**METODO DE DISEÑO DE LA INGENIERIA**

**FASE 1: PLANTEAMIENTO DEL PROBLEMA**

El planteamiento general del problema en el contexto de un sistema de gestión de tareas y recordatorios es crear una plataforma que permita a los usuarios gestionar sus actividades diarias de manera eficiente. El objetivo principal es abordar la necesidad común de organizar tareas pendientes y recordatorios de una manera que sea accesible y fácil de usar. Aquí se detallan algunos aspectos clave de este planteamiento:

* Almacenar tareas y recordatorios: Los usuarios deben poder agregar tareas y recordatorios al sistema. Donde se utiliza un identificador único como clave y la información relacionada, como título, descripción, fecha límite y prioridad, como valor. Esto garantiza un acceso rápido y eficiente a las tareas y recordatorios.
* Ordenación de Tareas: Para facilitar la gestión de tareas, se debe implementar una funcionalidad de ordenar. Los usuarios pueden optar por ordenar sus tareas y recordatorios por fecha límite o prioridad. Esto mejora la accesibilidad y la organización de las tareas almacenadas.
* Gestión de Prioridades: Se deben crear dos categorías para las tareas: "Prioritarias" y "No prioritarias". Esto es esencial para garantizar que las tareas críticas no se pasen por alto.
* Tareas No Prioritarias: Para las tareas que no se clasifican como prioritarias, se debe crear una categoría adicional que permita gestionarlas de acuerdo con su orden de llegada.
* Función de Deshacer: Implementar una función de "deshacer" que permite a los usuarios revertir la última acción realizada en el sistema. Se debe realizar un seguimiento de las acciones realizadas.

**FASE 2: RECOLECCIÓN DE INFORMACIÓN**

Una vez establecidas las necesidades y requisitos específicos del problema, el ingeniero empieza a recopilar la información y los datos pertinentes necesarios para resolverlo. En este proceso, es importante considerar el tipo de información necesaria y las técnicas adecuadas para su recopilación, que dependerán de la naturaleza del problema en cuestión.

En la investigación del problema, la búsqueda de información bibliográfica es un apoyo fundamental en esta fase. La revisión bibliográfica permite conocer lo que otros han aprendido sobre problemas relacionados, lo que puede ser muy útil para el ingeniero en la selección de las técnicas y herramientas adecuadas para resolver el problema en cuestión.

Para satisfacer los requerimientos establecidos teniendo en cuenta las necesidades del usuario, se tomará en cuenta un ejemplo de aplicación como lo es Google Keep. Google Keep es una aplicación de gestión de tareas y recordatorios desarrollada por Google que permite crear notas, listas de tareas y recordatorios con funciones de colaboración, etiquetas, colores, búsqueda y archivo. Para abordar problemas de gestión de tareas y recordatorios, se sugieren ideas como desarrollar una aplicación personalizada que se integre con el calendario, implementar un sistema de prioridades, mantener un historial de acciones, diseñar una interfaz de usuario intuitiva. Estas soluciones pueden adaptarse a las necesidades específicas de los usuarios y mejorar la eficiencia en la organización de tareas y eventos.

Por tanto, se debe desarrollar una aplicación que permita a los usuarios almacenar y organizar tareas y recordatorios en una tabla hash. La aplicación contará con una interfaz de usuario intuitiva que facilitará la adición, modificación, eliminación y visualización de tareas, además de la capacidad de ordenarlas utilizando heapsort u algoritmos de ordenamiento distintos. Las tareas podrán categorizarse como prioritarias o no prioritarias, con una cola de prioridades para las primeras y un enfoque FIFO para las segundas. Se mantendrá un registro de acciones de los usuarios, lo que posibilitará deshacer acciones anteriores. La aplicación se sincronizará en múltiples plataformas y se tomarán medidas para garantizar la seguridad de los datos del usuario.

**FASE 3: BUSQUEDA DE SOLUCIONES CREATIVAS**

Desde la perspectiva del uso de estructuras de datos, aquí tienes algunas alternativas para abordar los requerimientos del sistema de gestión de tareas y recordatorios:

Uso de una estructura de datos de tabla hash: Cumple con el requisito de almacenar tareas y recordatorios. Cada entrada en la tabla hash podría ser un objeto que almacena información como título, descripción, fecha límite y prioridad. Para el identificador único, podrías utilizar una clave generada o un identificador único de base de datos.

Lista enlazada con prioridades: En lugar de utilizar una cola de prioridades separada, podrías implementar una lista enlazada para las tareas prioritarias y otra para las tareas no prioritarias. Cada nodo de la lista enlazada podría contener la información de la tarea y un puntero al siguiente nodo. De esta manera, las tareas prioritarias se pueden organizar en función de su importancia al seguir la lista enlazada de prioridades.

Estructura de datos de pila para el seguimiento de acciones: Para implementar el método de deshacer, puedes utilizar una pila (LIFO) para realizar un seguimiento de las acciones realizadas por el usuario. Cada elemento de la pila podría contener información sobre la acción realizada y los detalles de la tarea afectada, lo que permite revertir las acciones de manera eficiente.

Ordenación con estructura de datos específica: Para ordenar las tareas por fecha límite o prioridad, podrías implementar un algoritmo de ordenación personalizado que trabaje directamente con la estructura de datos de la tabla hash. Por ejemplo, puedes diseñar una variante de heapsort que ordene las tareas en función de la fecha límite o la prioridad sin necesidad de copiar los datos a otra estructura.

Uso de índices o mapas adicionales: Para optimizar aún más la búsqueda y ordenación de tareas, se podría mantener índices o mapas adicionales que relacionen la fecha límite y la prioridad con las tareas correspondientes. Esto aceleraría las operaciones de búsqueda y ordenación sin necesidad de recorrer la tabla hash completa.

Lista de Tareas: Una lista de tareas y recordatorios que incluye detalles como título, fecha límite y prioridad. Los usuarios deben poder ordenar esta lista por fecha límite o prioridad con botones de ordenación por algoritmo como BubbleSort y SelectionSort.

Estas alternativas tienen en cuenta el uso eficiente de las estructuras de datos para cumplir con los requisitos del sistema y proporcionar una gestión de tareas y recordatorios efectiva. La elección dependerá en gran medida del lenguaje de programación y las bibliotecas que utilices, así como de tus preferencias de diseño.

**FASE 4: TRANSICIÓN DE LA LLUVIA DE IDEAS A LOS DISEÑOS PRELIMINARES**

Aterrizaremos las ideas en diseños preliminares para visualizar cómo podría ser la estructura de datos de un sistema de gestión de tareas y recordatorios que cumpla con los requerimientos mencionados. Aquí hay algunos elementos clave en estos diseños preliminares:

**Ideas Preliminar:**

* Uso de índices o mapas adicionales: Para optimizar aún más la búsqueda y ordenación de tareas, podrías mantener índices o mapas adicionales que relacionen la fecha límite y la prioridad con las tareas correspondientes. Esto aceleraría las operaciones de búsqueda y ordenación sin necesidad de recorrer la tabla hash completa.

Se descarta la idea anterior por la complejidad que implica mantener distintos tipos de mapas para la correlación de actividades y sus fechas. Para el almacenamiento resulta menos complejo en términos de mantenimiento el uso de una tabla Hash que mantener varios mapas.

* Lista de Tareas: Una lista de tareas y recordatorios que incluye detalles como título, fecha límite y prioridad. Los usuarios deben poder ordenar esta lista por fecha límite o prioridad con botones de ordenación.

**Ideas Final:**

3. Categorización: Una forma de categorizar las tareas como "Prioritarias" o "No prioritarias". Esto podría lograrse con una etiqueta o un icono junto a cada tarea.

4. Formulario de Agregar Tarea: Un formulario donde los usuarios puedan ingresar detalles de una nueva tarea, como título, descripción, fecha límite y prioridad.

5. Función de Deshacer: Un botón o enlace que permita a los usuarios deshacer la última acción realizada, como agregar, modificar o eliminar una tarea.

Estructura de Datos:

En cuanto a la estructura de datos, es fundamental para el almacenamiento eficiente de las tareas y la gestión de prioridades. Aquí hay una descripción de cómo podrían utilizarse las estructuras de datos:

1. Tabla Hash: Cada tarea o recordatorio se almacena en una tabla hash. La clave podría ser un identificador único, como un número o una combinación de letras y números. El valor sería un objeto que contiene información detallada sobre la tarea, como título, descripción, fecha límite y prioridad.

2. Cola de Prioridades: Para las tareas prioritarias, se utilizaría una cola de prioridades. Las tareas se insertan en la cola según su nivel de importancia (por ejemplo, alta prioridad primero). Esto garantiza que las tareas más críticas se manejen primero.

3. Lista enlazada o FIFO: Para las tareas no prioritarias, se podría utilizar una lista enlazada o una estructura FIFO (First-In, First-Out). Esto significa que las tareas no prioritarias se gestionarán en el orden en que se agregaron al sistema.

**FASE 5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN**

NOTA: ESTA EVALUACIÓN SE REALIZARÁ SOBRE MÓDULOS CON UN PORCENTAJE ASIGNADO, MÓDULOS = (Función Deshacer, Orden de tareas no prioritarias, Orden de tareas prioritarias, Almacenamiento de Datos)

Criterio A, la solución propuesta ofrece una solución que cumple los requisitos:

[5] Completamente, [3] Parcialmente, [1] Con dificultades

Criterio B, la solución propuesta es óptima en un contexto real:

[5] Completamente, [3] Parcialmente, [1] Con dificultades

Criterio C, la solución propuesta tiene una complejidad en el peor de los casos.

[5] Constante [4] Logarítmica [3] Lineal [2] N Logarítmica [1] [0] Exponencial

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IDEA / COMPONENT** | **Función Deshacer** | **Orden de tareas no prioritarias** | **Orden de tareas prioritarias** | **Almacenamiento de Datos** |  |
| **IDEA FINAL** | 10% | 20% | 30% | 40% |  |
| **The proposed solution delivers a solution that meets the following requirements** | 5 | 5 | 5 | 5 |  |
| **The proposed solution is optimal in a real-world context.** | 5 | 5 | 5 | 5 |  |
| **Time complexity** | Not applicable | 2 | 2 | 5 | **TOTAL** |
| **Score** | 5 | 4 | 4 | 5 | 4.5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IDEA / COMPONENT** | **Función Deshacer** | **Orden de tareas no prioritarias** | **Orden de tareas prioritarias** | **Almacenamiento de Datos** |  |
| **IDEA PRELIMINAR** | 10% | 20% | 30% | 40% |  |
| **The proposed solution delivers a solution that meets the following requirements** | 5 | 5 | 5 | 5 |  |
| **The proposed solution is optimal in a real-world context.** | 3 | 3 | 3 | 5 |  |
| **Time complexity** | Not applicable | 2 | 2 | 2 | **TOTAL** |
| **Score** | 4 | 3.333333333 | 3.333333333 | 4 | 3.666666667 |

**FASE 6: ELABORACIÓN DE INFORMES Y PLIEGOS DE CONDICIONES**

**Especificación del Problema**

**Problema:** Desarrollar un sistema de gestión de tareas y recordatorios.

**Entradas:**

Agregar, modificar o eliminar tareas.

Ordenar tareas por fecha límite o prioridad.

Deshacer la última acción.

**Salidas:**

Lista de tareas con detalles.

Tareas ordenadas según elección del usuario.

Categorización de tareas en "Prioritarias" o "No prioritarias."

Confirmación de acciones realizadas.

Reversión de la última acción.

**Pseudocodigo:**Estructura Tarea:

Cadena titulo

Cadena descripcion

Fecha fechaLimite

Entero prioridad

TablaHash tareas

ColaPrioridades tareasPrioritarias

Cola tareasNoPrioritarias

Pila accionesRealizadas

Función AgregarTarea(titulo, descripcion, fechaLimite, prioridad):

nuevaTarea = CrearTarea(titulo, descripcion, fechaLimite, prioridad)

tareas[GenerarClaveUnica()] = nuevaTarea

Si prioridad == "Prioritaria":

tareasPrioritarias.Encolar(nuevaTarea)

Sino:

tareasNoPrioritarias.Encolar(nuevaTarea)

accionesRealizadas.Apilar("Agregar tarea", nuevaTarea)

Función ModificarTarea(clave, nuevoTitulo, nuevaDescripcion, nuevaFechaLimite, nuevaPrioridad):

Si clave existe en tareas:

tareaModificada = tareas[clave]

tareaModificada.titulo = nuevoTitulo

tareaModificada.descripcion = nuevaDescripcion

tareaModificada.fechaLimite = nuevaFechaLimite

tareaModificada.prioridad = nuevaPrioridad

accionesRealizadas.Apilar("Modificar tarea", tareaModificada)

Función EliminarTarea(clave):

Si clave existe en tareas:

tareaEliminada = tareas[clave]

Eliminar tarea de tareas

Si tareaEliminada.prioridad == "Prioritaria":

Eliminar tarea de tareasPrioritarias

Sino:

Eliminar tarea de tareasNoPrioritarias

accionesRealizadas.Apilar("Eliminar tarea", tareaEliminada)

Función OrdenarTareasPorFechaLimite():

tareasOrdenadas = OrdenarPorFechaLimite(tareas)

Retornar tareasOrdenadas

Función OrdenarTareasPorPrioridad():

tareasOrdenadas = OrdenarPorPrioridad(tareas)

Retornar tareasOrdenadas

Función Deshacer():

Si accionesRealizadas.NoEstaVacia():

últimaAccion = accionesRealizadas.Desapilar()

Si últimaAccion.accion == "Agregar tarea":

Eliminar tarea de tareas

Si últimaAccion.tarea.prioridad == "Prioritaria":

Eliminar tarea de tareasPrioritarias

Sino:

Eliminar tarea de tareasNoPrioritarias

Sino Si últimaAccion.accion == "Modificar tarea":

tareaOriginal = últimaAccion.tarea

tareas[tareaOriginal.clave] = tareaOriginal

Sino Si últimaAccion.accion == "Eliminar tarea":

tareaEliminada = últimaAccion.tarea

tareas[tareaEliminada.clave] = tareaEliminada

Si tareaEliminada.prioridad == "Prioritaria":

tareasPrioritarias.Encolar(tareaEliminada)

Sino:

tareasNoPrioritarias.Encolar(tareaEliminada)

**Diagrama de flujo**

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

**FASE 7: IMPLEMENTACIÓN DE DISEÑO**

**Implementación en lenguaje de programación (Java)**

**Análisis modifyTaskOrReminder**

**Complejidad Temporal:**

El algoritmo "modifyTaskOrReminder" tiene principalmente una complejidad de tiempo constante (O(1)), pero podría alcanzar una complejidad de O(n) en el peor caso al buscar y modificar elementos en la estructura de datos `taskStorage`. La complejidad depende del rendimiento de `taskStorage`, es la búsqueda en una HashTable la cual es O(n) en el peor caso dado que se busca de manera O(1) dentro del arreglo pero en el peor caso se realiza una búsqueda lineal dentro de la lista enlazada almacenada dentro de cada posición del arreglo.

**Complejidad Espacial:**

El análisis de complejidad espacial del algoritmo "modifyTaskOrReminder" es principalmente O(1) para las variables locales y objetos constantes, como el `Scanner` y las variables locales. Sin embargo, el espacio utilizado por la estructura de datos `taskStorage` y la pila `undoStack` dependerá de la cantidad de tareas, recordatorios y acciones de deshacer almacenadas en ellas. En promedio, el espacio ocupado por `taskStorage` podría ser O(n), donde 'n' es el número de elementos almacenados, y el espacio de `undoStack` será O(m), donde 'm' es el número de acciones almacenadas. En resumen, la complejidad espacial varía según la cantidad de datos y acciones almacenadas en las estructuras de datos mencionadas.

**Análisis searchValue de la clase HashTable**

**Complejidad Temporal:**

Cálculo de la función hash: El primer paso en una búsqueda en una tabla hash es calcular la función hash para la clave que estás buscando. En el caso ideal, el cálculo de la función hash debería ser O(1) y tomar tiempo constante. Sin embargo, si la función hash no está bien diseñada y produce muchas colisiones, el cálculo podría llevar más tiempo.

Acceso a la ubicación de la tabla hash: Una vez que se ha calculado la función hash, acceder a la ubicación de la tabla hash donde se espera que se encuentre la clave también debería ser O(1) en promedio. Sin embargo, si hay colisiones, podría requerir tiempo adicional para resolverlas.

Resolución de colisiones: Si hay colisiones en la ubicación de la tabla hash, es necesario resolverlas. El tiempo requerido para resolver una colisión depende de la estrategia de resolución de colisiones utilizada (por ejemplo, encadenamiento o direccionamiento abierto). En el peor caso, la resolución de colisiones podría llevar tiempo lineal O(n), donde 'n' es el número de elementos en la lista de colisiones.

En general, en un escenario ideal sin colisiones, la complejidad temporal de búsqueda en una tabla hash sería O(1), pero en situaciones de colisión, podría alcanzar O(n) en el peor caso.

**Complejidad Espacial:**

La complejidad espacial de una tabla hash dependerá de su tamaño y la cantidad de elementos almacenados. En promedio, si la tabla hash se dimensiona adecuadamente, la complejidad espacial es O(n), donde 'n' es el número de elementos.

Además, el espacio ocupado por la estructura de datos de colisiones (por ejemplo, listas enlazadas para el encadenamiento) también contribuirá a la complejidad espacial.

En resumen, la complejidad temporal de búsqueda en una tabla hash ideal es O(1), pero puede ser O(n) en el peor caso debido a colisiones. La complejidad espacial es en promedio O(n) considerando el tamaño de la tabla hash y las estructuras de datos de colisiones.