices}(k+1,j) + d_{i,k}d_{k,j} \}$
- Multiplicación matricer: no se puede reducir espacio
- Justificación con 1 argumento: el número de casos básicos depende tanto de la longitud de la linea del párrafo como de las longitudes de

No si antes decirte
Lo mucho que te voy a recordar

Pero me voy a graduar.
Mañana mi diploma y título he de
pagar

Llegó mi momento de despedirme
Tras años en los que has estado mi
lado.

Siempre me has ayudado
Cuando por exámenes me he
agobiado

Oh Wuolah wuolah
Tu que eres tan bonita

las palabras.

- Si se quiere reconstruir la solución (justificación) no se puede reducir espacio (dar argumentos)
- Justificación $\mathcal{O}(n \cdot L)$ (n : n° palabras)
- Justificación 1 argumento:
 $\text{parrafo}(i) = 0$ si $\text{caben}(i, n) \leftarrow \text{C.B.}$
 $\text{parrafo}(i) = \min_{\substack{i \leq j < n \\ \text{caben}(i, j)}} \{\text{penaliza}(i, j) + \text{parrafo}(j+1)\}$
- Última para mejorar la eficiencia al reconstruir (como con las matrices)
- Justificación + argumento: $\mathcal{O}(N^2)$ y espacio $\mathcal{O}(N)$
- Justificación + argumento: caras báse en las posiciones con un índice mayor