**Lab2 实验报告**

**221300034 高志轩**

存储的变量：在DMA类中，我主要存储了self.heap(模拟内存)，self.free\_blocks(存储空闲块)，self.allocated\_blocks(存储已分配的块)。

数据结构设计：

1. self.heap：使用列表进行模拟。
2. self.free\_blocks：使用列表进行模拟，列表元素是二元组，形如[ (start, size) ]。每一个元素表示heap中空闲块的起始位置和大小。
3. self.allocated\_blocks：使用字典进行模拟，其中每一个键对应的值也是一个字典，形如{id : {‘start’: start, ‘size’: size, ‘value’: value}}。id表示分配块的编号，start表示分配块的起始位置，size表示大小，value是一个列表，存储该块的内容。self.allocated\_blocks的格式也就是self.data()需要返回的格式。

分配策略：

1. 对于malloc：

我采用了Best-fit和Worst-fit相结合的方法。

一开始的时候我只采用了Best-fit的策略，发现外部碎片的指标比较高。我发现当我只使用Best-fit的策略时，当策略匹配到一个best-fit块，但是这个best-fit块只比所需要分配的大小只多一点点时，将会产生一个非常小的碎片，这个碎片未来可能比较难以被使用，所以将会留下一个较小的碎片。

但是，注意到如果Best-fit策略匹配到一个大小恰好与所需大小相等的块，这是非常好的，因为随着这个块的放入，外部碎片一定会减少，所以如果遇到这样的完全匹配的块时，我们将直接退出空闲列表的遍历，并分配内存。

于是，我考虑了当出现找到的best-fit块大小比所需的只大一点时，将会使用Worst-fit策略，将分配大小最大的空闲块给所需内存。其中，切换策略的判断条件为：not equal and (best\_fit\_size - size) < best\_fit\_size \* ratio，其中ratio为一个常数比例。

1. 对于compact：

注意到，malloc操作并不会增加外部碎片，于是compact操作只需要在free的时候考虑。为了使外部碎片最少，理想情况下是每一次free操作后都进行compact，但是将会降低执行效率。

于是，我使用了另一个比率来进行判断，也就是如果self.frag / self.size > self.ratio，此时则进行compact。我通过维护self.frag来记录当前有的碎片数，如果碎片数与heap大小之比大于一个比例时，此时进行compact。