Lecture 4-1

Linked List

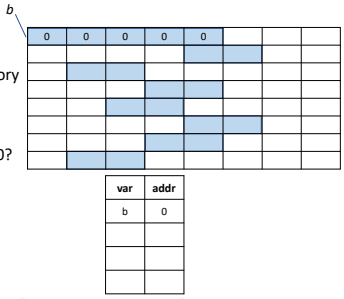
Teera Siriteerakul

1

Limitation of Array

revisited

- We have 45 bytes of memory but we **cannot** allocate an array of size 10.
- Now, how can we make a collection of data of size 10?



var	addr
b	0

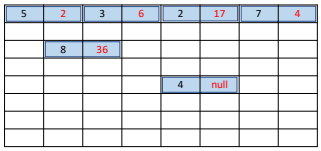
2

Enter the Linked List

- A data structure where a unit of data contain the data itself and a reference to the next data.
  - Made possible by class in JAVA
- Let see a simple example

3

Simple Example



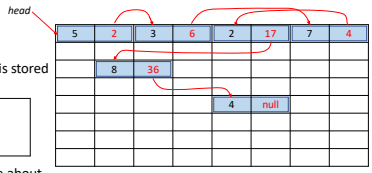
- Again, let's assume there are 64 bytes of dynamic memory
- Assume we want to store the following data: 5 3 7 2 8 4
- Here is what we can do.
- We will call each set of data and address of the next data **a node**.

Note: Only variable we have is head

var	address
head	0

4

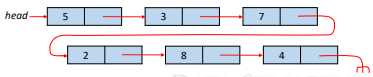
Visualizing Linked List



- Typically, one unit of data is stored in a class call Node, like so

```
public class Node {
    int data;
    Node next;
}
```

- Since we do not really care about the addresses, we usually using arrow to refer to the next data unit. Like this:



var	address
head	0

5

Linked List Operations

- We are to analyze the following Linked List operations
  - Random access (retrieve/update)
  - Add/Insert unordered/ordered array
  - Search in unordered/ordered array
  - Delete unordered/ordered array

6

MyLinkedList.java

LinkedListTester.java

```
public class LinkedListTester {
   public static void main(String[] args) {
      MyLinkedList mList = new MyLinkedList();

      // your code here

      System.out.println(mList.toString());
   }
}
```

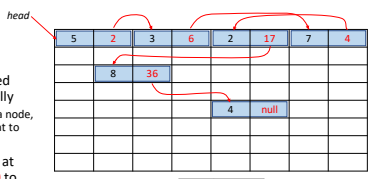
MyLinkedList.java

```
public class MyLinkedList {
   public class Node {
      int data;
      Node next;
      public Node(int d) {
         data = d;
      }
   }
   Node head = null;

   // your code here

   public String toString() {
      StringBuffer sb = new StringBuffer("head ");
      Node p = head;
      while(p!=null) {
         sb.append("--> ");
         sb.append(p.data);
         sb.append(" ");
         p = p.next;
      }
      sb.append("-> null");
      return new String(sb);
   }
}
```

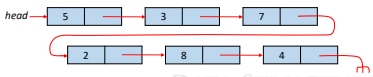
JAVA Inner class
Refer to as MyLinkedList.Node

7

Accessing Linked List Data



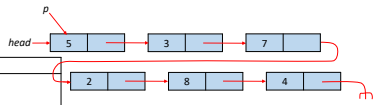
- Random access data in linked list must be done sequentially
  - Before we do anything with a node, we must have a variable point to that node, typically call **p**.
- For example, to access data at index=3, we must go from 0 to 2 to 6 to 4 to access 2
  - Thus, big-O is O(1), O(n), and O(n) respectively



var	address
head	0

8

Getter/Setter




```
getAt() & setAt()
public int getAt(int i) {
    Node p = head;
    while(i>0) {
        p = p.next;
        i--;
    }
    return p.data;
}

public void setAt(int d, int i) {
    Node p = head;
    while(i>0) {
        p = p.next;
        i--;
    }
    p.data = d;
}
```

- Both methods are `O(1), O(n), O(n)`
- What happen when head is `null`?
- We cannot test these methods right now.
  - Let wait until we implement **`add()`**
- Typically, we will not use linked list this way.

9

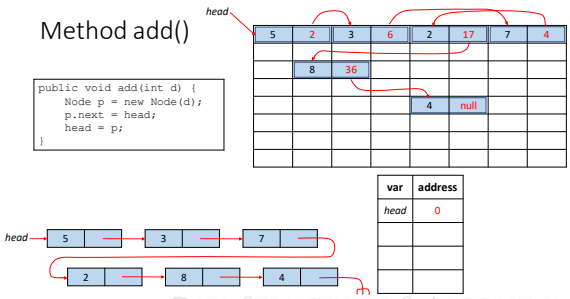
Adding Data into a Linked List

- Where should we put a new data?
  - Two choices: at the head, or at the tail.
  - One of them is O(1), the other is O(n)
- The correct choice is at the head
  - because to get to the tail, we need O(n)
- The steps are simple:
  - Create a new **node**, put the **data** in
  - Point **next** of that node to **head**
  - Point **head** to that **node**
- Let implement it

10

Method add()

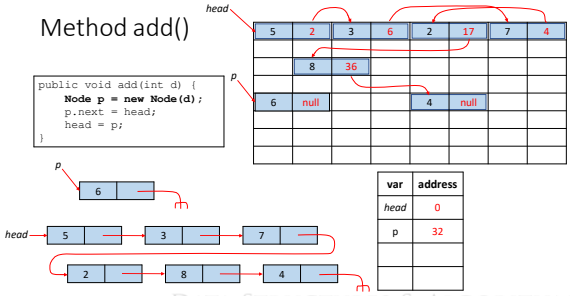
```
public void add(int d) {
    Node p = new Node(d);
    p.next = head;
    head = p;
}
```



11

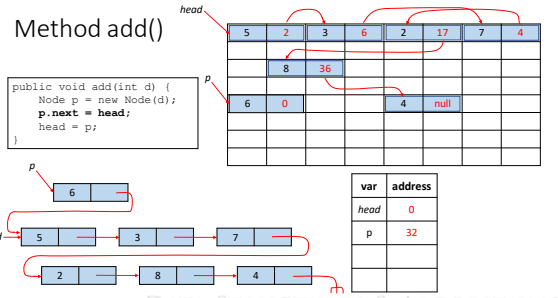
Method add()

```
public void add(int d) {
    Node p = new Node(d);
    p.next = head;
    head = p;
}
```



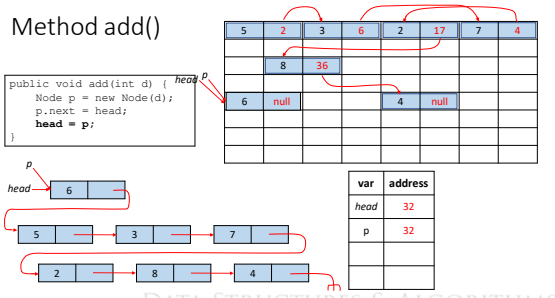
12

Method add()



13

Method add()



14

Summary

- Linked List is a data structure where we only keep address of the first data node and each data node keep address of the next one.
- We can set/get at a specific index in O(n) times.
  - So, we do not typically do it.
- We can add a new data using O(1) time if we insert at the front.

15