Red Black Tree

Shashwat Gupta (14IE10028)

March 3, 2016

1 Insertion

The following is the code for insertion of a node into the Red Black tree. The code is similar to insertion in a binary search tree. The insertion code is followed by a insertion fix code which does the required rotations and colour changing to balance and meet the conditions of the red black tree.

The following is the Code for insert

```
node* insert(node* z, node* root)
{
    node *x = (node*)malloc(sizeof(node));
    node *y = (node*)malloc(sizeof(node));
    y=root;
    x=root;
    while(x->extime<=1000)
    {
        y=x;
        if(z->extime<x->extime)
            x=x->lChild;
        else
            x=x->rChild;
    }
    z->parent=y;
    if(y->extime>1000)
        root=z;
    else if (z->extime<y->extime)
        y->lChild=z;
    else
        y->rChild=z;
    z->lChild=&nil;
    z->rChild=&nil;
    z->colour=RED;
    root=insertFix(z,root);
    return root;
}
```

Listing 1: insert

The following is the Code for insertfix

Listing 2: insert fix

2 Rotations

The following are the codes for left and right rotations respectively. These codes are responsible for the changes in the pointer structure as well as the changes in colour. The following is the Code

```
node* leftRotate(node* x, node* root)
{
    node *y = (node*)malloc(sizeof(node));
    y=x->rChild;
    x->rChild=y->lChild;
    if(y->lChild->extime<=1000)
        y->lChild->parent=x;
    y->parent=x->parent;
    if(x->parent->extime>1000)
        root=y;
    else if(x==x->parent->lChild)
        x->parent->lChild=y;
    else
        x->parent->rChild=y;
    y->lChild=x;
    x->parent=y;
    return root;
}
```

Listing 3: left

The following is the Code

```
node* rightRotate(node* x, node* root)
{
    node *y = (node*)malloc(sizeof(node));
    y=x->lChild;
    x->lChild=y->rChild;
    if(y->rChild->extime<=1000)
        y->rChild->parent=x;
    y->parent=x->parent;
    if(x->parent->extime>1000)
        root=y;
    else if(x==x->parent->lChild)
        x->parent->lChild=y;
    else
        x->parent->rChild=y;
    y->rChild=x;
    x->parent=y;
    return root;
}
```

Listing 4: Right

3 Deletion

The deletion Subroutine takes in the node to be deleted and the root of the tree and removes the node from the tree. The delete along with the delete fix functions are responsible for removing the node from the tree and the rotations and colour changing operations.

The following is the Code

```
node* delete(node* z, node* root)
{
    node *x, *y;
    if(z->lChild->extime>1000||z->rChild->extime>1000)
        y=z;
    else
        y=treeSuccessor(z);
    if(y->lChild->extime<=1000)
        x=y->lChild;
    else
        x=y->rChild;
    x->parent=y->parent;
    if(y->parent->extime>1000)
        root=x;
    else if(y==y->parent->lChild)
        y->parent->lChild=x;
    else
        y->parent->rChild=x;
    if(y!=z)
    {
        z->extime=y->extime;
        z->id=y->id;
        z->priority=y->priority;
        z->colour=y->colour;
    }
    if(y->colour==BLACK)
        deleteFix(x,root);
    return y;
}
```

Listing 5: delete

The following is the Code

```
node* deleteFix(node *x, node *root)
{
    node *w;
    while((x->extime<=1000)&&(x->colour==BLACK))
    {
        if(x==x->parent->lChild)
        {
            w=x->parent->rChild;
            if(w->colour==RED)
            {
                w->colour=BLACK;
                x->parent->colour=RED;
                root=leftRotate(x->parent,root);
                w=x->parent->rChild;
            }
            if( w->lChild->colour==BLACK&&w->rChild->colour==BLACK)
            {
                w->colour=RED;
                x=x->parent;
            }
```

```
            else
            {
                if(w->rChild->colour==BLACK)
                {
                    w->lChild->colour=BLACK;
                    w->colour=RED;
                    root=rightRotate(w,root);
                    w=x->parent->rChild;
                }
                w->colour=x->parent->colour;
                x->parent->colour=BLACK;
                w->rChild->colour=BLACK;
                root=leftRotate(x->parent,root);
                return x;
            }
        }
        else
        {
            w=x->parent->lChild;
            if(w->colour==RED)
            {
                w->colour=BLACK;
                x->parent->colour=RED;
                root=rightRotate(x->parent,root);
                w=x->parent->lChild;
            }
            if( w->rChild->colour==BLACK&&w->lChild->colour==BLACK)
            {
                w->colour=RED;
                x=x->parent;
            }
            else
            {
                if(w->lChild->colour==BLACK)
                {
                    w->rChild->colour=BLACK;
                    w->colour=RED;
                    root=leftRotate(w,root);
                    w=x->parent->lChild;
                }
                w->colour=x->parent->colour;
                x->parent->colour=BLACK;
                w->lChild->colour=BLACK;
                root=rightRotate(x->parent,root);
                return x;
            }
        }
    }
    return root;
}
```

Listing 6: deleteFix

4 Process management

The folowwing are the codes for process create and schedule which are responsible for the checking of N and creation of new nodes and processing inserted nodes.

The following is the Code

```
node* processCreate(int liveproc, node* root)
{
    node *new = (node*)malloc(sizeof(node));
    new->id=liveproc;
    new->extime=rand()%1000+1;
    new->priority=rand()%4+1;
    new->colour=RED;
    new->lChild=&nil;
    new->rChild=&nil;
    new->parent=&nil;
    root=insert(new,root);
    return root;
}
```

Listing 7: process create

The following is the Code

```
void processSchedule(node *root)
{
    node *x;
    x=treeMin(root);
    delete(x,root);
    x->extime=x->extime-(50*x->priority);
    insert(x,root);
}
```

Listing 8: process schedule

5 Time Complexity Calculation

Time complexity has the recursive relation:

$$T(n) = T(n/2) + \theta(1)$$

where n is the number of nodes.

Solving this equation by recursive method yields T(n) $\in O(logn)$.

The equation holds for the Insertion aswell as the Deletion subroutine