Data Structures and Algorithms [SBE201] (Spring 2020)
Report 1

Linked Lists

Asem Mohamed Alaaeldin Sec. 0 BN. 0

Sunday 12th April, 2020

1 Problem Set

1.1 Linked List Size

```cpp
struct IntegerNode
{
  int data;
  IntegerNode *next;
};

int size( IntegerNode *front )
{

}
```

1. Problem

A) Implement the function `size` that returns the size of a given linked list (count of elements).

B) Provide a time complexity estimate using the Big-O notation.

C) Can you provide a recursive version of the `size` function?

1. Solution

1.2 Linked List Operations

```cpp
#include <iostream>
struct IntegerNode
{
  int data;
  IntegerNode *next;
};
void funx(node* front)
{
  if(front == nullptr) return;
  fun1(front->next);
  std::cout << front->data << " ";
}
```

2. Problem

A) What does the function `funx` do?

B) What is the output would be if the input linked list is represented in order as: `5->90->300->7->55`

C) What is the time complexity of such a function.

2. Solution

1.3 Doubly-Linked List

```
struct IntegerNode
{
  int data;
  IntegerNode *next;
  IntegerNode *back;
};

struct IntegersLL
{
  IntegerNode *front;
};

void insertAt( IntegersLL &list , int index, int data )
{

}
```

3. Problem

A) Implement a function `insertAt` to insert an element at arbitrary `index` in a **linked list**.

B) Provide a visual illustratoin to the steps in order to support that operation.

3. Solution

1.4 Circular Linked List

```
struct IntegerNode
{
  int data;
  IntegerNode *next;
  IntegerNode *back;
};

struct IntegersLL
{
  IntegerNode *front;
};

void pushFront( IntegerLL &list, int data )
{
    list.front = new node{ data , list.front };
```

```cpp
}

node *backNode( IntegerLL &list )
{
    node *temp = list.front;
    while( temp->next != nullptr )
        temp = temp->next;
    return temp;
}

void *pushBack( IntegerLL &list, double data )
{
    if( list.front == nullptr )
        return pushFront( list , data );
    else
    {
        node *back = backNode( list );
        back->next = new node{ data , nullptr };
    }
}

void removeBack( IntegerLL &list )
{
    if( isEmpty( list ))
        return;
    else if( list.front->next != nullptr )
        removeFront( list );
    else
    {
        IntegerNode *prev = list.front;
        while( prev->next->next != nullptr )
            prev = prev->next;
        delete prev->next;
        prev->next = nullptr;
    }
}

void printLL( IntegerLL &list )
{
    node *current = list.front;
    while( current != nullptr )
    {
        std::cout << current->data;
        current = current->next;
    }
}
```

4. Problem

The functions: `pushFront`, `backNode`, `pushBack`, `removeBack`, and `printLL` are a implemented earlier for a regular linked list. How would you change each function to work properly for a circular linked list that uses only a **front** pointer.

4. Solution