2110211 โครงสร้างข้อมูลเบื้องต้น

Queue

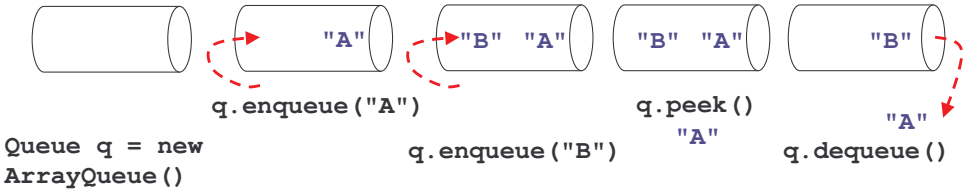
สมชาย ประสิทธิ์จูตระกูล

แถวคอย : queue

```
public interface Queue {
  public boolean isEmpty();
  public int size();
  public void enqueue(Object e);
  public Object peek();
  public Object dequeue();
}
```

FIFO

First-In First-Out



Queue คล้าย List

- Queue คือ list ที่เราเพิ่มปลายด้านหนึ่ง และลบที่ปลายอีกด้าน
- สร้าง queue ด้วย list แบบง่าย ๆ

```
public class ArrayListQueue implements Queue {
  private ArrayList list = new ArrayList();

  public boolean isEmpty() {return list.isEmpty();}
  public int size() {return list.size();}
  public void enqueue(Object e) {list.add(list.size(),e);}
  public Object peek() {
    if (isEmpty()) throw new NoSuchElementException ();
    return list.get(0);
  }
  public Object dequeue() {
    Object e = peek();
    list.remove(0);
    return e;
  }
}
```

Composition + Delegation

enqueue ใช้เวลา O(1) แต่ dequeue ใช้เวลา O(n)

สร้าง Queue ด้วย LinkedList

- เลือกใช้ Circular doubly linked list w/ header
- addLast และ removeFirst ใช้เวลา O(1)

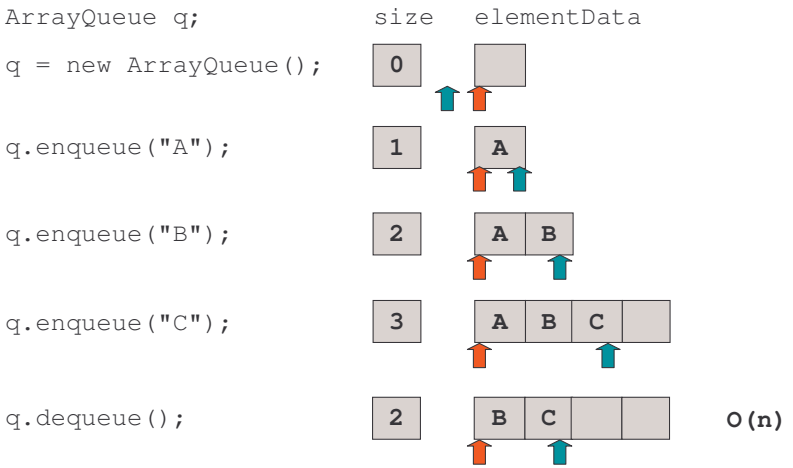
```
public class ArrayListQueue implements Queue {
  private LinkedList list = new LinkedList();

  public boolean isEmpty() {return list.isEmpty();}
  public int size() {return list.size();}
  public void enqueue(Object e) {list.addLast(e);}
  public Object peek() {
    if (isEmpty()) throw new NoSuchElementException ();
    return list.get(0);
  }
  public Object dequeue() {
    Object e = peek();
    list.removeFirst();
    return e;
  }
}
```

ทุก ๆ operations ใช้เวลา O(1) แต่เปลืองเพราะใช้ linked list

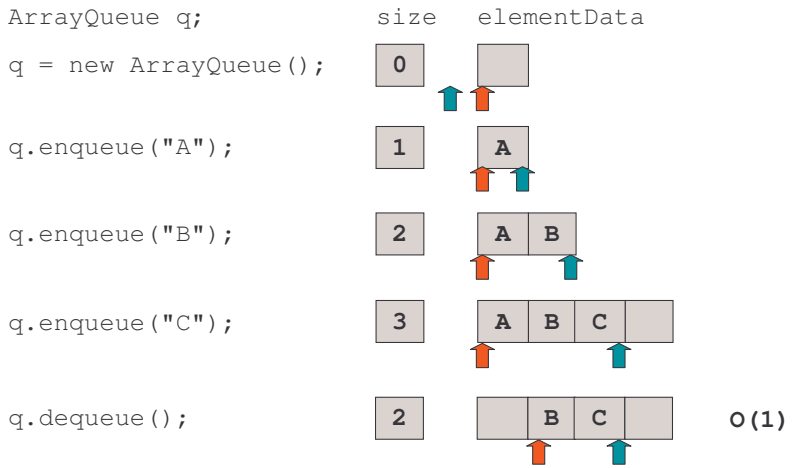
ArrayQueue : สร้าง Queue ด้วยอาเรย์

- เพิ่มที่ท้ายคิว ลบที่หัวคิว
- ให้หัวคิวอยู่ที่ index 0 เสมอ, ตอนลบต้องใช้เวลา O(n)



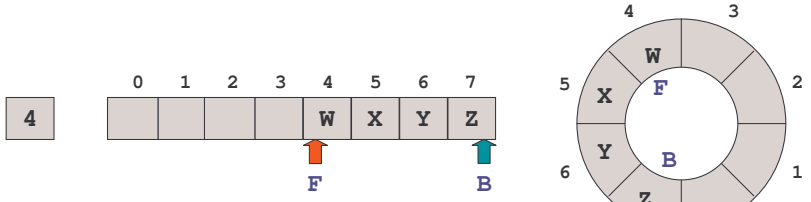
ArrayQueue : สร้าง Queue ด้วยอาเรย์

- เพิ่มที่ท้ายคิว ลบที่หัวคิว, จำ index ของหัวคิว และท้ายคิว
- เพิ่มด้านท้าย O(1) ลบ ใช้วิธีการเลื่อนตำแหน่งหัวคิว



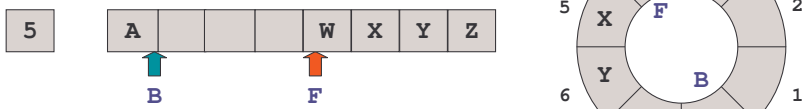
ใช้เนื้อที่ให้คุ้ม

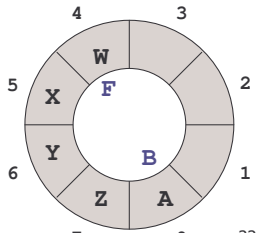
- ลบไปเรื่อย ๆ ตำแหน่งหัวคิว เพิ่มขึ้นเรื่อย ๆ
- มองอาเรย์ให้เป็นแบบ circular จะใช้เนื้อที่ได้เต็มที่



q.enqueue("A");

B = (B + 1) % elementData.length





ArrayQueue : สร้างด้วย circular array

```
public class ArrayQueue implements Queue {
  private Object[] elementData;
  private int size;
  private int front, back;

  public ArrayQueue() {
    elementData = new Object[10];
    front = 0; back = -1;
  }
  public boolean isEmpty() { return size == 0; }
  public int size() { return size; }
  public Object peek() {
    if (isEmpty()) throw new NoSuchElementException ();
    return elementData[front];
  }
  ...  // next slide
}
```

```
public class ArrayQueue implements Queue {
  ...
  public void enqueue(Object e) {
    if (size == elementData.length) {
      Object[] newA = new Object[2 * elementData.length];
      for (int i = 0, j = front; i < size; i++, j = inc(j)) {
        newA[i] = elementData[j];
      }
      front = 0; back = size - 1; elementData = newA;
    }
    back = inc(back); elementData[back] = e;
    elementData[back] = e;
    ++size;
  }
  public Object dequeue() {
    Object e = peek();
    elementData[front] = null; front = inc(front);
    --size;
    return e;
  }
  private int inc(int index) {
    return (index + 1) % elementData.length;
  }
}
```


การบ้าน : ArrayQueue

- ในคลาส ArrayQueue มี
  - front, back, และ size
- คำนวณ size จาก front และ back ได้หรือไม่ ?
- ถ้าได้ ก็กำจัด size ออกจาก ArrayQueue ได้
- ถ้าไม่ได้ จะทำอย่างไร จึงจะไม่ต้องเก็บ size

ตัวอย่างการใช้ queue

- งานที่ต้องใช้แถวคอยเพื่อเก็บข้อมูลโดยตรง
  - print spooling
  - I/O buffer
  - event queue ในระบบ GUI
  - packet forwarding ใน router
  - thread scheduling ใน jvm
- การจำลองระบบด้วยคอมพิวเตอร์ (simulation)
- Breadth-first search
- Radix sort
- ...

Radix Sort

Radix Sort

```
public static void radixSort(Integer[] data, int d) {
  Queue[] q = new ArrayQueue[10];
  for (int k = 0; k < d; k++) {
    for (int i = 0; i < q.length; i++)
      q[i] = new ArrayQueue();
    for (int i = 0; i < data.length; i++)
      q[getDigit(data[i],k)].enqueue(data[i]);
    for (int i = 0, j = 0; i < q.length; i++) {
      while(!q[i].isEmpty())
        data[j++] = (Integer) q[i].dequeue();
    }
  }
}
private static int getDigit(Integer v, int k) {
  int n = v.intValue();
  for (int i = 0; i < k; i++) n /= 10;
  return n % 10;
}
```

จำนวนหลักของข้อมูล

ขอเลขหลักที่ k ของจำนวนเต็ม v

การบ้าน : radixSort(...)

- เขียน public static void radixSort(int [] data)
  - โดยเรียกใช้ radixSort(Integer[] data, int d) ที่ได้เขียนมา
- เขียน public static void radixSort(String[ ] s)
  - s เป็นอาเรย์ของสตริงอังกฤษ (เฉพาะตัวอักษรใหญ่)

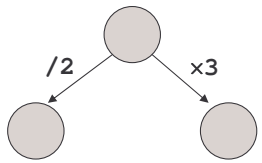
```
String [] s = {"BOY", "ANT", "ANN", "BE", "BET", CAT"};
```


ตัวอย่างการใช้ queue : x3 /2

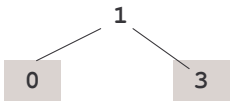
- ให้จำนวนเต็ม v
- เริ่มด้วย 1 จะต้องทำการ x3 และหรือ /2 (ปัดเศษทิ้ง) อย่างไร จึงมีค่าเท่ากับ v
- เช่น
  - v = 10 = 1x3x3x3x3/2/2/2
  - v = 31 = 1x3x3x3x3x3/2/2/2/2/2x3x3/2
- แก้ปัญหานี้อย่างไร ?
- ขอเสนอวิธีลุยทุกรูปแบบ

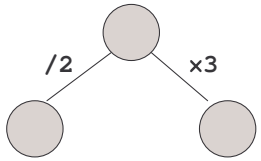
x3 /2 : ลุยทุกรูปแบบ

1

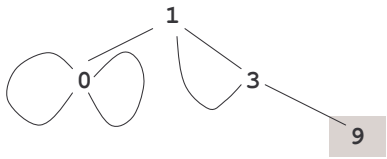


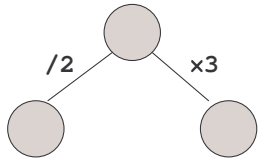
x3 /2 : ลุยทุกรูปแบบ



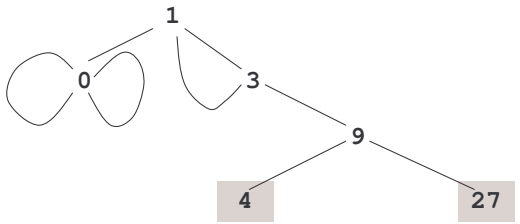


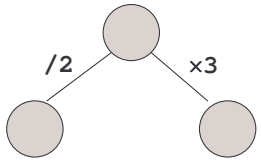
x3 /2 : ลุยทุกรูปแบบ



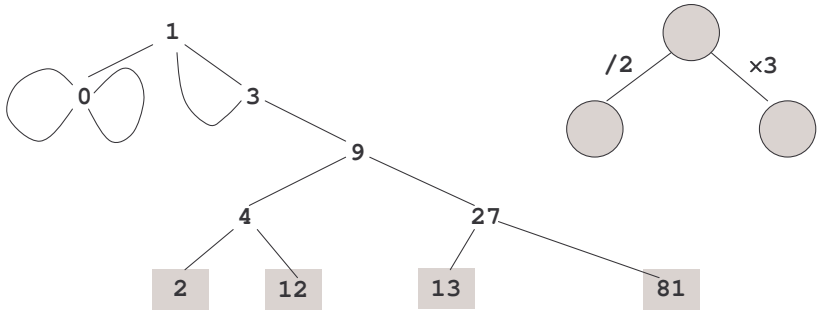


x3 /2 : ลุยทุกรูปแบบ

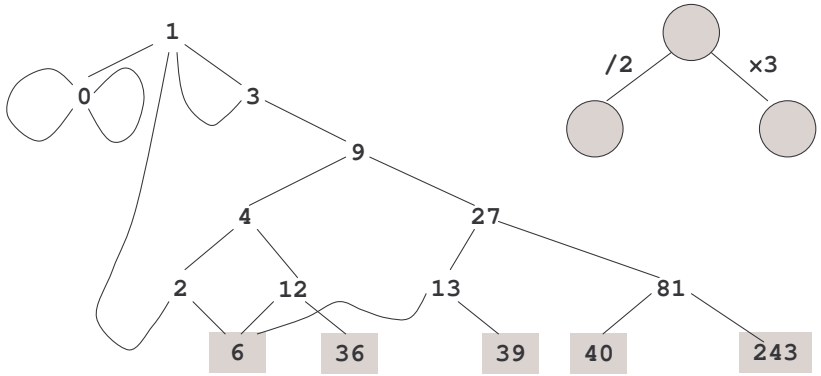




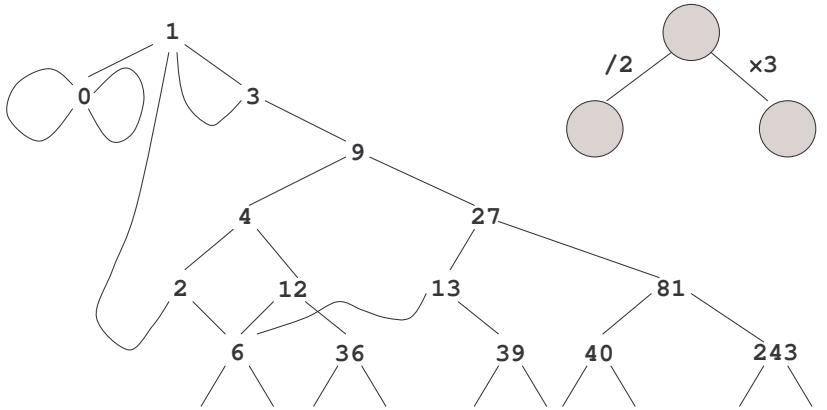
x3 /2 : ลุยทุกรูปแบบ



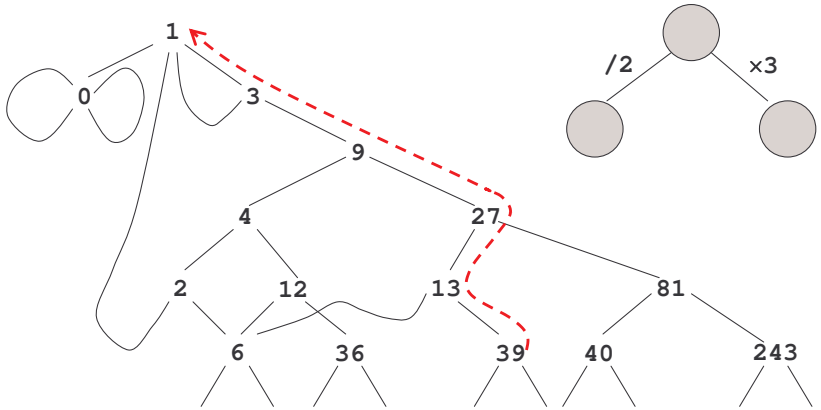
x3 /2 : ลุยทุกรูปแบบ



x3 /2 : ลุยทุกรูปแบบ



x3 /2 : ลุยทุกรูปแบบ

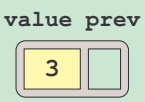


สมมติต้องการหาค่า 39 ก็จะหยุดตอนนี้ และพบคำตอบ

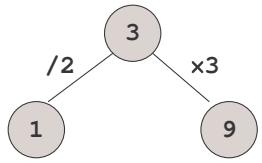
$39 = 1 \times 3 \times 3 \times 3 / 2 \times 3$

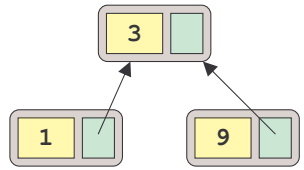
x3 /2 : nodes ต่าง ๆ

```
class Node {
  int value;
  Node prev;
  Node(int v, Node p) {
    this.value = v;
    this.prev = p;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Node)) return false;
    return this.value == ((Node) o).value;
  }
}
```



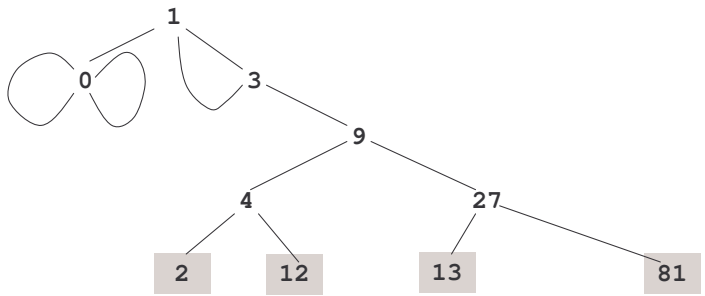
สอง nodes เท่ากันก็เมื่อ values ทั้งสองเท่ากัน





x3 /2 : ใช้ Set และ Queue

- ใช้ set เก็บทุก nodes ที่เคยผลิต
- ใช้ queue เก็บเฉพาะ nodes ที่ยังไม่แตกกิ่ง (node สีเทาๆ ในรูป)

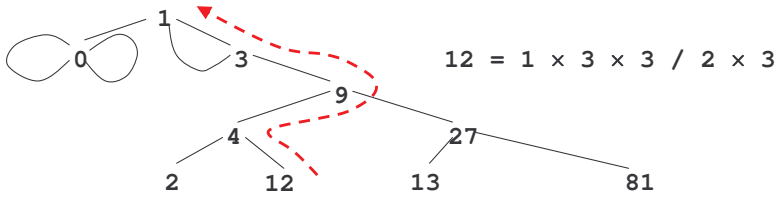


x3 /2 : ลุยทุกรูปแบบ

```
public static void main(String[] args) {
  Set set = new ArraySet();
  Queue q = new ArrayQueue();
  Node v = new Node(1,null);  // start with 1
  q.enqueue(v); set.add(v);
  int target = 31;            // 31 = ?
  while( !q.isEmpty() ) {
    v = (Node) q.dequeue();
    if (v.value == target) break;
    Node v1 = new Node(v.value*3, v);  // try x3
    Node v2 = new Node(v.value/2, v);  // try /2
    if (!set.contains(v1)) {q.enqueue(v1); set.add(v1);}
    if (!set.contains(v2)) {q.enqueue(v2); set.add(v2);}
  }
  if (v.value == target) printSolution(v);
}
```


x3 /2 : printSolution

```
static void printSolution(Node v) {
  if (v.prev != null) {
    printSolution(v.prev);
    System.out.print((v.prev.value / 2 == v.value) ?
                     "/2" : "x3");
  } else {
    System.out.print(1);
  }
}
```



x3 /2 : ใช้ stack แทน queue ก็ได้

```
public static void main(String[] args) {
  Set set = new ArraySet();
  Stack s = new ArrayStack();
  Node v = new Node(1,null);  // start with 1
  s.push(v); set.add(v);
  int target = 31;            // 31 = ?
  while( !s.isEmpty() ) {
    v = (Node) s.pop();
    if (v.value == target) break;
    Node v1 = new Node(v.value*3, v);  // try x3
    Node v2 = new Node(v.value/2, v);  // try /2
    if (!set.contains(v1)) {s.push(v1); set.add(v1);}
    if (!set.contains(v2)) {s.push(v2); set.add(v2);}
  }
  if (v.value == target) printSolution(v);
}
```

x3 /2

- ใช้ queue จะได้คำตอบสั้นที่สุด
  - queue : 31 = 1x3x3x3x3x3/2/2/2/2/2x3x3/2
  - stack : 31 = 1x3x3/2/2x3x3x3x3/2/2/2/2/2x3/2x3x3/2

  - queue : 29 = 1x3x3x3/2x3x3/2/2
  - stack : 29 = 1x3x3/2/2x3x3x3x3/2/2/2/2/2x3/2x3x3/2x3/2/2x3/2x3x3/2/2x3/2x3/2x3/2/2/2/2/2/2x3x3x3x3/2/2/2/2/2x3/2/2/2x3x3/2/2

  หมายเหตุ : รายละเอียดอยู่ในวิชา 2110327

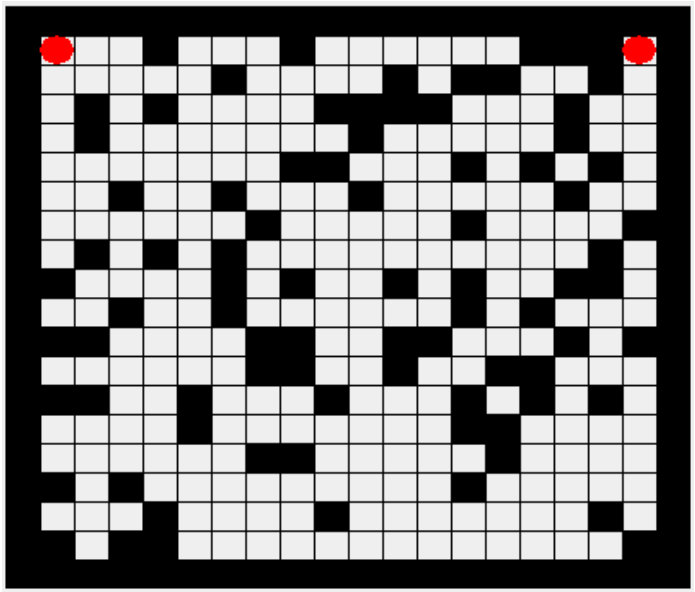
การบ้าน : x3 /2

- ลองสลับสองบรรทัดในโปรแกรม x3 /2 ที่ใช้ stack
- แล้วสั่งทำงานอีกครั้ง สังเกตผล หาเหตุผลว่าทำไมเป็นเช่นนั้น ?

```
if (!set.contains(v1)) {s.push(v1); set.add(v1);}
if (!set.contains(v2)) {s.push(v2); set.add(v2);}
```

```
if (!set.contains(v2)) {s.push(v2); set.add(v2);}
if (!set.contains(v1)) {s.push(v1); set.add(v1);}
```


การบ้าน : ใช้ queue หาเส้นทาง



การบ้าน : ใช้ queue หาเส้นทาง