Project documentation

Data structure course design

——Intersection of two ordered linked lists

Author name：Xuedi Liu
Number：1752985
instructor：Ying Zhang
College/Major：School of Software Engineering/Software Engineering

1. Analysis

1.1 Project background analysis

For two linked lists, if you want to ask for their intersection, you only need to traverse the linked list and compare its elements. For the situation given in this question: Both linked lists are non-descending linked lists, you can optimize and improve the method of finding the intersection of ordinary linked lists in terms of time complexity and space complexity.

1.2 Project function requirements

The core function of this project needs to analyze the elements of two linked lists, extract the same elements and store them in a new linked list. This linked list is the intersection of the two linked lists we require. In order to make the project more universal and make the project operation safer, this project encapsulates the method of operating linked lists and nodes.

2. Design

2.1 Algorithm design

- Find the intersection of two ordered linked lists, the essence of which is the operation of the linked list.
- The main idea is to use the pointers of the two linked lists to determine whether there are equal nodes in the two ordered linked lists.
- Because this question does not limit the space, for convenience, this program uses the method of constructing a new linked list to store the values of the equal nodes of two linked lists to find the intersection.

2.2 Algorithm Description

- First use header interpolation to construct two ordered linked lists **s1** and **s2** based on two rows of input;
- If **s1** or **s2** is an empty table, it outputs $NULL$ directly.
- Use pointers **p1**, **p2** to **s1** and **s2** to traverse the elements in the linked list;
- If the value of the element pointed to by **p1** is smaller than the value of the element pointed by **p2**, **p2** is shifted back by one;
- If the value of the element pointed to by **p2** is smaller than the value of the element pointed by **p1**, **p1** is shifted back by one; if the value of the element pointed by **p2** is less than the value of element pointed by **p1**, **p1** is moved back by one;
- If the values of the elements pointed by **p1** and **p2** are equal, the element value is added to **s3**; if the values of the elements pointed by **p1** and **p2** are equal, the element value is added to **s3**;
- If **p1** or **p2** is a null pointer, the loop ends;
- Print the values of all elements in **s3**.

2.3 Member and operational design

1. Node

```
class Node {
private:
    int _number;
public:
    Node *pre;
    Node *next;
```

```
    Node();

    Node(int number);

    int getNum();
    void insertAsPre(Node *t);//Insert as a precursor to this node
    void insertAsNext(Node *t);//As a successor to this node
};
```

Core function:

Constructor(Overload):

```
Node::Node() {
    _number = 0;
    this->pre = nullptr;
    this->next = nullptr;
}

Node::Node(int number) {
    this->_number = number;
    this->pre = nullptr;
    this->next = nullptr;
}
```

Insert the node as a precursor to the node:

```
void Node::insertAsPre(Node *t) {
    t->pre = this->pre;
    t->next = this;
    this->pre->next = t;
    this->pre = t;
}
```

2. LinkList

```
class LinkList {
private:
    int _size;
public:
    Node *header;
    Node *trailer;

    LinkList();

    void insertAsLast(Node *t);//将该节点插入在链表的最后
    void traverse();//统计
};
```

Core function:

Constructor:

```
LinkList::LinkList() {
    header = new Node;
    trailer = new Node;
    header->next = trailer;
    header->pre = nullptr;
    trailer->next = nullptr;
    trailer->pre = header;
    _size = 0;
}
```

Insert node at the end of the linked list:

```
void LinkList::insertAsLast(Node *t) {
    trailer->insertAsPre(t);
    this->_size++;
}
```

Traverse the Linklist:

```
void LinkList::traverse() {
    if (_size == 0) {
        std::cout << "NULL";
    } else {
        auto p = this->header->next;
        for (; p != this->trailer; p = p->next) {
            std::cout << p->getNum();
            if (p->next != this->trailer) {
                std::cout << ' ';
            }
        }
    }

}
```

3. Main function

```
int main() {
    int t = 2;//
    int num = 0;//Node data
    Node *tNode;
    auto *s1 = new LinkList;
    auto *s2 = new LinkList;


    while (1) {
        cin >> num;
```

```
        if (num == -1) {
            break;
        } else {
            tNode = new Node(num);
            s1->insertAsLast(tNode);
        }
    }

    while (1) {
        cin >> num;
        if (num == -1) {
            break;
        } else {
            tNode = new Node(num);
            s2->insertAsLast(tNode);
        }
    }

    LinkList *s3 = intersection(s1,s2);

    s3->traverse();
    return 0;
}
```

```
LinkList *intersection(LinkList *s1, LinkList *s2) {
    Node *tNode;
    auto *s3 = new LinkList;
    auto p1 = s1->header->next, p2 = s2->header->next;
    while (p1 != s1->trailer && p2 != s2->trailer) {
        if (p1->getNum() == p2->getNum()) {
            tNode = new Node(p1->getNum());
            s3->insertAsLast(tNode);
            p1 = p1->next;
            p2 = p2->next;
        } else if (p1->getNum() > p2->getNum()) {
            p2 = p2->next;
        } else {
            p1 = p1->next;
        }
    }
    return s3;
}
```

3. Realization and Test

3.1 General condition

There are several equal values in the two sequences

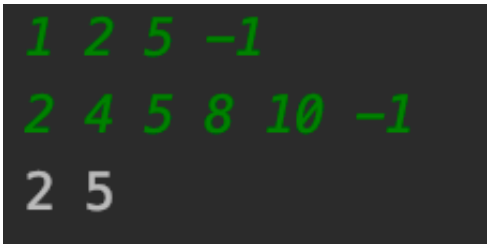
input:

- 1 2 5 -1
- 2 4 5 8 10 -1

Expected outcome:

2 5

Experimental result:



3.2 The intersection is empty

Input an odd sequence, an even sequence, and the intersection of the two sequences is null

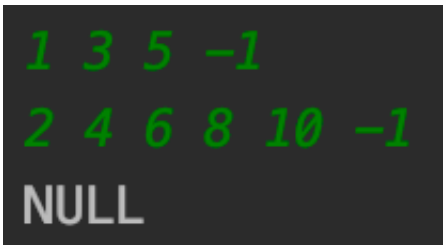
input:

- 1 3 5 -1
- 2 4 6 8 10 -1

Expected outcome:

NULL

Experimental result:



3.3 Completely intersect

The two sequences entered are exactly equal

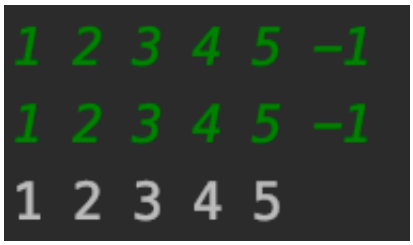
input:

- 1 2 3 4 5 -1
- 1 2 3 4 5 -1

Expected outcome:

1 2 3 4 5

Experimental result:



3.4 One of the sequences is contained

The case where one of the input two sequences completely belongs to the intersection

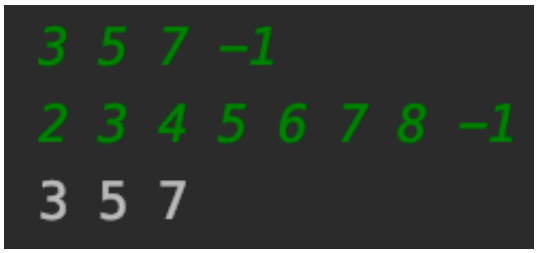
input:

- 3 5 7 -1
- 2 3 4 5 6 7 8 -1

Expected outcome:

3 5 7

Experimental result:



3.5 One of the sequences is empty

One of the two sequences of the input is an empty set

input:

- -1
- 10 100 1000 -1

Expected outcome:

NULL

Experimental result: