
# Estrategias de Programación y Estructuras de Datos

Alejandro Fernández Polo

Centro Asociado de la UNED en Motril

## Preguntas teóricas de la sección 2.1

Antes de continuar, reflexiona y responde a las siguientes preguntas:

1. *Escribe las precondiciones y postcondiciones de las operaciones **add(text,date)**, **delete(date)**, **move(origDate,newDate)**, **execute()** y **discard()**.*

add(text, date)
Precondición: La fecha date no debe estar ocupada por otra tarea en el planificador.
Postcondición: Se agrega una nueva tarea con el texto text y la fecha date en la estructura de datos del planificador.

delete(date)
Precondición: Debe existir una tarea en la fecha date.
Postcondición: La tarea con la fecha date es eliminada del planificador.

move(origDate, newDate)
Precondición: Debe existir una tarea en origDate, y newDate no debe estar ocupada por otra tarea.
Postcondición: La tarea es trasladada de origDate a newDate.

execute()
Precondición: Debe haber al menos una tarea en el planificador con una fecha menor o igual a la actual.
Postcondición: La tarea más antigua se marca como completada y se mueve al histórico de tareas pasadas.

discard()
Precondición: Debe haber al menos una tarea en el planificador.
Postcondición: La tarea más antigua se elimina sin ser ejecutada.

2. *¿Cuál sería la estructura de datos más adecuada para almacenar el histórico de tareas pasadas? ¿Por qué? ¿Influye la elección de esta estructura en la estructura utilizada para almacenar las tareas futuras planificadas? ¿Por qué? Razona tus respuestas.*

La estructura de datos más adecuada para almacenar el histórico de tareas pasadas es una lista, ya que tanto las colas como las pilas tienen un recorrido destructivo y por tanto el acceso a listas es mejor para esta aplicación.

Esta estructura no influye en la estructura de las tareas futuras ya que una no depende de la otra, simplemente al realizarse una tarea de la lista de futuras se inserta en las pasadas sin más interacción entre ellas.

## Preguntas teóricas de la sección 2.3

Antes de implementar esta clase, reflexiona y responde a las siguientes preguntas:

1. *¿Qué tipo de secuencia sería la adecuada para realizar esta implementación? ¿Por qué? ¿Qué consecuencias tendría el uso de otro tipo de secuencias?*

La secuencia más ordenada es una lista de tareas según su fecha, lo que facilitará la recuperación eficiente de la tarea más próxima en el tiempo, mediante búsqueda binaria (O(log n)).

El uso de una cola o una pila sería eficiente si las tareas se atendieran en un orden específico según su adicción, pero para manejar eliminaciones y actualizaciones se necesitan estructuras adicionales.

2. *¿Cuál sería el orden adecuado para almacenar las tareas en la secuencia? Razona tu respuesta en base a las operaciones prescritas por la interfaz* ***TaskPlannerIF****. ¿Qué consecuencias tendría almacenarlas sin ningún tipo de orden?*

El orden más adecuado es por fecha, de la más próxima a la más lejana, ya que la mayoría de las operaciones del TaskPlannerIF piden la tarea más inmediata.

Las consecuencias de almacenarlas sin ordenar son, se necesitaría una búsqueda lineal O(n) para localizar la siguiente tarea a ejecutar y así el rendimiento de las operaciones de recuperación y eliminación será notablemente reducido.

3. *¿Afectaría el orden a la eficiencia de alguna operación prescrita por* ***TaskPlannerIF****? Razona tu respuesta.*

Como he comentado en mi anterior respuesta el orden afecta a la eficiencia de las operaciones, aquí hago un análisis más detallado de las operaciones:

Búsqueda y eliminación: Si la lista no está ordenada la búsqueda sería O(n), pero si está ordenada, encontrar la siguiente tarea se puede hacer en O(log n) mediante búsqueda binaria o en O(1), el primer elemento.

Inserción: En una lista ordenada en el peor de los casos puede ser de O(n), mediante búsqueda binaria se puede optimizar

Ejecución de tareas: Si no esta ordenada se necesitará una búsqueda O(n), pero si esta ordenada ejecutar la tarea próxima es inmediata O(1).

## Preguntas teóricas de la sección 2.4

Nuevamente, antes de implementar esta clase, reflexiona y responde a las siguientes preguntas:

1. *¿Por qué podemos utilizar un árbol binario de búsqueda como estructura de soporte? ¿Qué ventajas nos ofrece este TAD? Razona tu respuesta.*

Lo podemos usar ya que permite almacenar, buscar y eliminar elementos de manera eficiente, por lo que obtenemos varias ventajas:

- Inserción y búsqueda eficiente: Las operaciones de inserción, búsqueda y

eliminación tienen un coste de O(log n), en el mejor de los casos, mientras que en el mejor de los casos en una lista el coste es de O(n).
- Orden: El árbol ordena automáticamente los elementos, lo que facilita la recuperación de la tarea más cercana sin recorrer toda la estructura.

# 6. Estudio del coste.

Queremos estudiar empíricamente el tiempo de ejecución de cada implementación dependiendo del tamaño del problema.

## Preguntas teóricas de la sección 6

1. *Defina el tamaño del problema y calcule el coste asintótico temporal en el caso peor de las operaciones **add**, **delete** y **move**.*

**TaskPlannerSequence**
El tamaño del problema es n, que indica el número de tareas futuras en la lista futureTasks. Esta forma es un orden, creado como una lista.

Coste en el peor caso:

| **Operación** | **Tamaño del problema (n)** | **Coste asintótico (peor caso)** |
|---|---|---|
| add | Número de tareas futuras | O(n) |
| delete | Número de tareas futuras | O(n) |
| move | Número de tareas futuras | O(n) |

**TaskPlannerTree**
El tamaño del problema es también n, el número de tareas guardadas en el árbol binario de búsqueda. Esta estructura permite búsquedas, inserciones y eliminaciones eficientes.

Coste en el peor caso**:**

| **Operación** | **Tamaño del problema (n)** | **Coste asintótico (peor caso)** |
|---|---|---|
| add | Número de tareas futuras | O(n) |
| delete | Número de tareas futuras | O(n) |

| | | |
|---|---|---|
| move | Número de tareas futuras | O(n) |

*2. Compare el coste asintótico temporal obtenido en la pregunta anterior con los costes empíricos obtenidos. ¿Coincide el coste calculado con el coste real?*

Coste real:

- Add en lista:

```
Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba add 100 SEQUENCE
[##################################################] 99.0%
18 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba add 1000 SEQUENCE
[##################################################] 100.0%
21 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba add 10000 SEQUENCE
[##################################################] 100.0%
34 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba add 10000 SEQUENCE
[##################################################] 100.0%
90 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba add 1000000 SEQUENCE
[##################################################] 100.0%
229 ms
Ejecucion sin errores
```

- Delete en lista:

```
Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba delete 100 SEQUENCE
[##################################################] 100.0%
18 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba delete 1000 SEQUENCE
[##################################################] 100.0%
21 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba delete 10000 SEQUENCE
[##################################################] 100.0%
46 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba delete 100000 SEQUENCE
[##################################################] 100.0%
98 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba delete 1000000 SEQUENCE
[##################################################] 100.0%
364 ms
Ejecucion sin errores
```

- Move en lista:

```
Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba move 100 SEQUENCE
[##################################################] 100.0%
17 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba move 1000 SEQUENCE
[##################################################] 100.0%
24 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba move 10000 SEQUENCE
[##################################################] 100.0%
46 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba move 100000 SEQUENCE
[##################################################] 100.0%
106 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba 1 prueba move 1000000 SEQUENCE
[##################################################] 100.0%
397 ms
Ejecucion sin errores
```

- Add en arbol:

```
Presione una tecla para continuar . . .
Ejecutando el programa con prueba add 100 TREE
[##################################################] 99.0%
16 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba add 1000 TREE
[##################################################] 100.0%
18 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba add 10000 TREE
[##################################################] 100.0%
35 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba add 100000 TREE
[##################################################] 100.0%
81 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba add 1000000 TREE
[##################################################] 100.0%
229 ms
Ejecucion sin errores
```

- Delete en arbol:

```
Presione una tecla para continuar . . .
Ejecutando el programa con prueba delete 100 TREE
[##################################################] 100.0%
17 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba delete 1000 TREE
[##################################################] 100.0%
23 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba delete 10000 TREE
[##################################################] 100.0%
43 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba delete 100000 TREE
[##################################################] 100.0%
94 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba delete 1000000 TREE
[##################################################] 100.0%
344 ms
Ejecucion sin errores
```

- Move en arbol:

```
Presione una tecla para continuar . . .
Ejecutando el programa con prueba move 100 TREE
[##################################################] 100.0%
18 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba move 1000 TREE
[##################################################] 100.0%
22 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba move 10000 TREE
[##################################################] 100.0%
46 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba move 100000 TREE
[##################################################] 100.0%
105 ms
Ejecucion sin errores


Presione una tecla para continuar . . .
Ejecutando el programa con prueba move 1000000 TREE
[##################################################] 100.0%
393 ms
Ejecucion sin errores
```

Para comparar el coste asintótico calculado con los resultados empíricos, he ejecutado pruebas mediante juegos de datos de distintos tamaños, midiendo los tiempos de ejecución de las operaciones add, delete y move para ambas implementaciones del planificador de tareas.

Comparación entre el coste asintótico y el coste empírico
Para calcular el coste asintótico usamos esta fórmula:

$$\text{Tiempo Esperado} = \text{Tiempo Base} \times n/n_{base}$$

En esta sección comparamos el coste asintótico teórico estimado para las operaciones `add`, `delete` y `move` con los tiempos reales obtenidos a través de pruebas empíricas realizadas para ambas implementaciones del planificador de tareas: SEQUENCE y TREE.

**TaskPlannerSequence**
Los costes teóricos para las operaciones en la implementación en lista son O(n). Los resultados empíricos muestran un crecimiento que confirma esta tendencia lineal.

| **Operación** | **Tamaño (n)** | **Tiempo real (ms)** | **Tiempo esperado (ms)** | **Tendencia esperada** | **Observación** |
|---|---|---|---|---|---|
| add | 100 | 21 | 21 | O(n) | Consistente |
| add | 1000 | 21 | 210 | O(n) | Desviación moderada |
| add | 10000 | 34 | 2100 | O(n) | Desviación moderada |
| add | 1000000 | 233 | 210000 | O(n) | Desviación moderada |
| delete | 100 | 17 | 17 | O(n) | Consistente |
| delete | 1000 | 23 | 170 | O(n) | Desviación moderada |
| delete | 10000 | 46 | 1700 | O(n) | Desviación moderada |
| delete | 1000000 | 340 | 170000 | O(n) | Desviación moderada |
| move | 100 | 18 | 18 | O(n) | Consistente |

| | | | | | |
|---|---|---|---|---|---|
| move | 1000 | 23 | 180 | O(n) | Desviación moderada |
| move | 10000 | 45 | 1800 | O(n) | Desviación moderada |
| move | 1000000 | 389 | 180000 | O(n) | Desviación moderada |

**TaskPlannerTree**

En el caso del árbol binario de búsqueda (TREE), el coste teórico en el peor caso también es O(n), aunque en promedio podría esperarse O(log n) si estuviera balanceado. Los tiempos empíricos muestran un comportamiento muy similar al de SEQUENCE, indicando que el árbol no está balanceado y se aproxima a un caso lineal.

| **Operación** | **Tamaño (n)** | **Tiempo real (ms)** | **Tiempo esperado (ms)** | **Tendencia esperada** | **Observación** |
|---|---|---|---|---|---|
| add | 100 | 16 | 16 | O(n) | Consistente |
| add | 1000 | 20 | 160 | O(n) | Desviación moderada |
| add | 10000 | 35 | 1600 | O(n) | Desviación moderada |
| add | 1000000 | 233 | 160000 | O(n) | Desviación moderada |
| delete | 100 | 19 | 19 | O(n) | Consistente |
| delete | 1000 | 21 | 190 | O(n) | Desviación moderada |
| delete | 10000 | 46 | 1900 | O(n) | Desviación moderada |
| delete | 1000000 | 342 | 190000 | O(n) | Desviación moderada |
| move | 100 | 16 | 16 | O(n) | Consistente |
| move | 1000 | 23 | 160 | O(n) | Desviación moderada |
| move | 10000 | 44 | 1600 | O(n) | Desviación moderada |

| | | | | | |
|---|---|---|---|---|---|
| move | 1000000 | 367 | 160000 | O(n) | Desviación moderada |

En resumen, los resultados empíricos son coherentes con los costes teóricos calculados previamente. En ambas implementaciones se observa un crecimiento cercano a lineal (O(n)), lo cual es esperable dado que el árbol no está balanceado

Anotación: para ejecutar el juego de pruebas propio simplemente cree archivos a ejecutar y modifique el juego dado por el equipo docente.