数据结构课程设计

北京郵電大學

姓　　名 tobeApe6eHok

学　　院 人工智能学院

专　　业 智能科学与技术

班　　级 111111111111

学　　号 1111111111

班内序号 1

指导教师 周延泉

2022 年　4 月

目录

数据结构课程设计实验一

一、实验题目

设计一个算法，实现在非空单链表中的某一 p 结点之前插入新结点 q 的操作，要求插入操作的时间复杂度尽可能小。

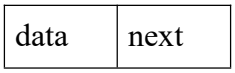


图 1 结点结构

结构类型定义如下：

```
Template<class T>
struct Node
{
    T data;
    struct Node<T> * next;
}
```

二、题目分析与算法设计

1、设计思路：

首先需要定义一个链表的结构：我定义的链表结构是有头节点的链表，头节点不存放任何数据，只是用于初始化链表的头，这样空链表也有相应的地址，在进行操作时就不需要进行多余的判断，更加方便。

其次，一个链表如果要在指定的结点之前插入一个节点，需要考虑如何指定节点，主要有三种方法：

（1）给定该节点的地址

（2）给定该节点存放的数据

（3）给定该节点是第几个位置

针对这三种方法，其实是由重叠的部分，（2）、（3）种方法相对于第一种方

法的区别只是需要先找到指定的元素，找到之后就可以得到该元素的地址，就可以直接进行指定地址的插入方法。（2）、（3）函数只需要实现不同的查找节点的方法，插入时只需要调用第一种方法即可。

2、算法描述：

由 1 的分析可得，只需要实现两种查找节点方法和指定地址的插入方法即可。

对于根据位置查找的方法，首先要判断位置是否合法，如果位置是小于等于

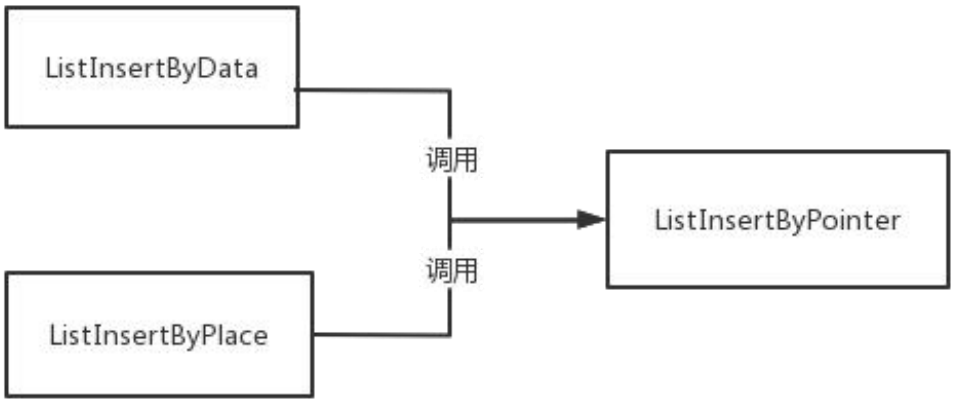
0，那么表示位置在头节点或者之前，不合法；如果位置大于节点的总个数，也是不合法的。其次，只需要从头遍历一遍链表并且记录当前节点的位置就可以找到指定位置的节点。

对于根据数据查找的方法，只需要遍历一遍链表，依次比较数据，如果数据相同，就找到了该节点。

对于指定地址的插入方法，我们已经得到了节点的位置，想在该节点之前插入节点。一种插入方法是得到该节点之前的那个节点，然后直接在其后插入即可；另一种方法是在该节点之后插入节点，然后将该节点的数据放入后面的那个新节点，然后将要插入的数据放入到该节点，就完成了前插的效果。此处我选择第二种方法。

3、函数调用关系：

由 1 的分析可得，函数的调用关系是：



三、数据结构描述

程序中我定义了一个 List 类，在类中定义了题目中提供的节点 struct 结构体，这样可以对各种操作进行封装，使程序更健壮：

```
//创建一个模板类
template<typename T>
class List {
private:
    //在模板类中定义一个对应的结构体,用于定义节点的结构
    struct Node {
        T data;                 //存储节点的数据
        struct Node* next;      //存储下一个节点的地址
    };
    Node* head;                 //链表的头节点
    Node* tail;                 //链表的尾节点
    int len;                    //记录链表的长度,头节点不计算在内
public:
    List();
    ~List();
    void PushBack(T value);
    void ListPrintf();
    void ListInsertByPlace(int place, T value);
    void ListInsertByData(T target, T value);
    void ListInsertByPointer(Node* ptr, T value);
};
```

四、程序清单

我的实验环境是：win10 + VS2019，以下是在机器上运行结果正确的代码：

```
#include<iostream>
using namespace std;

//创建一个模板类
template<typename T>
class List {
private:
    //在模板类中定义一个对应的结构体,用于定义节点的结构
    struct Node {
        T data;                 //存储节点的数据
        struct Node* next;      //存储下一个节点的地址
    };
    Node* head;                 //链表的头节点
    Node* tail;                 //链表的尾节点
    int len;                    //记录链表的长度,头节点不计算在内
public:
    List();                                             //链表的构造函数
    ~List();                                            //链表的析构函数
    void PushBack(T value);                             //向链表的最后一个位置插入一个节点,主要用于用户创建链表时
    void ListPrintf();                                  //依次打印链表中的节点,主要用于验证链表的插入效果
    void ListInsertByPlace(int place, T value);         //指定第几个节点(第0个节点是头节点),在该节点前插入节点,value是要插入的数据
```

```
    void ListInsertByData(T target, T value);          //指定数据,在含有该数据的第一个节点前插
入节点,value 是要插入的数据
    void ListInsertByPointer(Node* ptr, T value);      //指定一个节点的地址,在该地址的节点前插
入节点(用户一般不会使用),该函数被另外两个插入函数调用
};
//构造函数:
//1、初始化头节点,此时链表只有头节点,所以头节点就是尾节点
//2、由于头节点不计算在链表长度内,所以此时链表长度为 0
template<typename T>
List<T>::List() {
    tail = head = (Node*)malloc(sizeof(Node));//使用 malloc 申请内存
    len = 0;
}
//析构函数:
//逐个将所有的节点的内存都释放掉
template<typename T>
List<T>::~List() {
    Node* tmp = NULL;//tmp 用于记录要释放内存的节点地址
    while (head != NULL)
    {
        tmp = head;
        head = head->next;
        free(tmp);//使用 free 释放内存
    }
    len = 0;//节点都释放掉之后,链表长度变成 0 (头节点也没有了)
}
//在链表的最后加一个节点,用户可以反复使用这个函数初始化要构造的链表
template<typename T>
void List<T>::PushBack(T value) {
    Node* tmp = (Node*)malloc(sizeof(Node));//加入一个节点
    if (tmp) {//如果申请空间成功
        //在 tail 后插入该节点,并且 tail 指向该节点
        tmp->data = value;
        tmp->next = NULL;
        tail->next = tmp;
        tail = tmp;
        ++len;
    }
    else {
        cout << "无法申请空间" << endl;
    }
}
//将链表的节点的 data 依次打印出来,用于验证链表确实完成了在某个节点前插入节点的功能
template<typename T>
void List<T>::ListPrintf() {
    Node* tmp = head->next;
    while (tmp != NULL) {
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}
//根据节点的地址插入值为 value 的节点
//该函数一般不会被用户调用,该函数被另外两个插入函数调用
template<typename T>
void List<T>::ListInsertByPointer(Node* ptr, T value) {
    Node* newNode = (Node*)malloc(sizeof(Node));//先申请节点内存空间
    if (newNode) {//如果内存空间申请成功
        //先将指定节点的数据 copy 到下一个空节点
        //然后将要插入的数据放到指定节点上,这样就完成了插入
        newNode->data = ptr->data;
```

```
        newNode->next = ptr->next;
        ptr->data = value;
        ptr->next = newNode;
        //此处需要考虑一个特殊情况:
        //如果 tail 是要找的节点,由于插入的机制时在其后插入一个节点,所以 tail 就不能保证指向最后一
个节点了
        //所以此处进行特殊判断
        if (tail->next != NULL)
        {
            tail = tail->next;
        }
        //插入了一个节点,链表长度+1
        ++len;
    }
    else {
        cout << "无法申请空间" << endl;
    }
}
//根据指定的数据找到节点,并在其之前插入节点
template<typename T>
void List<T>::ListInsertByData(T target, T value) {
    Node* tmp = head->next;
    //首先找到 data==target 的节点
    while (tmp != NULL && tmp->data != target)
    {
        tmp = tmp->next;
    }
    //如果 tmp==NULL,那么说明链表中没有这样的节点
    if (tmp != NULL)
    {
        ListInsertByPointer(tmp, value);//找到了该节点,就可以使用根据地址插入的函数插入节点
    }
    else
    {
        cout << "未找到目标节点" << endl;
    }
}
//根据位置在节点前插入节点
template<typename T>
void List<T>::ListInsertByPlace(int place, T value) {
    //判断指定的位置是否有效
    if (place > len || place <= 0) {
        cout << "位置错误" << endl;
    }
    else {
        Node* tmp = head->next;
        //找到该位置的节点
        while (--place)
        {
            tmp = tmp->next;
        }
        ListInsertByPointer(tmp, value);//找到了该节点,就可以使用根据地址插入的函数插入节点
    }
}
//以下是测试用例
int main()
{
    List<int> list;
    int n,choice, t1, t2, p;
    cout << "请输入初始链表的节点数:" << endl;
    cin >> n;
```

```
    cout << "请输入各个节点的值:" << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> t1;
        list.PushBack(t1);
    }
    while (1)
    {
        cout << "当前的链表是:" << endl;
        list.ListPrintf();
        cout << "请输入要进行的操作:" << endl;
        cout << "1、在链表的最后插入" << endl;
        cout << "2、按照位置在特定节点前插入节点" << endl;
        cout << "3、在含有特定数据的节点前插入节点" << endl;
        cin >> choice;
        if (choice == 1)
        {
            cout << "请输入要插入的节点数据" << endl;
            cin >> t1;
            list.PushBack(t1);
        }
        else if (choice == 2)
        {
            cout << "请输入指定节点的位置和要插入的数据" << endl;
            cin >> p >> t1;
            list.ListInsertByPlace(p, t1);
        }
        else if (choice == 3)
        {
            cout << "请输入指定节点的数据和要插入的数据" << endl;
            cin >> t1 >> t2;
            list.ListInsertByData(t1, t2);
        }
        else
        {
            cout << "输入错误" << endl;
        }
        system("cls");
    }
}
```

五、程序复杂度分析

1、时间复杂度分析：

假设总共有 n 个节点，在程序中主要有三种用户需求操作：

（1）ListInsertByPointer():该函数指定了节点的位置，只需要创建一个新节点并且进行数据的设置即可，所以时间复杂度是 O(1)

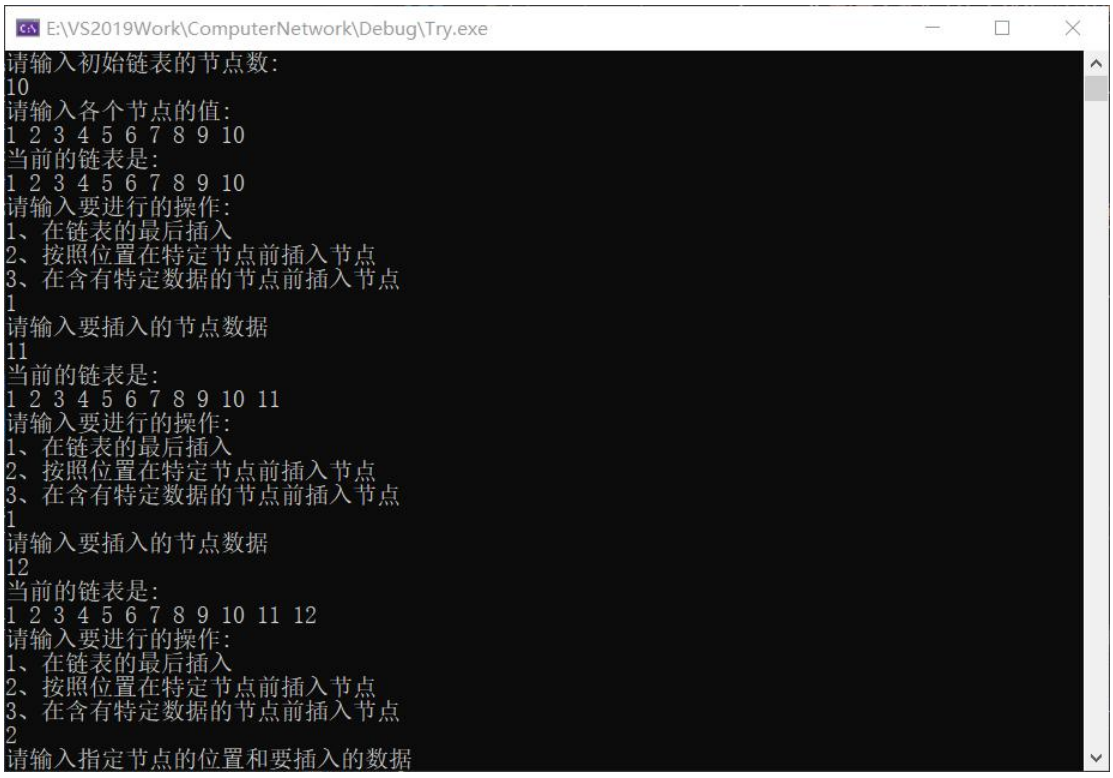
（2）ListInsertByData()&ListInsertByPlace:这两个函数都需要遍历链表查找指定的节点，然后调用 ListInsertByPointer()，查找最坏的时间复杂度是 O(n)，故 O(n)+O(1)=O(n)

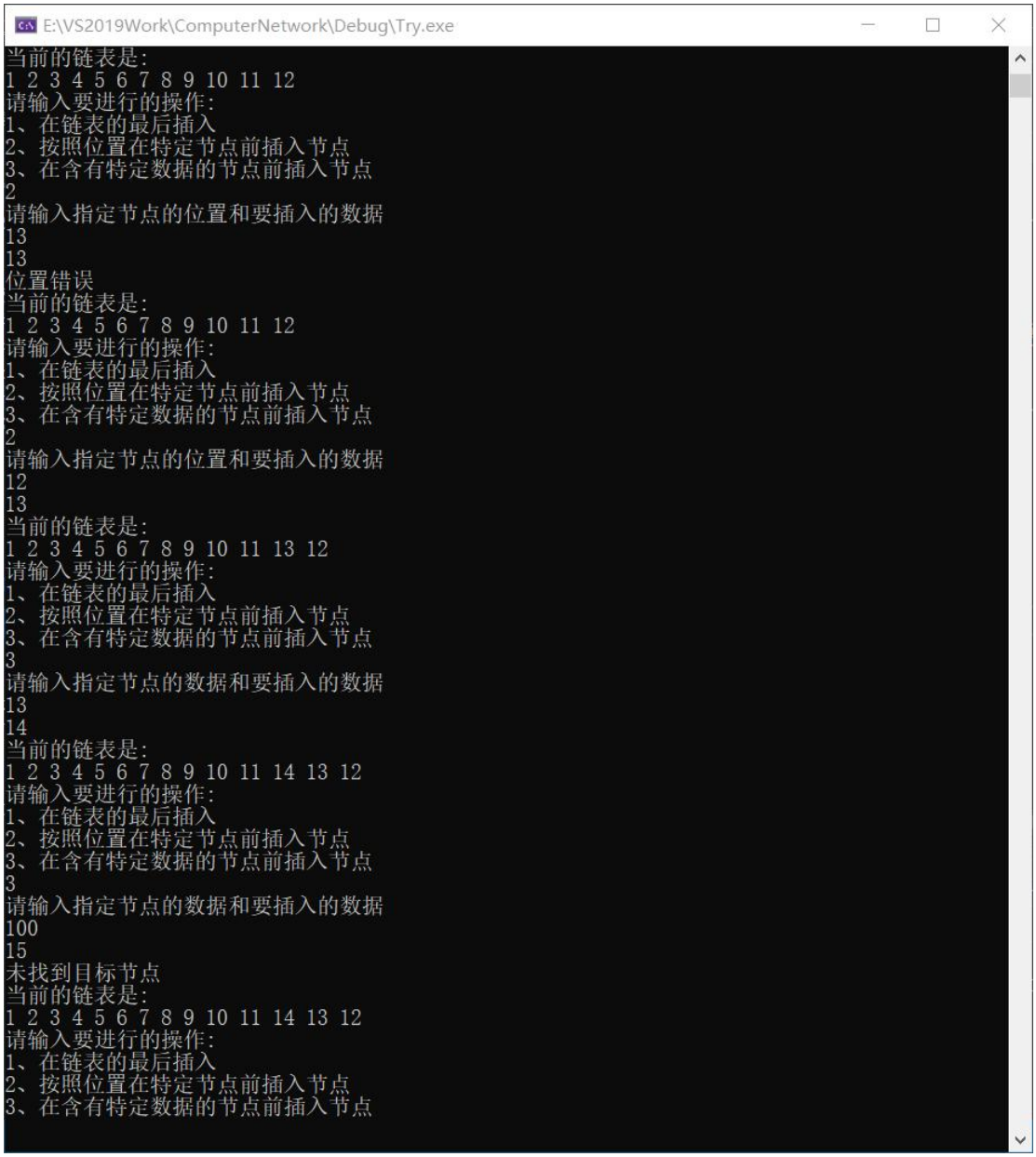
2、空间复杂度分析：

每个节点都要使用相应的空间，其余不需要额外的空间开销，假设总共有 n 个节点，空间复杂度是 O(n)

六、程序运行结果

以下是我进行的测试：
1、首先初始链表为 10 个节点:1 2 3 4 5 6 7 8 9 10
2、然后在最后插入 11 和 12
3、在第 13 个节点前插入 13，提示位置错误
4、在第 12 个节点前插入 13
5、然后在数据为 13 的节点前插入 14
6、最后在数据为 100 的节点前插入 15，提示无该节点




E:\VS2019Work\ComputerNetwork\Debug\Try.exe
当前的链表是:
1 2 3 4 5 6 7 8 9 10 11 12
请输入要进行的操作:
1、在链表的最后插入
2、按照位置在特定节点前插入节点
3、在含有特定数据的节点前插入节点
2
请输入指定节点的位置和要插入的数据
13
13
位置错误
当前的链表是:
1 2 3 4 5 6 7 8 9 10 11 12
请输入要进行的操作:
1、在链表的最后插入
2、按照位置在特定节点前插入节点
3、在含有特定数据的节点前插入节点
2
请输入指定节点的位置和要插入的数据
12
13
当前的链表是:
1 2 3 4 5 6 7 8 9 10 11 13 12
请输入要进行的操作:
1、在链表的最后插入
2、按照位置在特定节点前插入节点
3、在含有特定数据的节点前插入节点
3
请输入指定节点的数据和要插入的数据
13
14
当前的链表是:
1 2 3 4 5 6 7 8 9 10 11 14 13 12
请输入要进行的操作:
1、在链表的最后插入
2、按照位置在特定节点前插入节点
3、在含有特定数据的节点前插入节点
3
请输入指定节点的数据和要插入的数据
100
15
未找到目标节点
当前的链表是:
1 2 3 4 5 6 7 8 9 10 11 14 13 12
请输入要进行的操作:
1、在链表的最后插入
2、按照位置在特定节点前插入节点
3、在含有特定数据的节点前插入节点