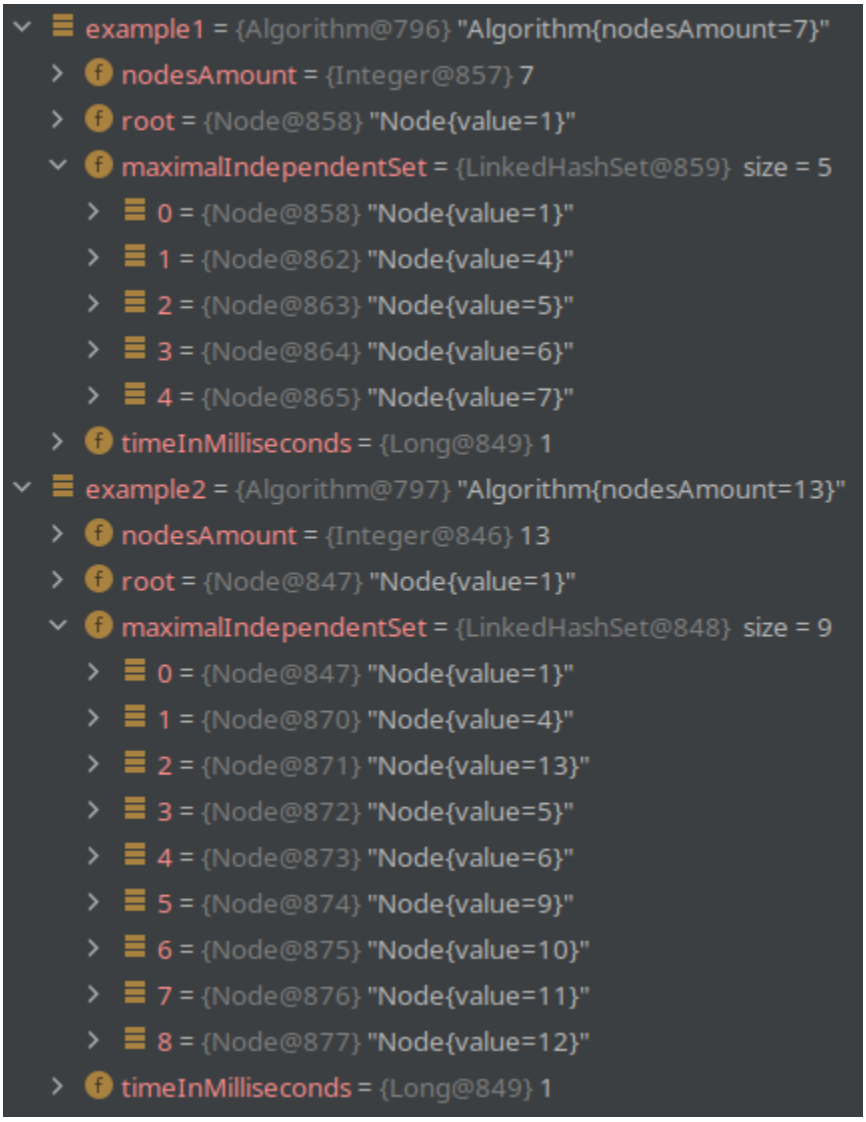
Błażej Piekoś 273210

Wyniki dla przykładów z zajęć:



Wyniki dla przykładów losowych:

Kod:

```
package model;

import com.google.common.collect.Lists;
import com.google.common.collect.Sets;
import java.util.List;
import java.util.Objects;
import java.util.Set;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class Node {

  private int value;
  private List<Node> children = Lists.newArrayList();
  private Set<Node> independentSet = Sets.newLinkedHashSet();
  private Integer independentSetSize = 0;

  public Node(
      int value
  ) {
    this.value = value;
  }

  public Boolean addChild(
      Node child
  ) {
    if (this.children == null) {
      this.children = Lists.newLinkedList();
    }
    this.getChildren().add(child);
    return Boolean.TRUE;
  }

  public Boolean addChildren(
      List<Node> children
  ) {
    if (this.children == null) {
      this.children = Lists.newLinkedList();
    }
    this.getChildren().addAll(children);
    return Boolean.TRUE;
  }
```

```
public List<Node> getGrandChildren(
) {
  List<Node> grandChildren = Lists.newLinkedList();
  for (Node child : this.getChildren()) {
    grandChildren.addAll(
        child.getChildren()
    );
  }
  return grandChildren;
}

public boolean addIndependentSetNode(
    Node node
) {
  if (this.independentSet == null) {
    this.independentSet = Sets.newLinkedHashSet();
  }
  this.independentSet.add(node);
  return Boolean.TRUE;
}

public boolean addIndependentSetNodes(
    Set<Node> nodes
) {
  if (this.independentSet == null) {
    this.independentSet = Sets.newLinkedHashSet();
  }
  this.independentSet.addAll(nodes);
  return Boolean.TRUE;
}

public boolean addIndependentSetNodes(
    List<Node> nodes
) {
  if (this.independentSet == null) {
    this.independentSet = Sets.newLinkedHashSet();
  }
  this.independentSet.addAll(nodes);
  return Boolean.TRUE;
}

public Set<Node> getChildrenIndependentSet(
) {
  Set<Node> independentSet = Sets.newLinkedHashSet();
  for (Node child : this.getChildren()) {
    independentSet.addAll(
        child.getIndependentSet()
    );
  }
```

```
    return independentSet;
  }

  @Override
  public boolean equals(
      Object o
  ) {
    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    Node node = (Node) o;
    return value == node.value;
  }

  @Override
  public int hashCode() {
    return Objects.hash(value);
  }

  @Override
  public String toString() {
    return "Node{" + "value=" + value + '}';
  }

}
```

```java
package model;

import com.google.common.collect.Sets;
import java.util.Set;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class Algorithm {

 private Integer nodesAmount;
 private Node root;
 private Set<Node> maximalIndependentSet = Sets.newHashSet();
 private Long timeInMilliseconds = 0L;

 public Algorithm(
    Node root,
    Integer nodesAmount
 ) {
  this.root = root;
  this.nodesAmount = nodesAmount;
  this.run();
 }

 /**
  * Metoda uruchamiajaca algorytm, zbierająca wyniki i mierząca czas.
  */
 public int run() {
  long start = System.currentTimeMillis();
  Integer maximalIndependentSetSize = this.processNode(this.root);
  this.timeInMilliseconds = System.currentTimeMillis() - start;
  this.maximalIndependentSet = this.root.getIndependentSet();
  return maximalIndependentSetSize;
 }

 /**
  * Główna metoda algorytmu
  */
 public Integer processNode(
    Node node
 ) {
  // Jeśli węzeł został już przetworzony zwróć jego ciąg
  if (!node.getIndependentSet().isEmpty()) {
   return node.getIndependentSet().size();
  }
  // Jeśli węzał nie ma dzieci to należy do ciągu
```

```
  if (node.getChildren().isEmpty()) {
    node.getIndependentSet().add(node);
    return node.getIndependentSet().size();
  }
  // Szukanie ciągu
  int setSize = this.findSet(node);
  return setSize;
}

/**
 * Szukanie ciągu dla danego węzła
 */
public Integer findSet(
    Node node
) {
  // Znajdujemy długość ciągu gdy przynależą do niego dzieci aktualnie przetwarzanego ciągu
  Integer childSetSize = this.findChildSet(node);
  // Znajdujemy długość ciągu gdy przynależą do niego wnukowie aktualnie przetwarzanego ciągu
  Integer grandChildSetSize = this.findGrandChildSet(node);
  // Sprawdzamy, który ciąg jest dłuższy
  Integer result = Math.max(childSetSize, grandChildSetSize);
  // Przypisujemy wyniki aktualnie przetwarzanego węzła
  if (grandChildSetSize > childSetSize) {
    node.addIndependentSetNode(node);
  }
  node.addIndependentSetNodes(
      node.getChildrenIndependentSet()
  );
  node.setIndependentSetSize(result);
  return result;
}

/**
 * Szukanie ciągu w wersji z dziećmi
 */
public Integer findChildSet(
    Node parent
) {
  // Początkowa wartość ciągu w wersji dzieci
  Integer childSetSize = 0;
  // Przetwarzanie dzieci
  for (Node child : parent.getChildren()) {
    childSetSize = childSetSize + this.processNode(child);
  }
  return childSetSize;
}

/**
 * Szukanie ciągu w wersji z wnukami
```

```
     */
    public Integer findGrandChildSet(
        Node parent
    ) {
      // Początkowa wartość ciągu w wersji wnukowie (1, bo aktualnie przetwarzany węzeł)
      Integer grandChildrenSetSize = 1;
      // Przetwarzanie wnuków
      for (Node grandChild : parent.getGrandChildren()) {
        grandChildrenSetSize = grandChildrenSetSize + this.processNode(grandChild);
      }
      return grandChildrenSetSize;
    }

    @Override
    public String toString() {
      return "Algorithm{" + "nodesAmount=" + this.nodesAmount + '}';
    }

}
```