基于红黑树的区间搜索扩展数据结构

吴杰 3180106238

2019 年 12 月 11 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.

```
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.

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

```cpp
    nil = new Node;
    nil->color = BLACK;
    Node *r = new Node;
    r->low=low;
    r->high=high;
    r->max=high;
    r->data = low;
    r->left = nil;
    r->right = nil;
    r->parent = nil;
    root = r;
};
```

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.

```cpp
template <class TYPE>
int BinarySearchTree<TYPE>::insert(Node *_new)
{
    Node *y = this->nil;
    Node *x = this->root;
    // Make sure that _new is a single node binary tree.
    _new->left = _new->right = this->nil;
    while (x != this->nil)
    {
        //new_______________________________________________________________
        x->max=_new->high>x->max?_new->high:x->max;
        //new -end_______________________________________________________________
    y = x;
    // The nodes less than to the left, then these greater than or
    // equal to to the right.
    if (_new->data < x->data)
        x = x->left;
    else
        x = x->right;
    }
    _new->parent = y;
    if (y == this->nil)
    this->root = _new;
    // Here the decision rule has to same as above.
    else if (_new->data < y->data)
    y->left = _new;
    else
    y->right = _new;
    return 0;
};
```

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.

```
int BinaryTree<TYPE>::RightRotate(Node *_x)
{
    Node *y = _x->left;
    _x->left = y->right;
    if (y->right != nil)
    y->right->parent = _x;
    y->parent = _x->parent;
    if (_x->parent == nil)
    root = y;
    // if _x is _x's parent's left child,
    else if (_x == _x->parent->left)
    // set y as _x's parent's left child.
    _x->parent->left = y;
    else
    _x->parent->right = y;
    y->right = _x;
    _x->parent = y;


    //new________________________________________________________________
    int max1=y->high,max2=_x->high;
    if(_x->left)max2=max(max2,_x->left->max);
    if(_x->right)max2=max(max2,_x->right->max);
    _x->max=max2;

    if(y->left)max1=max(max1,y->left->max);
    max1=max(max1,_x->max);
    y->max=max1;

    //new -end_______________________________________________________________
};
```

```
template <class TYPE>
int BinaryTree<TYPE>::LeftRotate(Node *_x)
{
   Node *y = _x->right;
   _x->right = y->left;
   if (y->left != nil)
   y->left->parent = _x;
   y->parent = _x->parent;
   if (_x->parent == nil)
   root = y;
   // if _x is _x's parent's left child,
   else if (_x == _x->parent->left)
   // set y as _x's parent's left child.
   _x->parent->left = y;
   else
   _x->parent->right = y;
   y->left = _x;
   _x->parent = y;

   //new_______________________________________________________________
   int max1=y->high,max2=_x->high;
   if(_x->left)max2=max(max2,_x->left->max);
   if(_x->right)max2=max(max2,_x->right->max);
   _x->max=max2;

   if(y->right)max1=max(max1,y->right->max);
   max1=max(max1,_x->max);
   y->max=max1;

   //new -end_______________________________________________________________
};
```

2.5

文件：行数　RedBlackTree.h 85 RedBlackTree.templates.h 192

实现功能　增加查找重叠区间树的函数.

```
Node *IntervalTreeSearch(Node *source);
   /**
    *find a node in this tree
    *this node's interval and source's interval overlap
    */
```

```
template <class TYPE>
typename RedBlackTree<TYPE>::Node* RedBlackTree<TYPE>::IntervalTreeSearch(RedBlackTree
     <TYPE>::Node *source)
{
   RedBlackTree<TYPE>::Node *t=this->root;
   while(t && (t->high<source->low || source->high<t->low))
   {
      if(t->left && t->left->max>=source->low)
         t=t->left;
      else
         t=t->right;
   }
   return t;
};
```

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 :

```
source = $(wildcard *.cpp)
object = $(patsubst %.cpp, %.o, $(source))

main: $(object)
       g++ -o $@ $(object)

%.o : %.cpp
       g++ -c $<

debug: $(source)
       g++ -o main $(source) -g

clean:
       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'.

```cpp
void test_interval_search_tree2()
{
    //test for interval serch trees
    //input and output format are below.
    std::cout<<"Input format:"<<std::endl;
    std::cout<<"The first line includes three numbers:n low high.n is the interval
         tree node you need input,low and high are interval which we want to find if
         there is a node overlap with it.Followed by n lines,each line include two
         numbers which are the interval's endpoints"<<std::endl;
    std::cout<<"Output format:"<<std::endl;
    std::cout<<"Output a interval tree first(based on RBT),then output the interval we
          find or if we can't find it,output 'None'"<<std::endl;
    std::cout<<"Example:"<<std::endl<<"Input:"<<std::endl;
    std::cout<<"5 3 5"<<std::endl;
    std::cout<<"1 2"<<std::endl;
    std::cout<<"3 4"<<std::endl;
    std::cout<<"6 7"<<std::endl;
    std::cout<<"8 9"<<std::endl;
    std::cout<<"10 14"<<std::endl;
    std::cout<<"Output:"<<std::endl<<std::endl;
    test_interval_search_tree1();

    std::cout<<"Now it's your show time:"<<std::endl;

    int n,l,h,temp1,temp2;
    std::vector<std::vector<int> > a;
    std::vector<int> a1;
    std::cin>>n>>l>>h;
    for(int i=0;i<n;i++)
    {
        std::cin>>temp1>>temp2;
        a1.push_back(temp1);
        a1.push_back(temp2);
        a.push_back(a1);
        a1.clear();
    }
    RedBlackTree<int> A;
    for(int i=0;i<a.size();i++)
    {
        BinaryTreeNode<int> *t=new BinaryTreeNode<int>(a[i][0],a[i][1]);
        A.insert(t);
    }
    A.display();
    BinaryTreeNode<int> *temp=new BinaryTreeNode<int>(l,h);
    BinaryTreeNode<int> *res=A.IntervalTreeSearch(temp);

    if(res && (res->low||res->low)){
        std::cout<<res->low<<" ";
        std::cout<<res->high<<std::endl;
    }
    else
        std::cout<<"None"<<std::endl;
}
```

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]

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

Case2　：

```
input:
6 14 16
4 8
5 11
7 10
17 19
```

[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

```
15 18
21 23
output:
15 18
```

Case3 :

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