COMP102

Part 1

Data Types and Structures

Lecture 9

Record Maintenance Using Linked Lists

Linked lists have beeen used to build two further ADTs in which specific ordering protocols applied: the ADTs **Stack** and **Queue**.

Another use of dynamic ADTs is in maintaining collections of information, or **database records** to allow:

1. ordering by value of a particular field (or **key**).
2. insertion of new records in the 'correct' place.
3. deletion of 'inactive' records.
4. reporting of data held in a record to be reported when its key is given.

The next part of COMP102, describes some general and powerful systems that are used in constructing **Databases**.

The properties and design theory of such systems form a major part of Computer Science.

The the set of methods described in this lecture effect only a very simplistic approach.

In this lecture the aims are:

1. To illustrate some of the elements in a typical database systems
2. To indicate how linked lists can be used in such systems.

A 'Toy' Example: Video Collections

As an example of some of the questions raised in database design, consider an application similar to the standard 'Library Collection' example.

Information regarding films in video collection:

Below are some data one might wish to record:

An **Identifying number** cf. ISBN/ISSN

title

Original **language**

BBFC (or other) Certificate

Director

ID	Title	Language	Cert.	Director
01	Ai No Corrida	Japanese	NC-17	Oshima
02	Le Mepris	French	15	Godard
03	Black Rain	English	18	Ridley Scott
04	The Offence	English	18	Sydney Lumet
05	M	German	94	Fritz Lang
06	Showgirls	English	18	P. Verhoeven
07	Dirty Weekend	English	18	M. Winner
08	Deux ou trois choses que je sais d'elle	French	15	Godard
09	Krótki film o milosc	Polish	15	Kieslowski
10	Novecento	Italian	18	Bertolucci
11	On the Waterfront	English	PG	Elia Kazan
12	After Hours	English	15	M. Scorsese
13	Unforgiven	English	15	C. Eastwood
14	The Unforgiven	English	PG	John Huston
15	Trois Couleurs: Bleu	French	15	Kieslowski
16	Tokyo monogatari	Japanese	U	Ozu
17	Taxi Driver	English	110	M. Scorsese
18	Triumph des Willens	German	E	L. Riefenstahl
19	The Vanishing	Dutch	107	G. Sluizer
20	The Vanishing	English	15	G. Sluizer

Having organised a collection of records of this form, typical **queries** could be,

a) List all **Titles** with a given **Director**

b) Give the total number of films with an 'adult' **Certificate** i.e. 18 or NC-17

c) List all the film **Titles** and **Directors** of records whose **Language** is not English.

The next part of this module will introduce a specific system for formulating queries such as these with respect to data organised in a given form.

In this lecture, realisations of the following methods are given:

These serve as a rudimentary framework for maintaining a collection of records where each has a unique key identifying it.

to determine if a given record (i.e. key) is present.

to return the record data associated with a given key.

to insert a new record.

to delete a record from a collection.

In order to facilitate the realisation it is useful to develop the linked list class of Lecture 6 to a form where an explicit Key field is specified:


```
public class Records
  {
  private class RecordCell
    {
    private Object Key;   // Unique Key value
    private Object Datum;
    private RecordCell Link;
    //
    private RecordCell(Object KeyVal,
                       Object head,
                       RecordCell next_cell)
      {
      Key=KeyVal; Datum = head; Link=next_cell;
      }
  }
  private RecordCell Head;
  private RecordCell Tail;
  private int CellCount;
  //****************
  // Instance Methods*
  //
  //  As with LinkedList class, and adding
  //***********************************
  //  Obtain the Object in the Key field
  //  at start
  //**********************************
  public Object GetHeadKey()
    {
    if (CellCount!=0)
      return Head.Key;
    else
      return null;
    }
```


Insertion and Deletion

Records are to be sorted using the **Key** value, (an Object):

Question:: If X, Y are Objects, what does $X < Y$ mean?

If X and Y are **Number** instances, can use arithmetic ordering.

If X and Y are **String** use lexicographic order.

Raises the problem:

What if X and Y are neither?

Answer: Assume the *toString()* instance method has been over-ridden in such a way to all X and Y to be ordered as *String*.


```
//*********************************//
// Compare the two Objects, X and Y://
// numeric if both are Number      //
// String otherwise                //
// Returns true if X `<' Y         //
//*********************************//
public static boolean IsBefore (Object X,
                                Object Y)
  {
  double xv, yv;
  if ( (X instanceof Number) &&
       (Y instanceof Number) )
    {
    xv = Double.valueOf(X.toString()).doubleValue();
    yv = Double.valueOf(Y.toString()).doubleValue();
    return (xv<yv);
    }
  else
    {
    return ((X.toString()).compareTo(Y.toString())<0);
    };
  }
```

Recall that, for user-defined Object classes, it is usual to over-ride the method toString().

A means for ordering Objects can be used in membership and reporting methods, i.e

```
//******************************//
// Determine if Key occurs in RB //
//******************************//
public static boolean IsMember (Object Key,
                                Records RB)
  {
  Records temp = RB;
  boolean found_key = false;
  while ((!found_key) && (!temp.IsEmpty()) )
    {
    if ( !IsBefore(Key,temp.GetHeadKey()) &&
         !IsBefore(temp.GetHeadKey(),Key) )
      {
      found_key=true;
      }
    else
      temp = temp.GetTail();
    };
  return found_key;
  }
```



```
//*******************************************//
// Return the Record Data of the entry in RB //
// whose Key field matches the Key parameter //
//*******************************************//
public static Object GetRecord (Object Key,
                                Records RB)
  {
  Records temp = RB;
  Object result = null;
  boolean found_key = false;
  while ((!found_key) &&
         (!temp.IsEmpty()) )
    {
    if ( !IsBefore(Key,temp.GetHeadKey()) &&
         !IsBefore(temp.GetHeadKey(),Key) )
      {
      result = temp.GetHeadDatum();
      found_key=true;
      }
    else
      temp = temp.GetTail();
    };
  return result;
  }
```

There are 4 cases when inserting a record **Entry** with key **Key** into a collection of records **RB**:

1. **Key** occurs in **RB**.
2. **RB** is empty.
3. **Key** precedes the key of the first record in **RB**

4. None of the above cases hold,

What actions do each of these cases entail?

In Case 1, return the collection unchanged.

For Case 2 and Case 3, the new record will be the new first record and so return the the result of invoking the instance method AddHead with **Entry** and **Key**.

Finally, in Case 4, the only possibility is that **Key** comes **after** the key value held in the first record of **RB**.

The pair (Key,Entry) has to inserted into the collection formed by all records in **RB** after the first. So we recursively insert the new record into the tail of the list and append the existing first element onto the result.


```
public static Records InsertRecord (Object Key,
                                    Object Entry,
                                    Records RB)
  {
  Records res=RB;   // Result returned.
  Records temp;
  if (IsMember(Key,RB))
    {
    return res;
    }
  else if ( (RB.IsEmpty()) ||
            (IsBefore(Key,RB.GetHeadKey())) )
    {
    res.AddHead(Entry,Key);
    return res;
    }
  else
    {
    temp = new Records();
    temp.AddHead(RB.GetHeadDatum(),
                 RB.GetHeadKey());
    return
      Concatenate(temp,
          InsertRecord(Key,Entry,RB.GetTail()));
    };
  }
```

Note the (adapted) **Concatenate** method from Lecture 6.

A similar breakdown applies to of deleting a record with a specified key from an ordered collection.

If **Key is not present no action is required.**

If the key field of the first record in RB precedes **Key** then the record is deleted from

the collection formed by the tail of **RB**.

Otherwise, **Key** must be the **first** record and so the result required is the tail of **RB**.

```
public static Records DeleteRecord (Object Key,
                                    Records RB)
  {
  Records res = RB;
  Records temp = new Records();
  if (!IsMember(Key,RB))
    {
    return res;
    }
  else if (IsBefore(RB.GetHeadKey(),Key) )
    {
    temp.AddHead(RB.GetHeadDatum(),
                 RB.GetHeadKey());
    res =
      Concatenate(temp,
             DeleteRecord(Key,RB.GetTail()));
    return res;
    }
  else
    {
    res = RB.GetTail();
    return res;
    };
  }
```


Summary

1. Maintaining information in the form of record structures is one of the major applications of computer systems and is the central topic of study in the field of Database Systems.
2. Using linked lists as a dynamic ADT (with an imposed ordering regime) is an extremely crude method for providing some rudimentary database features. This use is, however, unsuitable for all but very small scale applications.
3. Among the drawbacks to lists are: searching for a given key in an ordered list of keys will (on average) require inspecting half the records: for all but small (10s-100s) numbers this length of time becomes unacceptable.
4. Some of the drawbacks of a simple linked list structure can be overcome by more intricate dynamic forms, one such - Binary Trees will be considered in the next lecture.
5. The study of suitable forms of data organisation to deal with very large record systems (10^6 + records) continues to be an important research domain in Computer Science.