Data Structures and Algorithms (3)

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Chapter 3 Stacks and Queues

- Stacks
- Applications of Stacks
  - Implementation of Recursion using Stacks
- **Queues**

Definition of queues

- **First In First Out**
  - Linear lists that limit accessing point
    - Release elements according to the order of arrival
    - All the insertions occur at one end of the list and all the deletions occur at the other end
- **Main elements**
  - front
  - rear

Main operations of queues

- Insert an element into the queue (enQueue)
- Remove an element from the queue (deQueue)
- Get the element in the front (getFront)
- Judge whether the queue is empty (isEmpty)

Abstract data type of queues

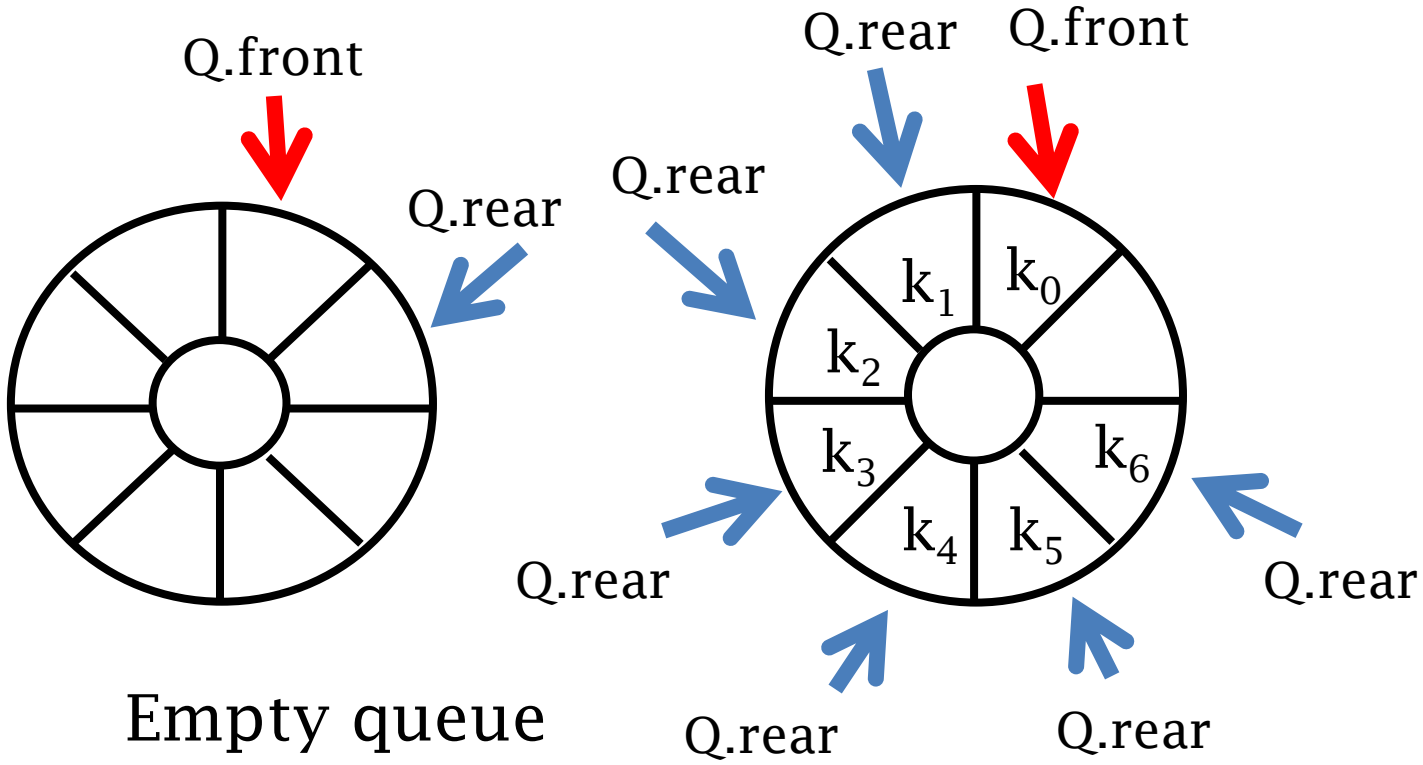
```
template <class T> class Queue {
public:                    // operation set of the queue
   void clear();           // change into empty queue
   bool enQueue(const T item); // insert item into the end of the queue, return true if
succeed, otherwise return false
   bool deQueue(T & item) ;
    // return the front element of the queue and remove it,return true if succeed
   bool getFront(T & item);
    // return the front element of the queue and do not remove it,return true if succeed

   bool isEmpty(); // return true if the queue is empty
   bool isFull();  // return true if the queue is full
};
```

Implementation of Queues

- **Sequential queue**
  - **The key point** is how to prevent false overflow
- **Linked queue**
  - Use single linked list to store, every element in the queue corresponds to a node in the linked list

Queue : Ring(true pointers)

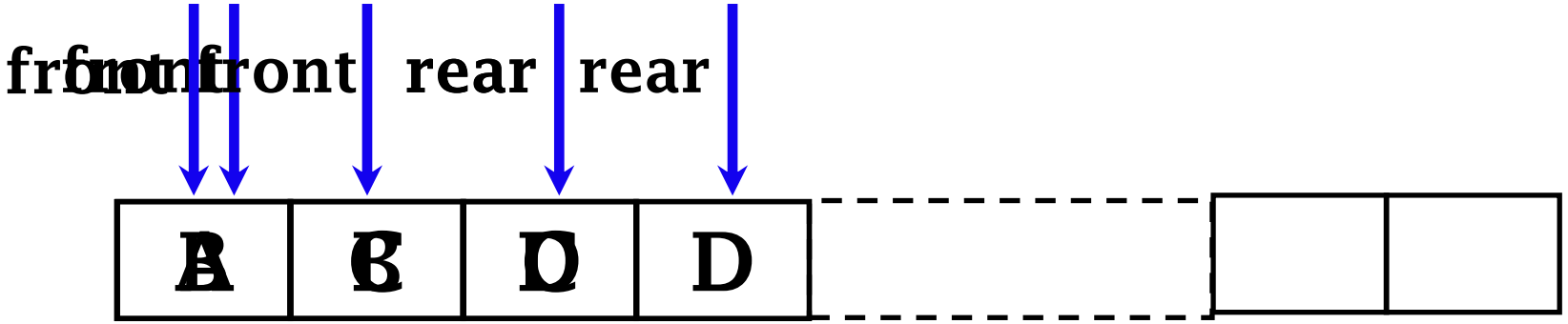


Class definition of sequential queues

```
class arrQueue: public Queue<T> {
private:
    int mSize;              // The size of array to store the queue
    int front;              // Subscript used to show the position of the front of the
queue
    int rear;               // Subscript used to show the position of the end of the
queue
    T * qu;                 // Array used to put queue elements of type T
public:                     // operation set of the queue
    arrQueue(int size);     // create an instance of the queue
    ~arrQueue();            // delete the instance and release space
}
```

The maintenance of sequential queue

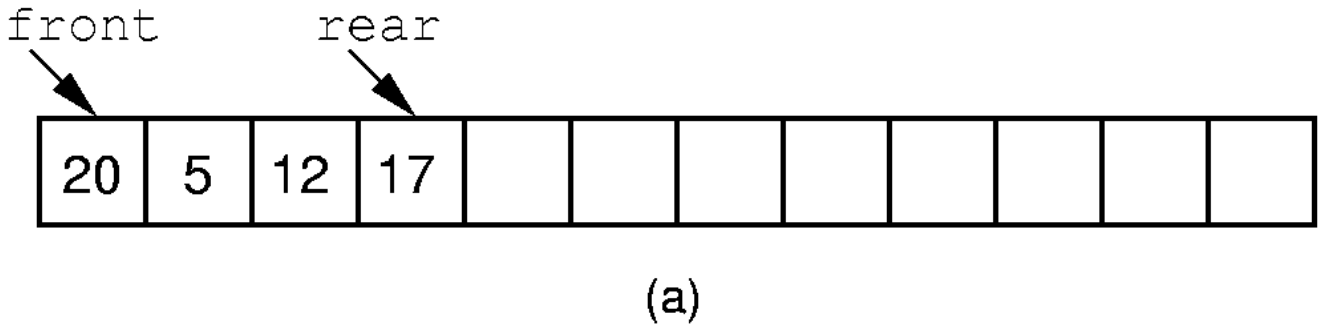
- Rear refers to



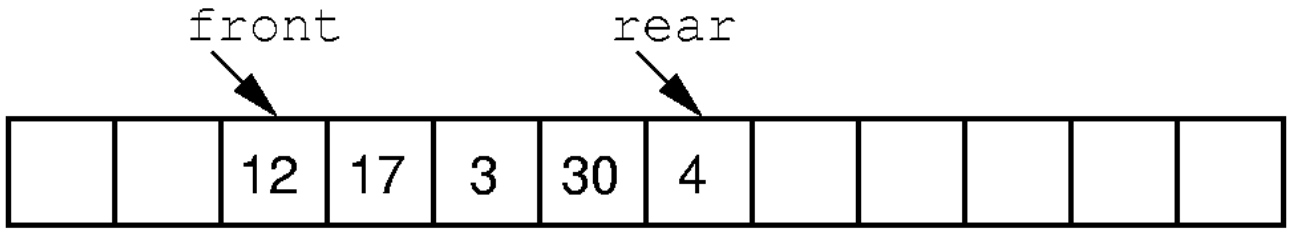
insert delete

The maintenance of sequential queue

- Front and rear are all true pointers



(a)



(b)

Implementation code of sequential queues

```
template <class Elem> class Aqueue : public Queue<Elem> {
private:
   int size;                  // The maximum capacity of queue
   int front;                 // The pointer of the front element of the queue
   int rear;                  // The pointer of the end element of the queue
   Elem *listArray;           // The array that store the elements
public:
   AQueue(int sz=DefaultListSize) {
      // Let the array for storage leave one more empty place
      size = sz+1;// size is the length of the array , and the max length of queue sz
      rear = 0; front = 1;     // you may assign rear=-1; front=0
      listArray = new Elem[size];
   }
   ~AQueue() { delete [] listArray; }
   void clear() { front = rear+1; }
```

Implementation code of sequential queues

```
bool enqueue(const Elem& it) {
    if (((rear+2) % size) == front) return false;
                        // There is only one empty place for the queue to be full
    rear = (rear+1) % size; // It needs to be moved to the next empty place first
    listArray[rear] = it;
    return true;
}
bool dequeue(Elem& it) {
    if (length() == 0) return false;
                              // the queue is empty
    it = listArray[front];      // move out of the queue first and then move the front subscript
    front = (front+1) % size;   // Increase in the formula of ring
    return true;
}
```

Implementation code of sequential queues

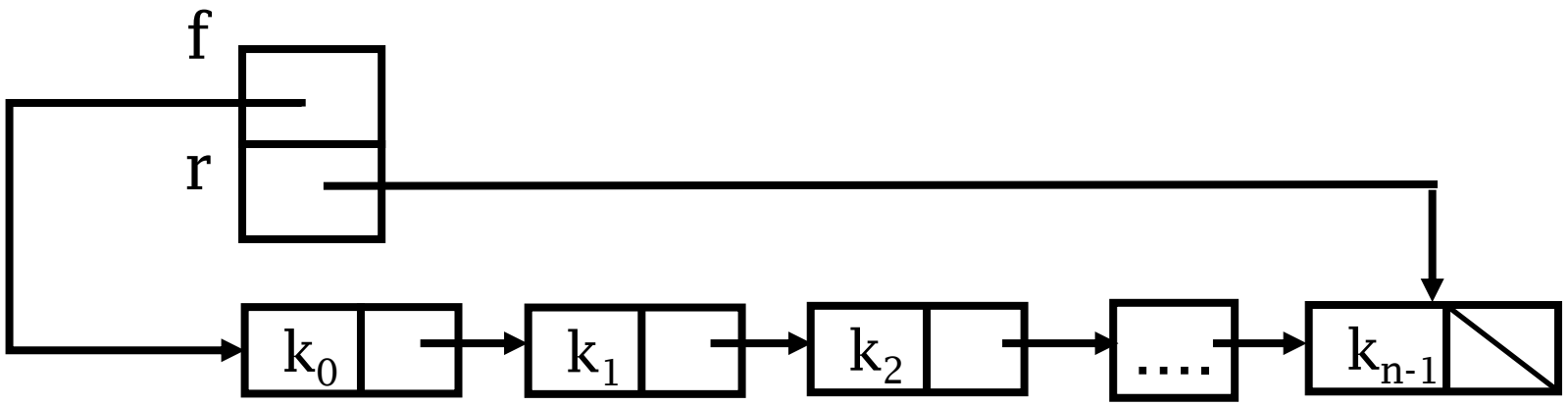
```
   bool frontValue(Elem& it) const {
    if (length() == 0)
                return false;      // the queue is empty
    it = listArray[front]; return true;
}
int length() const {
    return (size +(rear - front + 1)) % size;
}
```

Questions

- 1. You are given a queue with the length of n, you can just use the variable of front and rear, what is the largest number of elements that the queue can contain? Please give details of the derivation.
- 2. If you don't want to waste storage unit of the queue, what kind of other methods can you use ?

Representation of linked queues

- Singed linked list queue
- The direction of the linked pointer if from the front of the queue to the end of the queue



Class definition of linked queues

```
template <class T>
class lnkQueue: public Queue<T> {
private:
    int size;                       // the number of elements in the queue
    Link<T>* front;                 // the pointer of the front element of the queue
    Link<T>* rear;                  // the pointer of the end element of the queue
public:                             // operation set of the queue
    lnkQueue(int size);             // create an instance of the queue
    ~lnkQueue();                    // delete the instance and release space
}
```

Implementation code of linked queues

```
bool enQueue(const T item) {
//  insert the element to the end of the queue
   if (rear == NULL)       {      //if the queue is empty
          front = rear = new Link<T> (item, NULL);
   }
   else  {                  // add new elements
          rear-> next = new Link<T> (item, NULL);
          rear = rear ->next;
   }
   size++;
   return true;
}
```

Implementation code of linked queues

```
bool deQueue(T* item)  {
// return the front element of the queue and remove it
    Link<T> *tmp;
    if (size == 0)  {
// the queue is empty and no elements can be bring out of the
queue
        cout << "The queue is empty" << endl;
        return false;
    }
    *item = front->data;
    tmp = front;
    front = front -> next;
    delete tmp;
    if (front == NULL)
        rear = NULL;
    size--;
    return true;
}
```

Comparison between sequential queue and linked queue

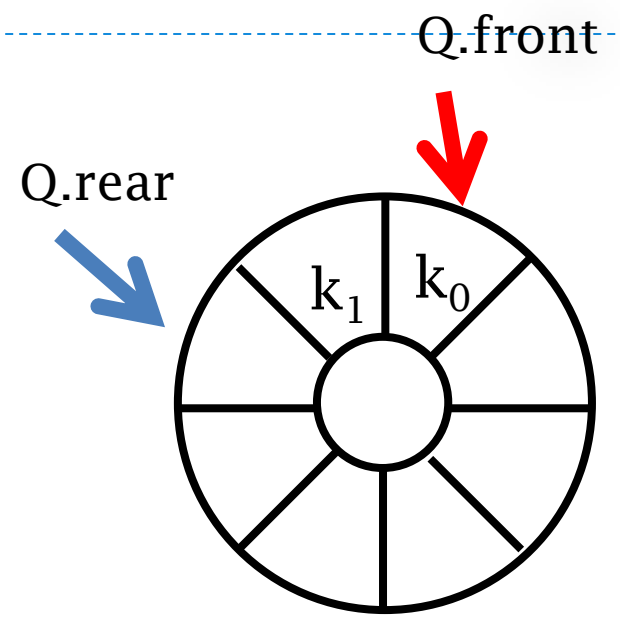
- **Sequential queue**
  - Fixed storage space
- **Linked queue**
  - Use in the cases when the maximum size cannot be estimated

Both of them are not allowed to access internal elements of the queue

Applications for queues

- All the applications that meet the characteristics of FIFO can use queue as the way of data organization or intermediate data structure
- Scheduling or buffering
  - Message buffer
  - Mail buffer
  - The communication between computer hardware equipment also need queue as a data buffer
  - Resource management of operating system
- BFS

Questions



- Linked list are usually implemented by using linked list, why not use doubly linked list?

- And, if we apply false-pointers to a tail of a sequential queue, what is the difference from the case of true-pointers we have introduced?

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)