Análisis Problema Estudiantes e implementación

Enrique Jara Escobar

Análisis

Mientras que otros algoritmos pierden ciclos de CPU repitiendo cálculos innecesarios, este diseño utiliza la técnica de memorización para registrar cada decisión tomada. Es una estrategia de optimización para asegurar que, si el sistema vuelve a encontrar una clave (nombre) idéntica, no desperdicie recursos volviendo a evaluar su estado. Es la diferencia entre un proceso que escala linealmente y uno que colapsa bajo el peso de la redundancia.

La implementación de la programación dinámica en este sistema se fundamenta en un intercambio estratégico de espacio por tiempo, optimizando el rendimiento mediante memorización. El núcleo algorítmico trasciende la mera evaluación de la condición nota ≥ 3, priorizando una interrogación preliminar a la caché interna (memoria_clasificacion) para evitar cómputos redundantes. Si el subproblema específicamente, la clasificación asociada a un nombre dado ya ha sido resuelto, se recupera el resultado en tiempo constante O(1), obviando por completo la lógica condicional subyacente. En caso de un registro inédito, se procede a su evaluación y persistencia en el diccionario, habilitando accesos futuros acelerados. Este paradigma eleva la eficiencia del sistema de manera progresiva ante datos redundantes, transmutando potenciales derroches computacionales en una ventaja competitiva de escalabilidad.

En cuanto a límites de ejecución y eficiencia de memoria, el análisis de complejidad temporal y espacial denota un perfil predecible y resiliente. El escenario óptimo emerge en entradas con alta densidad de duplicados, donde las recuperaciones se aproximan a la inmediatez absoluta. Por el contrario, el caso adverso se materializa con registros únicos, elevando la complejidad espacial a O(n) en su cota superior. No obstante, esta aproximación supera técnicamente a la recursividad pura, al erradicar riesgos de desbordamiento de pila (stack overflow) y asegurar una latencia mínima garantizada.

Requerimientos

1.1 Requerimientos Funcionales

RF-01

El sistema debe recibir como entrada un número entero n seguido de pares de datos (nombre y nota).

RF-02

El sistema debe validar que n sea un número entero mayor o igual a 0 para asegurar la integridad del proceso.

RF-03

El sistema debe parsear la cadena de entrada para extraer correctamente los nombres y las notas de los estudiantes.

RF-04

El sistema debe clasificar a los estudiantes en "Aprobados" y "Reprobados" utilizando una estructura de Programación Dinámica (Memoización).

RF-05

El sistema debe aplicar el criterio de evaluación donde una nota \ge 3 se clasifica como aprobado y una nota < 3 como reprobado.

RF-06

El sistema debe almacenar en un diccionario los resultados ya calculados para evitar evaluaciones redundantes de nombres duplicados.

RF-07

El sistema debe retornar tres listas: la lista completa de estudiantes procesados, la lista de aprobados y la lista de reprobados.

1.2 Requerimientos No Funcionales

RNF-01

El sistema debe ejecutarse con una complejidad temporal O(n), garantizando una respuesta inmediata incluso con volúmenes de datos considerables.

RNF-02

El sistema debe utilizar memoria eficiente O(n) mediante el uso de diccionarios para el almacenamiento de estados.

RNF-03

El código debe emplear nombres de variables descriptivos y una estructura modular para facilitar su mantenimiento.

RNF-04

El código debe incluir comentarios técnicos que expliquen la lógica de la programación dinámica aplicada.

RNF-05

El sistema debe manejar el flujo de ejecución sin interrupciones abruptas, procesando únicamente la cantidad de datos especificada por n.

2. Historias de Usuario

HU-01 Clasificación de Estudiantes

Como docente, quiero ingresar una lista de estudiantes con sus respectivas notas, para obtener de manera organizada y eficiente quiénes aprobaron y quiénes reprobaron la materia, asegurando que el sistema no trabaje de más si existen nombres repetidos.

Tabla costo/tiempo

Complejidad Temporal	Complejidad Espacial	Costo de Procesamiento
O(n)	O(n)	Bajo
O(1)	O(1)	Mínimo
O(1)	O(1)	Medio
O(n)	O(n)	Optimizado

Grafica mejor/peor caso