Act 3.4 - Actividad Integral de Árboles (Evidencia Competencia)

De manera individual, desarrolla la solución del siguiente problema:

A causa del COVID-19, muchos puertos han reducido sus operaciones. Debido a esto, y con el fin de dar un mejor servicio, la compañía naviera "International Seas, Ltd." ha pedido a sus clientes que envíen sus pedidos en orden creciente por prioridades. De esta manera, la compañía podrá determinar el correcto orden de envío de todos los pedidos.

En un principio, esto parecía una buena idea. Si bien, mezclar dos listas ordenadas, A = {a1, a2, ..., an} y B = {b1, b2, ..., bm}, es un trabajo sencillo (recuerdas el algoritmo Mergesort). De hecho, el número de comparaciones a realizar serían (m + n - 1) en el peor caso. Pero, pronto empezaron a llegar más y más listas de los diversos clientes y pronto el departamento de Logística y envío se han visto sobrepasado.

Veamos cuál es el problema. Por ejemplo, supongamos que tenemos un conjunto de pedidos de envío {P1, P2, P3, P4, P5} con tamaños {20, 5, 8, 7, 4} respectivamente. Si realizamos las siguientes mezclas:

```
P1 y P2  Z1, serían 20 + 5 - 1 = 24 comparaciones.
Z1 y P3  Z2, serían 24 + 8 - 1 = 31 comparaciones.
Z2 y P4  Z3, serían 31 + 7 - 1 = 37 comparaciones.
Z3 y P5  Z4, serían 37 + 4 - 1 = 40 comparaciones.
```

Lo que nos da un total de 132 comparaciones.

Desarrolla una solución que, dada un conjunto de pedidos y sus tamaños, calcule el menor número de comparaciones posibles:

Solución:

Se desarrolló un programa que utiliza distintas estructuras de datos para utilizar y comparar de manera óptima los datos.

Se decidió implementar un Binary Search Tree, una estructura de datos que ayudaría a organizar los datos de una manera óptima para manipularlo, una vez implementado el BST, se hace un inOrder, algoritmo que ordena de menor a mayor los datos, en este caso el inOrder estaría metiendo los datos a un stack para hacer una suma, una vez se suman los datos se regresa la suma total, esta es interpretada como la menor suma de comparaciones posibles.

La implementación del stack fue algo fundamental a la hora de sumar todos los elementos, esto porque necesitaba al final sumar las comparaciones individuales, las cuales se encontraban en dicho stack.

Test cases:

Test case 1: 5 listas; 20, 4, 5, 7, 8.

```
Listas en inOrden: 4 5 7 8 20
Comparaciones minimas totales: 83
Press any key to continue . . .
```

Test case 2: 7 Listas: 12, 43, 3, 6, 9, 1, 6.

```
Listas en inOrden: 1 3 6 6 9 12 43
Comparaciones minimas totales: 151
Press any key to continue . . .
```

Test case 3: Listas vacías.

```
Listas en inOrden:
Comparaciones minimas totales: 0
Press any key to continue . . .
```

Test case 4: 6 Listas; 1 1 1 1 1 1.

```
Listas en inOrden: 1 1 1 1 1 1
Comparaciones minimas totales: 5
Press any key to continue . . .
```

De esta manera se calcula de manera mas eficiente la manera en la que se le dará prioridad a los pedidos para hacerlo lo más rápido posible.

importancia y eficiencia del uso grafos en una situación problema de esta naturaleza.

Es muy importante que a la hora de diseñar un programa que vaya a manipular datos, implementar estructuras de datos que puedan ayudar a la correcta realización y, sobre todo, a la eficiencia de un programa.

El uso de grafos puede ayudar a esto, tienen distintas maneras de ser implementados para distintas situaciones, optimizan el tiempo y espacio de cada acción, cuando implementamos una estructura de datos, podemos manipular valores de una manera muy sencilla pero que a la ves es ordenada y optima.

En este problema se pueden implementar estructuras de datos de una manera muy natural que ayude a la correcta realización del proyecto, ya que se manipulan datos continuamente, también se necesita optar por una privacidad de los datos.

Algoritmos utilizados:

inOrder: atraviesas desde el subárbol izquierdo a la raíz y luego al subárbol derecho, este algoritmo presenta una complejidad de O(n).

Insertar un número: En este caso, dependiendo del valor del nodo se tiene que insertar en una posición distinta, por lo que a veces recorre todo el árbol hasta encontrar una hoja que se adecúe, por esto, tomando en cuenta un BST lineal, su máxima complejidad será de O(n).