The ADT Queue and its implementations.

Monday, October 21, 2019 4:51 PM

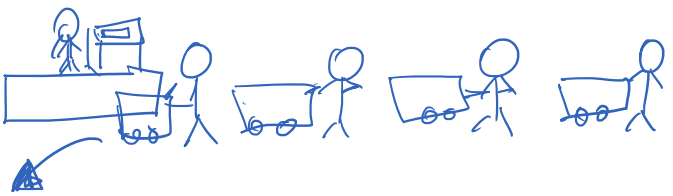
A queue is a sequence of elements of the same type

$Q_1 = \langle a, r, p, q, s \rangle$ (front: a, back: s)

Operations

- front (Q_1) = 'a'
- enqueue (Q_1, x) = $\langle a, r, p, q, s, x \rangle$ adds elements to the back
- dequeue (Q_1) = $\langle r, p, q, s \rangle$ removes elements at the front.

Intuition:

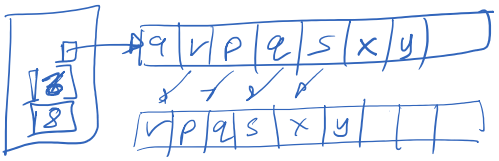


The Data Structure ArrayQueue

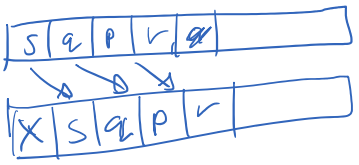
$Q_1 = \langle a, r, p, q, s \rangle$

enqueue = insert_back

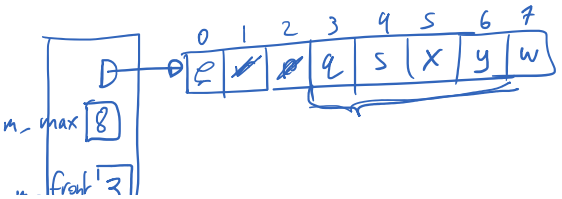
dequeue = remove_front

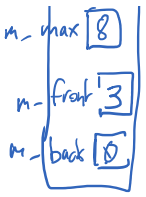


Idea #1 enq = insert_front, deq = remove_back



Idea #2





enqueue(x)
enqueue(y)

dequeue()
dequeue()
dequeue()

front()
return m_data[m_front]

enqueue(w)
enqueue(e)

"circular array"

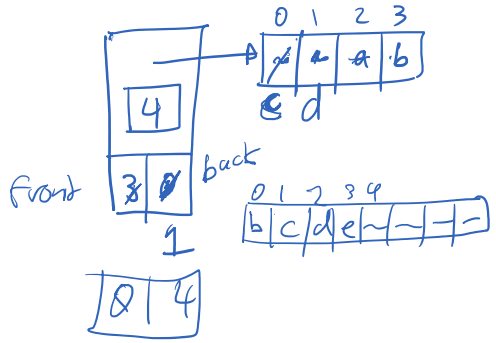
% mod operator.

enqueue(x)

m_data[m_back] = x;
m_back = (m_back + 1) % m_max;
if m_back is catching up to m_front:
grow()

dequeue()

m_front = (m_front + 1) % m_max

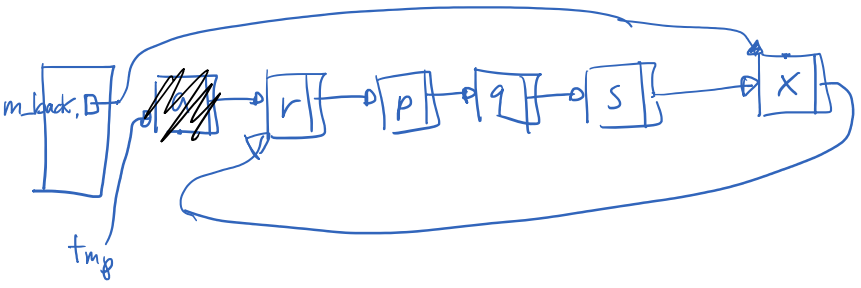


enqueue(b)
dequeue()
enqueue(c)
enqueue(d)
enqueue(e)

The Data Structure LinkedQueue

$Q_1 = \langle q, r, p, q, s \rangle$

circular Linked List.



enqueue(x)

enqueue(x)

tmp

```
LLNode* tmp = LLNode(x, m_back->m_next)
m_back->m_next = tmp
m_back = tmp;
```

dequeue()

```
LLNode* tmp = m_back->m_next
m_back->m_next = m_back->m_next->m_next
delete tmp.
```