Análisis Laboratorio 10

Grupo 3.

Alejandro Sánchez Caro **Cod** 202420972
Ana María Cuesta Ortiz **Cod** 202420380
Nicolás Sánchez Portilla **Cod** 202420978

Preguntas de análisis

a) ¿Qué estructura de datos subyacente se utiliza comúnmente para implementar una cola de prioridad en Python?

La estructura de datos subyacente que se utiliza comúnmente para implementar una cola de prioridad en Python es el heap binario, implementado sobre un arreglo dinámico. La elección del arreglo se da porque este permite representar un árbol binario completo de forma implícita y eficiente: en lugar de usar nodos con punteros (SingleLinked), las relaciones estructurales se codifican mediante simples operaciones aritméticas sobre los índices. Por ejemplo, el hijo izquierdo de un nodo siempre está en la posición i está en $2i$, y el derecho en $2i + 1$. Gracias a esta representación, el heap aprovecha el acceso directo a cualquier posición del arreglo en tiempo constante, permitiendo que operaciones como inserción y eliminación del mínimo se realicen en sin reordenar toda la estructura. Además, al ser dinámico, el arreglo puede crecer según se necesite, gestionando automáticamente la memoria y evitando costos adicionales por redimensionamientos manuales. Esta combinación de acceso rápido, mantenimiento estructural eficiente y escalabilidad automática hace que el arreglo dinámico sea la estructura de datos subyacente escogida para implementar la cola de prioridad en Python.

b) ¿Cuál es la diferencia entre una cola FIFO tradicional y una cola de prioridad?

La diferencia principal entre una cola FIFO tradicional y una cola de prioridad está en el criterio de orden para el procesamiento de los elementos: en una cola FIFO (First-In, First-Out), los elementos se atienden en el mismo orden en que llegan, sin importar su importancia; en cambio, en una cola de prioridad, cada elemento tiene una prioridad asociada, y siempre se atiende primero el de mayor (o menor) prioridad, sin importar el orden de llegada. Esto hace que las colas de prioridad sean más adecuadas para sistemas donde algunos elementos tienen mayores niveles de urgencia, sin importar el orden de llegada. En conclusión, la gran diferencia entre ambas estructuras radica en el criterio de orden de las colas.

c) ¿Qué módulo proporciona Python para trabajar fácilmente con colas de prioridad?

El módulo que proporciona Python para trabajar fácilmente con colas de prioridad se llama: `heapq`. Este módulo implementa un min heap sobre listas estándar de Python, permitiendo insertar elementos y extraer el de menor valor en tiempo logarítmico. `heapq` no define una clase

propia de cola de prioridad, sino que ofrece funciones como heappush, heappop y heapify para manipular directamente listas como si fueran heaps.

d) ¿Qué ventajas tiene el uso de una cola de prioridad sobre una lista ordenada manualmente?

El uso de una cola de prioridad presenta ventajas significativas sobre una lista ordenada manualmente, principalmente en términos de eficiencia computacional. Mientras que insertar un nuevo elemento en una lista ordenada requiere recorrerla para ubicar la posición correcta (lo que puede tomar tiempo lineal en el peor caso), una cola de prioridad basada en heap permite insertar y extraer el elemento de mayor prioridad en tiempo logarítmico. Además, las colas de prioridad garantizan que el orden de prioridad se mantenga automáticamente tras cada operación, reduciendo errores humanos y simplificando el código. Esto las hace más adecuadas para aplicaciones donde se requieren múltiples inserciones y extracciones eficientes.

e) Si dos elementos tienen la misma prioridad, ¿cómo decide la cola cuál atender primero?

Si dos elementos tienen la misma prioridad, el criterio de desempate depende de la forma en la que está estructurada la cola de prioridad. Por ejemplo, el ya mencionado modulo heapq decide cuál atender primero basándose en el orden en que fueron insertados. Para asegurar un comportamiento predecible, es común incluir un contador incremental o índice de llegada como segundo valor en la tupla (por ejemplo, (prioridad, contador, elemento)), lo que garantiza que en caso de empate en la prioridad, se preserve el orden de inserción y se evite ambigüedad en la comparación. Además, esto es suponiendo un muy importante fundamento de las colas de prioridad: la tricotomía. Es decir, para cada elemento a y b comparados en la lista de prioridad existen únicamente tres opciones: o a>b, o a=b o b<a. Entonces todos los elementos deben ser comparables entre sí.

f) ¿Qué se debe hacer para que los elementos personalizados puedan ser almacenados en una cola de prioridad en Python?

Para que los elementos personalizados puedan ser almacenados en una cola de prioridad en Python, es necesario que estos elementos cumplan la propiedad de tricotomía mencionada en la preguntada anterior. Es decir, tomando como ejemplo un elemento a insertar en un heapq (el módulo de colas de prioridad de Python), este debe ser compatible con funciones como __lt__ (menor que) o __ht__. Además, si lo que se quiere es insertar elementos asociados a una prioridad, se puede almacenar una tupla donde el primer valor sea la prioridad numérica y el segundo el objeto personalizado, como en (prioridad, objeto). Esto permite que heapq ordene los elementos correctamente usando la prioridad como criterio. Esta segunda opción es especialmente útil cuando los objetos no implementan comparadores o cuando se desea separar claramente los datos de su prioridad.

g) ¿Qué situaciones del mundo real se pueden modelar con colas de prioridad? Mencione al menos dos.

Las colas de prioridad son muy útiles en el mundo real pues existen varias situaciones donde se debe ordenar por cierto criterio de urgencia o relevancia. Un primer ejemplo es el sistema de triage en emergencias médicas: cuando llegan múltiples pacientes a un hospital, no se atiende primero al que llegó antes, sino al que presenta una condición más crítica (como un paro cardíaco frente a una fractura leve). Aquí, la cola de prioridad permite gestionar la atención médica basada en niveles de gravedad, asignando prioridad alta a los casos de vida o muerte. Un segundo ejemplo se encuentra en los algoritmos de planificación de procesos en sistemas operativos, donde tareas críticas del sistema (como actualizaciones de seguridad o gestión de memoria) tienen prioridad sobre procesos de usuario (como reproducción de música). El planificador usa una cola de prioridad para decidir cuál proceso accede a la CPU, mejorando el rendimiento global del sistema y evitando bloqueos. En ambos casos, la cola de prioridad garantiza un uso eficiente y justo de los recursos disponibles.

h) En un sistema de atención médica, ¿cómo se puede usar una cola de prioridad para organizar a los pacientes?

En un sistema de atención médica, una cola de prioridad puede usarse para organizar a los pacientes según la gravedad de su condición, de modo que los casos más urgentes sean atendidos primero, independientemente del orden de llegada. A cada paciente se le asigna una prioridad basada en criterios médicos (por ejemplo, nivel de dolor, riesgo vital, signos vitales inestables), y esta prioridad determina su posición en la cola. Así, un paciente con un infarto sería atendido antes que uno con un resfriado leve, asegurando una distribución más justa y eficiente de recursos médicos que son (desafortunadamente) limitados.

i) ¿Cómo afectaría al rendimiento usar una lista simple en lugar de una estructura especializada como heapq para manejar prioridades?

Usar una lista simple en lugar de una estructura especializada como heapq para manejar prioridades afectaría negativamente el rendimiento, especialmente en operaciones repetidas de inserción y extracción. En una lista simple, mantener el orden por prioridad requiere ordenar o buscar manualmente el lugar adecuado, lo cual puede costar tiempo lineal O(n) en cada operación. En cambio, un heap como el que implementa heapq permite realizar inserciones y extracciones del mínimo en tiempo logarítmico (O(log n)), lo que lo hace mucho más eficiente para grandes volúmenes de datos o cuando se requiere alta frecuencia de operaciones.

j) ¿Qué complejidad tiene la inserción y extracción en una cola de prioridad basada en heap?

En una cola de prioridad basada en un heap binario, tanto la inserción como la extracción del elemento de mayor prioridad (en un min-heap, el menor) tienen una complejidad de O(log n), donde *n* es el número de elementos en la estructura. Esta eficiencia se logra gracias a las operaciones de “swim” (al insertar) y “sink” (al eliminar), que reorganizan el heap manteniendo su propiedad estructural sin necesidad de ordenar toda la colección. Esto es posible pues la estructura ya estaba organizada antes de la inserción. Esto representa una mejora significativa frente a estructuras como listas ordenadas manualmente, donde estas operaciones pueden costar hasta O(n).