Software Testing HW2

Created by	Ⓣ Timmy Huang
Created time	@June 23, 2024 5:22 PM
Tags	

(a) List all of the input variables, including the (abstract) state variables.

Method	Input
constructor(capacity)	state of queue, capacity
enqueue(element)	state of queue, element
dequeue()	state of queue
is_empty()	state of queue
is_full()	state of queue

(b) Define the characteristics of the input variables. Make sure you cover all input variables.

First, we develop characteristics related to each method.

Method	Input	Characteristics
constructor(capacity)	state of queue, capacity	capacity ≥ 0 (C1)
enqueue(element)	state of queue, element	type of element is “number” (C2), element is an number (C3), queue is not full (C4)
dequeue()	state of queue	queue is not empty (C5)
is_empty()	state of queue	queue is not empty (C5)
is_full()	state of queue	queue is not full (C4)

Then, we determine the characteristics of each input in each method.

Method	Input	Characteristics
constructor(capacity)	state of queue	C4, C5
	capacity	C1
enqueue(element)	state of queue	C4, C5
	element	C2, C3
dequeue()	state of queue	C4, C5
is_empty()	state of queue	C4, C5

is_full()	state of queue	C4, C5

(c) Partition the characteristics into blocks. Designate one block in each partition as the "Base" block.

Characteristics	Partitions	Base
C1	{true, false}	true
C2	{true, false}	true
C3	{true, false}	true
C4	{true, false}	true
C5	{true, false}	true

(d) Define values for each block.

Characteristics	Block	Value (e.g.)
C1	true	3
	false	-1
C2	true	2
	false	“abc”
C3	true	2
	false	Number.NaN
C4	true	[1, ,] (capacity == 3)
	false	[1, 2, 3](capacity == 3)
C5	true	[1, ,] (capacity == 3)
	false	[, ,] (capacity == 3)

(e) Define a test set that satisfies Base Choice Coverage (BCC). Write your tests with the values from the previous step. Be sure to include the test oracles.

Method	Characteristics	Test	Infeasible Test	# of Tests
constructor(capacity)	C1, C4, C5	{TTT, FTT, TFT, TTF}	none	4
enqueue(element)	C2, C3, C4, C5	{TTTT, FTTT, TFTT, TTFT, TTTF}	none	5
dequeue()	C4, C5	{TT, FT, TF}	none	3
is_empty()	C4, C5	{TT, FT, TF}	none	3
is_full()	C4, C5	{TT, FT, TF}	none	3

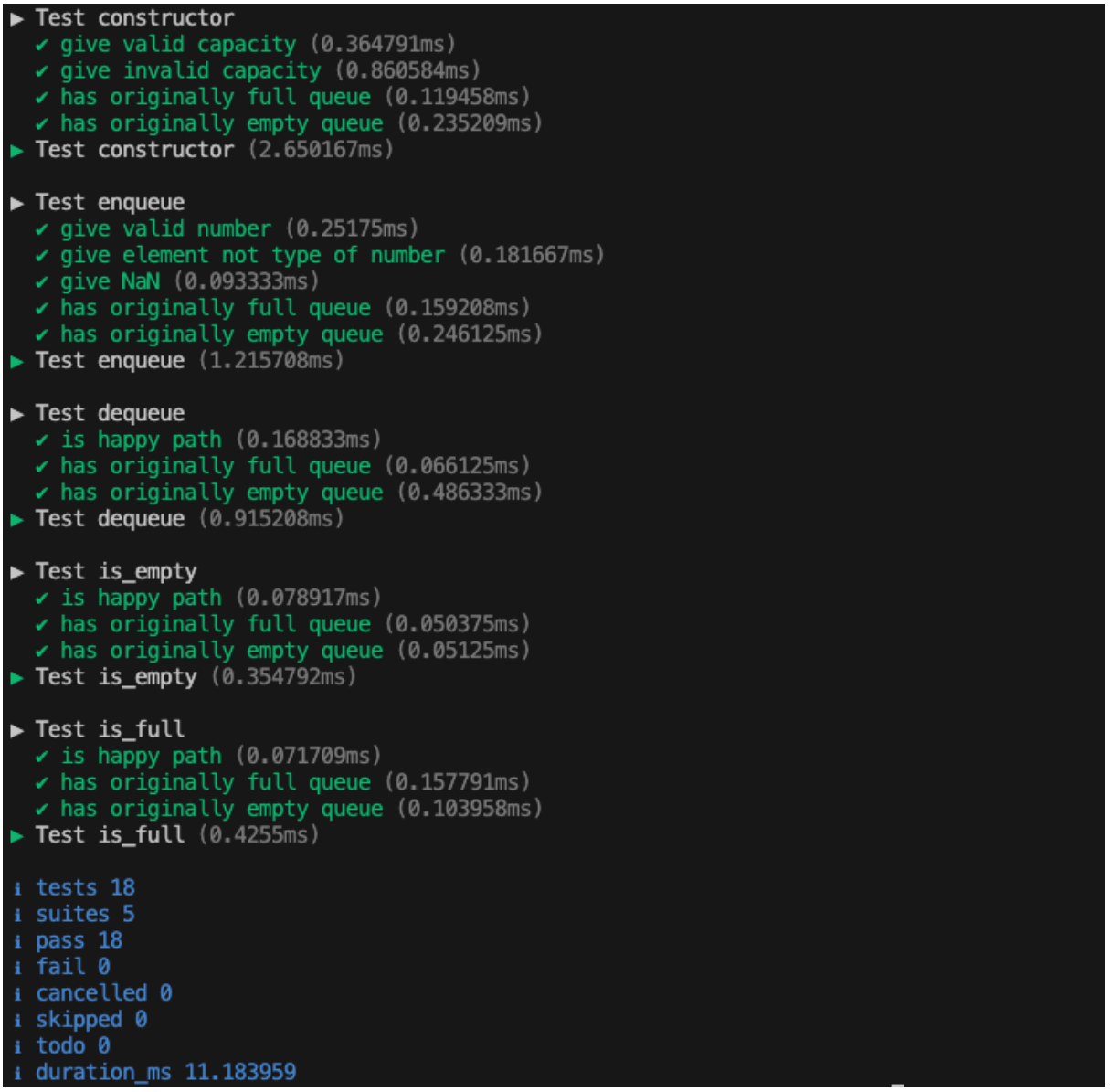
```
describe("Test constructor", () => {
  beforeEach(() => {
    queue = new BoundedQueue(3);
    queue.enqueue(1);
  });

  it("give valid capacity", () => {
    queue = new BoundedQueue(2);
    assert.strictEqual(
      queue.toString(),
      "[] is_empty(): true, is_full(): false"
    );
    assert.strictEqual(queue.capacity, 2);
  });

  it("give invalid capacity", () => {
    const err = new RangeError("capacity is less than 0");
    assert.throws(() => (queue = new BoundedQueue(-1)), err);
  });

  it("has originally full queue", () => {
    queue.enqueue(456);
    queue.enqueue(789);
    queue = new BoundedQueue(2);
    assert.strictEqual(
      queue.toString(),
      "[] is_empty(): true, is_full(): false"
    );
    assert.strictEqual(queue.capacity, 2);
  });

  it("has originally empty queue", () => {
    queue.dequeue();
    queue = new BoundedQueue(2);
    assert.strictEqual(
      queue.toString(),
      "[] is_empty(): true, is_full(): false"
    );
    assert.strictEqual(queue.capacity, 2);
  });
```

```
});

describe("Test enqueue", () => {
  beforeEach(() => {
    queue = new BoundedQueue(3);
    queue.enqueue(1);
  });

  it("give valid number", () => {
    queue.enqueue(2);
    assert.strictEqual(
      queue.toString(),
      "[1, 2] is_empty(): false, is_full(): false"
    );
  });

  it("give element not type of number", () => {
    const err = new RangeError("element is invalid");
    assert.throws(() => queue.enqueue("adc"), err);
  });

  it("give NaN", () => {
    const err = new RangeError("element is invalid");
    assert.throws(() => queue.enqueue(Number.NaN), err);
  });

  it("has originally full queue", () => {
    queue.enqueue(2);
    queue.enqueue(3);
    const err = Error("queue is full");
    assert.throws(() => queue.enqueue(4), err);
  });

  it("has originally empty queue", () => {
    queue.dequeue();
    queue.enqueue(2);
    assert.strictEqual(
      queue.toString(),
      "[2] is_empty(): false, is_full(): false"
    );
  });
```

```
});

describe("Test dequeue", () => {
  beforeEach(() => {
    queue = new BoundedQueue(3);
    queue.enqueue(1);
  });

  it("is happy path", () => {
    assert.strictEqual(queue.dequeue(), 1);
    assert.strictEqual(
      queue.toString(),
      "[] is_empty(): true, is_full(): false"
    );
  });

  it("has originally full queue", () => {
    queue.enqueue(2);
    queue.enqueue(3);
    assert.strictEqual(queue.dequeue(), 1);
    assert.strictEqual(
      queue.toString(),
      "[2, 3] is_empty(): false, is_full(): false"
    );
  });

  it("has originally empty queue", () => {
    assert.strictEqual(queue.dequeue(), 1);
    const err = new Error("queue is empty");
    assert.throws(() => queue.dequeue(), err);
  });
});

describe("Test is_empty", () => {
  beforeEach(() => {
    queue = new BoundedQueue(3);
    queue.enqueue(1);
  });

  it("is happy path", () => {
    assert.strictEqual(queue.is_empty(), false);
```

```
  });

  it("has originally full queue", () => {
    queue.enqueue(2);
    queue.enqueue(3);
    assert.strictEqual(queue.is_empty(), false);
  });

  it("has originally empty queue", () => {
    assert.strictEqual(queue.dequeue(), 1);
    assert.strictEqual(queue.is_empty(), true);
  });
});

describe("Test is_full", () => {
  beforeEach(() => {
    queue = new BoundedQueue(3);
    queue.enqueue(1);
  });

  it("is happy path", () => {
    assert.strictEqual(queue.is_full(), false);
  });

  it("has originally full queue", () => {
    queue.enqueue(2);
    queue.enqueue(3);
    assert.strictEqual(queue.is_full(), true);
  });

  it("has originally empty queue", () => {
    assert.strictEqual(queue.dequeue(), 1);
    assert.strictEqual(queue.is_full(), false);
  });
});
```

And we have the result:


▶ Test constructor
✔ give valid capacity (0.364791ms)
✔ give invalid capacity (0.860584ms)
✔ has originally full queue (0.119458ms)
✔ has originally empty queue (0.235209ms)
▶ Test constructor (2.650167ms)
▶ Test enqueue
✔ give valid number (0.25175ms)
✔ give element not type of number (0.181667ms)
✔ give NaN (0.093333ms)
✔ has originally full queue (0.159208ms)
✔ has originally empty queue (0.246125ms)
▶ Test enqueue (1.215708ms)
▶ Test dequeue
✔ is happy path (0.168833ms)
✔ has originally full queue (0.066125ms)
✔ has originally empty queue (0.486333ms)
▶ Test dequeue (0.915208ms)
▶ Test is_empty
✔ is happy path (0.078917ms)
✔ has originally full queue (0.050375ms)
✔ has originally empty queue (0.05125ms)
▶ Test is_empty (0.354792ms)
▶ Test is_full
✔ is happy path (0.071709ms)
✔ has originally full queue (0.157791ms)
✔ has originally empty queue (0.103958ms)
▶ Test is_full (0.4255ms)
ℹ tests 18
ℹ suites 5
ℹ pass 18
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 11.183959