МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

по дисциплине «АиСД»

на тему: «Линейные динамические структуры»

Выполнил: студент гр. ИП-32

Суховенко Э.С.

Принял: преподаватель

Процкая М.А.

Гомель 2022

**Цель работы:** Изучить основы работы с линейными динамическими структурами (списками, очередями, стеками, кольцевыми буферами и т.д.)

**Вариант 12**

|  |  |
| --- | --- |
| 12 | Из двух однонаправленных списков сформировать новый следующим образом: сперва записать четные по индексу элементы первого списка, затем – четные по индексу элементы второго списка. |

Задание 1. Разработка программы на основе варианта задания

* Написать 2 программы по созданию, просмотру, добавлению, удалению, сортировке и решению поставленной задачи для линейного списка (список, стек и/или очередь).
* В первой в качестве структуры данных использовать линейный список с указателем на следующий элемент, во второй – динамический массив с дополнительным полем – номер следующей записи (или двумя). ОБЯЗАТЕЛЬНО разобраться с сортировкой списков, а во второй программе - с удалением элементов.
* В программе нельзя использовать дополнительные массивы.
* В программе не использовать стандартные типы, использовать массивы и списки List (ArrayList

Задание 2. Практическое задание - Анализ алгоритма задачи

Выполнить анализ временных характеристик выбранной операции для варианты задачи.

Рассчитать:

* асимптотическую оценку сложности
* наилучшую, наихудшую и среднюю оценки

Подготовить отчет по практическому заданию, содержащий:

* псевдокод операции на упрощенном языке
* выкладки расчета асимптотической оценки
* выкладки или результаты тестирования на различных по объему набору данных

Код программы

import { ForwardList } from './lists/forwardList/forwardList'  
import { ArrayList } from './lists/arrayList/arrayList'  
import { IList } from './lists/IList'  
  
const a = [-1, 1, -2, 2, -3, 3, -4, 4]  
const b = [-5, 5, -6, 6, -7, 7]  
  
const mergeLists = <T>(listClass: { new(): IList<T> }, first: IList<T>, second: IList<T>): IList<T> => {  
 const resultList = new listClass()  
 for (let i = 1; i < first.length; i += 2) {  
 resultList.pushBack(first.getValueByIndex(i))  
 }  
 for (let i = 1; i < second.length; i += 2) {  
 resultList.pushBack(second.getValueByIndex(i))  
 }  
 return resultList  
}  
  
console.time('forward list')  
console.log(JSON.stringify(mergeLists(ForwardList, new ForwardList(a), new ForwardList(b)).toArray()))  
console.timeEnd('forward list')  
  
console.time('array list')  
console.log(JSON.stringify(mergeLists(ArrayList, new ArrayList(a), new ArrayList(b)).toArray()))  
console.timeEnd('array list')

import { IList } from '../IList'  
  
export class ArrayList<T> implements IList<T>{  
 array: Array<T>  
  
 length: number  
  
 constructor(array: T[] = []) {  
 this.array = [...array]  
 this.length = this.array.length  
 }  
  
 erase(index: number): void {  
 if (index < 0 || index >= this.length) throw new *Error*('ERASE: Index is out of bound')  
 this.array = [...this.array.slice(0, index), ...this.array.slice(index + 1)]  
 this.length -= 1  
 }  
  
 getValueByIndex(index: number): T {  
 if (index < 0 || index >= this.length) throw new *Error*('GET\_VALUE\_BY\_INDEX: Index is out of bound')  
 return this.array[index]  
 }  
  
 insert(index: number, value: T): void {  
 if (index < 0 || index > this.length) throw new *Error*('INSERT: Index is out of bound')  
 this.array = [...this.array.slice(0, index), value, ...this.array.slice(index)]  
 this.length += 1  
 }  
  
 popBack(): void {  
 if (!this.length) throw new *Error*('POP\_BACK: Array doesn\'t contain any element')  
 this.erase(this.length - 1)  
 }  
  
 pushBack(value: T): void {  
 this.insert(this.length, value)  
 }  
  
 sort(comparator: (a: T, b: T) => boolean = (a: T, b: T) => a < b): void {  
 for (let i = 0; i < this.length; ++i) {  
 let minIndex = i  
 let minValue: T = this.getValueByIndex(i)  
 for (let j = i + 1; j < this.length; ++j) {  
 if (comparator(this.getValueByIndex(j), minValue)) {  
 minValue = this.getValueByIndex(j)  
 minIndex = j  
 }  
 }  
 this.swapByIndex(i, minIndex)  
 }  
 }  
  
 swapByIndex(firstIndex: number, secondIndex: number): void {  
 if (firstIndex < 0 || firstIndex >= this.length) throw new *Error*('SWAP\_BY\_INDEX: First index is out of bound')  
 if (secondIndex < 0 || secondIndex >= this.length) throw new *Error*('SWAP\_BY\_INDEX: Second index is out of bound');  
 [this.array[firstIndex], this.array[secondIndex]] = [this.array[secondIndex], this.array[firstIndex]]  
 }  
  
 toArray(): T[] {  
 return [...this.array.slice(0, this.length)]  
 }  
}

export interface IList<T> {  
 length: number  
 getValueByIndex: (index: number) => T  
 insert: (index: number, value: T) => void  
 erase: (index: number) => void  
 swapByIndex: (firstIndex: number, secondIndex: number) => void  
 sort: (comparator: (a: T, b: T) => boolean) => void  
 pushBack: (value: T) => void  
 popBack: () => void  
 toArray: () => T[]  
}

import { ForwardListNode } from './forwardListNode'  
import { IList } from '../IList'  
  
export class ForwardList<T> implements IList<T>{  
 root: ForwardListNode<T> | null  
  
 length: number  
  
 constructor(array: T[] = []) {  
 this.root = null  
 this.length = 0  
 array.forEach(elem => this.pushBack(elem))  
 }  
  
 private getNodeByIndex = (index: number): ForwardListNode<T> | null => {  
 if (index < 0 || index >= this.length) throw new *Error*('GET\_NODE\_BY\_INDEX: Index is out of bound')  
 let currentIndex = 0  
 let currentNode: ForwardListNode<T> = this.root  
 while (currentIndex != index && currentNode) {  
 currentIndex += 1  
 currentNode = currentNode.next  
 }  
 return currentNode  
 }  
  
 getValueByIndex = (index: number): T => {  
 if (index < 0 || index >= this.length) throw new *Error*('GET\_VALUE\_BY\_INDEX: Index is out of bound')  
 return this.getNodeByIndex(index)?.value  
 }  
  
 insert = (index: number, value: T): void => {  
 if (index < 0 || index > this.length) throw new *Error*('INSERT: Index is out of bound')  
 const newNode = new ForwardListNode(value)  
 if (!this.length) {  
 this.root = new ForwardListNode<T>(value)  
 } else {  
 if (index - 1 >= 0 && index - 1 < this.length) {  
 const prevNode = this.getNodeByIndex(index - 1)  
 const curNode = prevNode?.next  
 prevNode.next = newNode  
 newNode.next = curNode  
 } else {  
 newNode.next = this.root  
 this.root = newNode  
 }  
 }  
 this.length += 1  
 }  
  
 erase = (index: number): void => {  
 if (index < 0 || index >= this.length) throw new *Error*('ERASE: Index is out of bound')  
 if (index - 1 >= 0 && index - 1 < this.length) {  
 const prevNode = this.getNodeByIndex(index - 1)  
 const curNode = prevNode?.next  
 prevNode.next = curNode?.next  
 } else {  
 this.root = this.root?.next  
 }  
 this.length -= 1  
 }  
  
 swapByIndex = (firstIndex: number, secondIndex: number): void => {  
 if (firstIndex < 0 || firstIndex >= this.length) throw new *Error*('SWAP\_BY\_INDEX: First index is out of bound')  
 if (secondIndex < 0 || secondIndex >= this.length) throw new *Error*('SWAP\_BY\_INDEX: Second index is out of bound')  
 const leftValue = this.getValueByIndex(firstIndex)  
 const rightValue = this.getValueByIndex(secondIndex)  
 this.erase(firstIndex)  
 this.insert(firstIndex, rightValue)  
 this.erase(secondIndex)  
 this.insert(secondIndex, leftValue)  
 }  
  
 sort = (comparator: (a: T, b: T) => boolean = (a: T, b: T) => a < b): void => {  
 for (let i = 0; i < this.length; ++i) {  
 let minIndex = i  
 let minValue: T = this.getValueByIndex(i)  
 for (let j = i + 1; j < this.length; ++j) {  
 if (comparator(this.getValueByIndex(j), minValue)) {  
 minValue = this.getValueByIndex(j)  
 minIndex = j  
 }  
 }  
 this.swapByIndex(i, minIndex)  
 }  
 }  
  
 pushBack = (value: T): void => {  
 this.insert(this.length, value)  
 }  
  
 popBack = (): void => {  
 if (!this.length) throw new *Error*('POP\_BACK: Array doesn\'t contain any element')  
 this.erase(this.length - 1)  
 }  
  
 toArray = (): T[] => {  
 const resultArray: T[] = []  
 let currentNode = this.root  
 for (let i = 0; i < this.length; ++i) {  
 resultArray.push(currentNode.value)  
 currentNode = currentNode.next  
 }  
 return resultArray  
 }  
}

export class ForwardListNode<T> {  
 public value: T | null  
  
 public next: ForwardListNode<T> | null  
  
 constructor(value: T, next: ForwardListNode<T> = null) {  
 this.value = value  
 this.next = next  
 }  
}

Результат

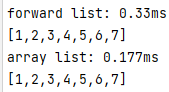


Рисунок 1

**Задание 2:**

*Асимптотическая оценка сложности алгоритма:*

Для ForwardList – O(n^2), потому что взятие элемента по индексу требует O(n) операций, и таких взятий требуется тоже O(n)

Для ArrayList ­­ - O(n), взятие элемента по индексу O(1)

*Наилучшую, наихудшую и среднюю оценки:*

ForwardList: worst: O(n^2) = average = best

ArrayList: worst: O(n) = average = best

*Псевдокод операции на упрощённом языке:*

mergeLists: listType, list1, list2

result = listType()

for i: 1..list1.len – 1, i = i + 2

result.add(list1[i])

for i: 1..list2.len – 1, i = i + 2

result.add(list2[i])

return result

*Выкладки расчета асимптотической оценки:*

list[i] – в зависимости от способа расположения данных в памяти может занимать O(n) операция для однонаправленного списка, либо O(1) для массива

Каждый из циклов занимает линейное относительно кол-ва элементов кол-во операций, и на каждом шаге цикла происходит взятие элемента по индексу, следовательно итоговая асимптотическая сложность алгоритма:

O(n \* O(list.get(index)))

*Выкладки или результаты тестирования на различных по объему набору данных:*

Будем менять размер первого массива:

При кол-ве элементов 20:

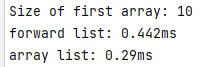


Рисунок 2

При кол-ве элементов 100:

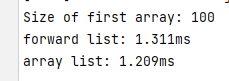


Рисунок 3

При кол-ве элементов 10000:

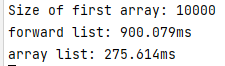


Рисунок 4

Как видно из тестов, при увеличении размера исходных данных время растёт квадратично для списка и линейно для массива.

Выводы: изучил основы работы с линейными динамическими структурами (списками, очередями, стеками, кольцевыми буферами и т.д.)