

Laboratorium 8

Szymon Kałuża

2022-05-10

1 Implementacja listy

```
1 public class ArrOne{
2     private Integer[] arr;
3     public ArrOne(int value){
4         Integer[] vv = new Integer[1];
5         vv[0] = value;
6         this.arr = vv;
7     }
8     public void add(int value){
9         Integer[] val = new Integer[arr.length + 1];
10        for(int j = 0; j < arr.length; j++){
11            val[j] = arr[j];
12        }
13        val[arr.length] = value;
14        this.arr = val;
15    }
16    public void removeTail(){
17        Integer[] val = new Integer[arr.length - 1];
18        for(int j = 0; j < arr.length - 1; j++){
19            val[j] = arr[j];
20        }
21        this.arr = val;
22    }
23    public int naiveSearch(int x){
24        for(int j = 0; j < arr.length; j++){
25            if(arr[j] == x) return j;
26        }
27        return -1;
28    }
29 }
```

2 Implementacja stosu i kolejki

2.1 Stos

```
1 class MyStack{
2     public Integer[] quess;
3
4     public MyStack(int value){
5         Integer[] val = new Integer[1];
6         val[0] = value;
7         this.quess = val;
8     }
9     public void add(int value){
10        Integer[] val = new Integer[quess.length + 1];
11        for(int j = 0; j < quess.length; j++){
12            val[j] = quess[j];
13        }
14        val[quess.length] = value;
15        this.quess = val;
16    }
17
18    public void remove(){
19        Integer[] val = new Integer[quess.length - 1];
20        for(int j = 0; j < quess.length - 1; j++){
21            val[j] = quess[j];
22        }
23        this.quess = val;
24    }
25
26    public void show(){
27        for(int j = 0; j < quess.length; j++){
```

```

28         System.out.println(queess[j] + ", ");
29     }
30 }
31 }

```

2.2 Kolejka

```

1 class Ques{
2     public Integer [] queess;
3
4     public Ques(int value){
5         Integer [] val = new Integer [1];
6         val[0] = value;
7         this.queess = val;
8     }
9     public void add(int value){
10        Integer [] val = new Integer [queess.length + 1];
11        for(int j = 0; j < queess.length; j++){
12            val[j] = queess[j];
13        }
14        val[queess.length] = value;
15        this.queess = val;
16    }
17
18    public void remove(){
19        Integer [] val = new Integer [queess.length - 1];
20        for(int j = 1; j < queess.length; j++){
21            val[j - 1] = queess[j];
22        }
23        this.queess = val;
24    }
25
26    public void show(){
27        for(int j = 0; j < queess.length; j++){
28            System.out.println(queess[j] + ", ");
29        }
30    }
31 }

```

3 Implementacja dodatkowych metod do listy

3.1 Dodawanie elementu

```

1 public void add(int value){
2     Integer [] val = new Integer [queess.length + 1];
3     for(int j = 0; j < queess.length; j++){
4         val[j] = queess[j];
5     }
6     val[queess.length] = value;
7     this.queess = val;
8 }

```

3.2 Odwracanie listy

```

1 public void reverse(){
2     int temp;
3     for(int j = 0; j < arr.length/2; j++){
4         temp = arr[j];
5         arr[j] = arr[arr.length - (j + 1)];
6         arr[arr.length - (j + 1)] = temp;
7     }
8 }

```

3.3 Odwracanie wszystkich kluczy o wartości key

```

1 public void removeKey(int key){
2     for(int j = 0; j < arr.length; j++){
3         if(arr[j] == key){
4             Integer [] val = new Integer [arr.length - 1];
5             for(int i = 0, k = 0; i < arr.length - 1; i++, k++){
6                 if(i == j){
7                     k++;

```

```

8         }
9         val[i] = arr[k];
10    }
11    this.arr = val;
12 }
13 }
14 }

```

3.4 Wstawianie elementu w określone miejsce na liście

```

1 public void insertKeyByIndex(int key, int index){
2     Integer[] val = new Integer[arr.length + 1];
3     for(int j = 0, k = 0; j < arr.length; j++, k++){
4         if(j == index){
5             val[k] = key;
6             k++;
7         }
8         val[k] = arr[j];
9     }
10    this.arr = val;
11 }

```

4 Praca na zbiorach

4.1 Czy a należy do zbioru A

```

1 public boolean isAnElementOf(ArrOne arrOne){
2     if(arr.length > arrOne.getNumberOfElements()) return false;
3     boolean ps = false;
4     for(int j = 0; j < arr.length; j++){
5         ps = false;
6         for(int i = 0; i < arrOne.getNumberOfElements(); i++){
7             if(arrOne.getElementByIndex(i) == arr[j]) ps = true;
8         }
9         if(!ps) return false;
10    }
11    return true;
12 }

```

4.2 Tworzenie listy składającej się z części wspólnej zbiorów a i A

```

1 public Integer[] intersectionWith(ArrOne arrOne){
2     Integer[] val = new Integer[0];
3     for(int j = 0; j < arr.length; j++){
4         for(int i = 0; i < arrOne.getNumberOfElements(); i++){
5             if(arr[j] == arrOne.getElementByIndex(i)){
6                 Integer[] temp = new Integer[val.length + 1];
7                 for(int k = 0; k < val.length; k++){
8                     temp[k] = val[k];
9                 }
10                temp[val.length] = arr[j];
11                val = temp;
12            }
13        }
14    }
15    return val;
16 }

```

4.3 Tworzenie listy będącej sumą zbiorów a i A

```

1 public Integer[] sumWith(ArrOne arrOne){
2     Integer[] val = new Integer[arr.length + arrOne.getNumberOfElements()];
3     for(int j = 0; j < arr.length; j++){
4         val[j] = arr[j];
5     }
6     for(int j = arr.length, k = 0; j < arr.length + arrOne.getNumberOfElements(); j++, k++){
7         val[j] = arrOne.getElementByIndex(k);
8     }
9     return val;
10 }

```

4.4 Sprawdzanie czy a jest podzbiorem zbioru A

```
1 public boolean isASubsetOf(ArrOne arrOne){
2     if(arr.length > arrOne.getNumberOfElements()) return false;
3     int number = 0;
4     for(int j = 0; j < arr.length; j++){
5         for(int i = 0; i < arrOne.getNumberOfElements(); i++){
6             if(arr[j] == arrOne.getElementByIndex(i)){
7                 number++;
8             }
9         }
10    }
11    if(number == arr.length) return true;
12    else return false;
13 }
```