



算法基础 Lab 4

区间树上重叠区间的查找算法

张芷苒

PB21081601

November 1, 2023

Part 1: 实验要求

对红黑树进行修改，使其成为一颗区间数，并实现区间树上的重叠区间查找算法。

Part 2: 区间树的数据结构

区间树的结点 x 包括区间 (interval)、max 值，并且每一个结点的 key 值为 $x.int.low$ 。

```
1 // 定义红黑树的颜色：红或黑
2 enum NodeColour {
3     RED,
4     BLACK
5 };
6
7 // 定义区间结构体，有低位和高位
8 struct IntervalSpan {
9     int lowPoint;
10    int highPoint;
11 };
12
13 // 定义红黑树的节点结构
14 struct TreeNode {
15     NodeColour nodeColour;
16     IntervalSpan span;
17     int keyValue;
18     int maxVal;
19     TreeNode *leftChild, *rightChild, *parent;
20 };
21
22 // 定义红黑树结构
23 struct TreeStructure {
24     TreeNode *treeRoot, *NILNode;
```

```
25 |};
```

Part 3: 算法设计思路

区间树由红黑树扩展而来，二者大部分操作均比较相似。对于本实验而言，主要区别如下：

在树初始化的时候，要将 tree.NILNode 的 max 值赋为负无穷，可以省去很多繁琐的判断。

```
1 void InitializeTree(TreeStructure &tree) {
2     tree.NILNode = new TreeNode;
3     tree.NILNode->nodeColour = BLACK;
4     tree.NILNode->maxValue = INT_MIN;
5     tree.treeRoot = tree.NILNode;
6     tree.treeRoot->parent = tree.NILNode;
7 }
```

每插入一个结点后，该节点会影响从此结点到根结点一条路径上的所有结点的 max 值，所以每插入一个结点都要进行一次自底向上的 max 值更新：

```
1 void AdjustMaxValue(TreeStructure &tree, TreeNode *z) {
2     TreeNode *currentNode = z;
3     while (currentNode != tree.NILNode) {
4         currentNode->maxValue = max(max(currentNode->leftChild->maxValue, currentNode->rightChild->maxValue),
5         currentNode->maxValue);
6     }
7 }
8
9 void InsertNode(TreeStructure &tree, TreeNode *z) {
10     TreeNode *temp1 = tree.NILNode;
11     TreeNode *temp2 = tree.treeRoot;
12
13     while (temp2 != tree.NILNode) {
14         temp1 = temp2;
15         if (z->keyValue < temp2->keyValue) {
16             temp2 = temp2->leftChild;
17         } else {
18             temp2 = temp2->rightChild;
19         }
20     }
```

```

21
22     z->parent = temp1;
23
24     if (temp1 == tree.NILNode) {
25         tree.treeRoot = z;
26     } else if (z->keyValue < temp1->keyValue) {
27         temp1->leftChild = z;
28     } else {
29         temp1->rightChild = z;
30     }
31
32     z->leftChild = tree.NILNode;
33     z->rightChild = tree.NILNode;
34     z->nodeColour = RED;
35
36     AdjustMaxValue(tree, z);
37     InsertFix(tree, z);
38 }

```

左旋和右旋也会改变 max 值，但只会改变左旋结点及其对应的孩子结点的 max，所以只需要对于旋转之后的两个点进行 max 值的更新（先对孩子结点更新，再对父亲结点更新）。代码详见源码，在此不赘述。

重叠区间的查找算法：

区间重叠的判断条件是：对两个闭区间 x, y ，若 $x.low > y.high$ 或 $y.low > x.high$ ，则区间不重叠，否则重叠。

```

1  // 判断两个区间是否有重叠
2  int IsOverlap(IntervalSpan x, IntervalSpan y)
3  {
4      return !(x.highPoint < y.lowPoint || x.lowPoint > y.highPoint);
5  }
6
7  // 查找与查询区间重叠的区间
8  TreeNode *SearchOverlapInterval(TreeStructure &tree, IntervalSpan querySpan)
9  {
10     TreeNode *currentNode = tree.treeRoot;
11     while (currentNode != tree.NILNode && !IsOverlap(querySpan, currentNode
12     ->span)) {
13         if (currentNode->leftChild != tree.NILNode && currentNode->leftChild
14     ->maxValue >= querySpan.lowPoint) {

```

```

15         currentNode = currentNode->leftChild;
16     } else {
17         currentNode = currentNode->rightChild;
18     }
19 }
20 return currentNode;
21 }

```

Part 4: 实验结果分析

运行程序，可以得到以下输出结果：

实际结果：

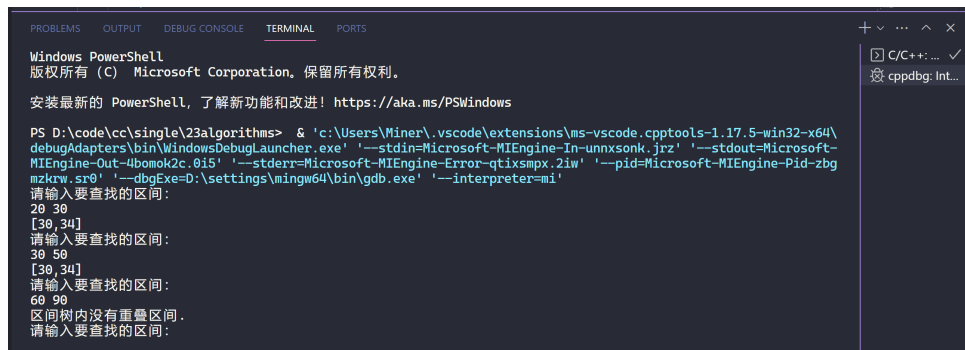
```

lab4 > v1 > LNR.txt
1  [0,1] red
2  [2,17] black
3  [4,15] red
4  [6,15] black
5  [7,14] red
6  [8,13] black
7  [9,18] red
8  [10,17] red
9  [11,22] black
10 [13,18] red
11 [14,22] black
12 [15,16] red
13 [16,25] red
14 [17,18] black
15 [18,23] black
16 [19,24] red
17 [21,24] black
18 [24,25] red
19 [30,34] black
20 [31,40] black
21 [32,43] red
22 [33,37] red
23 [34,43] black
24 [36,42] black
25 [38,43] black
26 [42,50] red
27 [43,45] red
28 [45,46] red
29 [48,49] black
30 [49,50] red
31

```

图 1: 输出结果

进行查询，可以得到以下查询结果：



```
Windows PowerShell
版权所有 (C) Microsoft Corporation. 保留所有权利。

安装最新的 PowerShell, 了解新功能和改进! https://aka.ms/PSWindows

PS D:\code\cc\single\23algorithms> & 'c:\Users\Miner\.vscode\extensions\ms-vscode.cpptools-1.17.5-win32-x64\
debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-unnxsonk.jrz' '--stdout=Microsoft-
MIEngine-Out-4bomok2c.0i5' '--stderr=Microsoft-MIEngine-Error-gtixsmpx.2iw' '--pid=Microsoft-MIEngine-Pid-zbg
mzkrw.sr0' '--dbgExe=D:\settings\mingw64\bin\gdb.exe' '--interpreter=mi'
请输入要查找的区间:
20 30
[30, 34]
请输入要查找的区间:
30 50
[30, 34]
请输入要查找的区间:
60 90
区间树内没有重叠区间。
请输入要查找的区间:
```

图 2: 查询结果

以上结果符合预期, 由此可以认为算法正确。

Part 5 实验总结

在实验过程中获得了以下收获:

- 对于数据结构的扩张——区间树有了更加深刻的理解
- 在扩张的过程中维护扩张的性质需要保证时间复杂度仍为 $O(\log n)$