Part A

思路分析

关键点在于cache结构体的设计与算法的实现。此外就是初始化cache，模式选择和输入输出读取。

具体实现

全局变量

```
// 缓存参数
int s, E, b, S;  // s: 组索引位数, E: 每组的行数, b: 块偏移位数, S: 组数
char file[100];   // 存储内存访问轨迹的文件名

// 缓存统计数据
int hit_count = 0, miss_count = 0, eviction_count = 0;

// 全局计数器，用于缓存替换策略
int count = -1;
```

输入输出

对于输入输出和模式选择，直接参考PPT中给出的`getopt`和`fscanf`操作即可，根据PPT上的实现能够处理好操作的读取。

cache设计

首先知道应该是一个二维数组，其中具体元素需要包括`tag`即标识位，此外，为了实现所需算法，还要一个变量用来记录时间。

```
struct Cache
{
    int tag;
    int time;
} **cache;
```

此外要对cache进行初始化，为防止segmentation fault，需要逐行开辟空间。又由于C中不能直接给结构体变量赋初始值，所以在初始化时还需要赋给初始值。

对于`tag`，需要初始化为1个负数，来标志空行，同时防止和真实tag相冲突。

对于`time`，可以任意赋给一个相同初始值。

算法实现

首先，有四种模式，对应四种访问操作。

1. `I`可以直接跳过。
2. 对于`L`和`S`，都是直接访问一次即可。
3. 对于`M`模式，在访问第一次之后，可以知道马上的第二次一定会命中，就不用再次访问，直接count加1就可以。

对于访问操作：

首先需要拿到数据的tag和set，利用位移操作实现。此外，增加了三个辅助变量，来实现判断，分别是标志第一个空行位置的`empty_set`，找到目前时间最大值以及对应下标的`max`和`max_index`.

```
    unsigned tag = addr >> (b + s);
    unsigned set = addr << (64 - b - s) >> (64 - s);//清除无效位，得到S。
    int empty_set = -1;
    int max = 1 << 31;//为保证正确更新，初始化为最小负数。
    int max_index = 0;
```

此后就是实现算法。

为了实现，初始化了一个全局变量`count`，每次循环的时候-1，每次进行行访问时，将其赋值给对应的`time`。相比于对所有cache的计数器加1，这能够用较小代价保证时间的正确更新。（最新访问的永远是数值最小的，访问时间越久远对应数值越大）。

在最后的循环中，同时进行匹配，空行判断，time最大值行判断。

如果匹配成功直接hit+1然后return。

如果匹配失败，空行判断成功，miss+1，然后替换空行。

如果都失败，miss+1，替换+1，然后替换对应的行。

最后就结束了。

Part B

32×32

直接利用ppt中的分块技术。

每次访问会读到同一行的8个整数，8行可以充满整个cache。

在转置时，按照8*8的块，除对角线上的，别的地方都能很好地解决。

处理对角线时，可以通过局部变量把读到的A的一整行存起来，再赋值给B实现。

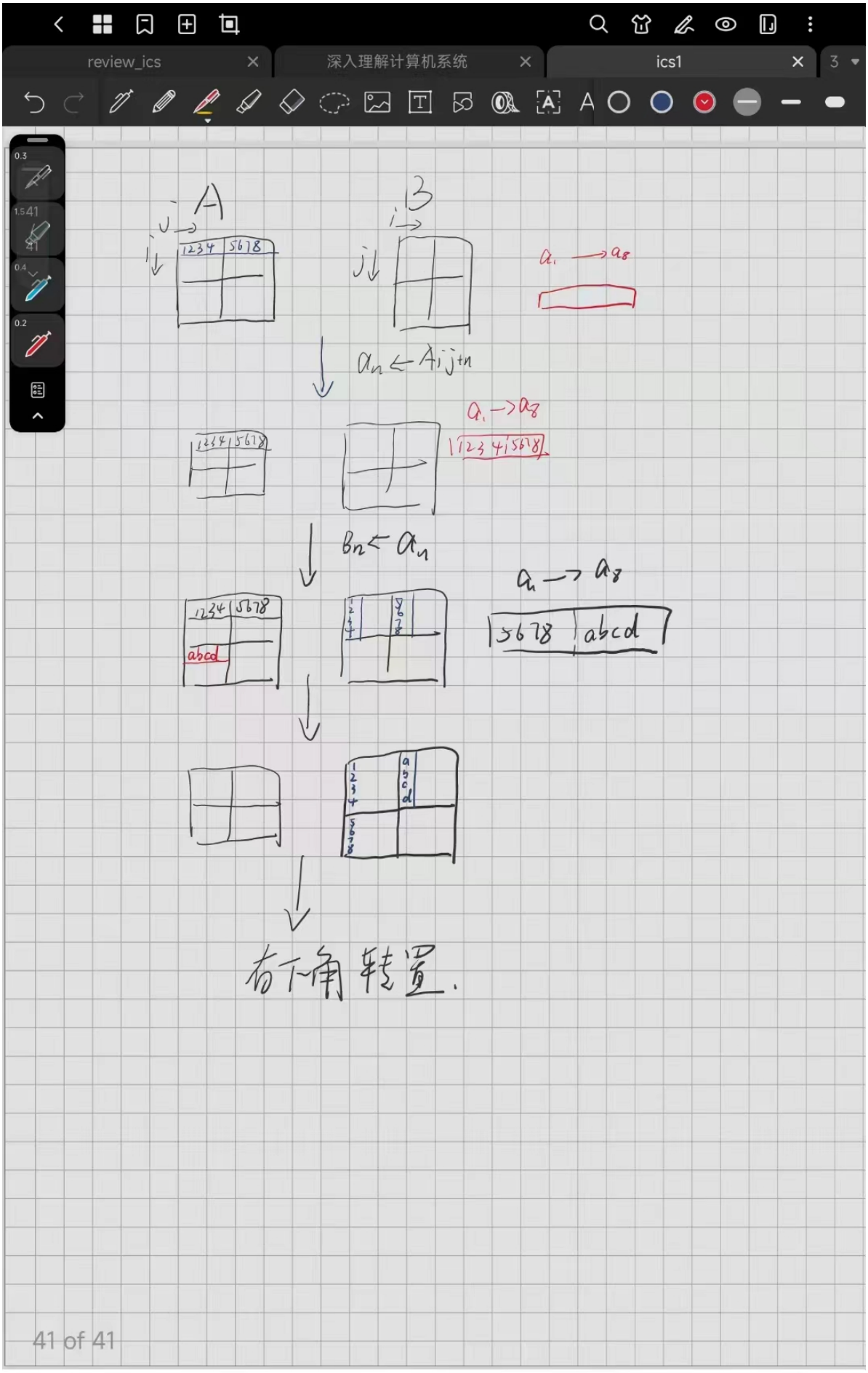
这样这个部分就能满分了。

64×64

直接按照上一题的8*8会由于冲突导致四千多次miss，因为这个时候转置对应的8*8块之间会发生大量冲突。

为了解决这个冲突问题，就需要一些操作，就是在大的分块的基础上再进行小的分块。分成4个4*4的。

具体过程如图：



也就是说，先把A左上角和右上角给局部变量，然后在B的左上角和右上角分别放下转置的A矩阵左上角和右上角。

此时B的左上角已经完成转置。

然后把B的右上角和A的左下角给局部变量。

分别对应B的左下角和右上角。

此时只剩右下角。

右下角直接可以用分块实现。

61×67

这个地方cache对应太难算了，想了很久没想明白具体对应关系。

最后就只能直接用16*16的最简单的分块了，刚好可以过。