# 《数据结构》课程作业

The Art of Compression

## 一、问题分析

本次任务旨在用色块状效果实现原始图片。处理后的图像会牺牲原始图像中不包含太多颜色变化的矩形中的颜色细节，但会在包含大量变化的原始图像区域中使用较小的矩形来保留细节。

图 1 原始图片与处理后预期图片

在指定算法时，题中做如下常规假设，

- 原点位置(0,0)位于图像的左上角；
- 矩形由其左上角和右下角的一对点指定；
- 矩形的左上角是距离原点最近的那个。 图像位置通常指定为（x，y），其中 x 是水平偏移量，y 是垂直偏移量。

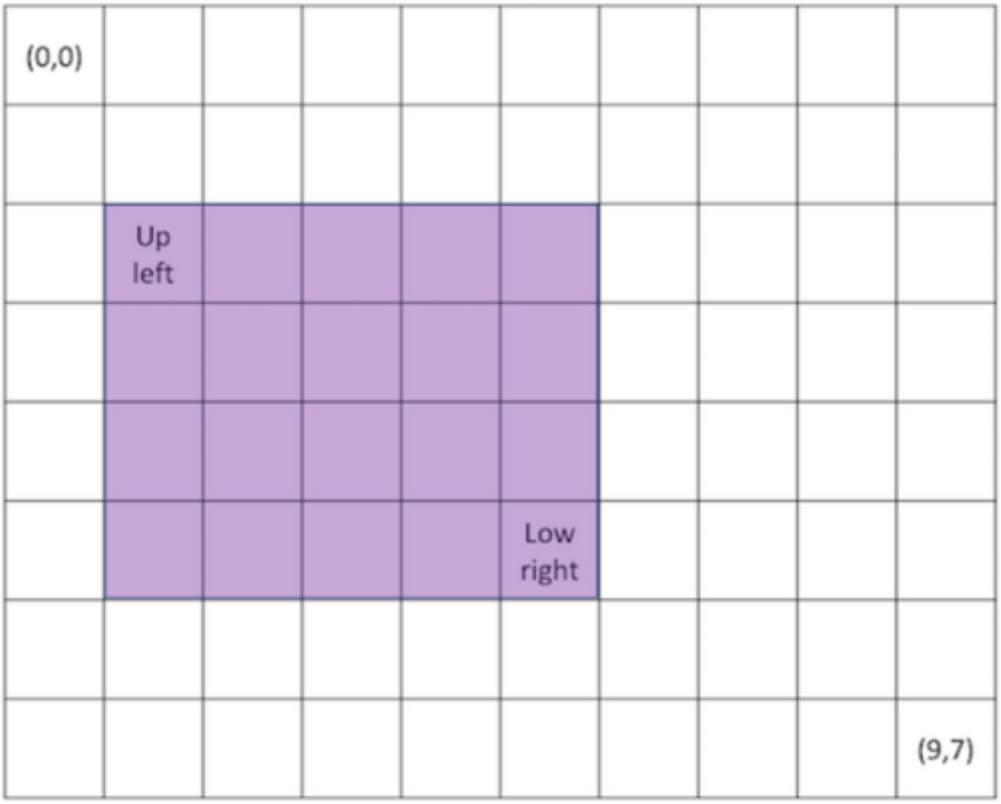


图 2 矩形的 ul 与 lr

原始图像将在内存中表示为二叉树，其节点包含有关矩形的信息，

- 每个节点都包含指定矩形的左上角和右下角点；
- 每个节点还包含一个像素，代表原始图像中矩形上的平均颜色；
- 当一个矩形分成两个较小的矩形时，父节点包含整个矩形，左边的孩子包含父母的左上角和一个新的右下角，右边的孩子包含一个新的左上角和父母的右下角；
- 矩形可以水平或垂直分割。

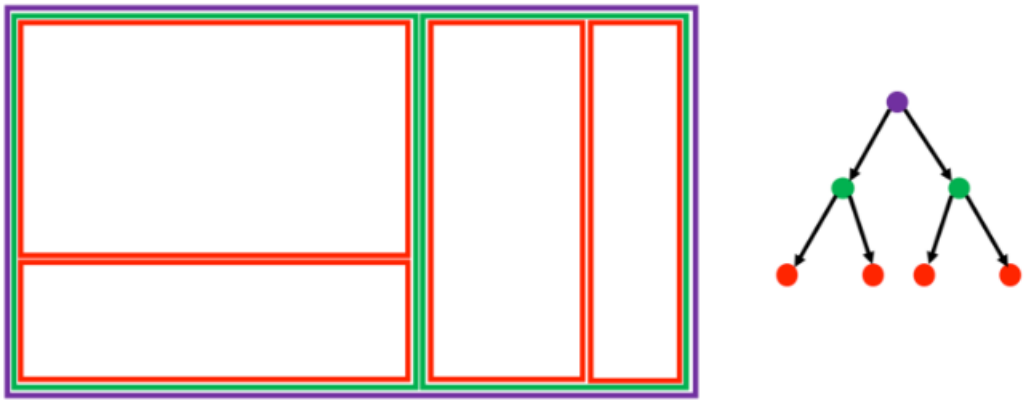

图 3 矩阵切割示意图

原始图像的储存形式为二叉树，包含所有的节点信息。此外，在切割时应尽量减少所得矩形的颜色变化。题中使用的可变性测量值是所有像素与矩形平均值的平方差的总和，即针对红色，绿色和蓝色的颜色通道独立计算，然后将它们相加以得到总差异性评分。

## 二、解决方案

查看代码包中的文件，题中已给出 PNG.cpp，RGBAPiexl.cpp 等文件，其中我们需要修改的主要是在 twoDtree.cpp，stats.cpp 文件中。

### (一) twoDtree 类

查看 twoDtree 代码中的内容，其中节点 Node 的构造函数已给出，分别定义右上和左下节点以及像素的平均值 Avg；并且已经给出析构函数以及“=”运算符的重载函数。我们需要补充以下部分，

1. 函数 twoDtree(PNG & imIn)

我们需要用此函数来确定“画布”的大小，图像左上角为图片的原点(0,0)位置，高度与宽度由原图像宽高确定。

2. 函数 buildTree

利用队列实现树的节点的切割。分别定义切割后右子树的左上角和左子树的右下角，并记录每次分割的右下角和左上角。可计算两个三角形得分之和，利用遍历方式找出最佳切割点。

3.函数 render

获得最后剪切获得的树，得到相应的像素值。

4.函数 prune

在修剪或切断二叉树的一部分时，题中设置两个参数（百分比和容差）用于评估子树适用于修剪的情况。 要修剪一个节点，我们只需删除它的左右子树，如果其子树中至少（大于或等于）百分比的树叶在其平均值的容限内（小于或等于），则修剪节点。颜色之间的距离被计算为每个颜色通道上像素值差的平方和。

5.函数 copy

此处我们遍历所有的节点，将其左右孩子传递给*this。

6.函数 clear

此处遍历所有的节点，然后删除 root。

**(二)stats 类**

stats 类仅用来构建 twoDtree。题中提醒使用颜色通道的差异评分值来衡量矩形的切割情况，所以在本类中需要计算所有颜色通道像素值的加和：（x，y）是从（0,0）到（x，y）的颜色像素值的累积和。

类似地，sumSq 向量是从（0,0）到（x，y）的累积平方和。统计矩形中像素的个数以及其偏差的平方和，从而在构建树时用于分割。

## 三、算法设计（流程图）

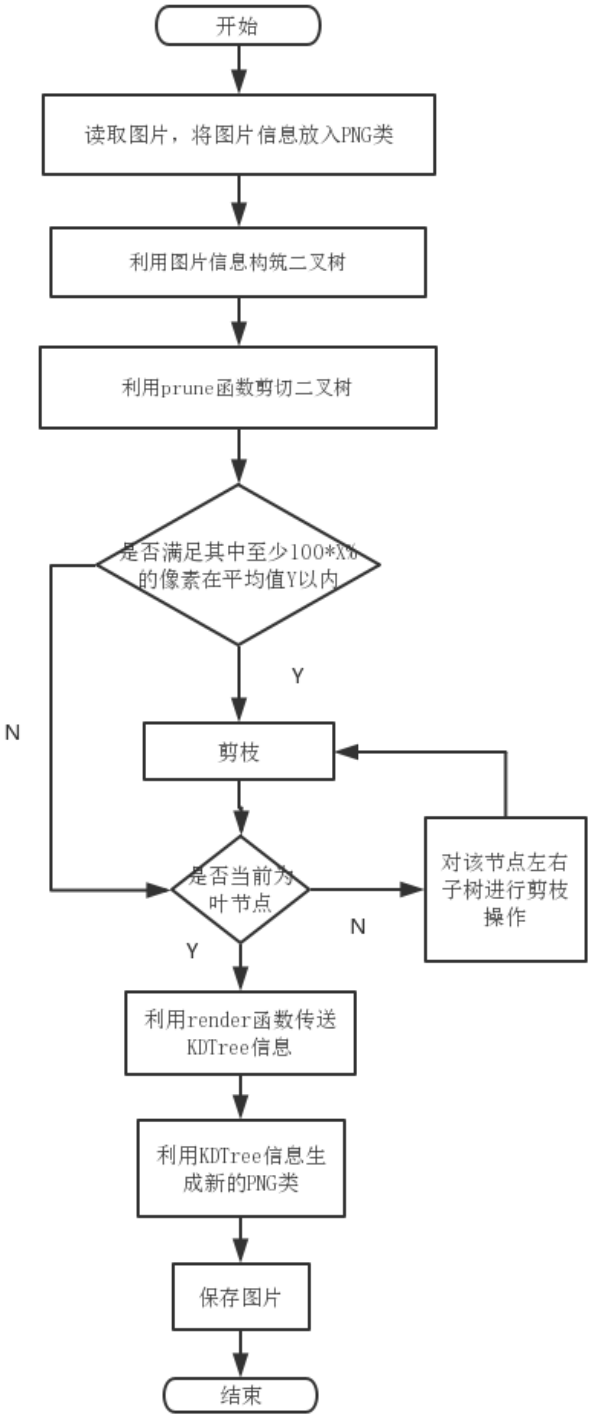


图 4 程序总体思路流程图

## 四、编程实现

（节选 twoDtree 文件的主要代码如下）

twoDtree::twoDtree(PNG & imIn)

```
{
    /* your code here */
    stats s(imIn);
    pair<int ,int> ul(0,0);
    pair<int,int> lr(imIn.width()-1,imIn.height()-1);
    height = imIn.height();
    width =imIn.width();
    root = buildTree(s,ul,lr);
}
twoDtree::Node * twoDtree::buildTree(stats & s, pair<int,int> ul, pair<int,int>
lr) {
    /* your code here */
    Node *temproot;
//定义临时节点指针
    Node *result;
//记录头节点指针的位置
    queue<Node*> t;
    Node* newroot = new Node(ul,lr,s.getAvg(ul,lr));
//新建根节点
    temproot = newroot;
    result = temproot;

    t.push(temproot);
```

```
    int count=0;
//设置一个计数器，若为偶数则水平切割，若为奇数则竖直切割
    while(!t.empty())
//利用队列进行层次建树
        {
            temproot = t.front();
            t.pop();
            pair<int,int>tul(temproot->upLeft);
            pair<int,int>tlr(temproot->lowRight);
            pair<int,int>r_ul(0,0);
//定义切割后右子树的左上角
            pair<int,int>r_lr(0,0);

//记录当前分割的矩形
        if((tlr.first-tul.first<2
            )||(tlr.second-tul.second<2))
//若矩阵规模小于 n*n 则不再分割
            continue;
        else
        {
            long long score=-1;
//定义切割后左子树的右下角
            if(count%2==0)
//水平切割
            {
```

```
                for(int i=tul.second+1;i<tlr.second;i++)
                {
                    pair<int,int>temp1(lr.first,i);
//记录每次分割的右下角和左上角
                    pair<int,int>temp2(ul.first,i);
                    long long temp =
s.getScore(tul,temp1)+s.getScore(temp2,tlr);
//计算两个三角形得分之和
                    if(temp>score)
//利用遍历方式找出最佳切割点
                    {
                        r_ul.first = tul.first; r_ul.second = i;
                        r_lr.first = tlr.first; r_lr.second = i;
                        score = temp;
                    }
                    //cout<<i<<endl;
                }
            }
            else
//竖直切割，思路同水平切割
            {

                for(int i=tul.first+1;i<tlr.first;i++)
                {
```

```
                    pair<int,int>temp1(i,lr.second);
                    pair<int,int>temp2(i,ul.second);
                    long long temp =
s.getScore(tul,temp1)+s.getScore(temp2,tlr);
                    if(temp>score)
                    {
                        r_ul.first = i; r_ul.second = tul.second;
                        r_lr.first = i; r_lr.second = tlr.second;
                        score = temp;
                    }
                }
            }
            count++;
//下一次切割换另一个维度
            Node *l_chirdnode=new
Node(temproot->upLeft,r_lr,s.getAvg(temproot->upLeft,r_lr));
            Node *r_chirdnode=new
Node(r_ul,temproot->lowRight,s.getAvg(r_ul,temproot->lowRight));
            temproot->right = r_chirdnode;
            temproot->left  = l_chirdnode;
            t.push(temproot->left);
            t.push(temproot->right);
```

```
            }
        }
        cout<<"切割次数 "<<count<<endl;
        return result;
}
PNG twoDtree::render(){
    /* your code here */
     PNG Newpng(width,height);
     Node *x = root;
     stack<Node*> t;
     t.push(root);
     while(!t.empty())
     {
          x=t.top();t.pop();
          if(x->right)
          {
               t.push(x->right);
          }
          if(x->left)
          {
               t.push(x->left);
          }
```

```
            if(x->right==NULL&&x->left==NULL)
            {
                RGBAPixel *tm;
                for(int i = x->upLeft.first;i<=x->lowRight.first;i++)
                {
                    for(int j = x->upLeft.second;j<=x->lowRight.second;j++)
                    {
                        tm = Newpng.getPixel(i,j);
                        *tm = x->avg;
                    }
                }
            }
        }
        cout<<"render complete"<<endl;
        return Newpng;
}
void twoDtree::prune(double pct, int tol){
    /* your code here */
    stack<Node*> t;
    Node *tre1 = root;
    t.push(root);
    while(!t.empty())
```

```
    {
        tre1 = t.top();t.pop();
        if(tre1->right)
        {
            t.push(tre1->right);
        }
        if(tre1->left)
        {
            t.push(tre1->left);
        }
        stack <Node*> s;
        int w_count=0,r_count=0;
        s.push(tre1);
        Node *tre2 = tre1;
        int ave =
(tre1->avg.a)*(tre1->avg.a)+(tre1->avg.b)*(tre1->avg.b)+(tre1->avg.g)*(tre1
->avg.g);
        while(!s.empty())
        {
            tre2 = s.top();s.pop();
            if(tre2->right)
            {
```

```
                s.push(tre2->right);
            }
            if(tre2->left)
            {
                s.push(tre2->left);
            }
            if(tre2->left==NULL&&tre2->right==NULL)
            {
                unsigned char r=tre2->avg.r;
                unsigned char g=tre2->avg.g;
                unsigned char b=tre2->avg.b;
                double temp = r*r+g*g+b*b;
                if ((temp>=ave-tol)&&(temp<=ave+tol))
                    r_count++;
                else
                    w_count++;
            }
        }
    }
}

void twoDtree::clear() {
```

```
    /* your code here */
        if(!root)return ;
        else
        {
            stack<Node*> t;
            t.push(root);
            while(!t.empty())
            {
                root = t.top();t.pop();
                if(root->right)
                    t.push(root->right);
                if(root->left)
                    t.push(root->left);
                delete root;

            }
        }
}

void twoDtree::copy(const twoDtree & orig){
    /* your code here */
     Node *tre2 = orig.root;
```

```
    Node* newroot = new Node(tre2->upLeft,tre2->lowRight,tre2->avg);
    root = newroot;
    Node *tre1 = root;
    if(tre2->left==NULL&&tre2->right==NULL)
    {
            Node* newnode = new
Node(tre2->upLeft,tre2->lowRight,tre2->avg);
            tre1 = newnode;
    }
    else
    {
            stack<Node*> s1;
            stack<Node*> s2;
            s1.push(tre1);s2.push(tre2);
            while(!s2.empty())
            {
                    tre1 = s1.top();s1.pop();
                    tre2 = s2.top();s2.pop();
                    if(tre2->right)
                    {
                            Node* r_node = new
Node(tre2->right->upLeft,tre2->right->lowRight,tre2->right->avg);
```

```
                    tre1->right= r_node;
                    s1.push(tre1->right);s2.push(tre2->right);
                }
                else
                {
                    tre1->right = NULL;
                }
                if(tre2->left)
                {
                    Node* l_node = new
Node(tre2->left->upLeft,tre2->left->lowRight,tre2->left->avg);
                    tre1->left= l_node;
                    s1.push(tre1->left);s2.push(tre2->left);
                }
                else
                {
                    tre1->left = NULL;
                }
            }
    }
}
```

（stats 文件的主要代码如下）

```
long long stats::getSum(char channel, pair<int,int> ul, pair<int,int> lr)
{
       long long int temp = 0;
        switch(channel)
        {
            case 'r':
                temp = sumRed[lr.first][lr.second] + sumRed[ul.first][ul.second]- sumRed[lr.first][ul.second] - sumRed[ul.first][lr.second];
                break;
            case 'g':
                temp = sumGreen[lr.first][lr.second] + sumGreen[ul.first][ul.second]- sumGreen[lr.first][ul.second] - sumGreen[ul.first][lr.second];
                break;
            case 'b':
                temp = sumBlue[lr.first][lr.second] + sumBlue[ul.first][ul.second]- sumBlue[lr.first][ul.second] - sumBlue[ul.first][lr.second];
                break;
        }
        return temp;
}
long long stats::getSumSq(char channel, pair<int,int> ul, pair<int,int> lr)
{
      long long int temp = 0;
        switch(channel)
        {
            case 'r':
                temp = sumsqRed[lr.first][lr.second] + sumsqRed[ul.first][ul.second]- sumsqRed[lr.first][ul.second] - sumsqRed[ul.first][lr.second];
                    break;
            case 'g':
                temp = sumsqGreen[lr.first][lr.second] + sumsqGreen[ul.first][ul.second]- sumsqGreen[lr.first][ul.second] - sumsqGreen[ul.first][lr.s
                    break;
            case 'b':
                temp = sumsqBlue[lr.first][lr.second] + sumsqBlue[ul.first][ul.second]- sumsqBlue[lr.first][ul.second] - sumsqBlue[ul.first][lr.secon
                    break;
        }
        return temp;
```

```
stats::stats(PNG & im)
{
    for(int i=0;i<im.width();i++)
    {
      sumRed.push_back(vector<long long> (im.height(),0));
      sumGreen.push_back(vector<long long> (im.height(),0));
      sumBlue.push_back(vector<long long> (im.height(),0));
      sumsqRed.push_back(vector<long long> (im.height(),0));
      sumsqGreen.push_back(vector<long long> (im.height(),0));
      sumsqBlue.push_back(vector<long long> (im.height(),0));
    }
    sumRed[0][0] = im.getPixel(0,0)->r;
    sumGreen[0][0] = im.getPixel(0,0)->g;
    sumBlue[0][0] = im.getPixel(0,0)->b;
    for(int i= 1;i<im.width();i++)
    {
        sumRed[i][0]=sumRed[i-1][0]+im.getPixel(i,0)->r;
        sumGreen[i][0]=sumGreen[i-1][0]+im.getPixel(i,0)->g;
        sumBlue[i][0] = sumBlue[i-1][0]+im.getPixel(i,0)->b;
    }
    for(int i=1;i<im.height();i++)
    {
        sumRed[0][i]=sumRed[0][i-1]+im.getPixel(0,i)->r;
        sumGreen[0][i]=sumGreen[0][i-1]+im.getPixel(0,i)->g;
        sumBlue[0][i] = sumBlue[0][i-1]+im.getPixel(0,i)->b;
    }
    for(int i = 1;i<im.width();i++)
        for(int j=1;j<im.height();j++)
        {
            sumRed[i][j] = im.getPixel(i,j)->r+sumRed[i-1][j]+sumRed[i][j-1]-sumRed[i-1][j-1];
            sumGreen[i][j] = im.getPixel(i,j)->g+sumGreen[i-1][j]+sumGreen[i][j-1]-sumGreen[i-1][j-1];
            sumBlue[i][j] = im.getPixel(i,j)->b+sumBlue[i-1][j]+sumBlue[i][j-1]-sumBlue[i-1][j-1];
        }
```

```
        sumsqRed[0][0] = (im.getPixel(0,0)->r)*(im.getPixel(0,0)->r);
        sumsqGreen[0][0] = (im.getPixel(0,0)->g)*(im.getPixel(0,0)->g);
        sumsqBlue[0][0] = (im.getPixel(0,0)->b)*(im.getPixel(0,0)->b);
        for(int i= 1;i<im.width();i++)
        {
            sumsqRed[i][0]=sumsqRed[i-1][0]+(im.getPixel(i,0)->r)*(im.getPixel(i,0)->r);
            sumsqGreen[i][0]=sumsqGreen[i-1][0]+(im.getPixel(i,0)->g)*(im.getPixel(i,0)->g);
            sumsqBlue[i][0] = sumsqBlue[i-1][0]+(im.getPixel(i,0)->b)*(im.getPixel(i,0)->b);
        }
        for(int i=1;i<im.height();i++)
        {
            sumsqRed[0][i]=sumsqRed[0][i-1]+(im.getPixel(0,i)->r)*(im.getPixel(0,i)->r);
            sumsqGreen[0][i]=sumsqGreen[0][i-1]+(im.getPixel(0,i)->g)*(im.getPixel(0,i)->g);
            sumsqBlue[0][i] = sumsqBlue[0][i-1]+(im.getPixel(0,i)->b)*(im.getPixel(0,i)->b);
        }
        for(int i = 1;i<im.width();i++)
            for(int j=1;j<im.height();j++)
            {
                sumsqRed[i][j] = (im.getPixel(i,j)->r)*(im.getPixel(i,j)->r)+sumsqRed[i-1][j]+sumsqRed[i][j-1]-sumsqRed[i-1][j-1];
                sumsqGreen[i][j] = (im.getPixel(i,j)->g)*(im.getPixel(i,j)->g)+sumsqGreen[i-1][j]+sumsqGreen[i][j-1]-sumsqGreen[i-1][j-1];
                sumsqBlue[i][j] = (im.getPixel(i,j)->b)*(im.getPixel(i,j)->b)+sumsqBlue[i-1][j]+sumsqBlue[i][j-1]-sumsqBlue[i-1][j-1];
            }
}
long long stats::getScore(pair<int,int> ul, pair<int,int> lr)
{
        long long Score = 0;// 计算面积时边长+1
        long long area = (ul.first-lr.first)*(lr.second-ul.second);
        Score += getSumSq('r',ul,lr) - getSum('r',ul,lr)*getSum('r',ul,lr)/area;
        Score += getSumSq('g',ul,lr) - getSum('g',ul,lr)*getSum('g',ul,lr)/area;
        Score += getSumSq('b',ul,lr) - getSum('b',ul,lr)*getSum('b',ul,lr)/area;
        //cout<<"score "<<score<<endl;
        return Score;
}

RGBAPixel stats::getAvg(pair<int,int> ul, pair<int,int> lr)
{
        long long area = (lr.first-ul.first+1)*(lr.second-ul.second+1);
        unsigned char r;
        unsigned char g;
        unsigned char b;

        r = getSum('r',ul,lr)/area;
        g = getSum('g',ul,lr)/area;
        b = getSum('b',ul,lr)/area;
        RGBAPixel temp(r,g,b);
        return temp;
}
long long stats::rectArea(pair<int,int> ul, pair<int,int> lr)
{
        long long area = (ul.first-lr.first+1)*(lr.second-ul.second+1);
        return area;
}
```

## 五、结果分析

最初在 CodeBlocks 编译时 hash 报错，后更换编译平台仍未能得到题目预设结果。

## 六、总结体会

1. codeblocks 中 hash 会编译报错“hash is not a member of ‘std’”，需更换编译平台；

2. 英语水平的短板越发显著。在翻译原题目文档时遇到了各种断句理解方面的问题；

3.在建树以及对树进行先序遍历时，使用栈的形式实现可以大大降低代码的时间复杂度；

4.在前期尝试过程中，我们不止一次的遇到“terminate called after throwing an instance of ‘std::bad_alloc’ what(): St9bad_alloc”内存泄漏问题。