# 集装箱问题：基于平面的启发式DFS算法

## 问题定义：

本算法是满足了多数高级需求的非在线算法。对箱子有如下约束条件：

1. 没有方向约束，箱子可以旋转。
2. 没有重量约束，不考虑箱子的承重问题。
3. 静态装箱，对全部的箱子进行一次性全部摆放完。
4. 块的摆放按照坐标系方向开始。
5. 需要稳定性，即上层箱子需要被下层箱子完全支撑，没有悬空的部分。

## 算法数据结构和思路：

本算法使用了块装载、最低平面的思路。块装载是将多个集装箱组成块，作为空间分配的最小单位。最低平面是问题归约思路，采用由底向上的方式完成物品的装载，既优先由内向外铺满底面，然后再向上堆放物品。大体过程是：首先由完全相同的箱子组成块，然后用块自底向上依次填充所选择的目标平面，并重新生成若干新的平面，然后不断重复上述过程来完成最终的摆放方案。

#### 2.1 块的定义：

块是包含多个箱子的长方体，是由单种箱子组成的。是最基本的空间分配单位。如果一个块中包含的箱子不同，或者摆放位置不同，都是不同的块。块表 block\_table是预先生成的按块体积降序排列的所有可能块的列表，用于迅速生成指定剩余空间的可行块列表。

#### 2.2 平面的定义

平面是指可用于摆放箱子的面。初始平面就是箱的整个底面，放入第一批箱子后，平面包括了同批箱子顶面形成的面和箱底面空余的部分。

## 最低水平面的启发构造：

#### 平面的选择：

(1) 为避免堆积过高，影响箱子堆放的稳定性，优先选择空间位置较低(即参考点z坐标较小)的平面；

(2) 若几个平面参考点z坐标相同，则优先选择面积较小的平面，因为面积小的平面在后面可能更难使用；

(3) 若这些平面的参考点z坐标与面积均相同，则优先选择相对较狭长的平面，同样因为狭长的平面在后面可能更加难以利用；

(4) 若以上3点均相同，先比较它们参考点的x坐标，选择x坐标最小的一个。若x坐标相同，则选择y坐标最小的一个。

具体的算法设计可以参考文件中的select\_plane()函数:

#### 3.2 将块放置在平面上时的选择：

(1) 箱子组成的“块”能最大限度地利用目标平面，即放入块后目标平面剩余面积最小；

(2) 若剩余面积相同，比较块的体积，保留最大体积的块。

具体的算法设计可以参考文件中的find\_block()函数。

#### 3.3 平面的合并：

当目前所选择的平面无法装入任何一个箱子时，就需要进行合并平面的合并。合并平面的要求是这两个平面有临边，合并的平面可以从平面列表和备用平面列表中选取。合并的测试条件有：

(1) 可以装入至少一个仍有剩余的物品(种类、方向不限)；

(2) 合并后新平面的面积是否比原先两个平面都大。

当目前所选择的平面无法和平面列表和备用列表中的所有平面合并时，就将该平面加入备用平面列表中。等待下次的合并。

具体的算法设计可以参考文件中的merge\_plane()函数

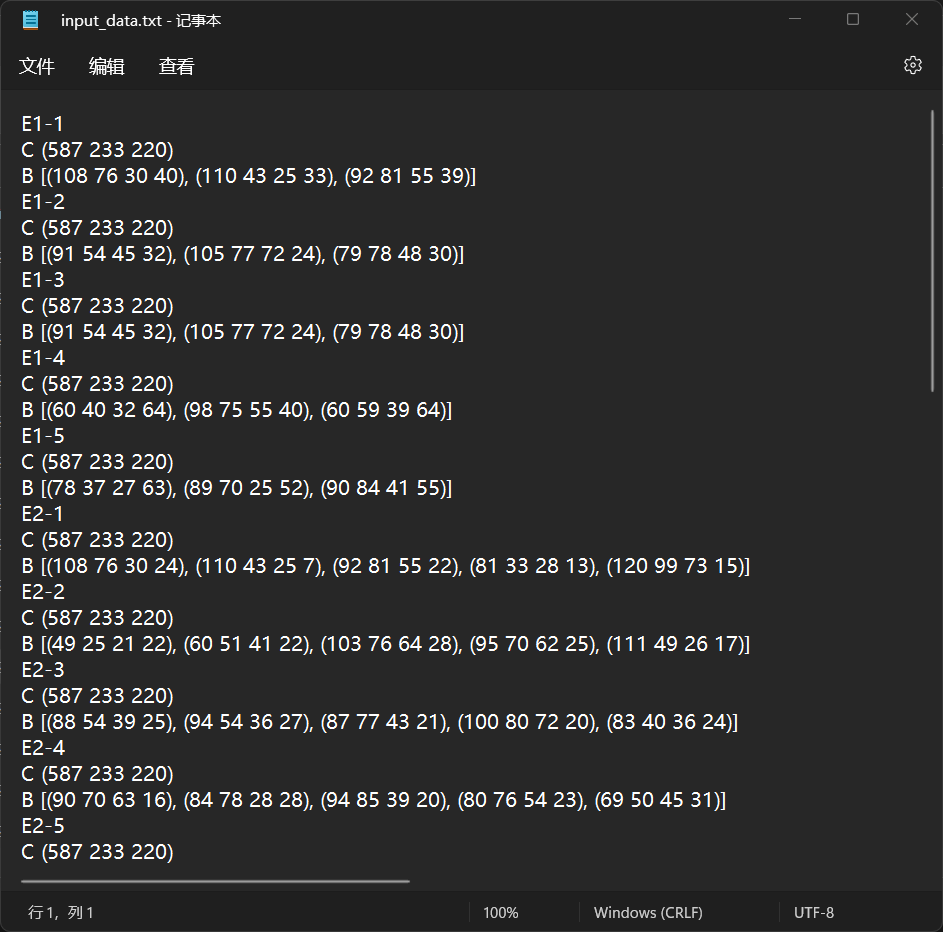
#### 3.4 新平面的产生和选择：

放置一个块时会产生3个平面：两个低层的平面（矩形）和一个高层的平面，此时应该使得低层平面中面积较大的一个取最大值，尽量让小矩形（可能被浪费）的面积更小。

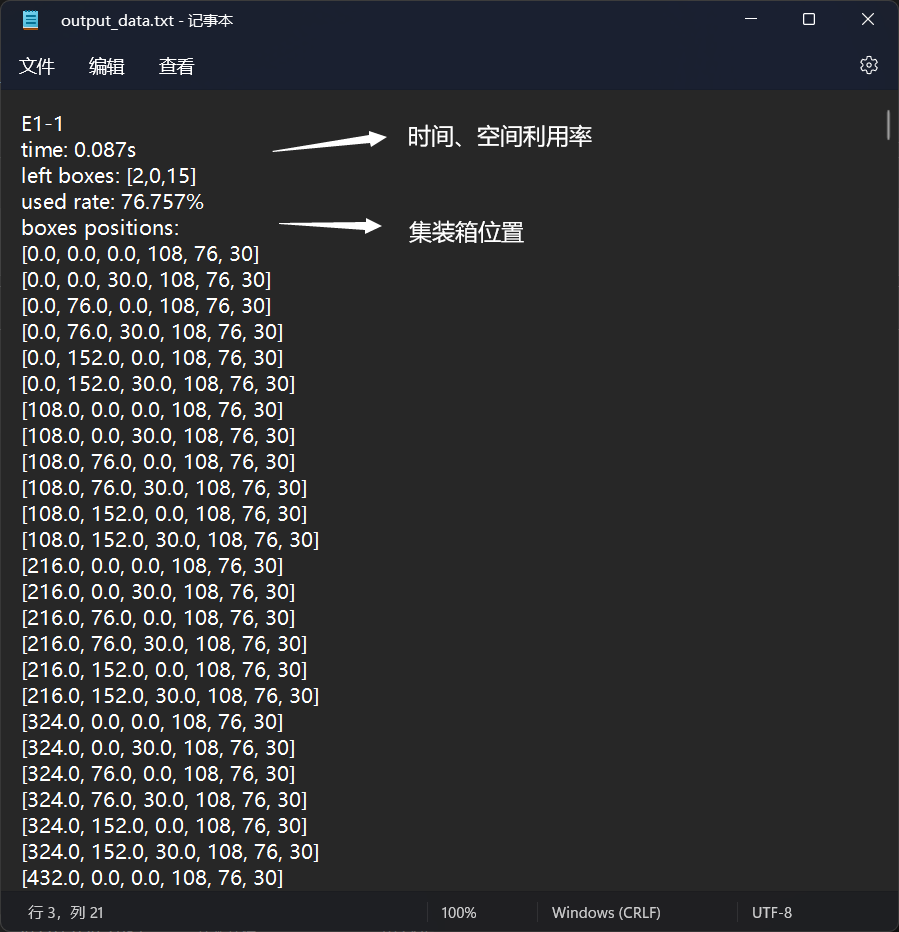
## 示例

#### 4.1基础要求：

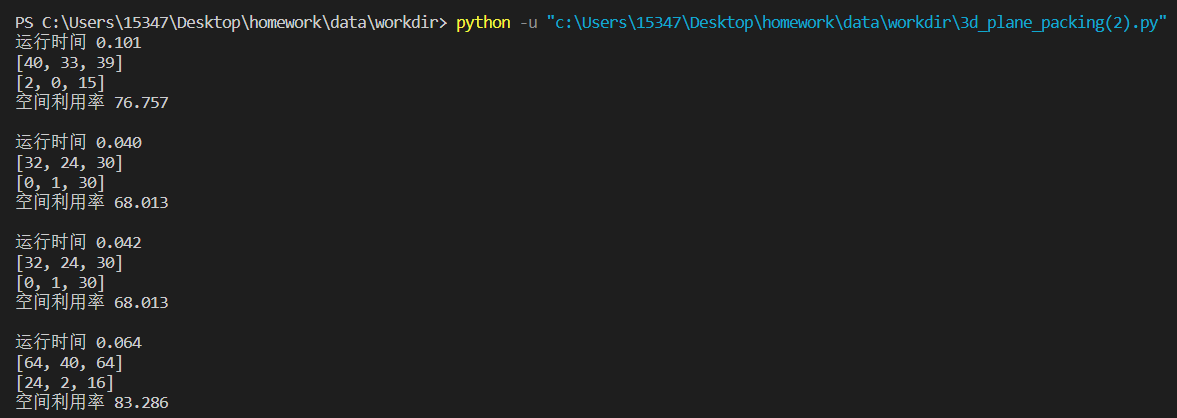
静态装箱，给定输入txt文件。输出所有情况的集装箱放置策略：



输入文件示意图



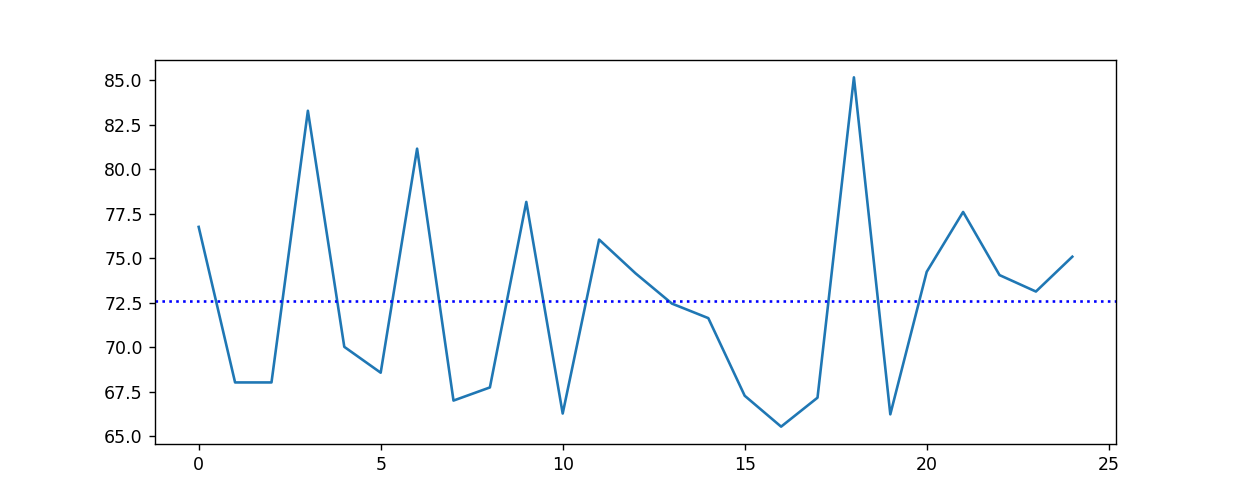
输出文件示意图



运行结果示意图

空间利用率：

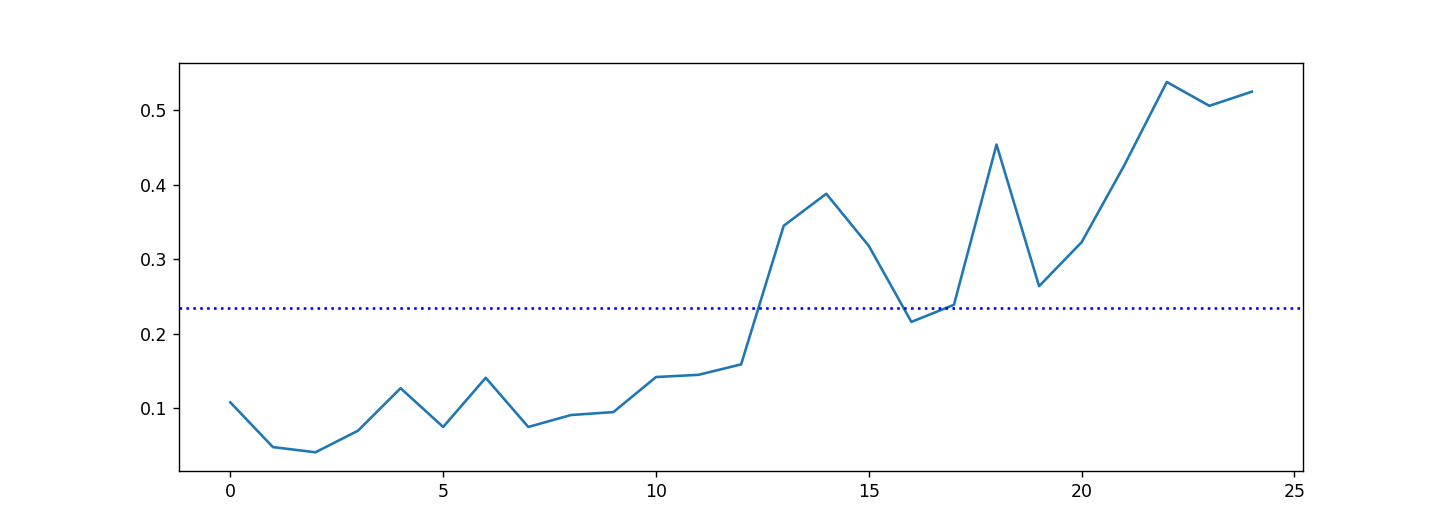
平均在75%左右，空间利用率和集装箱种类的数量无明显关系。



空间利用率的变化图

计算时间：

如图所示，解决集装箱问题的时间随着集装箱种类的增加而增加。最高时间不超过0.5秒。



问题解决时间的变化图

#### 高级要求1：参数考虑小数点后2位

当我们将箱子的长宽高设置为小数点后2位时，算法仍能运行，具体如下图所示：

文本

描述已自动生成

参数设置

图表, 折线图

描述已自动生成

运行结果示意图

可以看到，当我们将箱子的宽度设置为2位小数时，算法仍能正常对箱子进行摆放，说明我们的算法实现了这一部分的要求。

#### 4.3高级要求3：摆放顺序

我们的算法同样实现了将箱子从内到外、从上到下的摆放顺序。我们首先将箱子的参数设置为如下图所示：

文本

描述已自动生成

参数设置

此时的运行结果如下图所示：

图表, 雷达图

描述已自动生成

运行结果示意图

而当我们添加了一个箱子后，即参数修改为下图：

文本

描述已自动生成

参数设置

此时的运行结果则如下图所示：

图表, 雷达图

描述已自动生成

运行结果示意图

可以看出，算法的确是按照从内到外的顺序来摆放箱子的。

而对于从底向上，首先将参数设置为如下：

文本

描述已自动生成

参数设置

此时的运行结果如下图所示：

图表, 雷达图

描述已自动生成

运行结果示意图

而当我们将箱子个数增加后：

文本

描述已自动生成

参数设置

此时新增的箱子会摆放在旧的平面之上：

图表, 雷达图

描述已自动生成

运行结果示意图

通过这两者的对比，可以表明我们的算法的摆放顺序也是自底向上的。

#### 4.4高级要求4：箱子旋转

我们的算法也实现了让箱子可以以多个面为底开摆放，即集装箱的旋转。将箱子的参数如下设置：

文本

描述已自动生成

参数设置

可以看到，此时我们的箱子均为统一种类的箱子，而算法的运行结果则为：

图表, 图示

描述已自动生成

运行结果示意图

可以看到，虽然箱子的种类相同，然而摆放的方式却发生了改变，说明我们的算法实现了对箱子进行旋转，以此来增加容器的利用率。

#### 4.5高级要求5：实时求解

正如前文所提到的，本算法的时间复杂度较低，针对全部的输入数据，均可在0.5秒内求解出结果。