uc3m | Universidad Carlos III de Madrid

Autor: Profesores EDA

Estructura de Datos y Algoritmos. Tema 3 – Análisis de Algoritmos.

QUINTO TRABAJO INDIVIDUAL SEMANAL- Indica el orden de complejidad de los siguientes métodos. En los casos, que corresponda indica también el mejor y peor caso. Justifica tu respuesta:

Método	**O ()**	**Justificación**

El siguiente método de la clase DList, recibe una lista list2, y añade cada uno de los elementos de dicha lista al principio del objeto lista que invoca el método (objeto this). Por ejemplo, si this= {1,3,5} y list2={4,5}, this.merge(list2) tendrá como resultado que el objeto invocante contendrá los siguientes elementos {4,5,1,3,5}
 	
`public void merge(DList list2) {`
`if (!list2.isEmpty()) {`
`  for (DNode node=list2.header.next;node!=list2.trailer;node=node.next){`
`  DNode newNode = new DNode(node.elem);`
`  newNode.next = this.header.next;`
`  newNode.prev= this.header;`
`  this.header.next.prev = newNode;`
`  this.header.next = newNode;`
`  this.size++;`
`  }`
`}`
`}`
	O(n)
	
	+1
	+2+(n+1)+n
	+2*n
	+1*n
	+1*n
	+1*n
	+1*n
	+1*n
	
	Total:
	9n+4
		El mejor caso será que list2 esté vacía y seria constante O(1).
		
		El peor caso es que no esté vacía y tenga n elementos list2. Teniendo que añadir a this.list n elementos.
		
		El peor caso es el que tengo en cuenta para hallar la complejidad.

El método **removeEven** de la clase DList, que elimina todos sus elementos pares.
`public void removeEven() {`
`  for (DNode node= this.header.next;node!= this.trailer;node=node.next){`
`  if (node.elem%2==0){`
`  node.prev.next = node.next;`
`  node.next.prev= node.prev;`
`  this.size--;`
`  }`
`  }`
`}`
	O(n)
	
	+2+(n+1)+n
	+1*n
	+1*n
	+1*n
	+1*n
	
	Total:
	6n+3
		El mejor caso sería que la lista no tuviese ningún elemento, haría el for hasta la comparación y terminaría. O(1)
		
		El peor caso es que la lista tenga n elementos y todos fuesen pares. Este es el calculado.

El método **isEmpty** que comprueba si la lista está vacía o no.
`public boolean isEmpty() {`
`  return (this.header.next == this.trailer);`
`  }`
	O(1)
	
	+1
		Todos los casos son, que esté vacía o llena, y se trata solo de una comparación.

El método **contains** de la clase Queue, que comprueba si un elemento existe o no en la cola.
 	
`public boolean contains (String elem) {`
`  boolean found = false;`
`  for (int i = 0; i < this.size; i++) {`
`  String e = dequeue();`
`  if (elem.equals(e)) found = true;`
`  enqueue(e);`
`  }`
`  return found;`
`}`
	O(n)
	
	+2
	+2+(n+1)+n
	+2*n
	+2*n
	+1
	
	+1
	Total: 6n+7
		El mejor caso sería que la cola esté vacía, el for solo llega hasta la comparación y seguiría. O(1)
		
		El peor caso es que la cola tenga n elementos, tenga que recorrerla entera y no encontrarlo.

El método **insertAt** de la clase DList:
 	
`public void insertAt(int index, Integer elem) {`
`  DNode newNode = new DNode(elem);`
`  int i = 0;`
`  boolean inserted=false;`
`  for (DNode nodeIt = header; nodeIt != trailer && inserted==false; nodeIt = nodeIt.next) {`
`  if (i == index) {`
`  newNode.next = nodeIt.next;`
`  newNode.prev= nodeIt;`
`  nodeIt.next.prev= newNode;`
`  nodeIt.next = newNode;`
`  inserted=true;`
`  size++;`
`  }`
`  ++i;`
`}`
`  if (!inserted)`
`  System.out.println("DList: Insertion out of bounds");`
`}`
	O(n)
	
	+2
	+2
	+2
	+2+(n+1+n+1)+n
	+1*n
	+1
	+1
	+1
	+1
	+1
	
	+1*n
	
	+1
	
	Total: 5n+17
		El mejor caso será que la lista este vacía y solo llegue hasta la comparación con tráiler, saldría del for y haría el if que imprime por pantalla. O(1)
		
		El peor caso será que la lista tenga n elementos y lo tenga que colocar en la última posición, teniendo que recorrerlo entero y entrando en el if, y sumando cada iteración. Luego como lo inserta no entra en el if para imprimir por pantalla.