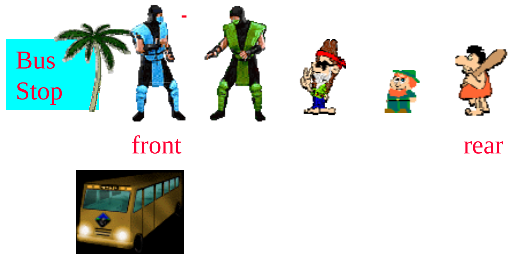
Queues

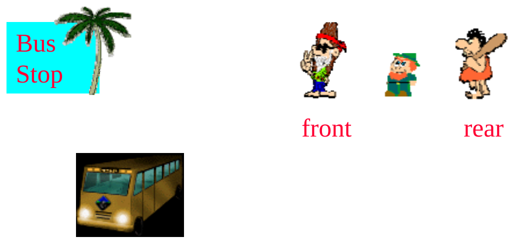
- Linear list.
- One end is called front.
- Other end is called rear.
- Additions are done at the rear only.
- Removals are made from the front only.
- FIFO (First In First Out)

Bus Stop Queue



Bus Stop Queue

Bus Stop Queue

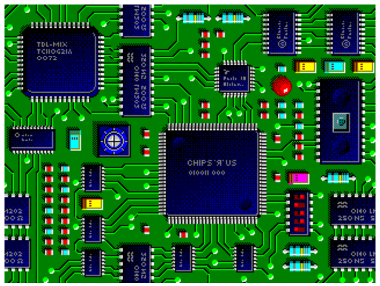


Bus Stop Queue

Revisit Of Stack Applications

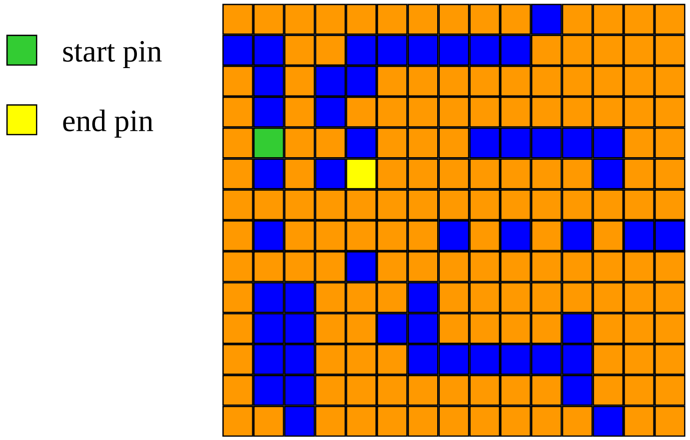
- Applications in which the stack cannot be replaced with a queue.
  - Parentheses matching.
  - Towers of Hanoi.
  - Method invocation and return.
- Application in which the stack may be replaced with a queue.
  - Rat in a maze.
    - Results in finding shortest path to exit.

Wire Routing



Represent as a grid in which components and already placed wires are denoted by blocked grid positions. (Can be used to solve the rat in the maze.)

Lee’s Wire Router

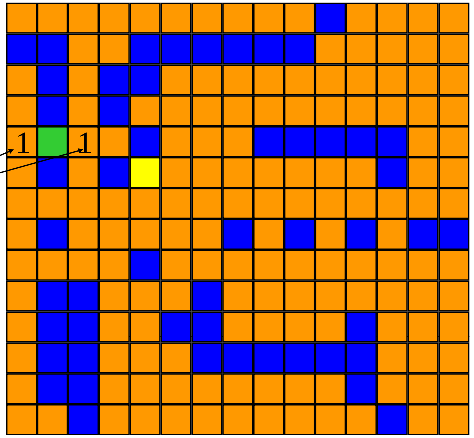


Label all reachable squares 1 unit from start.

Lee’s Wire Router

start pin

end pin



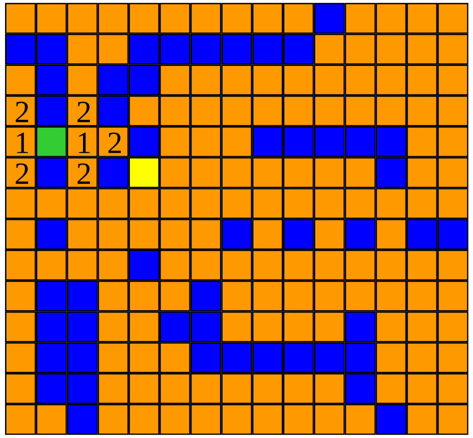
Store the position in the queue.

Label all reachable unlabeled squares 2 units from start.

Lee’s Wire Router

start pin

end pin

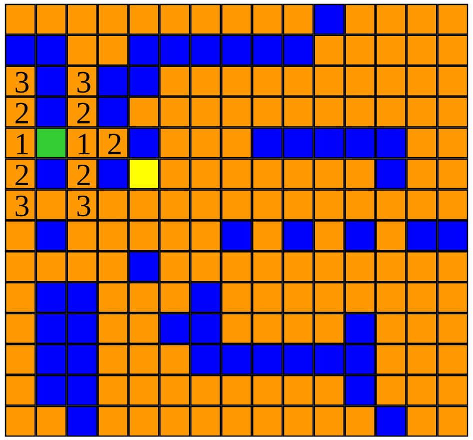


Label all reachable unlabeled squares 3 units from start.

Lee’s Wire Router

start pin

end pin

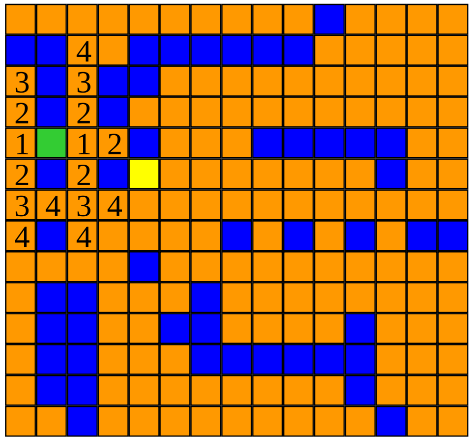


Label all reachable unlabeled squares 4 units from start.

Lee’s Wire Router

start pin

end pin

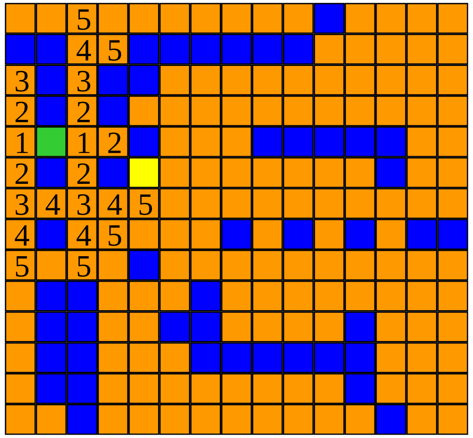


Label all reachable unlabeled squares 5 units from start.

Lee’s Wire Router

start pin

end pin

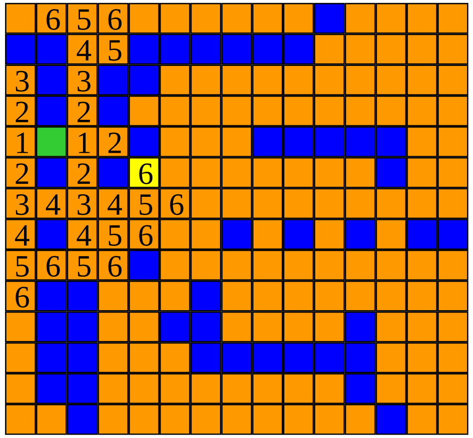


Label all reachable unlabeled squares 6 units from start.

Lee’s Wire Router

start pin

end pin

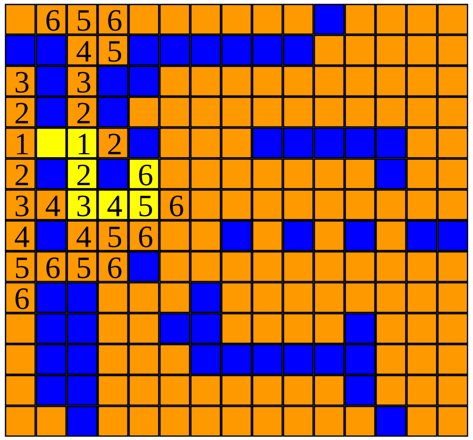


End pin reached. Traceback.

Lee’s Wire Router

start pin

end pin



End pin reached. Traceback.

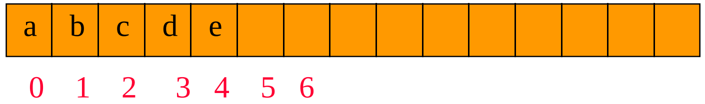
Queue Operations

- IsEmpty … return true iff queue is empty
- Front … return front element of queue
- Rear … return rear element of queue
- Push … add an element at the rear of the queue
- Pop … delete the front element of the queue

Queue in an Array

- Use a 1D array to represent a queue.
- Suppose queue elements are stored with the front element in queue[0], the next in queue[1], and so on.

Derive From arrayList



- Pop() => delete queue[0], shift other elements one step left
  - O(queue size) time
- Push(x) => if there is capacity, add at right end
  - O(1) time

O(1) Pop and Push

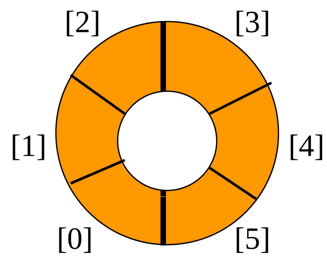
- to perform each opertion in O(1) time (excluding array doubling), we use a circular representation.

Custom Array Queue

- Use a 1D array queue.

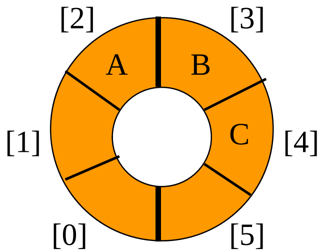
queue[]

- Circular view of array.



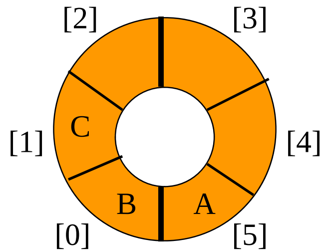
Custom Array Queue

- Possible configuration with 3 elements.



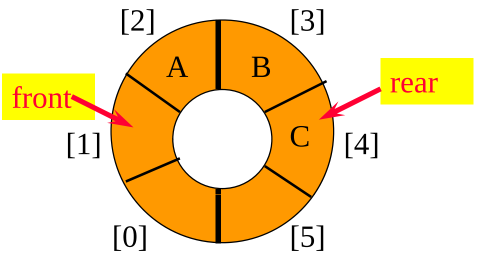
Custom Array Queue

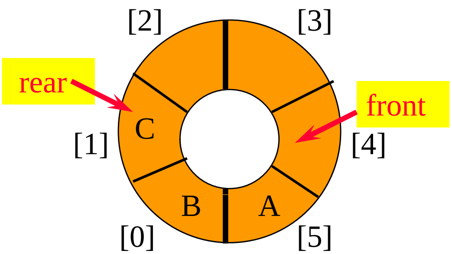
- Another possible configuration with 3 elements.



Custom Array Queue

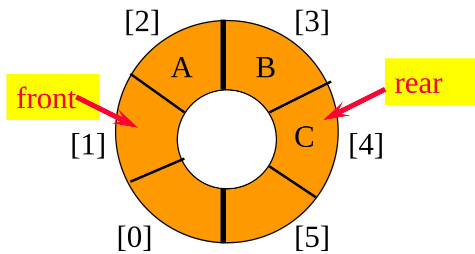
- Use integer variables front and rear.
  - front is one position counterclockwise from first element
  - rear gives position of last element





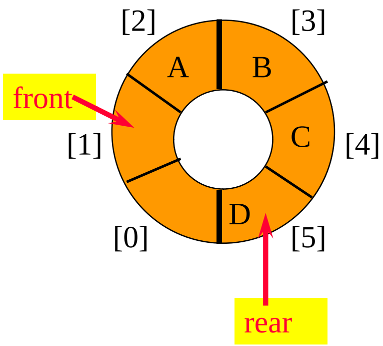
Push An Element

- Move rear one clockwise.



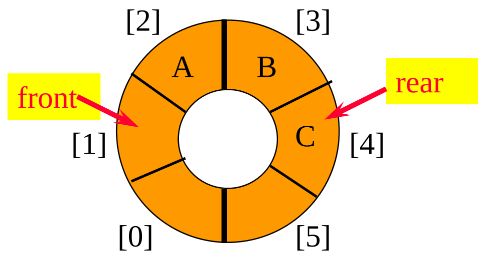
Push An Element

- Move rear one clockwise.
- Then put into queue[rear].



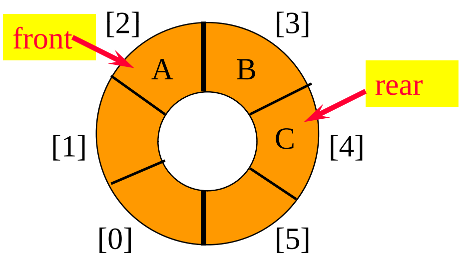
Pop An Element

- Move front one clockwise.



Pop An Element

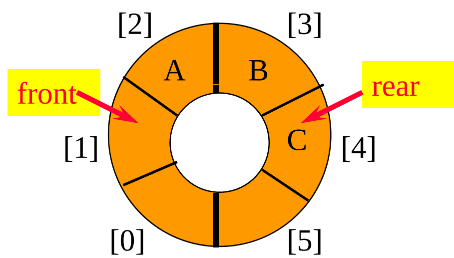
- Move front one clockwise.
- Then extract from queue[front].



Moving rear Clockwise

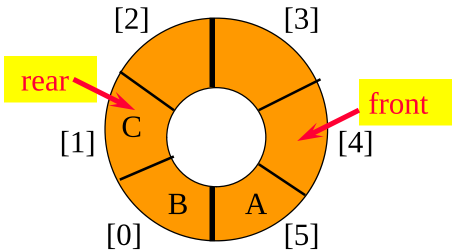
- rear++;

  if (rear = = capacity) rear = 0;

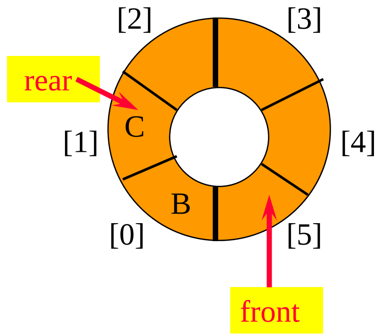


- rear = (rear + 1) % capacity;

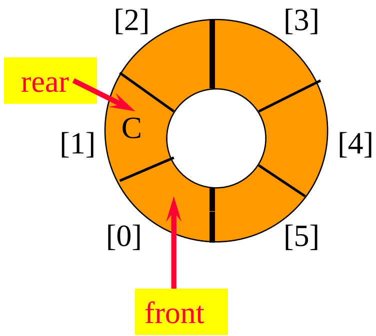
Empty That Queue



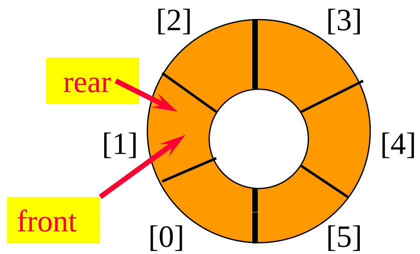
Empty That Queue



Empty That Queue

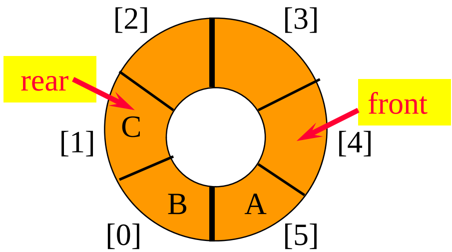


Empty That Queue

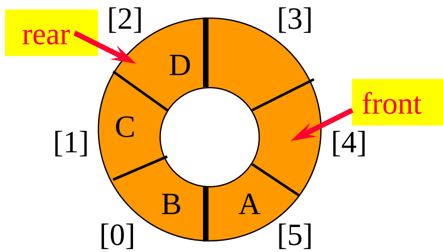


- When a series of removes causes the queue to become empty, front = rear.
- When a queue is constructed, it is empty.
- So initialize front = rear = 0.

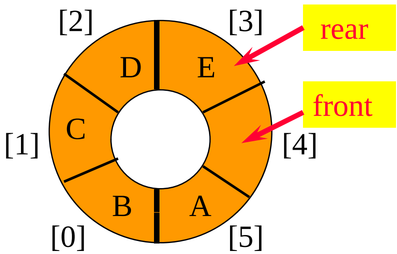
A Full Tank Please



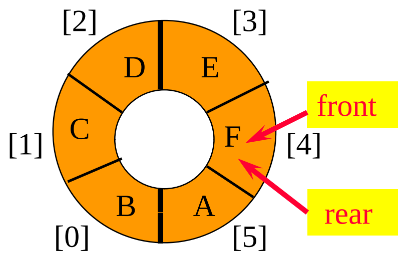
A Full Tank Please



A Full Tank Please



A Full Tank Please



- When a series of adds causes the queue to become full, front = rear.
- So we cannot distinguish between a full queue and an empty queue!

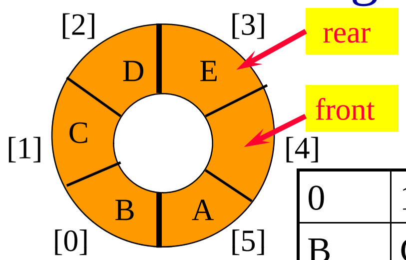
Ouch!!!!!

- Remedies.
  - Don’t let the queue get full.
    - When the addition of an element will cause the queue to be full, increase array size.
    - This is what the text does.
  - Define a boolean variable lastOperationIsPush.
    - Following each push set this variable to true.
    - Following each pop set to false.
    - Queue is empty iff (front == rear) && !lastOperationIsPush
    - Queue is full iff (front == rear) && lastOperationIsPush

Ouch!!!!!

- Remedies (continued).
  - Define an integer variable size.
    - Following each push do size++.
    - Following each pop do size--.
    - Queue is empty iff (size == 0)
    - Queue is full iff (size == arrayLength)
  - Performance is slightly better when first strategy is used.

Doubling Queue Capacity



Before enlarge array

0	1	2	3	4	5
B	C	D	E		A

After enlarge array

0	1	2	3	4	5	6	7	8	9	10	11
B	C	D	E		A						

Shift

0	1	2	3	4	5	6	7	8	9	10	11
B	C	D	E								A

Homework

- Sec. 3.5 Exercise 1 (a) P157
  - Trace the program. (Find a path through the maze with Lee’s Wire Router algorithm introduced in this section)