Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructura de datos y algoritmos

Pauta Tarea 0

3 de octubre de 2021

2º semestre 2021 - Profesores Yadran Eterovic, Martín Muñoz

Estructuras

- Estaciones (1 pt)
  Las estaciones son **struct**, que contiene un **id**(int) y un arreglo de **andenes**.
- Andenes (1 pt)
  Los andenes son **struct**, estos contienen **id**(int), **tren** y **fila de pasajeros**(Lista ligada).
- Trenes (1 pt)
  Los Trenes son **struct**, estos tiene **id**(int) y los **vagones**(lista ligada).
- Vagón (1 pt)
  El vagón es un **struct** que será usado como nodo en el atributo **vagones** del struct Tren. Lo importante es que vagón tenga un atributo **asientos**(arreglo).
- Filas (1 pt)
  Esta estructura podría no ser necesaria, la fila podría quedar implementada en el **struct** Pasajero como un puntero al siguiente pasajero.
- Pasajeros (1 pt)
  Los pasajeros son **struct**, estos contienen **id**(int), **destino**(int), **categoría**(int) y **siguiente_pasajero**(Lista ligada)

Consideraciones :

Asignar un punto al mencionar por que deben utilizar Lista Ligada y arrays cuando corresponde. (limitaciones de C y/o temas de eficiencia al indexar)

En caso de que el alumno no haga la estructura **Fila**, ya que lo implementó en la estructura **Pasajero** como lista ligada, asignar puntaje completo.

En caso de no tener un aparto para las estructuras pero referenciarlas en los eventos y como influyen en las complejidades asignar puntaje completo.

Complejidades teóricas :

Eventos :

- Para los eventos {Nuevo Tren, Pasajero, Remover, Desaparecer, Separar, Salir, Llegar}
  - Se calcula la complejidad del evento en base a las operaciones, se nota que son operaciones constantes $\mathcal{O}(1)$ para todos los casos. (0.3 pts por evento)
  - Se justifica correctamente que son operaciones constantes. (0.6 pts por evento)
  - El evento pasajero puede tener complejidad $\prime(n)$, con n cantidad de pasajeros en la fila, si justifica bien, es decir que su Lista Ligada es simple y no tiene referencia al tail o head se asigna puntaje completo.
- Para el evento Salir.
  - Se calcula la complejidad en base al estado del evento, es decir esta lleno $\mathcal{O}(n)$ o no está lleno $\mathcal{O}(n + a)$, n número total de pasajeros en al fila n y a cantidad de asientos vacíos. Se llega a una complejidad de $\mathcal{O}(n)$. (0.3 pts)
  - Se justifica la complejidad correctamente, ya sea con una descripción del algoritmo, fragmento del código o mediante palabras (0.6 pts)
  - en caso de tener un atributo que chequee si está lleno o no, complejidad de $\mathcal{O}(1)$ en caso de estar lleno y $\mathcal{O}(a)$ en caso de no estarlo, mención y justificación da pontaje completo
  - no hacer la diferenciación entre estar lleno o no otorga la mitad del puntaje.
- Para el evento Separar.
  - Se calcula la complejidad en base a la cantidad de asientos presentes en el tren, es decir $\mathcal{O}(n)$ donde n es la cantidad de asientos. (0.3 pts)
  - Se justifica la complejidad correctamente, ya sea con una descripción del algoritmo, fragmento del código o mediante palabras (0.6 pts)
  - No llegar a $\mathcal{O}(n)$ con n cantidad de asientos pero justificar una complejidad basada en el número de vagones otorga la mitad del puntaje.
- Eventos Estado Tren - Estación
  - Siendo n cantidad de vagones y m cantidad de asientos hay que llegar a una complejidad de $\mathcal{O}(n \cdot m)$ (0.2 pts por evento)

- justificar correctamente la complejidad, mediante código, palabras o una descripción del funcionamiento. (0.3 pts por evento)
- No llegar a $\mathcal{O}(n \cdot m)$ pero justificar una complejidad basada en el número de impresiones, es decir $\mathcal{O}(n)$ con n cantidad de atributos a chequear otorga mitad del puntaje

- Desaparecer

  - Se calcula la complejidad en base a la cantidad de asientos presentes en el tren y vagones, es decir $\mathcal{O}(nm)$ donde n es la cantidad de asientos y m la cantidad de vagones. (0.3 pts)
  - Se justifica la complejidad correctamente, ya sea con una descripción del algoritmo, fragmento del código o mediante palabras (0.6 pts)
  - No llegar a $\mathcal{O}(nm)$ pero justificar una complejidad basada en el número de solo vagones otorga la mitad del puntaje.

  Consideraciones sobre el puntaje :

  - Si se especifica que tiene algoritmos diferentes pero correctos con la complejidad bien calculada, se asigna puntaje completo
  - Si da la complejidad correcta sin justificar se asigna mitad del puntaje.
  - Si se argumenta correctamente que $O(nm) \sim O(n)$ Entonces se asigna puntaje completo (Una argumentacion correcta seria decir que $m << n$ tanto asi que se observa como contante. O utilizar un n general como el numero total de asientos del tren)

Ejemplo de informe:

Para estas complejidades se asume una implementacion utilizando Listas Ligadas para los Vagones y Array para los asientos. En caso de utilizar una modelacion diferente, ha de estar explicita en el informe. Y las complejidades variarian acorde.

- Nuevo Tren
  Este Evento crea el tren y se le asigna a una variable, el instanciar el tren con sus respectivos vagones y estos mismos con la capacidad se considera una operación constante, por lo que tiene una complejidad de $\mathcal{O}(1)$.

- Pasajero
  Este evento asigna a los pasajeros en los andenes según corresponda y sea posible, en caso de ser premium y haber espacio se sube de una al tren, lo que es una operación constante, en caso contrario debe ser asignado al final de su fila correspondiente. El asignarlos a la fila puede ser una operación constante $\mathcal{O}(1)$ en caso de tener una Lista Ligada con referencias al final de esta, o bien puede ser O(n) con n cantidad de pasajeros en la fila si tenemos una Lista Ligada simple con solo referencia al siguiente.

- Remover
  El evento remover busca sacar a los comercialggles colados, el indexar en arrays y remover un elemento es una operación constante y toma $\mathcal{O}(1)$.

- Salir
  Este evento indica que parte el tren, acá tenemos 2 casos.

  - Está lleno
    En caso de que el tren este lleno, lo cual deberíamos chequear con un atributo el tren simplemente parte, lo que es $\mathcal{O}(1)$. En caso de no tener un atributo y tener que revisar todos los asientes 1 a 1 esto tendría un costo $\mathcal{O}(n)$ con n la cantidad de asientos en el tren. Esta implementación se considera no óptima.

  - No esta lleno
    En caso de no estar lleno y chequearlo con un atributo, $\mathcal{O}(1)$, debemos "sentar.ª las personas que nos quedan, como debemos recorrer una LL esto nos tomara $\mathcal{O}(a)$con a cantidad de asientos vacíos, en caso de no hacer el chequeo $\mathcal{O}(1)$ sino que revisando todos los asientos, tendríamos una complejidad total de $\mathcal{O}(n + a)$ con n número de asientos en al vagón y a cantidad de asientos vacíos.

- Llegar
  Este evento baja a los pasajeros que les corresponde, Para chequear quienes se bajan podemos recorrer todo el vagón lo que toma $\mathcal{O}(n)$ con n cantidad de asientos, el bajar propiamente a un pasajero toma $\mathcal{O}(1)$. Subir a las personas en caso de tener asientos puede ser implementado de 2 formas :

  - Altiro

Cuando bajamos a una persona sabemos que va a quedar un asiento libre, por lo que podemos ahi mismo subir al primero en la fila correspondiente, lo que seria $\mathcal{O}(1)$, el re-ordenar la Lista Ligada de la fila también es $\mathcal{O}(1)$ al cambiar un par de punteros.

- Terminar y re-recorrer.
  Un Naive aproach sería primero bajarlos a todos y luego tener que volver a recorrer los asientos para ver cuales están vacíos, cada vez que vemos un asiento vació

- Desaparecer
  Este evento elimina un tren y todo lo que este contiene, Desaparecer en si tiene una complejidad $\mathcal{O}(1)$, liberar correctamente la memoria de todas las referencias tiene una complejidad $\mathcal{O}(n)$ con n cantidad de pasajeros dentro del tren.

- Separar
  Este evento separa un tren por sus vagones, Separar en si tiene una complejidad $\mathcal{O}(n)$ con n cantidad de asientos dentro del tren. Debido a que Hay que cambiar los vagones de destino del tren de destino según sea por o impar se debe recorrer todos los vagones del tren, pero además se debe cambiar el destino de cada pasajero, por lo que hay que recorrer todos los asientos de cada vagón.

- Estado Tren
  Este evento imprime la información del tren, esto es su id y si los asientos están ocupados o no, en caso de que no estén ocupados se imprime una X o la información del pasajero. Para poder hacer esto, se tienen que recorrer cada vagón(n) y los asientos(m) de dicho vagón, por lo que la complejidad teórica sería $\mathcal{O}(n \cdot m)$.

- Estación
  La complejidad de este evento es $\mathcal{O}(n \cdot m)$, esto ya que debes recorrer los andenes(n) de la estación indicada y para cada anden, debes recorrer la fila de pasajeros(m).