CSE 731-Software Testing

Project Report

Mutation Testing on Data Structure and Algorithms Using PITest and Strykar

Done By:
Deepanjali Ghosh MT2023119
Ketki Kerkar MT2023056

Project Aim:

The primary goal of this project is to apply Mutation Testing.This process is carried out using open-source tools, ensuring accessibility and practicality while rigorously testing the software for correctness and robustness.

Project Link: [Project Repo Link]

Code Used:

Algorithms play a critical role in software development, powering a wide range of applications to accomplish specific tasks efficiently. They are at the heart of systems ranging from simple programs to complex operations like data encryption, machine learning, and even space exploration. Given their significance, it is essential to ensure their correctness and reliability. To verify the accuracy and robustness of algorithms, we opted to apply mutation testing.

For this purpose, we utilized a diverse collection of data structure and algorithm (DSA) code that covers topics such as dynamic programming, graph algorithms, searching, sorting, queue, stack, linked list, and string manipulations. These implementations were tested in both Java and JavaScript. By focusing on these foundational algorithms, we aimed to validate their correctness under mutation testing and ensure their outputs remain accurate and reliable in various scenarios.

To be more specific, we have used the algorithms mentioned in this repo(Project Repo Link) for testing. The algorithms are used in both Java and Javascript.

Testing Strategy and Tools Used:

Mutation testing, also known as mutation analysis or program mutation, is a technique used to create new software tests and assess the effectiveness of current ones. It works by making minor changes to a program, resulting in altered versions known as mutants. The goal of testing is to identify and reject these mutants by triggering differences in behavior between the original program and the mutated version. Successfully detecting such differences is referred to as "killing the mutant.

There are 2 ways to “kill” a mutant:

- **Strongly Killing a Mutant**: A mutant is considered strongly killed if a test case produces an output or observable behavior that is different from the original program's output. In this scenario, the discrepancy in behavior directly demonstrates the presence of a fault in the mutant. This is the most robust form of mutant detection, as it ensures the mutation affects the program's visible behavior.
- **Weakly Killing a Mutant:**A mutant is weakly killed when a test case exercises the mutation and leads to a state (e.g., variable value) that differs from the original program, but this difference does *not* propagate to the program's final output or observable behavior. While the mutation is triggered, its effects are not visible in the external behavior of the program.

We opted for mutation testing as our testing approach, aiming to strongly eliminate the mutants.

1. IntelliJ IDEA: It is a widely used development environment tailored for Java and other programming languages. It offers a rich set of features, such as intelligent code suggestions, debugging capabilities, and seamless integration with testing frameworks.
2. PIT Mutation testing tool: Pitest is a mutation testing tool designed specifically for Java. It modifies the code in small ways (mutations) to assess whether your tests are robust enough to catch those changes.
3. VSCode: VS Code is a lightweight, versatile code editor widely used for JavaScript and other programming languages. It provides a user-friendly interface and is enriched with extensions for efficient development and testing.
4. Stryker: Stryker is a versatile mutation testing tool that supports various programming languages, including JavaScript, TypeScript, and C#. It complements Pitest for projects that involve languages other than Java.

Mutations Used

PIT, by default provides a set of mutation operators. These operators are listed below:

BOOLEAN_FALSE_RETURN	INCREMENTS_MUTATOR
BOOLEAN_TRUE_RETURN	INVERT_NEGS_MUTATOR
CONDITIONALS_BOUNDARY_MUTATOR	MATH_MUTATOR
EMPTY_RETURN_VALUES	NEGATE_CONDITIONALS_MUTATOR
PRIMITIVE_RETURN_VALS_MUTATOR	VOID_METHOD_CALL_MUTATOR
NULL_RETURN_VALUES	

More information about the mutation operators present in PITest can be found here.

Results Of Mutation Testing On The CodeBase(Java and Pitest):

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
45	97%	848/873	91%	690/755	92%	690/751

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.example.DynamicProgramming	13	98%	237/241	93%	253/271	94%	253/269
org.example.GraphAlgorithms	9	98%	207/212	91%	99/109	92%	99/108
org.example.LinkedList	1	99%	94/95	96%	52/54	98%	52/53
org.example.Queue	1	100%	27/27	95%	21/22	95%	21/22
org.example.Searching	4	94%	45/48	95%	53/56	95%	53/56
org.example.Sorting	8	95%	132/139	84%	118/140	84%	118/140
org.example.Stack	1	100%	17/17	100%	12/12	100%	12/12
org.example.String	8	95%	89/94	90%	82/91	90%	82/91

Report generated by PIT 1.17.1

Enhanced functionality available at arcmutate.com

1.Dynamic Programming:

Pit Test Coverage Report

Package Summary

org.example.DynamicProgramming

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
13	98%	237/241	93%	253/271	94%	253/269

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CoinChange.java	100%	18/18	95%	20/21	95%	20/21
EditDistance.java	100%	21/21	88%	22/25	88%	22/25
Knapsack.java	92%	12/13	100%	21/21	100%	21/21
LongestCommonSubsequence.java	100%	10/10	100%	15/15	100%	15/15
LongestIncreasingSubsequence.java	100%	44/44	90%	35/39	90%	35/39
LongestPalindromicSubsequence.java	100%	24/24	100%	14/14	100%	14/14
MatrixChainMultiplication.java	94%	15/16	94%	15/16	100%	15/15
PalindromePartitioning.java	100%	23/23	97%	28/29	97%	28/29
RodCutting.java	100%	9/9	100%	9/9	100%	9/9
ShortestCommonSupersequence.java	93%	27/29	90%	35/39	92%	35/38
StockBuyAndSell.java	100%	14/14	84%	16/19	84%	16/19
TargetSum.java	100%	13/13	93%	14/15	93%	14/15
UniquePaths.java	100%	7/7	100%	9/9	100%	9/9

Report generated by PIT 1.17.1

2.Graph Algorithms:

Pit Test Coverage Report

Package Summary

org.example.GraphAlgorithms

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
9	98%	207/212	91%	99/109	92%	99/108

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
BFS.java	100%	23/23	100%	3/3	100%	3/3
BellManFord.java	93%	13/14	92%	11/12	92%	11/12
Bipartite.java	95%	18/19	90%	9/10	90%	9/10
DFS.java	100%	19/19	100%	4/4	100%	4/4
Dijkstra.java	100%	31/31	75%	6/8	75%	6/8
FloydWarshall.java	94%	15/16	95%	18/19	95%	18/19
HamiltonianCycle.java	97%	31/32	93%	26/28	96%	26/27
KruskalsMST.java	97%	31/32	94%	16/17	94%	16/17
Prims.java	100%	26/26	75%	6/8	75%	6/8

Report generated by PIT 1.17.1

3.Linked List:

Pit Test Coverage Report

Package Summary

org.example.LinkedList

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	99%	94/95	96%	52/54	98%	52/53

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
LinkedList.java	99%	94/95	96%	52/54	98%	52/53

Report generated by PIT 1.17.1

4.Queue

Pit Test Coverage Report

Package Summary

org.example.Queue

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	100%	27/27	95%	21/22	95%	21/22

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
QueueClass.java	100%	27/27	95%	21/22	95%	21/22

Report generated by PIT 1.17.1

5.Searching

Pit Test Coverage Report

Package Summary

org.example.Searching

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
4	94%	45/48	95%	53/56	95%	53/56

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
BinarySearch.java	100%	12/12	92%	12/13	92%	12/13
ExponentialSearch.java	94%	16/17	96%	22/23	96%	22/23
JumpSearch.java	93%	13/14	93%	14/15	93%	14/15
LinearSearch.java	80%	4/5	100%	5/5	100%	5/5

Report generated by PIT 1.17.1

6.Sorting:

Pit Test Coverage Report

Package Summary

org.example.Sorting

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
8	95%	132/139	84%	118/140	84%	118/140

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
BubbleSort.java	89%	8/9	75%	9/12	75%	9/12
BucketSort.java	96%	27/28	78%	18/23	78%	18/23
CountingSort.java	94%	15/16	88%	14/16	88%	14/16
HeapSort.java	96%	22/23	78%	18/23	78%	18/23
InsertionSort.java	90%	9/10	90%	9/10	90%	9/10
MergeSort.java	100%	24/24	97%	31/32	97%	31/32
QuickSort.java	94%	17/18	81%	13/16	81%	13/16
SelectionSort.java	91%	10/11	75%	6/8	75%	6/8

Report generated by PIT 1.17.1

7.Stack:

Pit Test Coverage Report

Package Summary

org.example.Stack

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	100%	17/17	100%	12/12	100%	12/12

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
StackClass.java	100%	17/17	100%	12/12	100%	12/12

Report generated by PIT 1.17.1

8.String

Pit Test Coverage Report

Package Summary

org.example.String

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
8	95%	89/94	90%	82/91	90%	82/91

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Anagram.java	100%	6/6	100%	4/4	100%	4/4
FirstOccurance.java	80%	4/5	80%	4/5	80%	4/5
IsDigit.java	100%	7/7	100%	8/8	100%	8/8
IsPalindrome.java	100%	8/8	88%	7/8	88%	7/8
KMP.java	97%	31/32	73%	16/22	73%	16/22
LengthOfString.java	80%	4/5	100%	2/2	100%	2/2
RabinKarp.java	92%	22/24	97%	34/35	97%	34/35
ReverseString.java	100%	7/7	100%	7/7	100%	7/7

Report generated by PIT 1.17.1

Unit Mutation Operators Used:

LongestCommonSubsequence.java

```
1      package org.example.DynamicProgramming;
2
3      public class LongestCommonSubsequence {
4          public int lcs(String text1, String text2) {
5              int n = text1.length();
6              int m = text2.length();
7   2          int[][] dp = new int[n + 1][m + 1];
8
9   2          for (int i = 1; i <= n; i++) {
10  2              for (int j = 1; j <= m; j++) {
11  3                  if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
12  3                      dp[i][j] = 1 + dp[i - 1][j - 1];
13                     } else {
14  2                      dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
15                     }
16                 }
17             }
18  1          return dp[n][m];
19         }
20
21     }
```

Mutations

7	1. Replaced integer addition with subtraction → KILLED 2. Replaced integer addition with subtraction → KILLED
9	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
10	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
11	1. Replaced integer subtraction with addition → KILLED 2. Replaced integer subtraction with addition → KILLED 3. negated conditional → KILLED
12	1. Replaced integer subtraction with addition → KILLED 2. Replaced integer subtraction with addition → KILLED 3. Replaced integer addition with subtraction → KILLED
14	1. Replaced integer subtraction with addition → KILLED 2. Replaced integer subtraction with addition → KILLED
18	1. replaced int return with 0 for org/example/DynamicProgramming/LongestCommonSubsequence::lcs → KILLED

Mutations

16	1. Replaced integer subtraction with addition → KILLED 2. negated conditional → KILLED
20	1. Replaced integer addition with subtraction → KILLED
26	1. negated conditional → KILLED
28	1. replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED
30	1. Replaced integer subtraction with addition → KILLED
32	1. replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED
37	1. negated conditional → KILLED
39	1. replaced int return with 0 for org/example/Stack/StackClass::top → KILLED
41	1. replaced int return with 0 for org/example/Stack/StackClass::top → KILLED
46	1. negated conditional → KILLED 2. replaced boolean return with true for org/example/Stack/StackClass::isEmpty → KILLED

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

StackClass.java

```
1      package org.example.Stack;
2
3      public class StackClass {
4          private int maxSize;
5          private int[] stackArray;
6          private int top;
7
8          public StackClass(int size) {
9              maxSize = size;
10             stackArray = new int[maxSize];
11             top = -1;
12         }
13
14         // Method to push an element onto the stack
15         public void push(int value) {
16 2           if (top == maxSize - 1) {
17
18                 return;
19             }
20 1           stackArray[++top] = value;
21
22         }
23
24         // Method to pop an element from the stack
25         public int pop() {
26 1           if (top == -1) {
27
28 1               return -1;
29             }
30 1           int poppedElement = stackArray[top--];
31
32 1           return poppedElement;
33         }
34
35         // Method to peek the top element of the stack
36         public int top() {
37 1           if (top == -1) {
38
39 1               return -1;
40             }
41 1           return stackArray[top];
42         }
43
```

Mutations

16	1. Replaced integer subtraction with addition → KILLED 2. negated conditional → KILLED
20	1. Replaced integer addition with subtraction → KILLED
26	1. negated conditional → KILLED
28	1. replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED
30	1. Replaced integer subtraction with addition → KILLED
32	1. replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED
37	1. negated conditional → KILLED
39	1. replaced int return with 0 for org/example/Stack/StackClass::top → KILLED
41	1. replaced int return with 0 for org/example/Stack/StackClass::top → KILLED
46	1. negated conditional → KILLED 2. replaced boolean return with true for org/example/Stack/StackClass::isEmpty → KILLED

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Integration Mutation Operators Used:

```
public class MergeSort {

    public static void merge(int[] arr, int left, int mid, int right) {

        int n1 = mid - left + 1;
        int n2 = right - mid;

        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];


        System.arraycopy(arr, left, leftArray, 0, n1);
        System.arraycopy(arr, mid + 1, rightArray, 0, n2);


        int i = 0, j = 0;
        int k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                arr[k++] = leftArray[i++];
            } else {
                arr[k++] = rightArray[j++];
            }
        }

        while (i < n1) {
            arr[k++] = leftArray[i++];
        }


        while (j < n2) {
            arr[k++] = rightArray[j++];
        }
    }


    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;


            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);


            merge(arr, left, mid, right);
        }
    }
```

Active mutators

- CONDITIONALS_BOUNDARY
- CONSTRUCTOR_CALLS
- EMPTY_RETURNS
- EXPERIMENTAL_ARGUMENT_PROPAGATION
- EXPERIMENTAL_BIG_DECIMAL
- EXPERIMENTAL_BIG_INTEGER
- EXPERIMENTAL_MEMBER_VARIABLE
- EXPERIMENTAL_NAKED_RECEIVER
- EXPERIMENTAL_REMOVE_SWITCH_MUTATOR_[0-99]
- EXPERIMENTAL_SWITCH
- FALSE_RETURNS
- INCREMENTS
- INLINE_CONSTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NON_VOID_METHOD_CALLS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- REMOVE_CONDITIONALS_EQUAL_ELSE
- REMOVE_CONDITIONALS_EQUAL_IF
- REMOVE_CONDITIONALS_ORDER_ELSE
- REMOVE_CONDITIONALS_ORDER_IF
- REMOVE_INCREMENTS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Results Of Mutation Testing On The CodeBase(Javascript and Stryker):

All files

Mutants Tests

All files

932 97

File / Directory	Mutation Score: Of total	Mutation Score: Of covered	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	90.40	90.57	780	97	152	2	0	0	0	932	99	1031
DP	91.48	91.82	427	44	67	2	0	0	0	494	46	540
Searching	85.85	85.85	69	15	22	0	0	0	0	91	15	106
Sorting	85.93	85.93	122	28	49	0	0	0	0	171	28	199
linkedlist.js	93.79	93.79	122	9	14	0	0	0	0	136	9	145
queue.js	100.00	100.00	8	0	0	0	0	0	0	8	0	8
stack.js	96.97	96.97	32	1	0	0	0	0	0	32	1	33

1)Dynamic Programing

File / Directory	Mutation Score: Of total	Mutation Score: Of covered	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
DP	91.48	91.82	427	44	67	2	0	0	0	494	46	540
coinChange.js	92.68	92.68	34	3	4	0	0	0	0	38	3	41
EditDistance.js	95.45	95.45	38	2	4	0	0	0	0	42	2	44
knapsack.js	93.48	93.48	42	3	1	0	0	0	0	43	3	46
LongestCommonSubsequence.js	100.00	100.00	27	0	4	0	0	0	0	31	0	31
LongestIncreasingSubsequence.js	82.05	82.05	44	14	20	0	0	0	0	64	14	78
LongestPalindromicSubsequence.js	87.50	92.11	31	3	4	2	0	0	0	35	5	40
MatrixChainMultiplication.js	96.30	96.30	24	1	2	0	0	0	0	26	1	27
PalindromePartitioning.js	89.39	89.39	57	7	2	0	0	0	0	59	7	66
RodCutting.js	100.00	100.00	14	0	3	0	0	0	0	17	0	17
ShortestCommonSpersequence.js	92.00	92.00	52	6	17	0	0	0	0	69	6	75
StockBuyAndSell.js	93.75	93.75	29	2	1	0	0	0	0	30	2	32
TargetSum.js	92.86	92.86	25	2	1	0	0	0	0	26	2	28
UniquePaths.js	93.33	93.33	10	1	4	0	0	0	0	14	1	15

Coin Change:

All files / DP / coinChange.js

932 97

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
coinChange.js	92.68	92.68	34	3	4	0	0	0	0	38	3	41

Killed (34) Survived (3) Timeout (4)

```
class CoinChange {
    // Function to count the number of combinations
    static coinchange(coins, amount) {
        const combinations = new Array(amount + 1).fill(0);
        combinations[0] = 1;

        for (let coin of coins) {
            for (let i = coin; i <= amount; i++) {
                combinations[i] += combinations[i - coin];
            }
        }

        return combinations[amount];
    }

    // Function to find the minimum number of coins
    static minimumCoins(coins, amount) {
        const minimumCoins = new Array(amount + 1).fill(Infinity);
        minimumCoins[0] = 0;

        for (let i = 1; i <= amount; i++) {
            for (let coin of coins) {
                if (coin <= i) {
```

2)Linked List

linkedlist.js

Mutants Tests

All files / linkedlist.js

932 97

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
linkedlist.js	93.79	93.79	122	9	14	0	0	0	0	136	9	145

Killed (122) Survived (9) Timeout (14)

```
class Node {
    constructor(data) {
        this.data = data;
        this.next = null;
    }
}

function insertAtFront(head, newData) {
    const newNode = new Node(newData);
    newNode.next = head;
    return newNode;
}

function append(head, newData) {
    const newNode = new Node(newData);
    if (head === null) {
        return newNode;
    }
    let last = head;
```

BlockStatement Killed

3)Stack

932 97

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
stack.js	96.97	96.97	32	1	0	0	0	0	0	32	1	33

✅ Killed (32) 👽 Survived (1)

```
class Stack {
    constructor(size) {
        this.maxSize = size;
-       this.stackArray = [];
+       this.stackArray = ["Stryker was here"];
        this.topIndex = -1;  // Changed `top` to `topIndex` to avoid name conflict
    }

    // Method to push an element onto the stack
    push(value) {
        if (this.topIndex === this.maxSize - 1) {
            console.log("Stack overflow");
            return;
        }
        this.stackArray[++this.topIndex] = value;
        //console.log(value + " pushed into the stack");
    }

    // Method to pop an element from the stack
    pop() {
        if (this.topIndex === -1) {
            console.log("Stack underflow");
            return -1;
```

👽 ArrayDeclaration Survived (4:27) More

☂ Covered by 9 tests (yet still survived)

4)Sorting

Sorting

Mutants Tests

All files / Sorting

932 97

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
Sorting	85.93	85.93	122	28	49	0	0	0	0	171	28	199
BubbleSort.js	72.73	72.73	13	6	3	0	0	0	0	16	6	22
CountingSort.js	89.29	89.29	20	3	5	0	0	0	0	25	3	28
HeapSort.js	88.89	88.89	31	5	9	0	0	0	0	40	5	45
InsertionSort.js	84.21	84.21	12	3	4	0	0	0	0	16	3	19
MergeSort.js	86.44	86.44	31	8	20	0	0	0	0	51	8	59
QuickSort.js	88.46	88.46	15	3	8	0	0	0	0	23	3	26

5)Searching

All files / Searching

932 97

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
Searching	85.85	85.85	69	15	22	0	0	0	0	91	15	106
BinarySearch.js	95.45	95.45	13	1	8	0	0	0	0	21	1	22
ExponentialSearch.js	85.71	85.71	27	6	9	0	0	0	0	36	6	42
JumpSearch.js	77.42	77.42	21	7	3	0	0	0	0	24	7	31
LinearSearch.js	90.91	90.91	8	1	2	0	0	0	0	10	1	11

6)Queue

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
queue.js	100.00	100.00	8	0	0	0	0	0	0	8	0	8

✅Killed (8)

```
class Queue {
    constructor() {
        this.items = {}
        this.frontIndex = 0
        this.backIndex = 0
    }
    enqueue(item) {
        this.items[this.backIndex] = item
        this.backIndex++
        return item + ' inserted'
    }
    dequeue() {
        const item = this.items[this.frontIndex]
        delete this.items[this.frontIndex]
        this.frontIndex++
        return item
    }
    peek() {
        return this.items[this.frontIndex]
    }
    get printQueue() {
        return this.items;
    }
}


module.exports = Queue;
```