Date
1-NOV-2020

Assignment No. 08

21118 | E-1 (SE-1)

Problem Statement:

Second year Computer Engineering class, set A of students like Vanilla Ice-cream & set B of students like butterscotch Ice-cream. Write C++ program to store two sets using linked lists.

Compute & display:

a> Set of students who like both vanilla & butterscotch.

b> Set of students who like either vanilla or butterscotch or not both.

c> Number of students who like neither vanilla nor butterscotch.

Objectives:

a> To understand the concepts of OOP paradigm.

b> To understand usage of linked-list in creation of dynamic lists in C++.

Outcomes.

a> To implement set operations using linked-list data structure in cpp.

b> To write menu driven, modular OOP based (classes & objects) program in cpp.

c> To implement functions in cpp (methods in case of classes)

Hardware Requirement:

Manufacturer & Model: Acer, Swift 3

Processor: Intel core i5-8th gen. (8265U @ 1.8 GHz)

Installed memory: 8GB RAM, 512 GB SSD.

Architecture: 64-bit, windows 10 operating system

Software Requirement:

Date
Page

Operating System: Windows 10
Cpp version used: C++14
Compiler for C++: g++ (version 10.1.0)
IDE: Code-blocks (version: 20.03)

Theory:

Linked List:

~~Requir~~ Linked list is a linear data structure which store elements in squence. Unlike array they don't ocupy consecutive memory locations & also they cannot accesed by indexing (using subscript operator)

requirement:

The advantages of using linked list over arrays are.

1) The memory wastage in case of linked-list is very less compared to array as we use only required memory.
2) linked list are ~~very~~ benificial in some practical usage like ~~u~~ cache memory etc.
3) Insertion & deletion operations in linked list are less costlier compared to arrays.

Types.

1) Singly linked list
2) Doubly linked list
3) Circular linked list
4) More advanced like Unrolled linked list & skip lists etc.

Singly linked list: This consist of number of node where each node points to the next node.

node of the singly linked list can be represent as follow:

```
struct Node {
    int data;   // data or any info to be stored at node
    Node next;  // pointer to next node
}
```

Cost of operations in singly linked lists:

operation	Time.
Insertion at beginning	O(1)
Insertion at end	O(n)
Deletion at nth position	O(n)
Accessing nth node	O(n)

Pseudo Code:

```
Algorithm addNode( int data) {          // adds node with data given as i/p
    ~~Node~~ newNode ← getNewNode(data);
    if (head == null) : head ← newNode.
    else. { newNode→next ← head
        head ← newNode
    }
}

Algorithm deleteNode (int data){     // delete node of i/p data.
    ~~temp ← head~~
    if( head → data == data) {
        temp ← head
        head ← head →next
        delete temp
    }
    else {
        curr ← head, prev ← null
        while( curr && curr → data != data) {
            prev = curr,
            curr ← curr→next
        }
        if (curr == null) : print( element not found)
        else { prev→next ← ~~current~~ curr next
            delete curr
        }
    }
}
```

Operating System: Windows 10
Cpp Version used: C++14 ,
Compiler for C++: g++ (version 10.1.0)
IDE: Code-blocks (version: 20.03)

\# Theory:

Linked List:

~~Requir~~ Linked list is a linear data structure which store elements in squence. Unlike array they don't occupy consecutive memory locations & also they cannot accessed by indexing (using subscript operator)

Requirement:

The advatages of using linked list over arrays are

1> The memory wastage in case of linked-list is very less compared to array as we use only required memory.

2> Linked list are ~~very~~ benificial in some practical usage like ~~used~~ cache memory etc.

3> Insertion & deletion operations in linked list are less costlier compared to arrays.

Types:

1> Singly linked list
2> Doubly linked list
3> Circular linked list
4> More advanced like Unrolled linked list & skip lists etc.

Singly linked list: This consist of number of node where each node points to the next node

node of the singly linked list can be represent as below

```
struct Node {
    int data;   // data or any info to be stored at node
}   Node next;  // pointer to next node
```

Cost of operations in singly linked lists:

operation	Time
Insertion at beginning	O(1)
Insertion at end	O(n)
Deletion at ith position	O(n)
Accessing ith node	O(n)

Pseudo Code:

```
Algorithm addNode( int data){          // adds node with data given as i/p
    ~~node~~ newNode ← getNewNode(data);
    if (head == null) : head ← newNode.
    else.{ newNode → next ← head
           head ← newNode
         }
}

Algorithm deleteNode (int data){       // delete node of i/p data.
    ~~temp ← head~~
    if( head → data == data) {
        temp ← head
        head ← head → next
        delete temp
    }
    else {
        curr ← head, prev ← null
        while( curr && curr → data != data) {
            prev = curr,
            curr ← curr → next.
        }
        if (curr == null) : print( element not found)
        else { prev → next ← curr → next
               delete curr.
        }
    }.
}
```

```
Algorithm isPresent(int data) {        //check if node of ^data is
                                                  i/p
                                        present in the list/not
    temp ← head
    while (temp != null & temp→data != data)
        temp ← temp → next
    if (temp == null) return (false);
    else return (true);
}
```

ADT of classes.

The classes used in program are:

1) class Node {
```
private:
    int roll_no;
    string name;
    Node* next;
public:
    Node (int roll_no, string name) {   //constructor
        //initializes instance member variables
    }
}
```

2) class linked list {
```
private:
    Node* head;
public:
    linked list() { Constructor
    } // initializes instance member variables
    void addNode (int roll_no, string name) {
        // code to add node in linked list
    }

    void deleteNode (int roll_no.) {
        // code to delete node in linked list
    }
```

```
    void isPresent (int roll_no) {
      //checks wheather roll_no is present in list / not
    }
    void Print() {
    } // prints the entire list.
}

3) class Set {
   private:
     LinkedList students;
   public:
     void getInput() {
       // takes data of students from the user
     }
     void setToAnotherSet (Set S) {
     } // used to initialize current set with set S.

     void addStudent ( int roll_no, string name) {
     } // adds new student to the set.

     set getUnion( Set B) {
       //compute union of set with set B & returns it
     }
     Set get Intersection( Set B) {
       // compute intersection of set with set B & return it
     }
     Set get Diff ( set B) {
       //compute difference set & returns it
     }
     void printSet() {
     } // prints the entire set.
```

Analysis of Algorithms:

1) ~~Checking~~ Union of Set:

Time Complexity: The algorithm take O(m.n) time where m & n are the sizes of two sets.

Space Complexity: The algorithm Takes O(m+n) space to store the list of union set.

2) Intersection of Set:

Time Complexity: The algorithm requires O(m.n) time where m&n are the sizes of two sets.

Space Complexity: The algorithm req. O(min(m,n)) space to store the intersection set of two i/p sets.

3) Difference of Set:

Time Complexity: The algorithm require O(m×n) time where m & n are sizes of two i/p sets.

Space Complexity: The algorithm req. O(m) space to store difference.

All the operations which are required to calculate the number of students are just the variations of the above hence they will have same complexity.

Test Cases:

No.	TestCase Description	Expected o/p	Program o/p
1)	Universal Set: {{'A',1}, {'B', 2}, {'C',3}, {'D',4} {'E', 5}, {'F",6}, {"I",7}} {"X", Y} → X is the name of student & Y is the roll no.	vanilla & butterscotch {{"C",3}} either vanilla or butterscotch but not both. {{"B",2}, {"D",4},	vanilla & butterscotch {{"C",3}} either vanilla/ butterscotch but not both. {{"B",2}, {"D",4},

Vanill Set: {{"B", 2}, {"C",3}, {'D', 4}} Butterscotch Set: { {"C",3}, {'F",6}, {'I",7}}	{'F",6}, {'I",7}} neither vanillar nor butterscotch. {{"A",1}, {"E",5}}	{'F",6}, {'I",7}} neither vanilla nor butterscotch. {{"A",1}, {"E",5}}

Applications:

1) Implementation of other data structures like stack, queue, adj list representation of graph. etc.
2) Image viewer.
3) Prev & next page of browsers.
4) Music player

Conclusion:

Linked list provides upper-edge in case of memory usage over arrays. It can be used in practical scenarios but may not be useful always due to it's high access time compared to arrays.