CS213/293 Data Structure and Algorithms 2023

Lecture 4: Queue

Instructor: Ashutosh Gupta

IITB India

Compile date: 2023-08-10

Topic 4.1

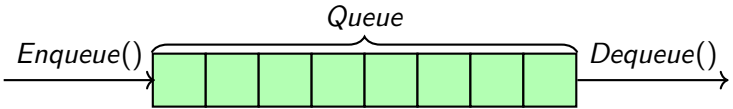
Queue

Queue

Definition 4.1

Queue is a container where elements are added and deleted according to the first-in-first-out (FIFO) order.

- Addition is called enqueue
- Deleting is called dequeue



Example 4.1

- *Entry into an airport*
- *Calling lift in a building (priority queue)*

Interface of queue

Reference: https://en.cppreference.com/w/cpp/container/queue

Queue supports four main interface methods

- `queue<T> q` : allocates new queue q
- `q.enqueue(e)` : Adds the given element e to the end of the queue. (push)
- `q.dequeue()` : Removes the first element from the queue. (pop)
- `q.front()` : access the first element .

Some support functions

- `q.empty()` : checks whether the queue is empty
- `q.size()` : returns the number of elements

Commentary: All literature uses the term enqueue and dequeue, but unfortunately C++ library uses push for enqueue and pop uses for dequeue. Other languages such as Java uses the term enqueue and dequeue.

Axioms of queue

1. `queue<T> q; q.enqueue(e); Assert(q.front() == e);`

2. `queue<T> q,q1; q.enqueue(e); q.dequeue(); Assert(q1 == q);`

3. ```
   q.enqueue(e1); Assume(q1 == q);
   q.enqueue(e2);
   Assert(q.front() == q1.front());
   ```

4. ```
   q.enqueue(e1); Assume(q1 == q);
   q.enqueue(e2);q.dequeue(); q1.dequeue();q1.enqueue(e2);
   Assert(q == q1);
   ```

Exercise 4.1

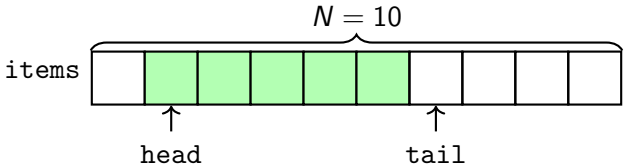
Why do the above four axioms define queue?

Topic 4.2

Array implementation of queue

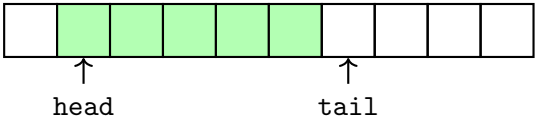
Array-based implementation

- Queue is stored in an array `items` in a circular fashion
- Three integers record the state of the queue
  1. `N` indicates the available capacity (N-1) of the queue
  2. `head` indicates the position of the front of the queue
  3. `tail` indicates position one after the rear of the queue

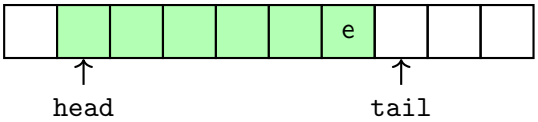


Enqueue operation on array

Consider the state of the queue

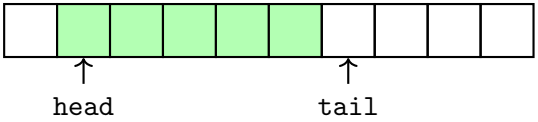


After enqueue(e) operation:

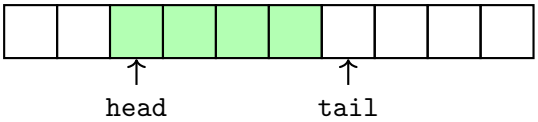


Dequeue operation on array

Consider the state of the queue



After dequeue() operation:

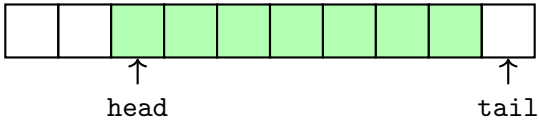


Exercise 4.2

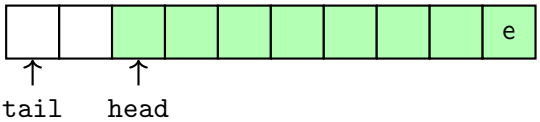
1. *Where will front() read from?*
2. *What is the size of the queue?*

Wrap around to utilize most of the array

Consider the state of the queue

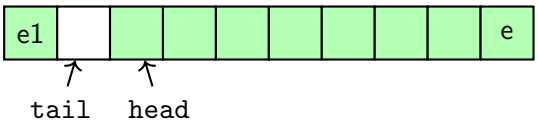


After enqueue(e) operation, we move the tail to 0.



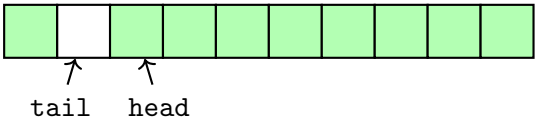
Wrap-around allows us to use the array repeatedly.

After another `enqueue(e1)` operation:

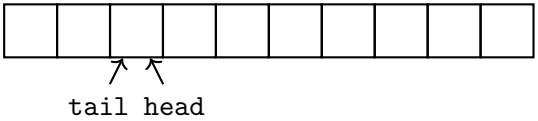


Full and empty queue

Full queue:



Empty queue:



Exercise 4.3

Can we use all N cells for storing elements?

Array implementation

The code is not written in exact C++; We will slowly move towards pseudo code to avoid clutter on slides.

```
int head = 0, tail=0, N = INITIAL_CAPCITY;

Object items[N];                    //Some initial size

bool empty()   { return (head == tail);   }

bool size()    { return (N+tail-head)%N; }

Object front() { return items[head];      }
```

Array implementation

```
void dequeue() {
  if( empty() ) throw Empty;              // Queue is empty
  free(items[head]);items[head] = NULL; // Clear memory
  head = (head+1)%N;      //Remove an element
}
```

```
void enqueue( Object x ) {
  if ( size() == N-1 ) expand(); // Queue is full;expand
  items[tail] = x;
  tail = (tail+1)%N; // insert element
}
```

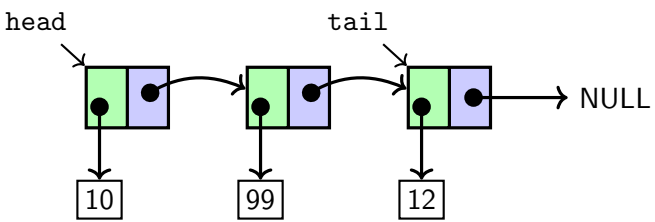
Topic 4.3

Queue via linked list

Linked lists

Definition 4.2

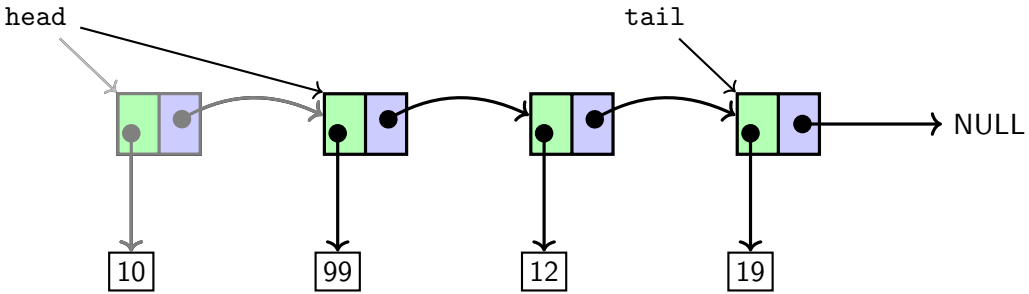
A linked list consists of nodes with two fields data and next pointer. The nodes form a chain via the next pointer. The data pointers point to the objects that are stored on the linked list.



Exercise 4.4

If we use a linked list for implementing a queue, which side should be the front of the queue?

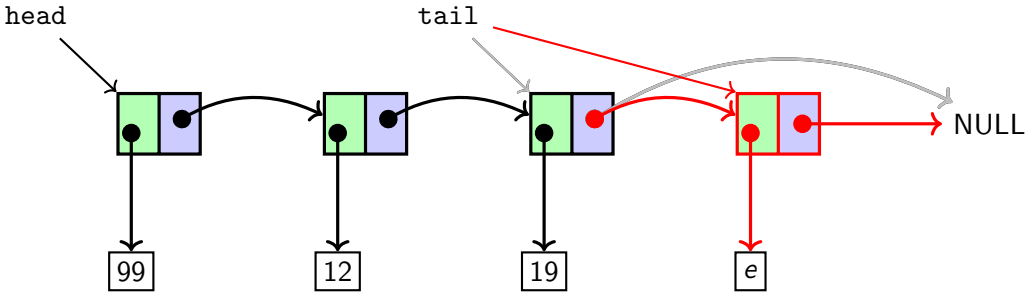
Dequeue in linked lists



Exercise 4.5

What happens to the object containing 10?

Enqueue(e) in linked lists



Exercise 4.6

a. *Which one is better: array or linked list?*

b. *Do we need the tail pointer?*

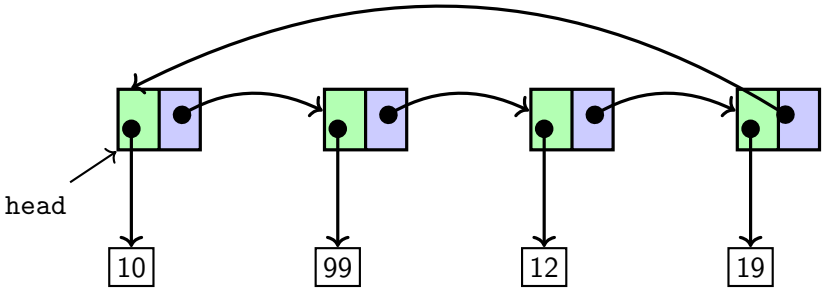
Topic 4.4

Circular linked list

Circular linked lists

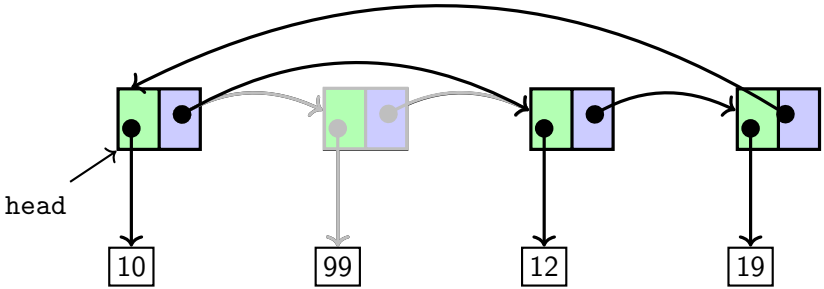
Definition 4.3

In a circular linked list, the nodes form a circular chain via the next pointer.

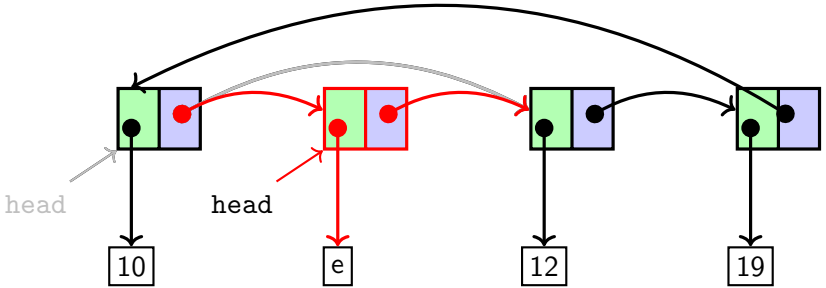


A head pointer points at some node of the circular list. A single pointer can do the job of head and tail.

Dequeue in circular linked lists



enqueue(e) in circular linked lists



Exercise 4.7

Which element should be returned by front()?

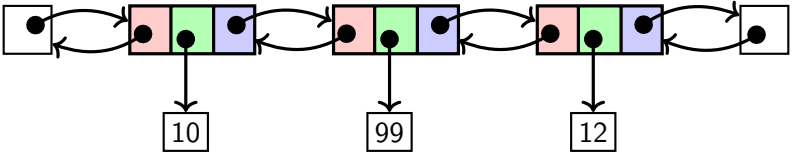
Topic 4.5

Dqueue via a doubly linked list

Doubly linked lists

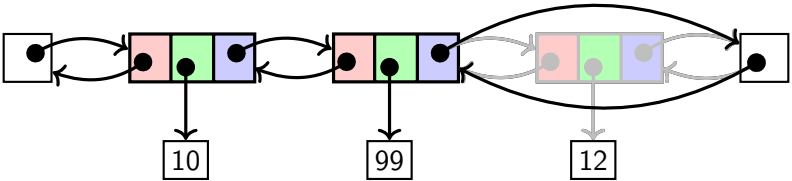
Definition 4.4

A doubly linked list consists of nodes with three fields prev, data, and next pointer. The nodes form a bidirectional chain via the prev and next pointer. The data pointers point to the objects that are stored on the linked list.



At both ends, two dummy or sentinel nodes do not store any data and are used to store the start and end points of the list.

Deleting a node in a doubly linked list



Deque (Double-ended queue)

Definition 4.5

Deque is a container where elements are added and deleted according to both last-in-first-out (LIFO) and first-in-first-out (FIFO) order.

Interface of Deque

Reference: https://en.cppreference.com/w/cpp/container/deque

Queue supports four main interface methods

- `deque<T> q` : allocates new queue q
- `q.push_back(e)` : Adds the given element e to the back.
- `q.push_front(e)` : Adds the given element e to the front.
- `q.pop_front()` : Removes the first element from the queue.
- `q.pop_back()` : Removes the last element from the queue.
- `q.front()` : access the first element .
- `q.back()` : access the first element .

Some support functions

- `q.empty()` : checks whether the stack is empty
- `q.size()` : returns the number of elements

We can implement the Deque data structure using the doubly linked lists.

Stack and queue via Deque

We can implement both stack and queue using the interface of deque.

Exercise 4.8

- *Which functions of deque implement stack?*
- *Which functions of deque implement queue?*

All modification operations are implemented in $O(1)$.

Exercise 4.9

Can we implement `size` in $O(1)$ in a doubly linked list?

Topic 4.6

Problems

Problem: reversing a linked list

Exercise 4.10

Give an algorithm to reverse a linked list. You must use only three extra pointers.

Commentary: **Solution:** Here is a code for reversing the linked list:

```
Node* current=head;
Node* next;
Node* prev=NULL;
while(current!=NULL){
    next=current->nptr;
    current->nptr=prev;
    prev=current;
    current=next;
    }
hptr=prev;
```

Can we do it with only two extra pointers? Please use `https://www.godbolt.org` to check the number of registers involved in your solution.

Problem: middle element

Exercise 4.11

Give an algorithm to find the middle element of a singly linked list.

Commentary: **Solution:** We may use two pointers `fast` and `slow` to traverse a linked list. In each iteration, fast moves two steps, and slow moves one step. When fast reaches Null, slow must be pointing to the middle. Will we have correct answer in both the odd and even cases?

Problem: messy queue

Exercise 4.12

The mess table queue problem: There is a common mess for k hostels. Each hostel has some N_1,...,N_k students. These students line up to pick up their trays in the common mess. However, the queue is implemented as follows: If a student sees a person from his/her hostel, she/he joins the queue behind this person. This is the "enqueue" operation. The "dequeue" operation is as usual, at the front. Think about how you would implement such a queue. What would be the time complexity of enqueue and dequeue? Do you think the average waiting time in this queue would be higher or lower than a normal queue? Would there be any difference in any statistic? If so, what?

Commentary: **Solution:** We may use queues $q_1, \ldots, q_k$ for people for each hostel. The hostels are put on another queue q depending on the arrival of their first student. The students of hostel i are served where their hostel is in the front of q and are served in order of q_i. If q_i becomes empty, then we dequeue the hostel i from q.

End of Lecture 4