Queues

August 8, 2020

1 Queues

- https://opendsa-server.cs.vt.edu/ODSA/Books/CS2/html/Queue.html
- https://en.cppreference.com/w/cpp/container/queue

1.1 Table of Contents

1.2 Introduction

- queue is a list-like data structure in which elements are inserted at the back and removed from front
  - less flexible than list
  - more efficient and easier to implement
- operates as **FIFO** (First-In, First-Out) data structure
- many applications require the limited form of insert and remove operations that queue provides
- mimics real-world queues (lines) e.g., line of customers at restaurants/banks, queue of cars in drive-through

1.2.1 Applications

- in computers, CPU scheduling, disk scheduling, file IO, network IO, etc.
- email queues, print queues

1.2.2 Operations

- enqueue : insert element at the back of queue
- dequeue : remove and return element from the front of the queue

1.3 Implementations of Queue as ADT

- Queue can be implemented using array or linked-list

1.3.1 Array implementation of Queue

- implementing array-based queue is as simple as stack

- below is the array-based queue

1.3.2 Visualization of Array-based Queue

https://opendsa-server.cs.vt.edu/ODSA/Books/CS2/html/Queue.html

[ ]:
```
#include <iostream>
#include <cassert>

using namespace std;
```

[ ]:
```
template<class T>
class ArrayQueue {
  private:
    size_t maxSize;
    size_t front, back;
    T * queue;
    size_t count;

  public:
    ArrayQueue(size_t mSize=100) { //constructor
      assert(mSize > 0);
      maxSize = mSize;
      queue = new T[maxSize];
      front = back = 0;
      count = 0;
    }

    // clear the queue
    void clear() { front = back = count = 0; }

    // get the size of the queue
    size_t size() { return count; }

    // check if queue is empty
    bool empty() { return count == 0; };

    // check if queue is full
    bool full() { return count == maxSize; }

    // return the max size
    size_t max_size() { return maxSize; }

    // add element to the end of queue
    void enqueue(T value) {
        if (full()) return;
        queue[back] = value;
        // circular increment
```

```
        back = (back+1)%maxSize;
        count++;
    }

    // remove and return the element from the front of the queue
    T dequeue() {
        T data = queue[front];
        //circular increment
        front = (front+1)%maxSize;
        count--;
        return data;
    }

    T next() {
        return queue[front];
    }
};
```

1.4 Test ArrayQueue Implementation

[ ]:
```
ArrayQueue<int> aQ(5);
```

[4]:
```
aQ.enqueue(10);
aQ.enqueue(20);
aQ.enqueue(30);
cout << "size of aQ = " << aQ.size();
```

```
size of aQ = 3
```

[5]:
```
cout << "front of the queue is: " << aQ.dequeue() << endl;
cout << "now the aQ size = " << aQ.size() << endl;
```

```
front of the queue is: 10
now the aQ size = 2
```

[12]:
```
aQ.enqueue(40);
aQ.enqueue(50);
aQ.enqueue(60);
cout << "size of aQ = " << aQ.size();
```

```
size of aQ = 5
```

[10]:
```
aQ.enqueue(70);
cout << "size of aQ = " << aQ.size();
```

```
size of aQ = 1
```

[8]:
```
cout << "max_size of aQ = " << aQ.max_size();
```

```
max_size of aQ = 5
```

[13]:
```
while (!aQ.empty()) {
    cout << " next element = " << aQ.dequeue() << endl;
}
```

```
 next element = 70
 next element = 40
 next element = 50
 next element = 60
 next element = 40
```

1.5 Linked Queue Implementation

- https://opendsa-server.cs.vt.edu/ODSA/Books/CS2/html/QueueLinked.html
- elements are inserted after the tail and removed only from the head of the list
- header nodes are not required because no special-cases need to be handled
- implementation and test is left as an assignment

[14]:
```
#include <iostream>
#include <cassert>

using namespace std;
```

[ ]:
```
template<class T>
struct Node {
  T data;
  Node<T> * next;
};
```

[ ]:
```
template<class T>
class LinkedQueue{
  private:
    size_t nodeCount;
    Node<T> * head;
    Node<T> * tail;
  public:
    //constructor
    LinkedQueue();

    // clear the Queue
    void clear();

    // get the size of the Queue
    size_t size();
```

```
    // check if Queue is empty
    bool empty();

    //insert data at the end of the Queue
    void enqueue(T value);

    // remove and return element from front of the Queue
    T dequeue();
};
```

1.6 Test Linked Queue Implementation

[ ]:
```
LinkedQueue<int> lQ;
```

[ ]:
```
lQ.enqueue(10);
lQ.enqueue(20);
lQ.enqueue(30);
cout << "size of lQ = " << lQ.size();
```

1.7 Exercises

1. Server - https://open.kattis.com/problems/server

- Ferry Loading III - https://open.kattis.com/problems/ferryloading3
- Foosball Dynasty - https://open.kattis.com/problems/foosball

[ ]:
```

```