Applications of Linked Lists

Linked List concept can be used to deal with many practical problems.

**Problem 1:** Suppose you need to program an application that has a pre-defined number of categories, but the exact items in each category is unknown.

**Solution:** Pre-defined number of categories implies that we can use a simple static structure like array to represent the categories. Since we do not know the number of items in each category, we can represent items in each category using a linked list. So what we need is an array of linked lists.

![Diagram of linked lists](image)

**More Examples**

You can also think of representing a web index using an array of linked lists, where array contains the keywords and linked lists contains the web URL’s where that keyword occurs.
Sparse Matrices

- Most problems can be modeled by a system of linear equations
  - Thousands of equations (constraints)
  - Thousands of variables (unknowns)
  - It is not practical to allocate n by n space to store a linear system
  - How much space is necessary for a matrix of size 10000x10000 where each entry is a double?
  - Sparse matrix representation

Designing the Node for Sparse Matrix

```java
public class Node {

    public Comparable data;
    public int row, col;
    public Node nextrow;
    public Node nextcol;

    Node(int row, int col, Comparable data) {
        this.data = data; this.row = row; this.col = col;
        nextrow = null; nextcol = null;
    }

    public String toString(){return data.toString();}
}
```
Other Advanced Operations on Linked Lists

Reversing a Linked List

Cloning a Linked List

Let us begin by evaluating the following code

```java
LinkedList list = new LinkedList();
list.prepend(new Node("first");
list.append(new Node("last");
LinkedList list2 = list;
```

The reference list2 is an alias, it points to the same object as list1. This means that once we change list, these changes occur in list2 as well. In this case, we consider list2 as a shallow copy of list1. In other words, list1 and list2 shares the same nodes in one linked list. In many cases it will be quite useful to have a deep copy of an object. To create a deep copy, we need to duplicate all nodes in the list. For this purpose, the Object class provides the method clone(), which we will override in the LinkedList class:

```java
public Object clone() throws java.lang.CloneNotSupportedException {
    LinkedList twin = new LinkedList();
    if(head == null) return twin;

    Node tmp = head;
    for(int i = 0; i < size; i++)
    {
        twin.append(tmp.data);
        tmp = tmp.getNext();
    }

    return twin;
}
```
Sorting a Linked List

- Sorting a List is one of the common operations that are performed.
- Sorting a linked list is not that trivial since linked list nodes cannot be randomly accessed.
- One useful way to sort a linked list is to remove a node from the old list and insert into a new list in order.
  - However for large lists, this is not very practical as insertion sort requires $O(n^2)$ operations.
- An alternative method is to define a toArray method for the linked list class that allows the linked list to be converted to an array first and then apply a more efficient algorithm like quick sort to sort.
  - Once sorted the array can be converted into a linked list again.

Questions

- Choosing the proper data structure depends on the application. Specify what data structure you would choose in each of the following cases. You can choose from a static array, singly linked list, circular LL, doubly LL, array of LL’s, multilinked list etc
  - A sorted file is given and a list in reverse order needs to be built in $O(n)$

  - An application requires a structure where new nodes can easily added to the front and back of a given node in $O(1)$

  - An application requires a data structure that can be randomly accessed

  - A set of entries needs to be sorted by a category first. Each category will receive an unknown number of entries

  - An application requires frequent insertions, generally in the same region

  - A list needs to be maintained in multiple sorted orders, but space for each entry can be allocated only once.