基于红黑树的区间搜索扩展数据结构

吴杰 3180106238

2019 年 12 月 10 日

1 扩展思路

1.1

选取区间新增树的属性：low，high，max，low 为 BST 树 key 值.

1.2

结点属性 max 作为扩展信息，max 为该所属结点为根的子树中包含的全部子树的区间的并中能包含的最大数值，即

$$x \to max = max\{x \to left \to max, x \to right \to max, x \to high\}$$

1.3

修改 insert 中的 max 属性因改动带来的影响修改左旋、右旋对 max 属性因改动带来的影响.

1.4

区间搜索树的功能实现.

1.5

BinaryTreeNode<type> 对区间的初始化.

2 具体实现

2.1

文件：行数 BinaryTreeNode.h 31

实现功能 增加新的属性：low，high，max，并以 low 作为 key (data) 值.

```cpp
class BinaryTreeNode
{
public:
    //new_____

    TYPE low;
    TYPE high;
    TYPE max;
    int flag=1;

    //new -end_____
    TYPE data;  /**< Satellite data, which is
                 * also a key value. */
    BinaryTreeNode *left;  /**< Left child. */
    BinaryTreeNode *right;  /**< Right child. */
    BinaryTreeNode *parent; /**< Parent. */
    int depth;  /**< Depth of the node. */
    int pos;  /**< Location in the current
                 * layer in the full binary
                 * tree (including the vacant
                 * node).*/
    bool color = RED;
}
```

2.2

文件：行数 BinaryTree.templates.h 34

实现功能 增加新的构造函数，输入区间端点值来构造树以 low 值为 key 增加 max，low，high 属性. 增加区间树定义，新的构造函数，用于输入区间初始化一个节点.

```cpp
template <class TYPE>
BinaryTree<TYPE>::BinaryTree(TYPE low,TYPE high)
{
```

```
 4     nil = new Node;
 5     nil->color = BLACK;
 6     Node *r = new Node;
 7     r->low=low;
 8     r->high=high;
 9     r->max=high;
10     r->data = low;
11     r->left = nil;
12     r->right = nil;
13     r->parent = nil;
14     root = r;
15 };
```

2.3

文件：行数 BinarySearchTree.templates.h 352

实现功能 Insert（node *）增加修改 max 属性的操作.

```
 1  template <class TYPE>
 2  int BinarySearchTree<TYPE>::insert(Node *_new)
 3  {
 4      Node *y = this->nil;
 5      Node *x = this->root;
 6      // Make sure that _new is a single node binary tree.
 7      _new->left = _new->right = this->nil;
 8      while (x != this->nil)
 9      {
10          //new_____
11          x->max=_new->high>x->max?_new->high:x->max;
12          //new -end_____
13      y = x;
14      // The nodes less than to the left, then these greater than or
15      // equal to to the right.
16      if (_new->data < x->data)
17          x = x->left;
18      else
19          x = x->right;
20      }
21      _new->parent = y;
22      if (y == this->nil)
23      this->root = _new;
24      // Here the decision rule has to same as above.
25      else if (_new->data < y->data)
26      y->left = _new;
```

```
27      else
28      y->right = _new;
29      return 0;
30  };
```

2.4

文件：行数 BinaryTree.templates.h 424 && 456

实现功能 修改红黑树左旋右旋的操作，保证节点的 max 在动态操作中保持正确性.

```
1   int BinaryTree<TYPE>::RightRotate(Node *_x)
2   {
3       Node *y = _x->left;
4       _x->left = y->right;
5       if (y->right != nil)
6       y->right->parent = _x;
7       y->parent = _x->parent;
8       if (_x->parent == nil)
9       root = y;
10      // if _x is _x's parent's left child,
11      else if (_x == _x->parent->left)
12      // set y as _x's parent's left child.
13      _x->parent->left = y;
14      else
15      _x->parent->right = y;
16      y->right = _x;
17      _x->parent = y;
18
19
20      //new_____
21      int max1=y->high,max2=_x->high;
22      if(_x->left)max2=max(max2,_x->left->max);
23      if(_x->right)max2=max(max2,_x->right->max);
24      _x->max=max2;
25
26      if(y->left)max1=max(max1,y->left->max);
27      max1=max(max1,_x->max);
28      y->max=max1;
29
30      //new -end_____
31  };
```

4

```
1   template <class TYPE>
2   int BinaryTree<TYPE>::LeftRotate(Node *_x)
3   {
4       Node *y = _x->right;
5       _x->right = y->left;
6       if (y->left != nil)
7       y->left->parent = _x;
8       y->parent = _x->parent;
9       if (_x->parent == nil)
10      root = y;
11      // if _x is _x's parent's left child,
12      else if (_x == _x->parent->left)
13      // set y as _x's parent's left child.
14      _x->parent->left = y;
15      else
16      _x->parent->right = y;
17      y->left = _x;
18      _x->parent = y;
19
20      //new_____
21      int max1=y->high,max2=_x->high;
22      if(_x->left)max2=max(max2,_x->left->max);
23      if(_x->right)max2=max(max2,_x->right->max);
24      _x->max=max2;
25
26      if(y->right)max1=max(max1,y->right->max);
27      max1=max(max1,_x->max);
28      y->max=max1;
29
30      //new -end_____
31  };
```

2.5

文件: 行数   RedBlackTree.h 85

实现功能   增加查找重叠区间树的函数.

```
1   Node *IntervalTreeSearch(Node *source)
2       {
3           Node *t=this->root;
4           while(t && (t->high<source->low || source->high<t->low))
5           {
6               if(t->left && t->left->max>=source->low)
```

```
 7              t=t->left;
 8          else
 9              t=t->right;
10      }
11      return t;
12  }
```

3   程序测试

3.1   编译连接

利用 Makefile 进行编译连接，在终端文件夹位置输入 make 即可编译连接，随后./main 运行程序实现相应功能.

Makefile 文件 ：

```
 1  source = $(wildcard *.cpp)
 2  object = $(patsubst %.cpp, %.o, $(source))
 3
 4  main: $(object)
 5          g++ -o $@ $(object)
 6
 7  %.o : %.cpp
 8          g++ -c $<
 9
10  debug: $(source)
11          g++ -o main $(source) -g
12
13  clean:
14          rm *.o main test *.exe html latex -fr
```

3.2   测试格式

输入格式   第一行输入三个数:n,low,high.low 和 high 为所需要查找的区间的端点值，随后输入 n 行数据，每一行有两个数，为区间树的每个节点的区间端点值.

输出格式   打印一颗红黑树，随后输出所查找到的区间树节点的区间端点值，若无该值，则输出'None'.

测试样例　在 test_cases.dat 中有三组测试样例以及相应的结果。

3.3　测试结果

Case1　：[1]

```
1  input:
2  5 3 5
3  1 2
4  3 4
5  6 7
6  8 9
7  10 14
8  output:
9  3 4
```

Case2　：

```
1   input:
2   6 14 16
3   4 8
4   5 11
5   7 10
6   17 19
7   15 18
8   21 23
9   output:
10  15 18
```

Case3　：

```
1   input:
2   6 12 14
3   4 8
4   5 11
5   7 10
6   17 19
7   15 18
8   21 23
9   output:
10  None
```

---

[1]这里的输出没有显示树的直观表示，而程序中输出存在树的直观显示