Chapter 5

Linked Lists

By: Hassan Adelyar

Linked List

- **Lists** can be implemented in **2 ways**:
  - **Array**
  - **Linked lists**
- Although **arrays** are good, but it has some **disadvantages**:
  - The **size** of array,
  - In an **unordered** array, searching is slow, whereas in an **ordered** array, update is slow.
  - In both kinds of array **deletion** is slow.

Linked List

- The right solution is to build the **data structure** from **small pieces**, and **add** a **new piece** whenever we need to make it larger.
- The **linked list** is a **versatile mechanism** suitable for use in many kinds of **general-purpose** databases.
- In a **linked list**, we store items **non-contiguously** rather than in the usual **contiguous array**.

Linked List

- To do so, we **store each object** in a **node** that contains the object and a **reference** to the next node in the list. **Each object** or node is a **record** which contain **several fields**.

Linked Data Structures

Linked List

- **Node-based** data structure,
- Nodes are **elements**, which may have one or more **pointers**,
- There is **no limit** on the number of elements,
- **Dynamic growing** and **shrinking -** the **key** for some **algorithms**.

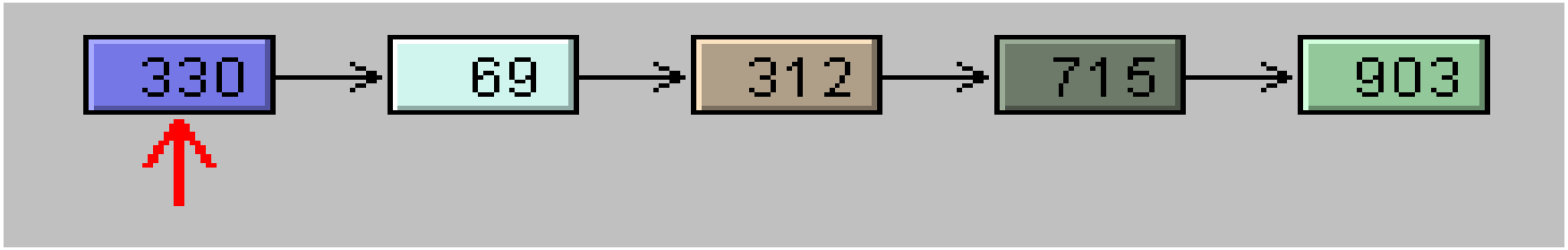
Linked List

- All the **dynamic data structures** we will build have certain **shared properties**.
  - We need a **pointer** to the **entire object** so we can find it,
  - Each cell contains one or more **data fields**, which is what we want to store,
  - Each cell contains a **pointer** field to at least one ``next'' cell. Thus much of the space used in linked data structures is not data.
  - We must be able to **detect** the **end** of the data structure. This is why we need the **NIL pointer**.
  - Just as the **stack reuses memory** when a procedure exits, **dynamic storage** must be **recycled** when we don't need it anymore.

Linked List

- In a **linked list**, each data item is **embedded** in a **link**. A **link** is an **object** of a class, called something like **link**. Because there are many similar links in a list, it makes sense to use a **separate class** for them, distinct from the linked list itself. Each link **object** contains a reference (usually called next) to the next link in the list. A **field** in the list itself contains a **reference** to the first link.

Linked List



Linked List

```
class Link
{
    public int idno;
    public String name;
    public Link next;

    public Link(int id, String stname)
    {
        idno = id;
        name = stname;
    }
}
```

Linked List

```
class LinkList
{
    private Link first;
    public LinkList()
    {
        first = null;
    }

    public boolean isEmpty()
    {
        return (first == null);
    }

    public void insertFirst(int id, String stname)
    {
        Link newLink = new Link(id,stname);
        newLink.next = first;
        first = newLink;
    }
```

Linked List

```
public void deleteFirst()
    {
        System.out.print(first.name + " - " + first.idno);
        first = first.next;
    }

    public void displayList()
    {
        System.out.print("List (first--->last:  ");
        Link selected = first;
        while(selected !=null)
        {
            System.out.println(selected.name + " - " + selected.idno);
            //System.out.print(" , ");
            selected = selected.next;
        }
        System.out.println("");
    }
}
```

Linked List

```
import java.util.Scanner;
class LinkListApp
{
    public static void main(String[] args)
    {
       LinkList theList = new LinkList();

       String studentname;
       int studentid;
     for(int i = 1; i<5; i++){
        System.out.print("Enter Student name: ");
        Scanner keyboard = new Scanner(System.in);
        studentname = keyboard.next();
        System.out.print("Enter Student id: ");
        Scanner keyboard2 = new Scanner(System.in);
        studentid = keyboard2.nextInt();
        theList.insertFirst(studentid,studentname);
      }
```

Linked List

```
        theList.displayList();

        while( !theList.isEmpty() )
            theList.deleteFirst();

        theList.displayList();
    }
}
```

Linked List

```
public void printpointer()
{
        System.out.println(first);
        System.out.println(first.next);
        System.out.println(first.idno);
        System.out.println(first.name);
}
```

Insert After

Linked List

```
public void insert(int id) {
      Link newLink = new Link(id);
      Link s = first;
      Link p = first;
      if (first == null) {
          newLink.next = first;
          first = newLink;
          return;
     }
      while(s !=null) {
          p = s;
          s = s.next;
     }
      p.next = newLink;
      newLink.next = null;
    }
```

Finding a specified links

Linked List

```
public link find (int key)
{
    link current = first ;
    while (current.idata != key )
    {
        if ( current.next = = null)
        return null;
        else
        current = current.next;
    }
    return current;
}
```

Deleting a specified links

Linked List

```
public link delete ( int key)
{
    link current = first;
    link previous = first;
    while ( current.idata != key )
    {
        if  ( current.next = = null)
        return null;
        else
        {
        previous = current;
        current = current.next;
        }
    }
    if ( current = = first)
        first = first.next;
    else
        previous.next = current.next;
    return current;
}
```