KAUNO TECHNOLOGIJOS UNIVERSITETAS

Duomenų struktūros (P175B014)

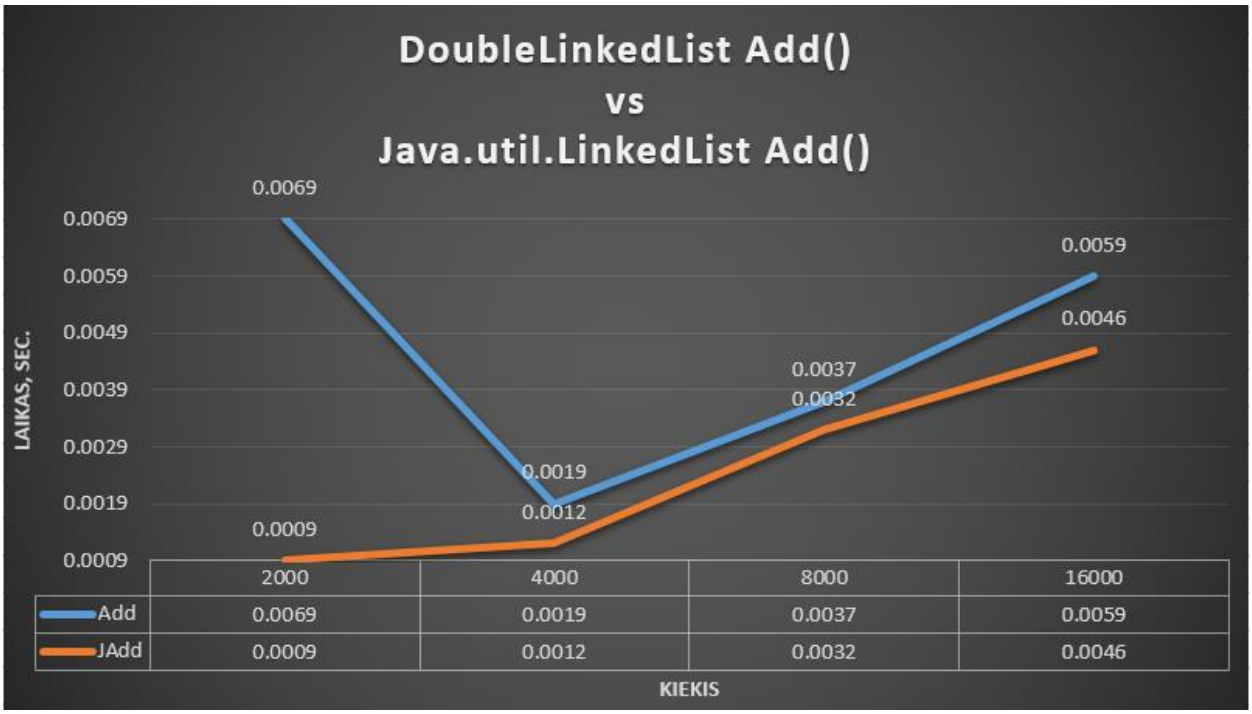
Projekto ataskaita

Atliko **Martynas Kemežys** gr. IF-8/1

Priėmė lekt. **Karčiauskas Eimutis**

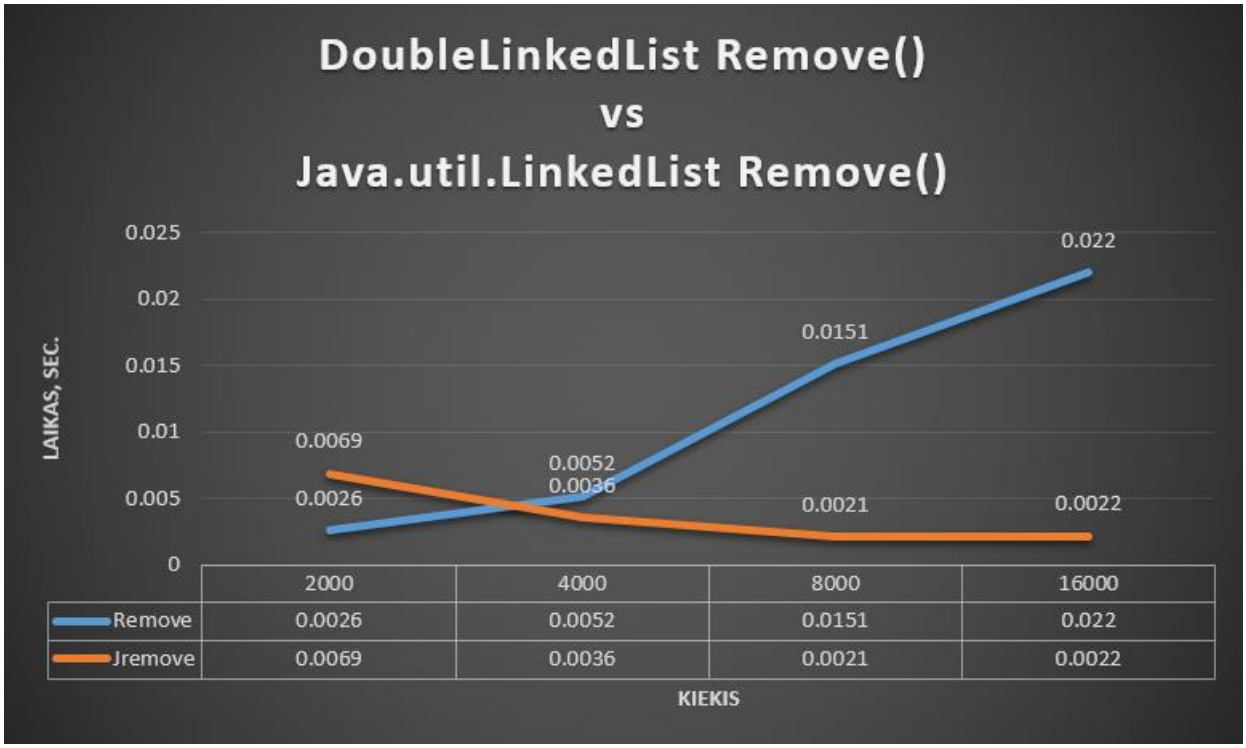
1. Metodų greitaveika edu.ktu.ds.util.DoubleLinkedList ir java.util.LinkedList

1.1 Metodas Add()



1.2 Metodų greitaveika edu.ktu.ds.util.DoubleLinkedList ir java.util.LinkedList

1.2 Metodas Remove()



2.Kodas

```
public class DoubleLinkedList<E extends Comparable<E>> implements List<E>, Iterable<E>, Cloneable {
   private Node<E> first;   // rodyklė į pirmą mazgą
   private Node<E> last;    // rodyklė į paskutinį mazgą
   private Node<E> current; // rodyklė į einamąjį mazgą, naudojama getNext
   private int size;        // sąrašo dydis, tuo pačiu elementų skaičius

   public DoubleLinkedList() {
   }
   @Override
   public boolean add(E e) {
      if (e == null) {
         return false;   // nuliniai objektai nepriimami
      }
      if (first == null) {
         first = new Node<>(e, first, last);
         last = first;
      } else {
         Node<E> e1 = new Node(e, null, last);
         last.next = e1;
         last = e1;
      }
      size++;
      return true;
   }
   public List<E> subList(int fromIndex, int toIndex)
   {
      DoubleLinkedList list = new DoubleLinkedList();
      Node<E> test = null;
      if(fromIndex > toIndex || toIndex > size) return null;
      Ks.oun("=======subList veikimas========");
      for (int i = fromIndex; i <= toIndex; i++) {
        test = first.findNode(i);
        list.add(test.element);
      }
      return list;
   }
   public boolean removeLastOccurrence(E e) {
      Node<E> temp = first, previous = null;
      Node<E> test = null;
      if (e == null) {

         return false;   // nuliniai objektai nepriimami

      }
      for (int i = 0; i < size; i++) {
         test = first.findNode(i);
         if(test.element.equals(e) && e != first.element)
         {
          previous = first.findNode(i-1);   //prieš audi
          temp = test.next;   //kas už audi
         }
      }
      if(previous == null)
      {
         first = first.next;
         size --;
         return true;
      }
```

```
    if (temp == null)
    {
    previous.next = null;
    size --;
    return true;
    }
    previous.next = temp;
size--;
return true;
}
public boolean addLast(E e) {
    if (e == null) {
        return false;   // nuliniai objektai nepriimami
    }
    if (first == null) {
        first = new Node<>(e, first, last);
        last = first;

    } else {
        Node<E> e1 = new Node(e, null, last);
        last.next = e1;
        last = e1;
    }
    size++;
    return true;
}
@Override
public int size() {
    return size;
}
@Override
public boolean isEmpty() {
    return first == null;
}
@Override
public void clear() {
    size = 0;
    first = null;
    last = null;
    current = null;
}
@Override
public E get(int k) {
    if (k < 0 || k >= size) {
        return null;
    }
    current = first.findNode(k);
    return current.element;
}
@Override
public E set(int k, E e) {
    if (k < 0 || k >= size)
        return null;

    Node<E> ref = first.findNode(k);
    E old = ref.element;
    ref.element = e;
    return old;
}
@Override
public boolean add(int k, E e) {
    if (e == null || k < 0 || k >= size)
```

```
        return false;

    if (k == 0) {
        first = new Node(e, first, last);
        return true;
    }

    Node<E> ref = first;
    for (int i = 0; i < k - 1; i++)
        ref = ref.next;

    ref.next = new Node(e, ref.next, last);
    return true;
}
@Override
public E getNext() {
    if (current == null) {
        return null;
    }
    current = current.next;
    if (current == null) {
        return null;
    }
    return current.element;
}
public E getPreviuos() {
    if (current == null) {
        return null;
    }
    current = current.previuos;
    if (current == null) {
        return null;
    }
    return current.element;
}
@Override
public E remove (int k) {
   if (k<0 || k>=size) return null;  // jei k yra blogas, grąžina null
                Node<E> actual = null;
   if (k == 0) {   // šaliname elementą, esantį pradžioje
      actual=first; first = actual.next;
      if (first == null) last=null;
   } else {    // šaliname elementą, esantį tolimesnėje sekoje
      Node<E> previous = first.findNode(k-1);
      actual = previous.next;
      previous.next = actual.next;
      if (actual.next == null) last = previous;
   }
   size--;
   return actual.element;
}
@Override
public DoubleLinkedList<E> clone() {
    DoubleLinkedList<E> cl = null;
    try {
        cl = (DoubleLinkedList<E>) super.clone();
    } catch (CloneNotSupportedException e) {
        Ks.ern("Blogai veikia metodas clone()");
        System.exit(1);
    }
    if (first == null) {
        return cl;
```

```
    }
    cl.first = new Node<>(this.first.element, null, null);
    Node<E> e2 = cl.first;
    for (Node<E> e1 = first.next; e1 != null; e1 = e1.next) {
      e2.next = new Node<>(e2.element, null, null);
      e2 = e2.next;
      e2.element = e1.element;
    }
    cl.last = e2.next;
    cl.size = this.size;
    return cl;
  }
  public Object[] toArray() {
    Object[] a = new Object[size];
    int i = 0;
    for (Node<E> e1 = first; e1 != null; e1 = e1.next) {
      a[i++] = e1.element;
    }
    return a;
  }
  @Override
  public java.util.Iterator<E> iterator() {
    return new Iterator();
  }
  class Iterator implements java.util.Iterator<E> {
    private Node<E> iterPosition;
    Iterator() {
      iterPosition = first;
    }
    @Override
    public boolean hasNext() {
      return iterPosition != null;
    }
    @Override
    public E next() {
      E d = iterPosition.element;
      iterPosition = iterPosition.next;
      return d;
    }
    public E previuos() {
      E d = iterPosition.element;
      iterPosition = iterPosition.previuos;
      return d;
    }
    @Override
    public void remove() {
      if (iterPosition == null) throw new IllegalStateException();
      Node x = iterPosition.previuos;
      Node y = iterPosition.next;
      x.next = y;
      y.previuos = x;
      if (current == iterPosition)
        current = y;
      else
        size--;
      iterPosition = null;
    }
  }
  private static class Node<E> {
    private E element;    // ji nematoma už klasės ListKTU ribų
    private Node<E> next; // next - kaip įprasta - nuoroda į kitą mazgą
    private Node<E> previuos;
```

```
    Node(E data, Node<E> next, Node<E> previuos) { //mazgo konstruktorius
      this.element = data;
      this.next = next;
      this.previuos = previuos;
    }
    public Node<E> findNode(int k) {
      Node<E> e = this;
      for (int i = 0; i < k; i++) {
        e = e.next;
      }
      return e;
    }
  }
}
```

3.Dvikrypčio sąrašo pliusai/minusai

Pliusai:

Galima vaikščioti per sąrašą pirmyn ir atgal. Esmė kaip reikia ištrinti ankstesnį mazga, nereikia pereiti visko vėl nuo pradžios, nes mazga visgi galima rasti su **.previous** rodykle.

Minusai:

- Santykinai sudėtinga įdiegti, reikia daugiau atminties saugojimui.
- Įterpimai ir ištrynimai užima gana daug laiko(priskiriant **.previous** rodykle).

4.Išvados

Projekto darbo metu vėl prisiminiau dvikrypčio sąrašo struktūra, sužinojau pranašumus. Palyginau su javos vienkrypčiu sąrašu, sužinojau kas greičiau veikia. Supratau kad dvikryptis sąrašas ilgiau atlieka veiksmus, užima daugiau atminties bet yra patogesnis paieškos funkcijai ir t.t.