Lecture 3: Queue Applications

CS2504/CS4092– Algorithms and Linear Data Structures

Dr. Kieran T. Herley

Department of Computer Science
University College Cork

2013/14

Summary

ADT Queue. Definition and Java packaging of same. Josephus problem. Simulation using ADT Queue.

ADT Queue

ADT Queue Another container abstraction

Analogy Simple single-file bank queue

- Customers enter queue at the rear
- Customers leave queue at front (having completed their transactions)

Observation

Queue respects First-In First-out (FIFO) discipline

ADT Queue

A queue is an abstract data type on which the following operations are defined:

enqueue(o): Insert item o at rear of queue. *Input:* *EltType*; *Output:* *None*.

dequeue(): Remove and return item at front of queue; illegal if queue is empty. *Input:* *None*; *Output:* *EltType*.

ADT Queue

A queue is an abstract data type on which the following operations are defined:

enqueue(o): Insert item o at rear of queue. *Input:* *EltType*; *Output:* *None*.

dequeue(): Remove and return item at front of queue; illegal if queue is empty. *Input:* *None*; *Output:* *EltType*.

size(): Return number of items in the queue. *Input:* *None*; *Output:* *int*.

isEmpty(): Return boolean indicating if queue is empty. *Input:* *None*; *Output:* *boolean*.

front(): Return, but do not remove, front queue item; illegal if queue is empty. *Input:* *None*; *Output:* *EltType*.

Illustration

Operation	Queue	Output
enqueue(A)	A	
enqueue(B)	A B	
front()	A B	A
enqueue(C)	A B C	
size()	A B C	3
dequeue()	B C	A
enqueue(D)	B C D	
dequeue()	C D	B
dequeue()	D	C
isEmpty()	D	false
dequeue()		D
isEmpty()		true
dequeue()		ERROR

Queue Applications

- Queue-like concepts used in OS implementations
  - print jobs waiting to be completed
  - processes waiting to be executed
- Simulations *e.g.* traffic simulation
  - model of city– road network
  - model of traffic patterns
  - model intersections as queues of vehicles waiting in each direction

Queues and Java

```
public class QueueExercise
{  public static void main (String [] args)
   {  QueueOfInt q = new QueueOfInt();int num;
      for (num = 1; num <= 6; num++)
      {  System.out. println ("Enqueuing " +num);
         q.enqueue(num);
      }
      while (!q.isEmpty())
      {  num = q.dequeue();
         System.out. println ("Dequeuing " +num);
      }
   }
}
```

The Josephus Problem

The Game

- N people are arranged in a circle with the person at the twelve-o-clock position in the circle holding a ball
- The token is passed clockwise around the circle from person to person; after M passes, the person holding the ball is eliminated
- Following the elimination, the circle closes ranks with the person after the eliminee picking up the ball and the game continues by repeating the above step
- The game continues in this fashion until only one person remains, who is deemed the winner

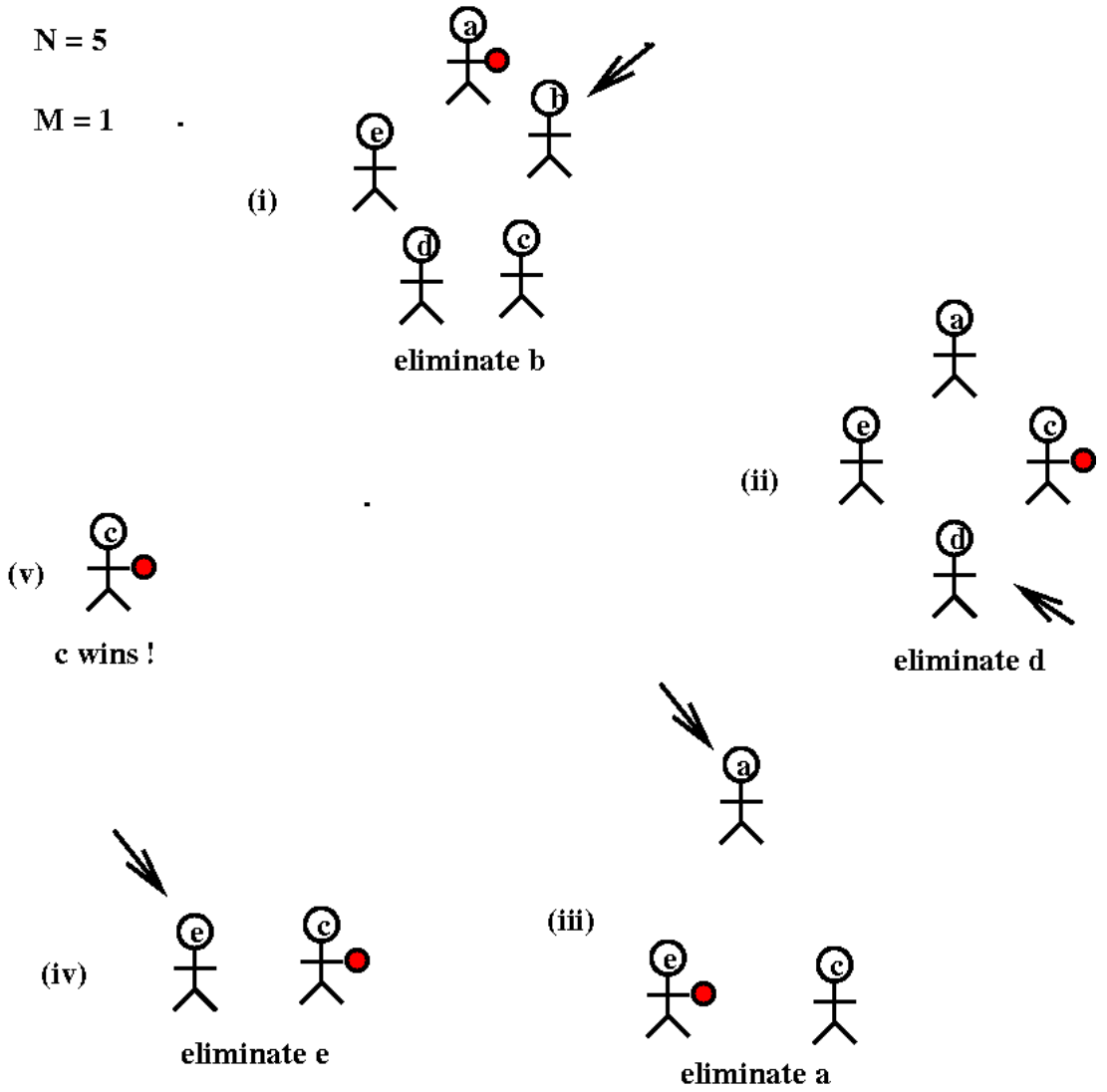
The Josephus Problem

The Game

- N people are arranged in a circle with the person at the twelve-o-clock position in the circle holding a ball
- The token is passed clockwise around the circle from person to person; after M passes, the person holding the ball is eliminated
- Following the elimination, the circle closes ranks with the person after the eliminee picking up the ball and the game continues by repeating the above step
- The game continues in this fashion until only one person remains, who is deemed the winner

Josephus Problem Given N and M, determine the winner

Illustration



How to Solve This?

Mathematical Approach

- Analyze problem based on number-theoretic characteristics of parameters n and m

Computer Science Approach

- Develop program to *simulate* the game
- Run programe (for specific n and m values) to determine winner

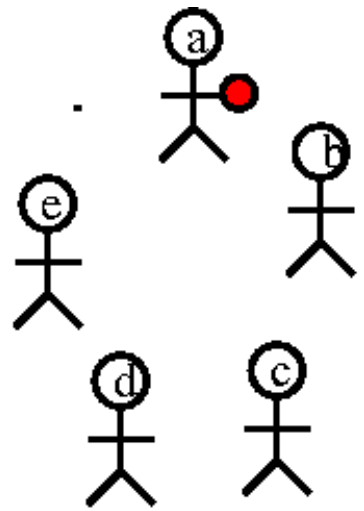
Sneak Preview

```
Queue<Integer> circle =
    new ArrayBasedQueue<Integer>();
for (int i = 1; i <= circleSize; i++)
{   circle.enqueue(i);
}
while ( circle.size() > 1)
{   for (int k = 0; k < stepSize; k++)
    {   circle.enqueue( circle.dequeue());
    }
    circle.dequeue();
}
int survivor = circle.front();
```

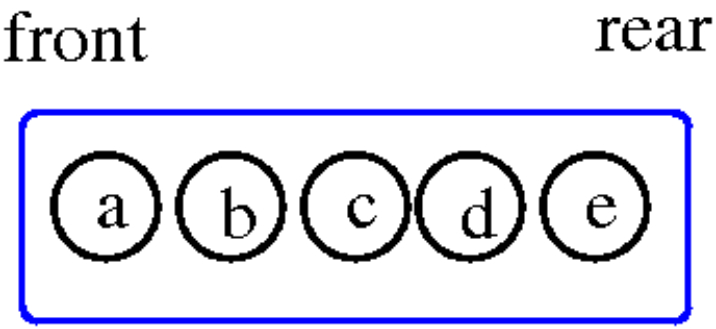
How to Model Circle?

Model the circle as a queue:

- the individuals appear in clockwise order in queue
- the front of the queue represents the person holding the ball

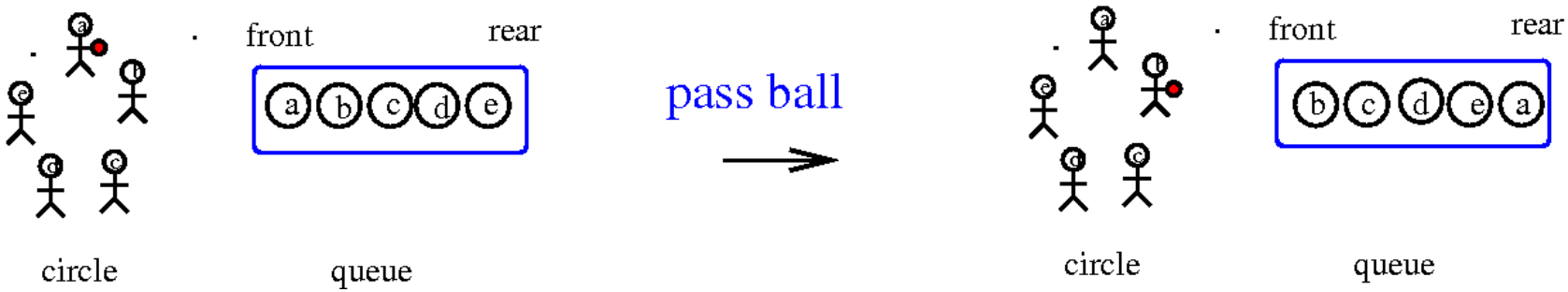


circle

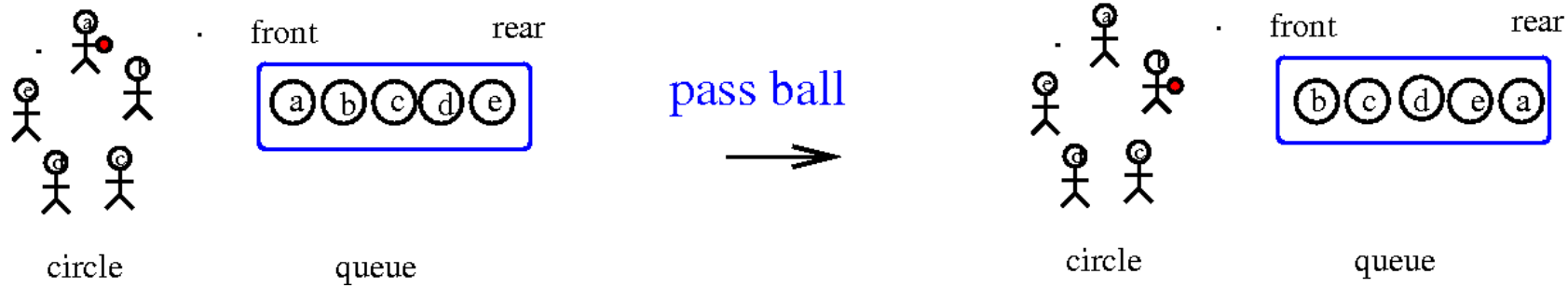


queue

How to Simulate Game cont'd



How to Simulate Game cont'd

- 

- To "pass the buck(ball)", move front queue item to the rear

Simulation cont'd

Simulation cont'd

- 

- To remove the person holding the ball, remove the front queue item

Pseudocode

```
Algorithm Josephus(n, m):
  for i ← 1 to n do
      circle.enqueue(i)
  while circle.size() > 1 do
      for j ← 1 to m do
          circle.enqueue(circle.dequeue())
      circle.dequeue()
  return circle.front()
```

- Algorithm expressed in semiformal *pseudocode* rather than Java, capturing algorithm's essentials
- Omits extraneous nitty gritty programming detail
  - no semicolons, declarations
  - trivial notational diffs e.g. $\leftarrow$ instead of $=$
- No { }; statement nesting reflected by indentation only

Issues

- Setting up the circle (numbered players)

```
for i ←1 to n do
    circle .enqueue(i)
```

Issues

- Setting up the circle (numbered players)

```
for i ←1 to n do
    circle .enqueue(i)
```

- Passing the ball clockwise once

```
circle .enqueue( circle .dequeue())
```

Issues

- Setting up the circle (numbered players)

```
for i ←1 to n do
    circle .enqueue(i)
```

- Passing the ball clockwise once

```
circle .enqueue( circle .dequeue())
```

Iterate m times for complete round

- Removing the victim (ball holder)

```
circle .dequeue()
```

Issues

- Setting up the circle (numbered players)

```
for i ←1 to n do
    circle .enqueue(i)
```

- Passing the ball clockwise once

```
circle .enqueue( circle .dequeue())
```

Iterate m times for complete round

- Removing the victim (ball holder)

```
circle .dequeue()
```

- Determining the last(only) man standing

```
circle . front ()
```

Complete Algorithm

```
Algorithm Josephus(n, m):
    for i ← 1 to n do
        circle.enqueue(i)
    while circle.size() > 1 do
        for j ← 1 to m do
            circle.enqueue(circle.dequeue())
        circle.dequeue()
    return circle.front()
```

Exercise: modify so players referred to by name rather than number.

Translating into Java

```
Algorithm Josephus(n, m):
  for i ←1 to n do
      circle .enqueue(i)
  while circle . size () > 1 do
      for j ←1 to m do
          circle .enqueue( circle .dequeue())
      circle .dequeue()
  return circle . front ()
```

Translating into Java

```
Algorithm Josephus(n, m):
  for i ←1 to n do
      circle .enqueue(i)
  while circle . size () > 1 do
      for j ←1 to m do
          circle .enqueue( circle .dequeue())
      circle .dequeue()
  return circle . front ()
```

```
Queue<Integer> circle =
    new ArrayBasedQueue<Integer>();
for (int i = 1; i <= circleSize; i++)
{   circle .enqueue(i);
}
while ( circle . size () > 1)
{   for (int k = 0; k < stepSize; k++)
    {   circle .enqueue( circle .dequeue());
    }
    circle .dequeue();
}
int survivor = circle . front ();
```

For now think of Queue<Integer> as being equiv. to QueueOfInt.

Acknowledgements

The bank-queue image is take from the `http://www.sxc.hu` online image repository its use here is consistent with the terms and conditions of that site.