Diseño de algoritmos I
Arocha, Juan 09-10055
Ferrando, Matteo 09-10285

Proyecto 3

Problema 1 ACMAKER

Descripción de la solución

En primer lugar, se tomó la frase como un solo string terminado en el caracter nulo. El algoritmo que se utilizó consta de 4 argumentos:

1. La posición en la frase a analizar (0 indizado)
2. El número de palabras que quedan por leer
3. La letra actual del acrónimo
4. Un booleano que indica si la palabra actual requiere todavía que se le asigne una letra del acrónimo.

Además, el mismo se divide en 5 casos:

1. Si se utilizaron todas las letras del acrónimo, se está viendo la última palabra y la misma ya tiene una letra asignada, se retorna 1 pues es una forma de asignar el acrónimo.
2. Si se usaron todas las letras del acrónimo y quedan palabras por analizar, o si ya se leyó toda la frase y quedan letras por asignar, se retorna 0 pues es una asignación inválida.
3. Si se encuentra un espacio y la palabra actual requiere una letra, se retorna 0 pues toda palabra necesita al menos una letra del acrónimo
4. Si el caracter actual de la frase es igual a la letra actual del acrónimo, se asigna la letra.
5. No se asigna la letra.

Técnica de optimización

Para optimizar el algoritmo, se utilizó la técnica de memoización con índices en:

1. I : posición en la frase
2. L: posición en el acrónimo
3. M: 1 si la palabra actual requiere letra del acrónimo, 0 en otro caso

Estrategia de programación dinámica: Top-Down

Complejidad en tiempo y espacio:

$O(N^2)$ en tiempo pues se debe llenar la matriz de programación dinámica en el peor caso la cual tiene tamaño $2 * N^2$, donde N es el largo de la frase.

$O(N^2)$ en espacio pues se debe almacenar la matriz de programación dinámica que tiene tamaño $2 * N^2$, donde N es el largo de la frase.

Problema 2 BABY

Descripción de la solución

Cada reina se identificó como la fila en la cual está ubicada actualmente, pues el problema dice que no es posible tener dos reinas en una misma fila. Se usó una máscara de bits donde el bit n denota si la fila n ya tiene bien ubicada la reina, y usando esta máscada de bits se fija una reina en una buena posición, y se ubicaba el resto recursivamente.

Técnica de optimización

Para optimizar el algoritmo, se utilizó la técnica de memoización con índices en:

1. Q: Fila de la reina actual
2. M: Máscara de bits que representa el tablero actual

Estrategia de programación dinámica: Top-Down

Complejidad en tiempo y espacio:

O(N * 2^N) en tiempo pues se debe llenar la matriz de programación dinámica en el peor caso, la cual tiene tamaño N * 2^N, donde N es el tamaño del tablero.

O(N * 2^N) en espacio, porque la tabla de programación dinámica es de tamaño N * 2^N.

Problema 3 BORW

Descripción de la solución

Se modeló la solución como un problema de desición. Se pinta de negro (si se puede), de blanco (si se puede) o no se pinta. La idea es mantener la posicion del último blanco pintado y del último negro pintado para poder satisfacer las restricciones a la hora de pintar.

Técnica de optimización

Para optimizar el algoritmo, se utilizó la técnica de memoización con índices en:

1. i: Posición en el arreglo de números
2. lb: Posición del último número pintado de negro
3. lw: Posición del último número pintado de blanco.

Estrategia de programación dinámica: Top-Down

Complejidad en tiempo y espacio:

$O(N^3)$ en tiempo pues se debe llenar la matriz de programación dinámica en el peor caso, la cual tiene tamaño N^3, donde N es el tamaño de la secuencia.

$O(N^3)$ en espacio pues se debe almacenar la matriz de programación dinámica, la cual tiene tamaño N^3, donde N es el tamaño del arreglo de entrada.

Problema 4 MAXWOODS

Descripción de la solución

En primer lugar, notamos que por la definición del problema, el número de fila indica implícitamente la dirección en que se debe recorrer dicha fila, siendo de izquierda a derecha si la fila es par y de derecha a izquiera en caso contrario. Por otro lado, para cada posición en el mapa, se necesita conocer la máxima cantidad árboles cortados en las posiciones anteriores posibles, las cuales incluyen la fila anterior a la posición actual, por lo que para calcular una fila se necesita únicamente la fila anterior y el valor de las posiciones anteriores en la misma fila.

Luego, teniendo en cuenta lo anterior, se usó un arreglo de tamaño M + 2, donde M es el número de columnas del mapa; las 2 posiciones extra son para simular los bordes del mapa, a donde no nos podemos mover.

Se revisa que la posición inicial no esté bloqueada, y de ser así, se inicializa en 0 y el resto en -1; luego, para cada posición en el mapa, se revisa si se puede llegar ahí, y de ser así, se calcula el máximo valor entre sus dos posibles posiciones anteriores.

Técnica de optimización

Para optimizar el algoritmo, se calcula cada fila usando la anterior, manteniendo una variable para el máximo valor encontrado hasta el momento.

Estrategia de programación dinámica: Bottom-Up

Complejidad en tiempo y espacio:

O(N*M) en tiempo pues se debe recorrer todo el mapa, donde N es el número de filas y M el de columnas.

O(M) en espacio pues se debe almacenar una sola fila en memoría.