Segunda Tarea Programada

Resolución de ejercicios de Arrays y Trees mediante programación dinámica

Elaborado por:

Jean Paul Rodríguez Flores 2020059156

Esteban Vargas Quirós 2016015616

Investigación de Operaciones

Grupo 40

Profesor José Castro

I Semestre del 2023

Ejercicio 5 Arrays

a) Crea una matriz de tamaño n x n llamada dp, donde dp[i][j] representa la longitud de la subsecuencia palindrómica más larga en la subcadena s[i:j+1].

Inicialmente, la matriz se llena con ceros, y luego se coloca un 1 en todas las entradas dp[i][i], ya que una cadena de un solo carácter siempre es palindrómica.

Luego, el algoritmo recorre todas las subcadenas más largas en orden creciente de longitud l, y para cada subcadena s[i:j+1], comprueba si los caracteres en los extremos coinciden. Si coinciden, la longitud de la subsecuencia palindrómica más larga en s[i:j+1] es igual a la longitud de la subsecuencia palindrómica más larga en la subcadena s[i+1:j] más 2. De lo contrario, la longitud de la subsecuencia palindrómica más larga en s[i:j+1] es igual a la máxima longitud de la subsecuencia palindrómica más larga en las subcadenas s[i+1:j] y s[i:j-1].

Finalmente, el algoritmo devuelve dp[0][n-1], que es la longitud de la subsecuencia palindrómica más larga en la cadena completa.

b) La idea detrás de esta solucion es que una supersecuencia palindrómica más corta de una cadena dada se puede obtener uniendo la cadena original y su reverso de tal manera que se garantice que la subcadena común más larga de la cadena original y su reverso es palindrómica.

Para encontrar la longitud de la subcadena común más larga, utilizamos una matriz dp de tamaño (n+1) x (n+1), donde n es la longitud de la cadena dada. La matriz dp[i][j] almacena la longitud de la subcadena común más larga de las subcadenas s[0:i] y rev_s[0:j], donde rev_s es la cadena dada revertida.

Luego, recorremos las subcadenas s[0:i] y rev_s[0:j] y comparamos los caracteres correspondientes. Si los caracteres son iguales, la longitud de la subcadena común más larga se incrementa en uno. De lo contrario, tomamos el máximo de la longitud de la subcadena común más larga de las subcadenas s[0:i-1] y rev_s[0:j] y la longitud de la subcadena común más larga de las subcadenas s[0:i] y rev_s[0:j-1].

Finalmente, la longitud de la supersecuencia palindrómica más corta se puede calcular utilizando la fórmula:

longitud_supersecuencia_palindromo_mas_corta = longitud_cadena + longitud_cadena - longitud_subcadena_comun_mas_larga

Donde longitud_cadena es la longitud de la cadena original, y longitud_subcadena_comun_mas_larga es la longitud de la subcadena común más larga calculada en la matriz dp.

c) En este código, utilizamos la función is_palindrome() para verificar si una subcadena dada es un palíndromo. Luego, inicializamos la matriz dp con valores infinitos y establecemos la diagonal principal en 1, ya que cualquier carácter individual es un palíndromo.

Luego, utilizamos dos bucles anidados para recorrer todas las subcadenas posibles de la cadena s. Si la subcadena es un palíndromo, establecemos dp[i][j] en 1. De lo contrario, intentamos

encontrar la mejor división utilizando diferentes valores de k. Para ello, utilizamos un tercer bucle y comparamos el valor actual de dp[i][j] con dp[i][k] + dp[k][j] + 1.

Finalmente, devolvemos el valor almacenado en dp[0][n], donde n es la longitud de la cadena s, que contiene el número mínimo de palíndromos necesarios para descomponer la cadena dada.

Ejercicio 25 Arrays

La matriz dp se utiliza para almacenar los resultados parciales del algoritmo. La posición i en la matriz representa el final de una subsecuencia creciente que termina en el número digits[i]. La longitud de esta subsecuencia se almacena en dp[i].

Para calcular dp[i], se itera sobre todos los números j anteriores a i. Si digits[j] es menor que digits[i], significa que se puede extender una subsecuencia creciente que termina en j agregando digits[i]. En ese caso, se actualiza dp[i] para que tenga la longitud máxima de todas las subsecuencias crecientes que se pueden formar agregando digits[i] a una subsecuencia que termina en un número anterior.

Finalmente, la longitud máxima de todas las subsecuencias crecientes se encuentra buscando el valor máximo en la matriz dp.

Ejercicio 25 Trees

La idea es representar cada empleado como un objeto Employee con un identificador único (id) y una calificación de diversión (fun). Cada objeto Employee también tiene una lista de objetos Employee que son sus subordinados directos (children).

El algoritmo se implementa mediante una función recursiva max_fun_party que toma como argumento el objeto Employee que representa al presidente de la compañía (la raíz del árbol jerárquico). La función devuelve una tupla con dos valores:

1. La suma de las calificaciones de diversión de los empleados que deben asistir a la fiesta.
2. La suma de las calificaciones de diversión de los empleados que no pueden asistir a la fiesta.

Para calcular la primera suma, la función recursivamente visita cada subordinado directo del presidente y suma su calificación de diversión, además de llamar recursivamente a max_fun_party en cada subordinado. Para calcular la segunda suma, la función hace lo mismo, pero sin incluir la calificación de diversión del presidente.

Finalmente, la función devuelve la máxima de las dos sumas.

Ejercicio 26 Trees

La función distribute_gifts recibe como argumento la raíz del árbol y devuelve el mínimo costo de cualquier etiquetado válido del árbol. La idea es utilizar una función recursiva label_node que asigne una etiqueta a cada nodo del árbol y devuelva el costo mínimo de cualquier etiquetado válido para el subárbol que tiene como raíz a ese nodo.

La función label_node recibe como argumentos el nodo actual y la etiqueta del nodo padre. Si el nodo actual es una hoja, simplemente se le asigna la opción de regalo que queda y se devuelve un costo de 0. Si el nodo actual no es una hoja, se prueba con cada opción de regalo para el nodo actual y se llama recursivamente a label_node en cada hijo del nodo actual con la opción de regalo que se está probando. El costo del etiquetado actual es la suma de los costos de los etiquetados de los hijos más el costo de asignar una etiqueta distinta a la del padre al nodo actual. Al final, se devuelve el costo mínimo.

La complejidad de este algoritmo depende del tamaño del árbol. En el peor caso, donde el árbol es una cadena, la complejidad es O(n^3), ya que hay n nodos y se prueban O(n) opciones de regalo para cada nodo, y cada llamada recursiva cuesta O(n) tiempo debido a la suma de los costos de los hijos. Sin embargo, en la práctica, el algoritmo debería ser mucho más rápido que eso, ya que muchas de las opciones de regalo se descartan temprano.

Ejercicio 27 Trees

a) Primero hicimos la función get_subordinates que toma el objeto root y utiliza la recursión para construir el diccionario subordinates que asocia a cada empleado con sus subordinados directos. Cada clave en el diccionario es un objeto Employee y el valor es un conjunto de objetos Employee que representan los subordinados directos del empleado correspondiente.

Definimos A[i][0] como la mínima cantidad de incomodidad que se puede tener al invitar a i nodos en el subárbol con raíz en el nodo i, incluyendo el nodo i, y A[i][1] como la mínima cantidad de incomodidad que se puede tener al invitar a i nodos en el subárbol con raíz en el nodo i, excluyendo el nodo i. Para calcular estos valores, podemos recorrer el árbol en postorden. Para cada nodo i, podemos calcular A[i][0] y A[i][1] como sigue:

- Si el nodo i es una hoja, entonces A[i][0] = max(0, w[i]) y A[i][1] = 0, donde w[i] es la incomodidad asignada al nodo i.
- Si el nodo i tiene un solo hijo, digamos j, entonces A[i][0] = max(0, w[i]) + A[j][1] y A[i][1] = A[j][0], es decir, podemos invitar al nodo i y no invitar a su hijo j, o no invitar al nodo i y sólo invitar al hijo j.
- Si el nodo i tiene dos hijos, digamos j y k, entonces A[i][0] = max(0, w[i]) + A[j][1] + A[k][1] y A[i][1] = min(A[j][0] + A[k][0], A[j][1] + A[k][0], A[j][0] + A[k][1]), es decir, podemos invitar al nodo i y no invitar a ninguno de sus hijos, o no invitar al nodo i y sólo invitar a uno de sus hijos, o invitar a todos los nodos.

  El valor buscado será A[root][0], donde root es la raíz del árbol.

b) Para el caso en que no hay restricciones en la jerarquía de la empresa, podemos modelar el problema como un problema de selección de subconjuntos. Podemos utilizar un algoritmo de fuerza bruta para encontrar el subconjunto de tamaño k con la menor suma de incomodidades. El enfoque de fuerza bruta implica generar todos los subconjuntos posibles de tamaño k y calcular la suma de incomodidades de cada uno. El subconjunto con la menor suma de incomodidades será nuestra solución óptima.