KAUNO TECHNOLOGIJOS UNIVERSITETAS

TECHNOLOGINIO PROJEKTO ATASKAITA

Studentas

Matas Palujanskas

KAUNAS 2022

TURINYS

Dekas

Dekas arba dviguba eilė (angl. Double Ended Queue) – tai tiesinė duomenų struktūra, kurioje elementus galima išimti ir įdėti tiek iš priekio, tiek iš galo. Dekas gali būti taikomas visur, kur galima taikyti steką arba eilę. Algoritmuose, kuriuose reikia dirbti su slenkančiu langu.

Deke saugomi duomenys yra Integer tipo, gali būti įdedami, tiek į pradžią tiek į galą.

Dekas buvo realizuotas dvikrypčio susietojo sąrašo pagrindu ir ciklinio masyvo pagrindu.

Susietasis dvikryptis sąrašas leidžia šalinti ir įterpti tiek eilės priekyje, tiek gale. Dvikrypčiame susietajame sąraše šios operacijos yra O(1) sudėtingumo, todėl ši duomenų struktūra puikiai tinka dekui realizuoti.

Realizuojant deką ciklinio masyvo pagrindu, algoritmas yra panašus kaip ir realizuojant eilę, tačiau šiuo atveju reikia realizuoti daugiau operacijų ir atlikti papildomų tikrinimų. Reikia leisti pradžios (angl. front) ir pabaigos (angl. rear) rodyklėms judėti laisvai po masyvą tiek pirmyn, tiek atgal.

Dvikrypčio susietojo sąrašo realizacija

```
//Deque LinkedList class
public class DequeLinkedList<E> implements Deque<E> {

    private Node<E> head;   // Link to the first element
    private Node<E> tail;   // Link to the last element
    private int size = 0;   // Size

    public DequeLinkedList()
    {
        tail = null;
        head = null;
    }

    //Adding element to front
    @Override
    public void addFirst(E elementToAdd) {
        if (elementToAdd == null) throw new IllegalArgumentException();

        Node<E> node = new Node<>(null, elementToAdd, null);
        if (head == null) {
            head  = node;
            tail = head;
        }
        else
        {
            node.next = head;
            head.previous = node;
            head = node;
        }
        size++;
    }

    //Returning first element
    @Override
```

```java
    public E getFirst() {
        if (this.head == null)
        {
            return null;
        }
        return this.head.value;
    }



    //Removing first element
    @Override
    public E removeFirst() {
        if (isEmpty())
        {
            throw new NoSuchElementException();
        }
        Node<E> toReturn = head;
        head = head.next;
        size--;
        return toReturn.value;
    }

    //Adding element to the end
    @Override
    public void addLast(E elementToAdd) {
        if (elementToAdd == null) throw new IllegalArgumentException();

        Node<E> node = new Node<>(null, elementToAdd, null);
        node.value = elementToAdd;

        if (head == null)
        {
            tail = node;
            head = tail;
        }
        else
        {
            tail.next = node;
            node.previous = tail;
            tail = node;
        }
        size++;
    }

    //Returning last element
    @Override
    public E getLast() {
        if (this.tail == null)
        {
            return null;
        }
        return this.tail.value;
    }

    //Removing last element
```

```
    @Override
    public E removeLast() {
        if (isEmpty())
        {
            throw new NoSuchElementException();
        }
        Node<E> toReturn = tail;
        tail = tail.previous;
        tail.next = null;
        size--;
        return toReturn.value;
    }

    //Checking if list is empty
    @Override
    public boolean isEmpty() {
        return size == 0;
    }

    //Node class
    private static class Node<E> {
        private E value;
        private Node<E> previous;
        private Node<E> next;

        private Node(Node<E> previous, E current, Node<E> next) {
            this.value = current;
            this.previous = previous;
            this.next = next;
        }
    }
    public int size()
    {
        return this.size;
    }
    public String toString()
    {
        StringBuilder deque = new StringBuilder();

        if (isEmpty())
        {
            deque.append("Deque is empty.");
            return deque.toString();
        }

        Node<E> first = head;
        deque.append("Deque elements: [ ").append(first.value).append(" ");
        while (first.next != null)
        {
            deque.append(first.next.value).append(" ");
            first = first.next;
        }
        deque.append("]");

        return deque.toString();
```

```
    }
}
```

Ciklinio masyvo realizacija

```
//Deque Array class
public class DequeArray<E> implements Deque<E> {

    private Object[] elements;
    private int headIndex; //head index
    private int tailIndex; //tail index
    private int numberOfElements; //amount of elements

    public DequeArray(int capacity) {
        if (capacity < 1) {
            throw new IllegalArgumentException("Capacity must be 1 or
higher");
        }
        elements = new Object[capacity];
    }

    //Ensuring capacity
    private void ensureCapacity()
    {
        if (size() == elements.length)
        {
            Object[] ensuredArray = new Object[elements.length * 2];
            for (int i = 0; i < numberOfElements; i++)
            {
                ensuredArray[i] = elements[headIndex];
                headIndex = (headIndex + 1) % elements.length;
            }
            headIndex = 0;
            tailIndex = numberOfElements;
            elements = ensuredArray;
        }
    }

    //Adding element to the front
    @Override
    public void addFirst(E element) {
        ensureCapacity();

        headIndex = decreaseIndex(headIndex);
        elements[headIndex] = element;
        numberOfElements++;
    }

    //Returning first element
    @Override
```

```
    public E getFirst() {
        return elementAtHead();
    }

    //Removing first element
    @Override
    public E removeFirst() {
        E element = elementAtHead();
        elements[headIndex] = null;
        headIndex = increaseIndex(headIndex);
        numberOfElements--;
        return element;
    }

    //Adding element to the end
    @Override
    public void addLast(E element) {
        ensureCapacity();

        elements[tailIndex] = element;
        tailIndex = increaseIndex(tailIndex);
        numberOfElements++;
    }

    //Returning last element
    @Override
    public E getLast() {
        return elementAtTail();
    }

    //Removing last element
    @Override
    public E removeLast() {
        E element = elementAtTail();
        tailIndex = decreaseIndex(tailIndex);
        elements[tailIndex] = null;
        numberOfElements--;
        return element;
    }

    //Checking if array is empty
    @Override
    public boolean isEmpty() {
        return false;
    }
    private E elementAtHead() {
        if (isEmpty()) {
            throw new NoSuchElementException();
        }
        E element = (E) elements[headIndex];
        return element;
    }

    private E elementAtTail() {
        if (isEmpty()) {
            throw new NoSuchElementException();
```

```
        }
        E element = (E) elements[decreaseIndex(tailIndex)];
        return element;
    }

    private int decreaseIndex(int index) {
        index--;
        if (index < 0) {
            index = elements.length - 1;
        }
        return index;
    }

    private int increaseIndex(int index) {
        index++;
        if (index == elements.length) {
            index = 0;
        }
        return index;
    }

    //Size of an array
    public int size()
    {
        return numberOfElements;
    }

    //ToString method
    public String toString()
    {
        String dequeElements = "";


        for (int i = 0,j=0; j<elements.length;i=(i+1)% elements.length,j++)
        {

            if (elements[i]!=null)
                dequeElements = dequeElements + elements[i].toString()+ " ";
            else if (elements[j]!=null)
            {
                //System.out.println("j is: " +j);
                dequeElements = dequeElements + elements[j].toString()+ " ";
            }
        }
        return dequeElements;
    }
}
```

Greitaveikos tyrimas

Atlikti greitaveikos tyrimai naudojant Java JMH ir palyginti atskirų realizacijų metodai (addFirst, addLast, removeFirst, removeLast, getFirst, getLast).

Testuojant greitaveiką, buvo generuojami atsitiktiniai Integer tipo elementai.

Asimptotiniai sudėtingumai

Operacija	Asimptotinis sudėtingumas
Įterpti priekyje	O(1)
Įterpti gale	O(1)
Ištrinti priekyje	O(1)
Ištrinti gale	O(1)
Imti elementą iš priekio	O(1)
Imti elementą iš galo	O(1)

Kompiuterio charakteristikos

Procesorius: AMD Ryzen 7 4800H 8 -Core Processor 2.90 GHz

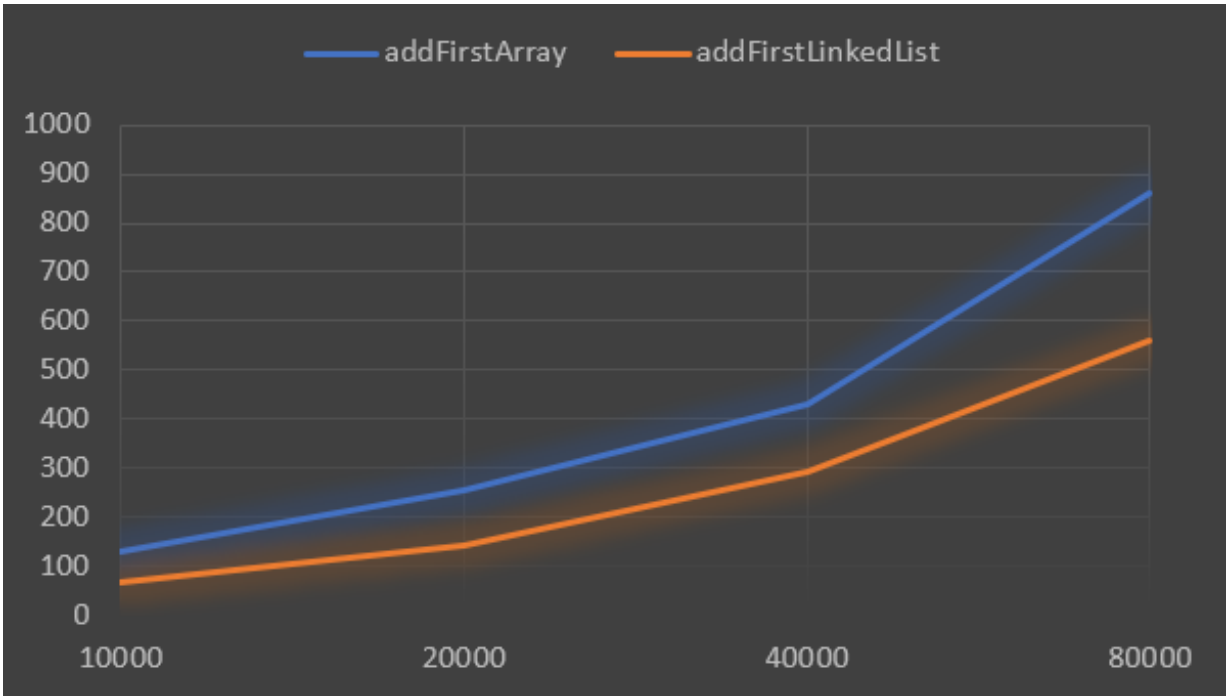
Installed RAM: 16 GB, (usable 15.4 GB)

Greitaveikos rezultatai

addFirst metodo greitaveikos rezultatai:

Benchmark	(elementCount)	Mode	Cnt	Score	Error	Units
Tests.Benchmark.addFirstArray	10000	avgt	5	129,083 ±	5,785	us/op
Tests.Benchmark.addFirstArray	20000	avgt	5	225,622 ±	20,075	us/op
Tests.Benchmark.addFirstArray	40000	avgt	5	431,437 ±	34,985	us/op
Tests.Benchmark.addFirstArray	80000	avgt	5	859,037 ±	65,031	us/op
Tests.Benchmark.addFirstLinkedList	10000	avgt	5	68,030 ±	7,695	us/op
Tests.Benchmark.addFirstLinkedList	20000	avgt	5	142,182 ±	11,843	us/op
Tests.Benchmark.addFirstLinkedList	40000	avgt	5	293,492 ±	62,870	us/op
Tests.Benchmark.addFirstLinkedList	80000	avgt	5	561,542 ±	122,981	us/op

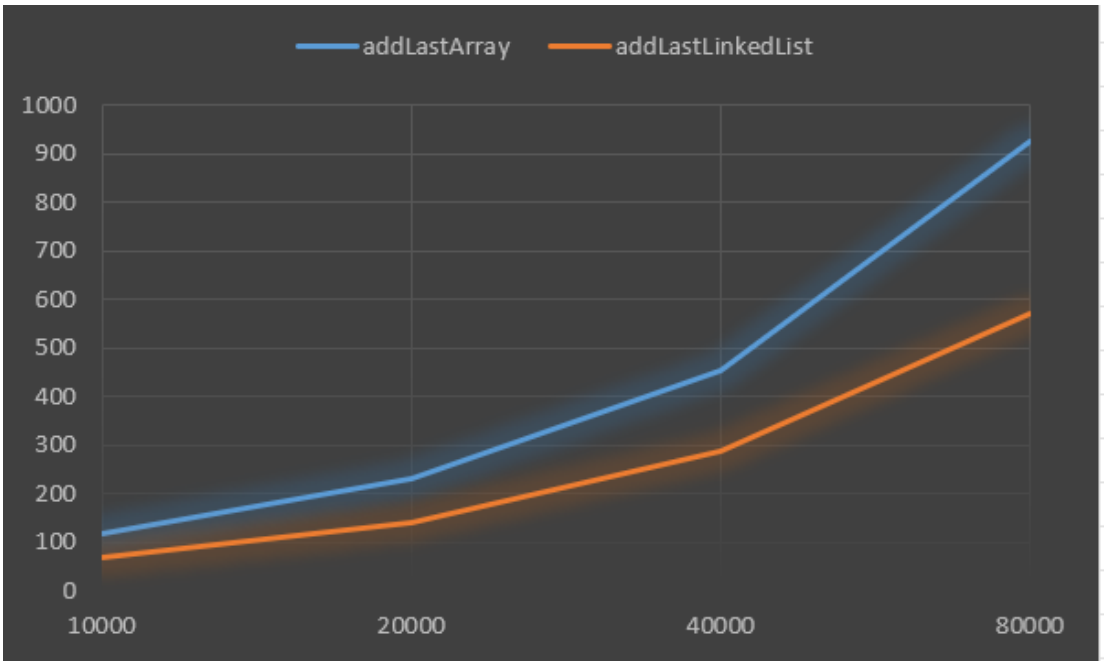
Testavimo grafikas:



addLast greitaveikos rezultatai:

Benchmark	(elementCount)	Mode	Cnt	Score	Error	Units
Tests.Benchmark.addLastArray	10000	avgt	5	118,703 ±	3,809	us/op
Tests.Benchmark.addLastArray	20000	avgt	5	231,578 ±	4,269	us/op
Tests.Benchmark.addLastArray	40000	avgt	5	456,677 ±	31,752	us/op
Tests.Benchmark.addLastArray	80000	avgt	5	926,863 ±	128,509	us/op
Tests.Benchmark.addLastLinkedList	10000	avgt	5	70,291 ±	3,427	us/op
Tests.Benchmark.addLastLinkedList	20000	avgt	5	142,361 ±	9,505	us/op
Tests.Benchmark.addLastLinkedList	40000	avgt	5	290,094 ±	25,156	us/op
Tests.Benchmark.addLastLinkedList	80000	avgt	5	572,469 ±	144,863	us/op

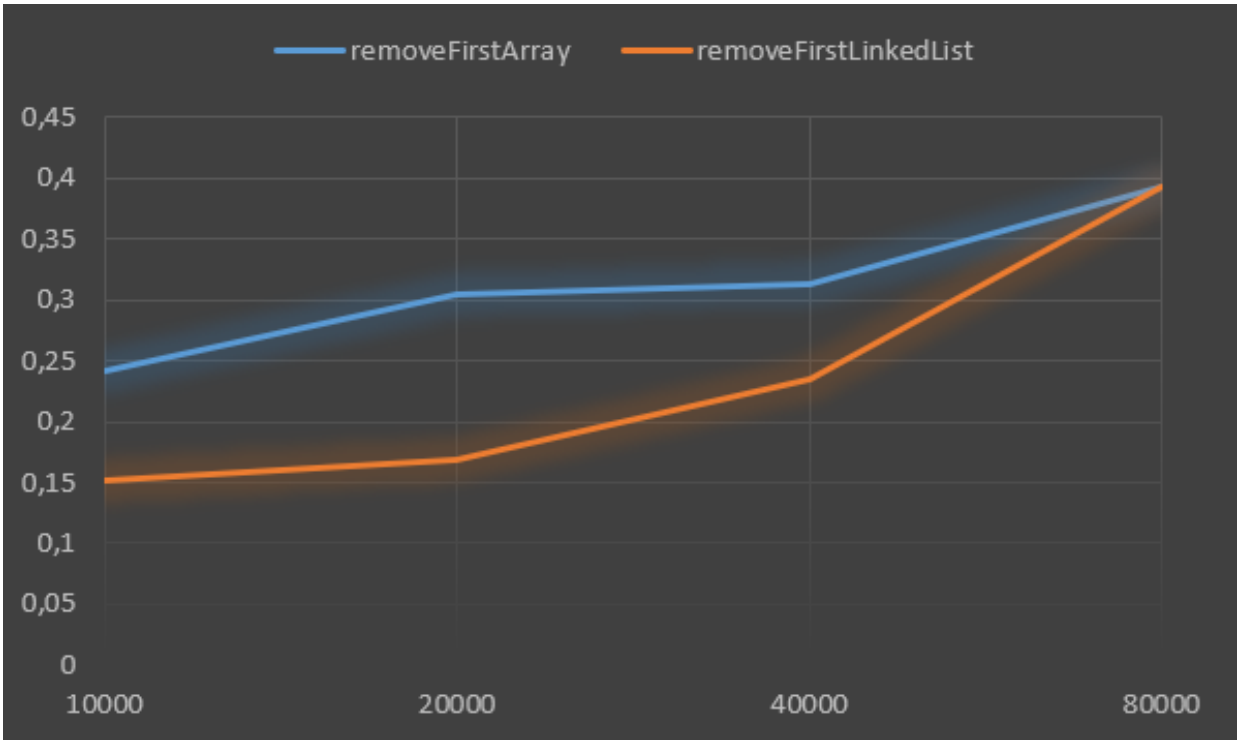
Testavimo grafikas:



removeFirst greitaveikos rezultatai:

Benchmark	(elementCount)	Mode	Cnt	Score	Error	Units
Tests.Benchmark.removeFirstArray	10000	avgt	5	0,242	± 0,030	us/op
Tests.Benchmark.removeFirstArray	20000	avgt	5	0,304	± 0,059	us/op
Tests.Benchmark.removeFirstArray	40000	avgt	5	0,313	± 0,025	us/op
Tests.Benchmark.removeFirstArray	80000	avgt	5	0,393	± 0,101	us/op
Tests.Benchmark.removeFirstLinkedList	10000	avgt	5	0,151	± 0,027	us/op
Tests.Benchmark.removeFirstLinkedList	20000	avgt	5	0,169	± 0,043	us/op
Tests.Benchmark.removeFirstLinkedList	40000	avgt	5	0,235	± 0,027	us/op
Tests.Benchmark.removeFirstLinkedList	80000	avgt	5	0,393	± 0,493	us/op

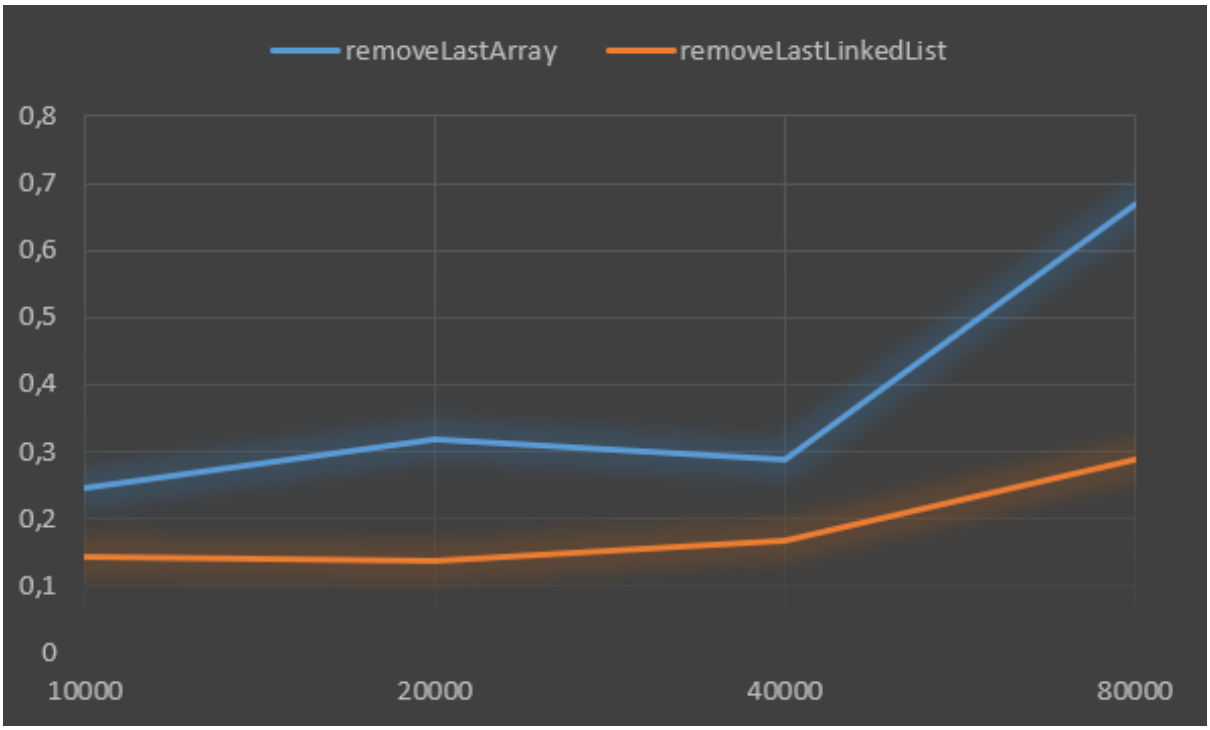
Testavimo grafikas:



removeLast greitaveikos rezultatai:

Benchmark	(elementCount)	Mode	Cnt	Score	Error	Units
Tests.Benchmark.removeLastArray	10000	avgt	5	0,246	± 0,061	us/op
Tests.Benchmark.removeLastArray	20000	avgt	5	0,317	± 0,068	us/op
Tests.Benchmark.removeLastArray	40000	avgt	5	0,287	± 0,112	us/op
Tests.Benchmark.removeLastArray	80000	avgt	5	0,668	± 0,641	us/op
Tests.Benchmark.removeLastLinkedList	10000	avgt	5	0,143	± 0,027	us/op
Tests.Benchmark.removeLastLinkedList	20000	avgt	5	0,138	± 0,033	us/op
Tests.Benchmark.removeLastLinkedList	40000	avgt	5	0,166	± 0,119	us/op
Tests.Benchmark.removeLastLinkedList	80000	avgt	5	0,289	± 0,215	us/op

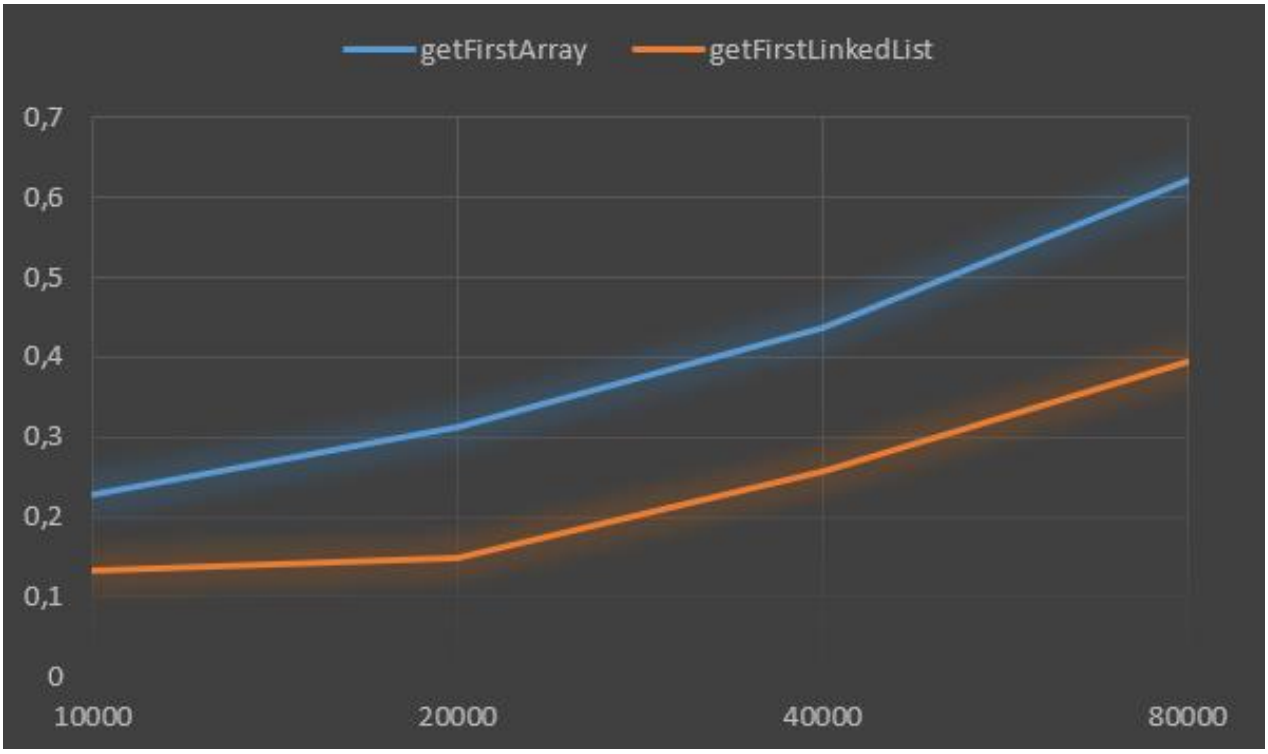
Testavimo grafikas:



getFirst greitaveikos rezultatai:

Tests.Benchmark.getFirstArray	10000	avgt	5	0,227 ± 0,011	us/op
Tests.Benchmark.getFirstArray	20000	avgt	5	0,313 ± 0,085	us/op
Tests.Benchmark.getFirstArray	40000	avgt	5	0,437 ± 0,167	us/op
Tests.Benchmark.getFirstArray	80000	avgt	5	0,623 ± 0,300	us/op
Tests.Benchmark.getFirstLinkedList	10000	avgt	5	0,133 ± 0,028	us/op
Tests.Benchmark.getFirstLinkedList	20000	avgt	5	0,150 ± 0,031	us/op
Tests.Benchmark.getFirstLinkedList	40000	avgt	5	0,258 ± 0,301	us/op
Tests.Benchmark.getFirstLinkedList	80000	avgt	5	0,395 ± 0,181	us/op

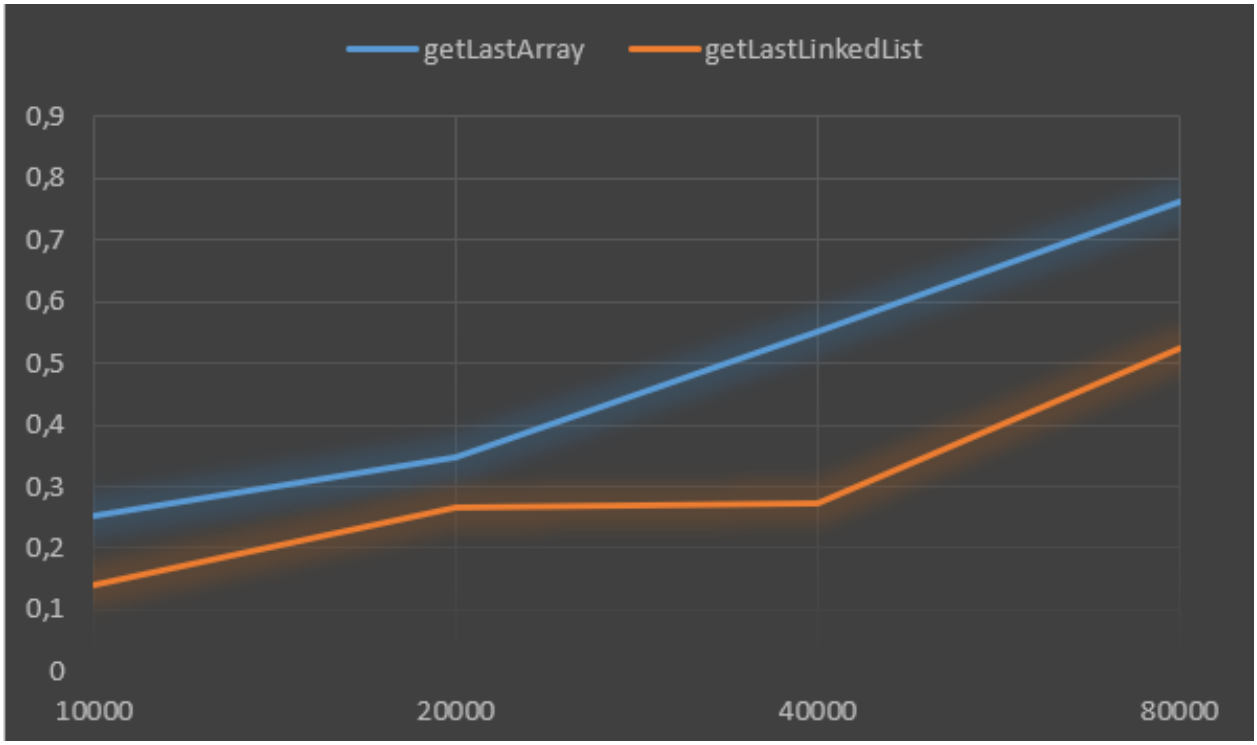
Testavimo grafikas:



getLast greitaveikos rezultatai:

Benchmark	(elementCount)	Mode	Cnt	Score	Error	Units
Tests.Benchmark.getLastArray	10000	avgt	5	0,251	± 0,053	us/op
Tests.Benchmark.getLastArray	20000	avgt	5	0,349	± 0,081	us/op
Tests.Benchmark.getLastArray	40000	avgt	5	0,550	± 0,368	us/op
Tests.Benchmark.getLastArray	80000	avgt	5	0,763	± 0,348	us/op
Tests.Benchmark.getLastLinkedList	10000	avgt	5	0,142	± 0,040	us/op
Tests.Benchmark.getLastLinkedList	20000	avgt	5	0,265	± 0,062	us/op
Tests.Benchmark.getLastLinkedList	40000	avgt	5	0,272	± 0,253	us/op
Tests.Benchmark.getLastLinkedList	80000	avgt	5	0,524	± 0,210	us/op

Testavimo grafikas:



Išvados

- Dekas buvo sėkmingai realizuotas tiek susietojo sąrašo, tiek ciklinio masyvo pagrindu.
- Testuojami metodai atitiko teorinius asimptotinius sudėtingumus, visi jie buvo O(1).
- Iš gautų greitaveikos rezultatų galime pastebėti, jog Deko realizacija susietojo sąrašo pagrindu veikia greičiau nei ciklinio masyvo. Taip yra todėl, nes masyvo pagrindu realizuotas Dekas, didėjant elementų kiekiui turi didinti ir talpą, tuo tarpu susietajame sąraše elementai yra „kabinami“ į grandinėlę. Gauti rezultatai atitinka ir įžangoje aptartas skirtingas Deko realizacijas, jog dvikrypčio susieto sąrašo duomenų struktūra puikiai tinka Dekui realizuoti.