实验二报告

程序语言：c++

姓名：郑子睿

学号：22920212204317

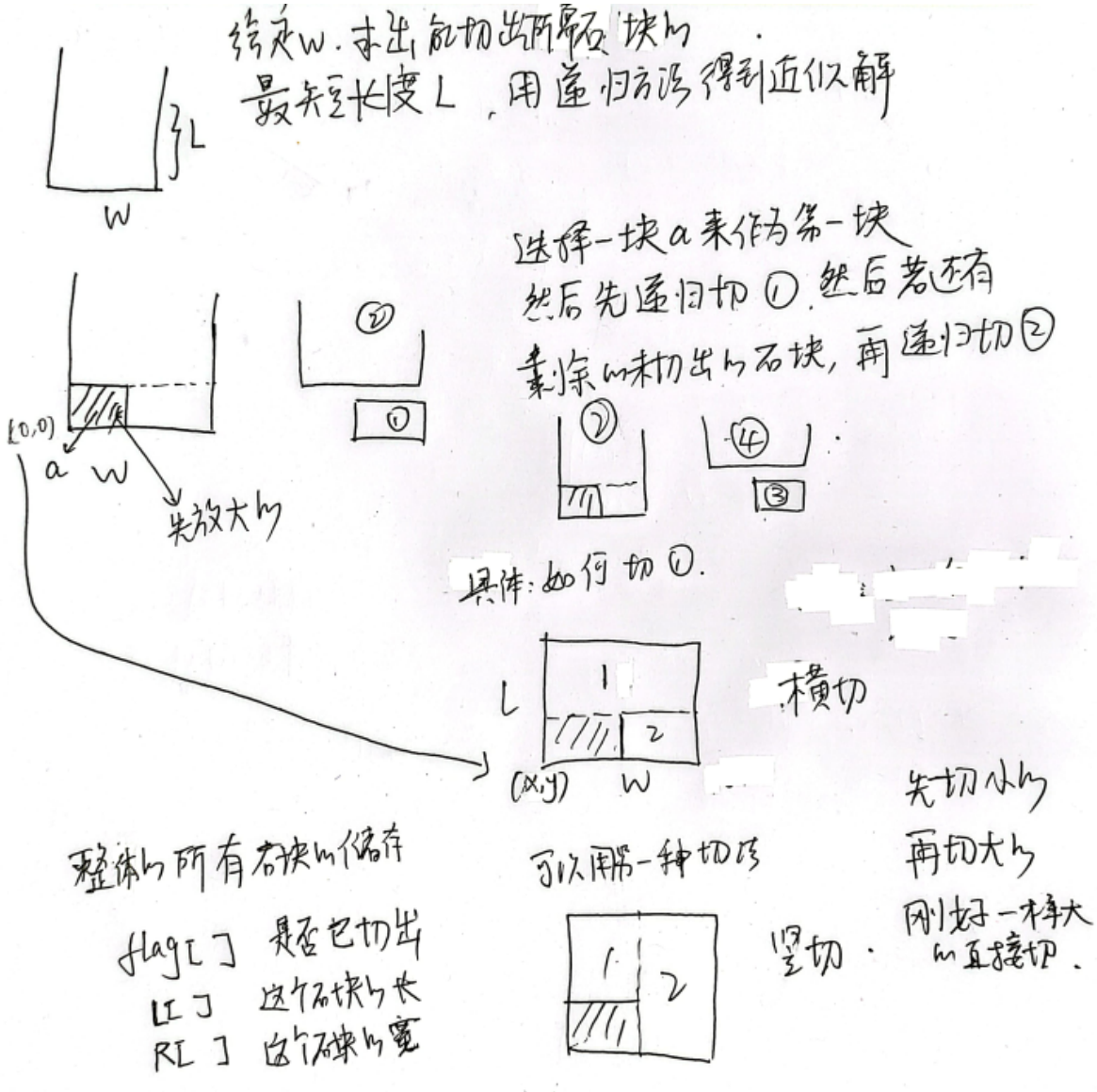
专业：计算机科学与技术

一、问题描述

一块石板，宽度固定为W，给定一些石块的宽和高，求能切出所给石块需要的需要消耗石板的最小高度H，用递归方法得到近似解

二、算法思想

本实验的重点在于如何设计递归方案。



结合上图，具体的算法如下：

1. 将所有石块以高度为第一关键字、面积为第二关键字排序。
2. 切割策略是每次优先切出高度最大的石块，采用先横切再竖切的方法。
3. 第2步将石板切成了一块高度无限的石板和一块高度固定的石板。
4. 先切高度固定的石板，同理优先切出可行的高度最大的石块，也是先横切再竖切，切出两块高度固定的石板。这两块按第4步同样的方式处理。
5. 再按第2步的方式切另一块高度无限的石板。
6. 如此进行，直到所有石块被切出。
7. 将石块切出的位置用矩形可视化，并计算利用率。

三、描述算法

```
// 切高度无限的石板，（x，y）为石板左下角的坐标
void cutInfStrip(double x, double y)
{
    // 如果已经没有待切的石块，返回
    if 已经没有待切的石块 then return;

    // 从前往后遍历待切的石块，找到第一个宽度比W小的石块，将它切出
    for (auto i : candidateRocks)
    {
        double wi = rocks[i].width, hi = rocks[i].height;
        if (rocks[i].width <= W) // 找到了符合条件的石块
        {
            order.push_back(i); // 将该石块加入切出次序
            candidateRocks.erase(i); // 在待切出石块中将它删除
            rocks[i].x = x, rocks[i].y = y + hi; // 计算出该石块的坐标
            maxH = max(maxH, rocks[i].y); // 更新最大利用高度
            // 分成切一个高度有限的石板和一个高度无限的石板两个子问题
            cutLimitStrip(x + wi, y, W - wi, hi);  // 先切有限制的石板
            cutInfStrip(x, y + hi); // 再切无限制的石板
            break;
        }
    }
}
```

```
// 切高度有限的石板（x，y）为石板左下角的坐标，w是石板宽度，h是石板高度
void cutLimitStrip(double x, double y, double w, double h)
{
    // 如果已经没有待切的石块，返回
    if (candidateRocks.empty()) return;
    // 如果该石板不合法，返回
    if (w <= 0 || h <= 0) return;

    // 从前往后遍历待切的石块，找到第一个可以切出的石块，将它切出
    for (auto i : candidateRocks)
    {
        double wi = rocks[i].width, hi = rocks[i].height;
        if (w >= wi && h >= hi)
        {
            order.push_back(i);
            candidateRocks.erase(i);

            rocks[i].x = x, rocks[i].y = y + hi;
            // 分成切两个高度有限的石块（两个子问题）
            cutLimitStrip(x + wi, y, w - wi, hi);
```

```
            cutLimitStrip(x, y + hi, w, h - hi);
            break;
        }
    }
}
```

四、验证算法

具体代码

1. 石块的数据结构

```
// 每个石块的结构体
struct Rock
{
    int id;  // 石块的编号
    double width, height; // 石块的宽度和高度
    double x, y; // 石块在可视化时的矩形方块的坐标（左上角）
};
vector<Rock> rocks; // 储存所有石块信息
set<int> candidateRocks; // 储存还没被切出的石块
vector<int> order; // 储存切出石块的次序
int n; // 要切出的石块的总数
double maxH, W; // maxH为石板被利用到的最大高度，W为石板的宽度
double totalArea = 0; // 石块面积之和
```

2. 切高度无限的石板

```
// 切高度无限的石板，（x, y）为石板左下角的坐标
void cutInfStrip(double x, double y)
{
    // 如果已经没有待切的石块，返回
    if (candidateRocks.empty()) return;

    // 从前往后遍历待切的石块，找到第一个宽度比W小的石块，将它切出
    for (auto i : candidateRocks)
    {
        double wi = rocks[i].width, hi = rocks[i].height;
        if (rocks[i].width <= W) // 找到了符合条件的石块
        {
            order.push_back(i); // 将该石块加入切出次序
            candidateRocks.erase(i); // 在待切出石块中将它删除
            rocks[i].x = x, rocks[i].y = y + hi; // 计算出该石块的坐标
            maxH = max(maxH, rocks[i].y); // 更新最大利用高度
            // 分成切一个高度有限的石板和一个高度无限的石板两个子问题
            cutLimitStrip(x + wi, y, W - wi, hi);  // 先切有限制的石板
            cutInfStrip(x, y + hi); // 再切无限制的石板
            break;
        }
    }
}
```

3. 切高度有限的石板

```

```

```cpp
// 切高度有限的石板（x, y）为石板左下角的坐标，w是石板宽度，h是石板高度
void cutLimitStrip(double x, double y, double w, double h)
{
    // 如果已经没有待切的石块，返回
    if (candidateRocks.empty()) return;
    // 如果该石板不合法，返回
    if (w <= 0 || h <= 0) return;

    // 从前往后遍历待切的石块，找到第一个可以切出的石块，将它切出
    for (auto i : candidateRocks)
    {
        double wi = rocks[i].width, hi = rocks[i].height;
        if (w >= wi && h >= hi)
        {
            order.push_back(i);
            candidateRocks.erase(i);

            rocks[i].x = x, rocks[i].y = y + hi;
            // 分成切两个高度有限的石块（两个子问题）
            cutLimitStrip(x + wi, y, w - wi, hi);
            cutLimitStrip(x, y + hi, w, h - hi);
            break;
        }
    }
}
```

4. 主函数

```cpp
int main()
{
    // 输入石块个数和石板宽度
    cin >> n >> W;

    double totalArea = 0; // 石块面积之和
    for (int i = 0; i < n; i ++ ) // 输入所有石块
    {
        Rock x{};
        cin >> x.id >> x.width >> x.height;
        totalArea += x.width * x.height;
        rocks.push_back(x);
    }
    // 对石块排序，先按高度从大到小，再按宽度从大到小
    sort(rocks.begin(), rocks.end(), [&](Rock a, Rock b) {
        if (a.height != b.height) return a.height > b.height;
        else return a.width > b.width;
    });

    // 创建待切出石块序列
    for (int i = 0; i < n; i ++ )
        candidateRocks.insert(i);

    // 开始切石板，以左下角为原点
    cutInfStrip(0, 0);

    // 计算利用率
    double utilizationRate = totalArea / (W * maxH);

```

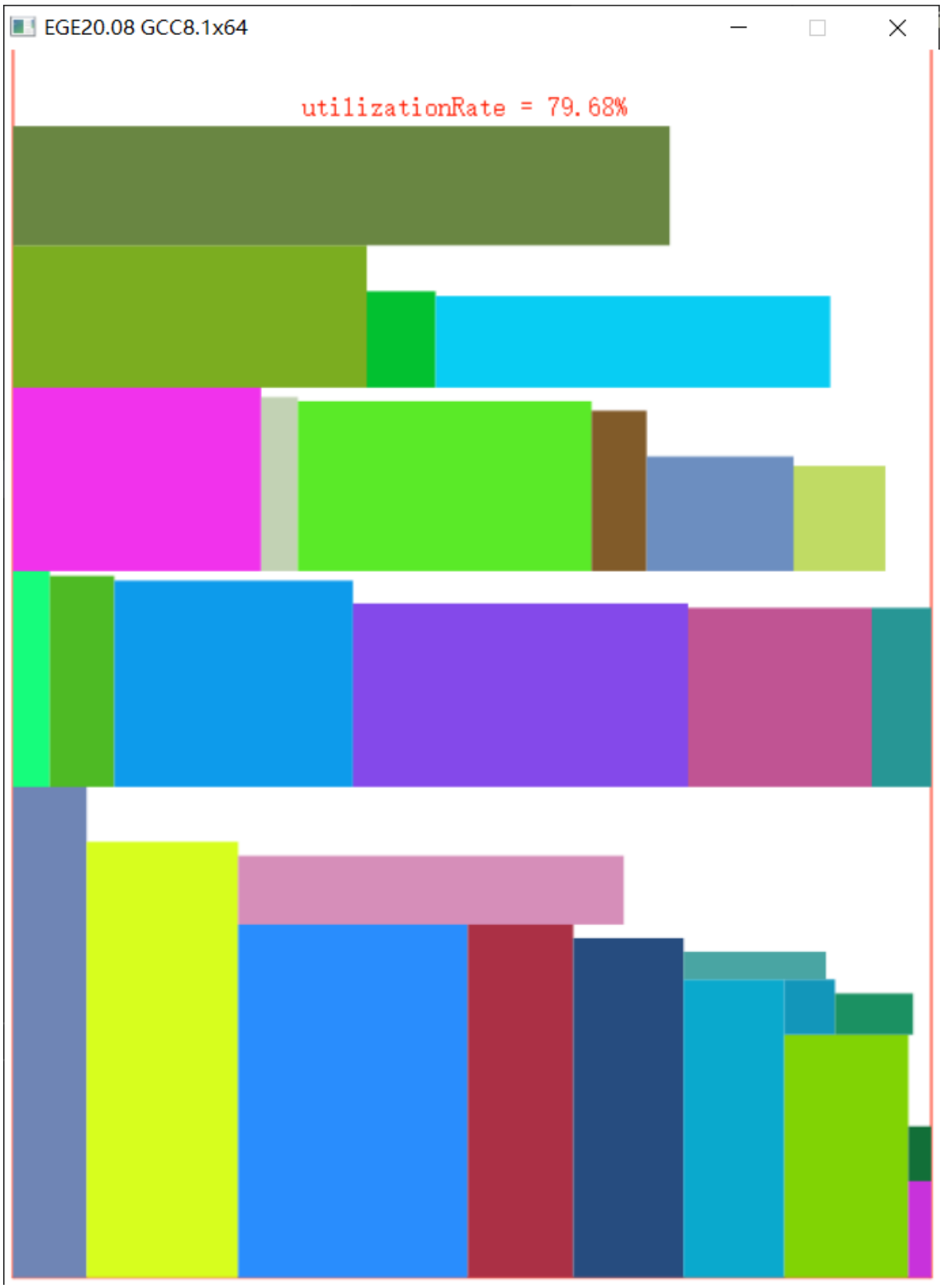
```
    // 初始化画布
    double graphWidth = 550, margin = 5;
    double widthSize = graphWidth - 2 * margin;
    double scale = widthSize / W; // 缩放比
    double heightSize = maxH * scale, graphHeight = heightSize + 50;
    initgraph(graphWidth, graphHeight, INIT_RENDERMANUAL);

    // 定义石板的左下角和右下角
    pair<double, double> st = {margin, graphHeight - margin}, ed =
{graphWidth - margin, graphHeight - margin};

    // 开启抗锯齿
    ege_enable_aa(true);
    // 设置画布背景颜色
    setbkcolor(0xffffff);

    // 画出石板的下、左、右边界
    setcolor(0xffff1b00);
    setlinewidth(1);
    ege_line(st.first, st.second, st.first, 0);
    ege_line(st.first, st.second, ed.first, ed.second);
    ege_line(ed.first, ed.second, ed.first, 0);

    // 将石块可视化输出
    randomize(); // 初始化随机数
    setwritemode(R2_BLACK);
    for (auto i : order)
    {
        Rock rock = rocks[i];
        // 计算石块对应的矩形缩放后的宽高
        double width = rock.width * scale, height = rock.height * scale;
        // 计算石块对应的矩形的左上角坐标
        double x = margin + (rock.x * scale), y = graphHeight - (rock.y *
scale + margin);

        // 随机设置石块的颜色
        int r = random(256), g = random(256), b = random(256);
        setfillcolor(EGEARGB(255, r, g, b));
        // 调用函数画出石块
        ege_fillrect(x, y, width, height);
    }
    // 在画布上打印利用率
    xyprintf(graphWidth / 2 - 100, graphHeight - margin - heightSize - 20,
"utilizationRate = %%%.2f", utilizationRate * 100);
    // 按任意键关闭画布
    getch();
    closegraph();
    return 0;
}
```

结果展示

1. 石块个数较小时利用率较低

`29个石块，石板宽度为200`

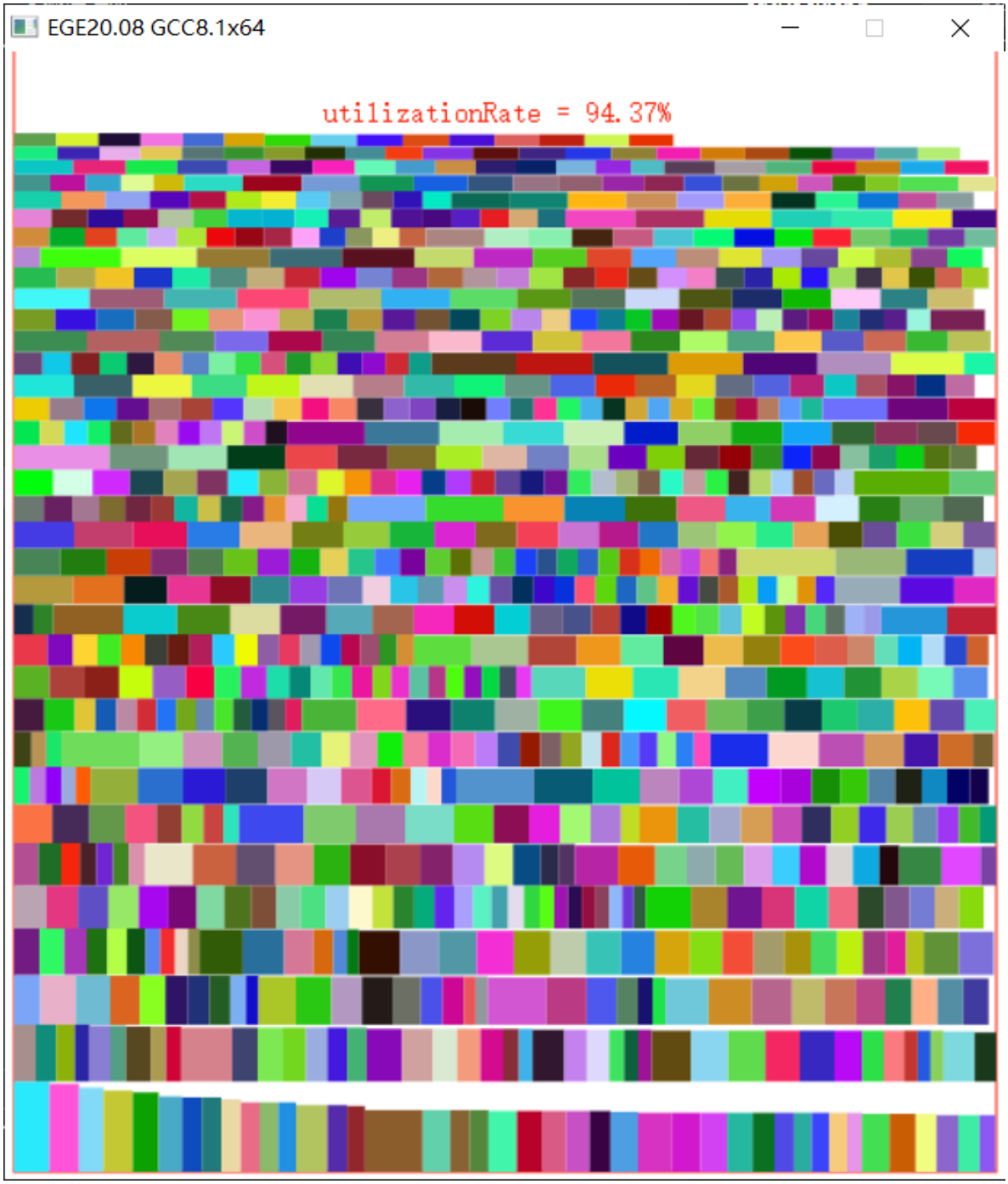
利用率一般在75%以上



2. 石块个数较大时利用率较大

`1000个石块，石板宽度为1000`

当石块数量上千时，利用率一般在90%以上



五、结论

本实验采用的是递归+贪心的做法，规定了切割的策略是优先切出高度大的石块，并且每次都是先横切再竖切，以达到最终消耗石板的高度最小的目的，同时兼顾了石板的利用率。从结果来看，当石块数量较小时，利用率在75%左右，效果欠佳；但当石块的数量较大时，利用率可达到90%以上，效果很好，由此可知这种方法适用于石块数量较多时（>1000）。

本实验可以改进：使用回溯算法或者动态规划算法得到最优解，而不仅仅局限于递归算法得到的近似解。