Queue[1]

R. Inkulu
http://www.iitg.ac.in/rinkulu/

[1] C code is developed on the board (and the same is not available in slides)

Introduction

- Motivation for First-In-Fist-Out (FIFO) data structure

  job scheduling

  message buffers

- Essential operations to be supported include:

  enqueue an element to queue

  dequeue an element from queue

Fixed-size queue

```
#define QUEUESIZE 1000
typedef struct {
   void *p[QUEUESIZE];
   int head;  //p[head] is the element to be dequeued
   int tail;  //enqueued element is placed at p[tail]
} Queue;

int initialize(Queue *q);
int enqueue(Queue *q, void *data);
void *dequeue(Queue *q);
```

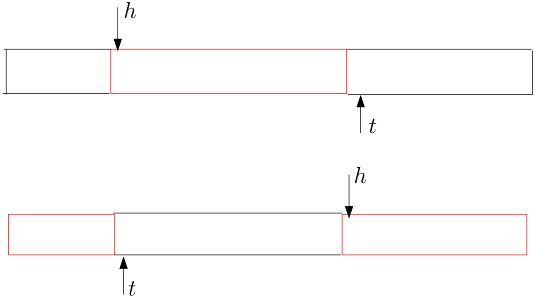
Fixed-size queue

```
#define QUEUESIZE 1000
typedef struct {
   void *p[QUEUESIZE];
   int head;  //p[head] is the element to be dequeued
   int tail;  //enqueued element is placed at p[tail]
} Queue;

int initialize(Queue *q);
int enqueue(Queue *q, void *data);
void *dequeue(Queue *q);
```

- drawback in viewing $p[]$ as a linear array: overflow may occur while many entries in $p[]$ are not utilized

Fixed-size circular queue[2]

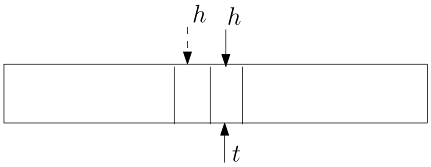


head can precede tail, and vice versa

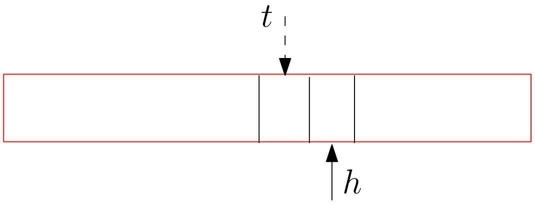
- array p in the queue data structure (mentioned above) is viewed as a circular
- move *head* and *tail* over $p[]$ using modular arithmatic

[2]considered to be the default queue: the word *circular* may not always be mentioned

Fixed-size circular queue (cont)



indicates queue is empty (reset tail to -1 and head to 0)



indicates queue is full

Worst-case time complexity

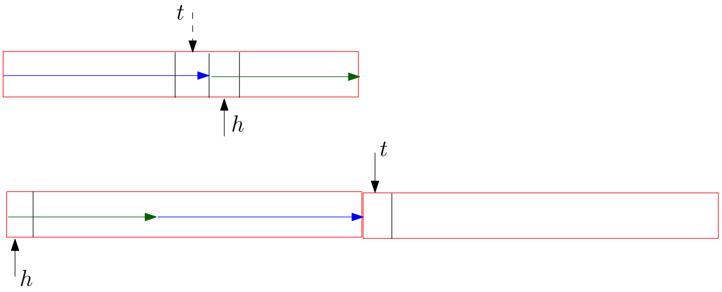
- enqueue: $O(1)$ time and $O(1)$ space
- dequeue: $O(1)$ time and $O(1)$ space

Dynamic-sized circular queue

```
typedef struct {
   int capacity;
   int head;  //p[head] is the element to be dequeued
   int tail;  //enqueued element is placed at p[tail]
   void **p;  //p is a pointer to an array of void *s
} Queue;

int initialize(Queue *q,
               int initialCapacity);
int destroy(Queue *q);
int enqueue(Queue *q, void *data);
void *dequeue(Queue *q);
```

Dynamic-sized circular queue



resize when the queue is full

- when the overflow occurs, double the size of buffer pointed by p
- when the queue is one-quarter full, halve the size of buffer pointed by $p \leftarrow$ homework
- in both of these cases, avoid using realloc (which implicitly frees old buffer) instead use malloc and explicitly free the old buffer

Deque

data structure having provision for adding/removing at either of its ends is known as a *doubly-ended queue* (a.k.a. *deque*[3])

homework: implement the dynamic-sized deque

[3]'deque' pronounced as 'deck'

Stack using Queue, and vice versa

Homework:

- Implement Stack using Queue and analyze the asymptotic time and space complexities of the resultant push and pop
- Implement Queue using Stack and analyze the asymptotic time and space complexities of the resultant enqueue and dequeue