Gheorghiu Ioana
Group: 913

Data Structures and Algorithms Project

Project Theme: 23. ADT Priority Queue – implementation on a singly linked list on an array

1. **Domain** of the Abstract Data Type

$\mathcal{PQ}$ = { pq | pq is a priority queue with elements (e, p), e ∈ TElem, p ∈ TPriority }

2. **Interface** of the Abstract Data Type

- init(pq, R)
  // Description: creates a new empty priority queue
  // Pre: R is a relation over the priorities, R: TPriority x Tpriority
  // Post: pq ∈ $\mathcal{PQ}$, pq is an empty priority queue

- destroy(pq)
  // Description: destroyes a priority queue
  // Pre: pq ∈ $\mathcal{PQ}$
  // Post: pq was destroyed

- push(pq, e, p)
  // Description: pushes (adds) a new element to the priority queue
  // Pre: pq ∈ $\mathcal{PQ}$, e ∈ TElem, p ∈ TPriority
  // Post: pq' ∈ $\mathcal{PQ}$, pq' = pq ⊕ (e, p)

- pop(pq, e, p)
  // Description: pops(removes) from the priority queue the element with the highest priority. It returns both the element and its priority
  // Pre: pq ∈ $\mathcal{PQ}$
  // Post: pq' ∈ $\mathcal{PQ}$, pq' = pq ⊖ (e, p)
  e ∈ TElem, p ∈ Tpriority, e is the element with the highest priority from the priority queue, p is its priority
  // Throws: an exception if the priority queue is empty

- top(pq, e, p)
  // Description: returns from the priority queue the element with the highest priority and its priority
  // Pre: pq ∈ $\mathcal{PQ}$
  // Post: pq' ∈ $\mathcal{PQ}$, pq' = pq ⊖ (e, p)
  e ∈ TElem, p ∈ Tpriority, e is the element with the highest priority from the priority queue, p is its priority
  // Throws: an exception if the priority queue is empty

Gheorghiu Ioana
Group: 913

- isEmpty(pq)
  // Description: checks if the priority queue is empty
  // Pre: pq ∈ $\mathcal{PQ}$
  // Post: isEmpty ← true, if pq has no elements
  false, otherwise

- increasePriority(pq, e, p)
  // Description: increases the priority of a given elements
  // Pre: pq ∈ $\mathcal{PQ}$, e ∈ TElem, p ∈ Tpriority
  // Post: pq ∈ $\mathcal{PQ}$, e ∈ TElem, p’ ∈ Tpriority, p’ > p, p’ is the new priority of e

3. **Representation** of the ADT Priority Queue on a singly linked list on an array

Pair:	Node:	SLLA:
elem: TElem p: TPriority	e: Pair next: Integer	elems: Node[] cap: Integer size: Integer head: Integer firstEmpty: Integer R: Relation

4. **Implementations** of the operations for ADT Priority Queue on a SLLA

subalgorithm initPriorirtyQueue(pq, R, capacity) is:

```
// Description: creates a new empty priority queue
// Pre: R is a relation over the priorities, R: TPriority x Tpriority
// Post: pq ∈ PQ , pq is an empty priority queue
// Complexity: O(n)
{
        pq.size ← 0
        pq.cap ← capacity
        pq.elems = ↑initNode(Node, pq.cap)
        for i ← 0, i < pq.cap, i++
                new_elems[i].next ← i + 1
        new_elems[pq.cap].next <- -1
        pq.head = -1
        pq.firstEmpty = 0
        pq.priority = R
}
```

Gheorghiu Ioana
Group: 913

<u>subalgorithm push(pq, a, t) is:</u>

```
    // Description: pushes (adds) a new element to the priority queue
    // Pre: pq ∈ 𝒫𝒬, a ∈ Airplane, t ∈ Double
    // Post: pq' ∈ 𝒫𝒬, pq' = pq ⊕ (e, p)
    // Complexities:
    //        Best case: θ(1)
    //        Worst case: θ(size)
    //        Average case: θ(size) amortized = (1 + 2 + … + size)/size = (Σ_{i=1}^{i<=size} i)/size = size(size+1)/(2 *size)
    //        (amortized, as we are not taking into consideration the resizing operation, considering the small
    //        number of times it should be performed
    {
        if size = capacity then
            resize(pq)
        pair ← init(Pair, e, p);
        if head = -1 then
            insertEmptyQueue(pq, pair);
        else
            prev_pos ← -1, current_pos ← head
            condition ← True
            while condition execute
                if current_pos = -1
                    condition = False
                else
                    airplane ← elems[current_pos].pair.airplane
                    if Relation.compare(a, airplane) == True
                        prev_pos ← current_pos
                        current_pos ← elems[current_pos].next
                    else condition ← False
            if prev_pos = -1
                insertFirst(pq, pair)
            else
                insertOnPosition(pq, pari, position)
        size = size + 1
    }
```

<u>subalgorithm resize(pq, factor) is:</u>

```
    // Description: multiplies the capacity of the priority queue by factor
    // Pre: pq ∈ 𝒫𝒬, factor ∈ Integer
    // Post: pq' ∈ 𝒫𝒬, pq.cap = pq.cap * factor
    // Complexity: O(size)
    {
        pq.cap ← pq.cap * factor
        new_elems[] ← init(Node, pq.cap)

        for i ← pq.size, i < pq.cap – 1, i++ execute
            new_elems[i].next ← i + 1
        new_elems[pq.cap].next <- -1
        pq.firstEmpty = pq.size

        for i ← 0, i < pq.size, i++ execute
            new_elems[i] ← pq.elems[i]
    }
```

Gheorghiu Ioana
Group: 913

<u>subalgorithm insertEmptyQueue(pq, pair) is:</u>

```
    // Description: adds a Pair to an empty Priority Queue
    // Pre: pq ∈ 𝒫𝒬, pair ∈ Pair
    // Post: pq' ∈ 𝒫𝒬, pair ∈ pq, pq.top = pq.elems[pq.head].pair
    // Complexity: O(1)
    {
        new_node ← init(Node, pair, -1);
        new_position ← pq.firstEmpty
        pq.firstEmpty ← pq.elems[new_position].next
        pq.elems[new_position] ← new_node
        pq.head ← new_position
    }
```

<u>subalgorithm insertOnPosition(pq, pair, prev_position) is:</u>

```
    // Description: adds a Pair to a Priority Queue
    // Pre: pq ∈ 𝒫𝒬, pair ∈ Pair
    // Post: pq' ∈ 𝒫𝒬, pair ∈ pq, pq.top = pq.elems[pq.head].pair
    // Complexity: O(1)
    {
        new_node ← init(Node, pair, -1);
        new_position ← pq.firstEmpty
        pq.firstEmpty ← pq.elems[new_position].next
        pq.elems[new_position] ← new_node
        pq.elems[prev_position].next ← new_position
    }
```

<u>subalgorithm insertFirst(pq, pair) is:</u>

```
    // Description: adds a Pair to the top of the Priority Queue
    // Pre: pq ∈ 𝒫𝒬, pair ∈ Pair
    // Post: pq' ∈ 𝒫𝒬, pair ∈ pq, pq.top = pq.elems[pq.head].pair
    // Complexity: O(1)
    {
        new_node ← init(Node, pair, pq.head);
        new_position ← pq.firstEmpty
        pq.firstEmpty ← pq.elems[new_position].next
        pq.elems[new_position] ← new_node
        pq.head ← new_position
    }
```

<u>function pop(pq) is:</u>

```
    // Description: pops(removes) and returns the pair with the highest priority from the priority queue
    // Pre: pq ∈ PQ
    // Post: pq' ∈ PQ , pq' = pq (e, p) ⊖ e TElem, p Tpriority, e is the element with the highest priority from the
            priority ∈ ∈ queue, p is its priority
    // Throws: an exception if the priority queue is empty
    // Complexity: O(1)
    {
        if pq.size = 0
            throw exception "Priority queue is empty!"
        int emptied_position = pq.head
        pair ← pq.elems[pq.head].pair
        pq.head = pq.elems[pq.head].next
        pq.elems[emptied_position].next ← pq.firstEmpty
        pq.firstEmpty ← emptied_position
        pop ← pair
    }
```

Gheorghiu Ioana
Group: 913

<u>function top(pq) is:</u>

```
    // Description: returns from the priority queue the element with the highest priority and its priority
    // Pre: pq ∈ PQ
    // Post: pq ∈ PQ, Pair contains the element with the highest priority from the priority queue and its priority
    // Throws: an exception if the priority queue is empty
    // Complexity: O(1)
    {
        if pq.size = 0
            throw exception “Priority queue is empty!”
        int emptied_position = pq.head
        pair ← pq.elems[pq.head].pair
        top ← pair
    }
```

<u>function isEmpty(pq) is:</u>

```
    // Description: checks if the priority queue is empty
    // Pre: pq ∈ PQ
    // Post: isEmpty ← true, if pq has no elements
                       false, otherwise
    // Complexity: O(1)
    {
        if pq.size = 0
            isEmpty ← true
        isEmpty ← false
    }
```

5. **Tests** for the Priority Queue ADT

<u>void Test::test()</u>

```
    {
        PriorityQueue pq{new TimePriority()};

        Airplane a1{ 1, "WizzAir", "Bucharest", 15.00 };
        Airplane a2{ 2, "WizzAir", "CLuj-Napoca", 14.30 };
        Airplane a3{ 4, "Lufthansa", "Munich", 14.45 };
        Airplane a4{ 3, "Blue Air", "Bucharest", 17.00 };

        // A1 IS ADDED FIRST IN THE PQ -> TEST INSERT IN EMPTY QUEUE
        pq.push(a1, a1.getHour());
        assert(pq.getSize() == 1);
        assert(pq.top().getAirplane().getID() == a1.getID());

        // A2 IS ADDED SECOND BUT WILL BE THE FIRST IN THE PQ -> TEST INSERT FIRST
        pq.push(a2, a2.getHour());
        assert(pq.getSize() == 2);
        assert(pq.top().getAirplane().getID() == a2.getID());

        // A3 WILL BE THE SECOND IN THE PQ -> TEST INSERT ON POSITION
        pq.push(a3, a3.getHour());
        assert(pq.getSize() == 3);

        // A4 WILL BE THE LAST IN THE PQ -> TEST INSERT LAST
        pq.push(a4, a4.getHour());
        assert(pq.getSize() == 4);
```

Gheorghiu Ioana
Group: 913

```
		// VERIFY IF A3 IS THE SECOND AND A4 IS THE FOURTH -> TEST TOP AND POP
		pq.pop();
		assert(pq.top().getAirplane().getID() == a3.getID());
		pq.pop(); pq.pop();
		assert(pq.top().getAirplane().getID() == a4.getID());

		// EXTRACT THE LAST ELEMENT -> TEST isEmpty
		assert(pq.isEmpty() == false);
		pq.pop();
		assert(pq.isEmpty() == true);
	}

void Test::testExceptions()
	{
		PriorityQueue pq{ new TimePriority() };

		// PRIORITY QUEUE IS EMPTY, CALLING POP OR TOP ON IT THROWS EXCEPTIONS
		try {
			pq.pop();
		}
		catch (std::exception & ex)
		{
			std::string exception = ex.what();
			assert(exception == "Priority queue is empty!");
		}

		try {
			pq.top();
		}
		catch (std::exception & ex)
		{
			std::string exception = ex.what();
			assert(exception == "Priority queue is empty!");
		}
	}
```

Gheorghiu Ioana
Group: 913

6. **Problem Statement** + **Justification**

Simulate the flights display in the arrival's terminal of an airport with two gates. You should be able to see at any time the last arrived airplanes for each of the two gates and also the next flight scheduled to arrive. Marking the next scheduled flight as arrived should put it on the display of the right gate. An airplane needs a total time of 30 minutes to disembark passengers and make the preparations for the next flight or leave the gate. If an airplane arrives before any gate can be emptied, it should be displayed that it is redirected to another terminal. You should also be able to add at any time a new airplane scheduled later for the day by adding its ID, company, origin and arrival time.

I have chosen the above problem statement because priority queues are of great use in managing events in simulations like an airport scenario. In this case the priority of the element is represented by the scheduled time of landing. All the main typical operations for a priority queue can be used in such a problem.

The Next Arrival display will always contain the data of the top of the priority queue, while the gate displays will show the data of the last popped items. Adding a new flight is done by entering info like ID, company and origin, and the last argument, arrival time represents the priority of the element: earlier flights are located nearer the top. The flight will later be displayed only at the right time according to its arrival time. Exceptions thrown by pop and top functions are handled by displaying the message "No incoming airplane!" instead of the next flight information.

7. **Pseudocode** for the solution of the problem

```
subalgoritm initController(ctr, PriorityQueue pq) is:
	// Description: initializes controller with a given priority queue of objects of type Airplanes
	// Pre: pq ∈ PQ
	// Post: ctr ∈ PQ
	// Complexity: O(1)
	{
		ctr.airplanes ← pq;
		ctr.gate_1 ←initPair()
		ctr.gate_2 ← initPair()
	}

subalgorithm set_landed(ctr) is:
	// Description: Redirects the arriving airplane to the right gate, but only if there is one available; pops an item
	from the priority queue, the one set as next arrival before
	// Post: if possible, gate_1 or gate_2 are assigned a new Airplane and last_gate is updated
	// Complexity: O(1)
	{
		if ctr.is_available_1 then
			ctr.gate_1 ← ctr.airplanes.pop()
			ctr.last_gate ← 1
		else if ctr.is_available_2 then
			ctr.gate_2 ← ctr.airplanes.pop()
			ctr.last_gate ← 2
		else ctr.airplanes.pop()
	}
```

Gheorghiu Ioana
Group: 913

```
function get_last_arrival(ctr) is:
	// Description: returns information about the last popped item, i.e., the last airplane arrived
	// Complexity: O(1)
	{
		if ctr.last_gate = 1 then
			function ← ctr.gate_1.airplane
		else
			function ← ctr.gate_2.airplane
	}

functions is_available_1(ctr) is:
	// Description: returns true if the airplane at gate 1 arrived for more than 30 minutes and false otherwise
	// Complexity: O(1)
	{
		next ← ctr.airplanes.top()
		if time between next.priority and ctr.gate_1.priority is less than 30 minutes then
			is_available_1 ← True
		else
			is_available_1 ← False
	}

functions is_available_2(ctr) is:
	// Description: returns true if the airplane at gate 2 arrived for more than 30 minutes and false otherwise
	// Complexity: O(1)
	{
		next ← ctr.airplanes.top()
		if time between airplanes is less than 30 minutes then
			is_available_2 ← True
		else
			is_available_2 ← False
	}

subalgorithm add_airplane(ctr, airplane) is:
	// Description: pushes a new pair to the priority queue
	// Pre: airplane Airplane
	// Post: pq' ∈ PQ, airplane and priority ∈ pq
	// Complexity of push operation
	{
		ctr.airplanes.push(airplane, airplane.arrivalTime);
	}
```