UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE CIENCIAS

Escuela Profesional de Ciencias de la Computación

Algoritmos y Estructuras de Datos

Examen Sustitutorio – Parte II

Fecha: 01.08.2023

Hora: 8:00hrs-9:30hrs

Lugar: Sala 3

Código \_20212196K\_\_\_\_ Apellidos y Nombres Aguilar Vitancio Victor Manuel

1. Implemente la interfaz *Stack* llamada *NoDupStack* que no permita que un duplicado del elemento existente se agregue en la pila. Definimos un elemento e2 como duplicado de e1 si e1.equals(e2) es true. Defina una nueva clase de excepción llamada *DuplicateExceptionAED* y lance esta excepción desde el método *push* cuando se hace un intento por agregar un duplicado. Use una lista enlazada para esta implementación. (4pts)
2. Considere el Caso del bloqueo mortal o puntos muertos: Ref. Código Github https://github.com/inwe-research-group/DrozdekCap8Graphs

Elija una de las tres implementaciones del caso para su demostración paso a paso y documentación detallada del Código. Determine si se ha omitido alguna lógica en el Código que resuelva el caso planteado. La presentación de la prueba será directamente desde github o el drive de google, cuyo link debe consignar como respuesta en el classroom (4pts).

Class DeadlockDetection

1. Se define la clase Transaction que representa una transacción con un nombre y una lista de operaciones.
2. En la función Main, se inicializan las estructuras de datos para almacenar las transacciones “transactions”, los recursos bloqueados “lockedResources”, y el gráfico de espera “waitGraph”.
3. Se lee la entrada del usuario y se procesa en un bucle hasta que se ingresa la palabra "end". Las operaciones son "read", "write" y "end". Cada vez que se realiza una operación, se actualiza la información en las estructuras de datos.
4. Cuando se encuentra una operación "end", la transacción se elimina de la lista de transacciones “transactions” y se liberan los recursos que tenía bloqueados “lockedResources”.
5. Si el recurso está bloqueado, la transacción se coloca en el gráfico de espera “waitGraph”. Si el recurso no está bloqueado, la transacción adquiere el recurso y se agrega a la lista de recursos bloqueados.
6. Una vez que se han procesado todas las operaciones, se busca un ciclo en el gráfico de espera utilizando la clase de detección de ciclos en un gráfico dirigido “findDeadlockCycle”. Si se encuentra un ciclo, significa que hay un deadlock. La función devuelve una lista de transacciones del ciclo.
7. El código muestra si se ha detectado un deadlock y, en caso afirmativo, cuál es la transacción que interrumpió el ciclo.

Un problema importante en el manejo de base de datos es evitar los puntos muertos entre las transacciones. Una transacción es una secuencia de operaciones en la base de datos. En las bases de datos grandes, muchas transacciones pueden ejecutarse al mismo tiempo. Esto puede conducir a inconsistencias si el orden de ejecución de las operaciones no se monitorea. No obstante, este monitoreo puede provocar que las transacciones se bloqueen entre sí, causando por consiguiente un punto muerto. Para detectar un punto muerto, se construye un grafo de espera con el fin de mostrar cual transacción espera a cuál. Use un mecanismo de bloqueo binario para implementar un grafo de espera. En este mecanismo si una transacción T accesa a un registro R, entonces T pone un candado en R y este registro no puede procesarse por ninguna otra transacción antes de que T se termine. Libere todos los candados puestos por una transacción T cuando T termina. La entrada se compone de los comandos siguientes: read(T,A), write(T,A), end(T). Por ejemplo si la entrada es: read(T1,A1), read(T2,A2), read(T1,A2) , write(T1,A2), end(T1)…. Entonces T1 se suspende cuando se intenta ejecutar el paso read(T1,A2) y edge(T1,T2) se crea, debido a que T1 espera a que T2 termine. Si T1 no tiene que esperar, resume su ejecución. Después de que cada grafo se actualiza, busque un ciclo en el grafo. Si un ciclo se detecta, interrumpa la ejecución de la transacción mas reciente T y ponga sus pasos al final de la entrada.

Observe que algunos registros pueden haber sido modificados por una transacción como esta, así que deben restaurarse a su estado anterior a que T iniciara. Pero una modificación como esta podría haber sido usada por otra transacción, la cual también debería interrumpirse. En este programa no aborde el problema de restaurar los valores de los registros (el problema de reducir las transacciones y de poner en cascada esta reducción). Concéntrese en la actualización y monitoreo del grafo de espera. Observe que, si una transacción se termina, su vértice debe eliminarse del grafo, el cual puede ser cualquier otra transacción que este en espera.