Assignment 3 Report

可描述整数连续区间的迭代器的设计

曹旭 大数据学院 16307110230

一、题目描述与分析

题目要求我们设计并实现一个迭代器，该迭代器包含一系列实数区间并迭代由区间覆盖的整数。通过调用迭代器的方法，可以“连接”附加区间（即当前和新区间的并集）。所有区间都是有限的，左闭和右开，两端都是整数（可以是正整数或负数）。题目也给出了运行过程中具体的指令，不过这些部分在自带的 main 函数中已经写好，无需改动。

初看此题，我立马就联想到可以用顺序存储和链式存储两种方法存储“区间”，比较来看，链式存储明显会优于顺序存储。这是因为链式存储可以自由的开辟空间，有利于实现“连接”附加区间的这一题目要求。同时，链式存储只需存储各个并区间首尾的整数即可，大大降低了内存空间利用率，虽然链式存储会损失查找等过程的时间复杂度，但相比起空间复杂度的优化，这点损失微乎甚微。

二、使用链表的解决方法

首先描述一下迭代器的定义：在计算机编程中，迭代器是一个对象，它使程序员能够遍历容器(container)，常用于遍历列表[1]。在 C++中，迭代器类似于 C 语言里面的指针类型，它提供了对对象的间接访问，相当于指针的泛化。

在本题中，我将使用链式存储来存储区间，在原有的基础上新定义一个 Node 结构，包含整型的 left、right 变量和指向下一个结构体的 next 指针。

```
struct Node
{
    int left;
    int right;
    position next;
};
```

当每次需要插入新的区间时，我们先判断链表是否空，如果空，直接在表末插入该区间，如果链表不空，从链表表头开始遍历链表，每遍历到一个链表单元，首先判断下一个单元空不空，如果空，说明已经是链表最末端，此时使用 if-else 语句进一步判断需要插入的结构是直接插在最末，还是与原最后的一个单元合并。

如果下一个单元不空，需判断要插入的区间的 left 是否大于当前单元的 right 并小于下一个单元的 right，若符合这一条件，则进一步比较要插入的区间的 right 和下一个单元的 left，分‘<’，‘>’，‘=’三种情况来讨论，这部分通过链表的 Insert 及 Delete 函数控制即可，参见后文的总结部分。

另一种情况是要插入的区间的 left 等于当前单元的 right 并小于下一个单元的 right，这里的操作同上，不过需要整合重复的区间。

若各个情况都不满足，说明可以插入的位置还没到达，继续向下一个单元移动，重复前面的判断。

综上所述，对于新区间的插入有以下 3 种子情形：

1、 和原有的区间单元都不重复，在链表中插入；

2、 和原有的区间部分重复，改变现有区间的左右数值即可；

3、 跨区间重复，需要使用链表的删除操作删除被覆盖的原区间，往后遍历链表直到找到重复的最后一个区间为止，修改其数值。

除此之外，还有一些小的细节可以提一下，例如：可以在初始化链表的时候在表头后

插入一个最小单元，如右图所示，在头结点后插入了 left=-999999，right=-999998 的结构体，这样就能解决头插的问题（总是可以插在这个结构的后面!）

在 iterate(void *iterator)输出操作时需要在区间左右相等时删除该区间，否则会造成输出错误。

```
void *init()
{
    List L;
    position P;
    position temp;
    L = malloc(sizeof(struct Node));
    L->next = NULL;
    MakeEmpty(L);
    P = L;
    temp = malloc(sizeof(struct Node));
    temp->left = -999999;
    temp->right = -999998;
    temp->next = P->next;
    P->next = temp;
    return L;
}
```

三、总结

由于插入新区间的这一过程是在遍历链表的过程中进行的，相当于一边查找，一边进行链表的修改、插入、删除等操作。每插入一个新区间，该方法的方法最坏时间复杂度为 O(n)（时间复杂度最坏情况：遍历到终点才插入），最坏空间复杂度为 O(1)（空间复杂度最坏情况：需要开辟新空间）

参考文献：

[1] https://en.wikipedia.org/wiki/Iterator