这份作业是关于计算机体系结构的研究生课程，具体是关于缓存（Cache）的模拟。作业的目标是使用Python编程语言来创建两种类型的缓存：直接映射缓存（Direct-Mapped Cache）和集合关联缓存（Set-Associative Cache）。这个缓存模拟器主要处理内存地址，而不关心数据的实际值。
作业的具体要求如下：

1. **环境准备**：
    - 需要在计算机上安装Python3。建议使用Ubuntu系统，并且可能需要通过pip3安装numpy库。
2. **直接映射缓存（Direct Mapped Cache）**：
    - **热身练习**：给定一系列的32位内存地址引用，要求根据直接映射缓存的规则（如块大小和缓存总大小）来判断每个引用是命中（hit）还是未命中（miss），并计算命中率。
    - **直接映射缓存模拟**：需要实现一个简单的直接映射缓存模拟器。具体需要实现以下几个Python函数：
        - `find_set`：根据地址返回集合（set）的值。
        - `find_tag`：根据地址返回标签（tag）的值。
        - `find`：根据地址判断是命中还是未命中。
        - `load`：在缓存未命中时，根据地址加载数据到缓存。
    - **验证**：验证通过Python缓存模拟器得到的命中率是否与热身练习中计算的一致。
3. **集合关联缓存（Set-Associative Cache）**：
    - **热身练习**：类似于直接映射缓存的热身练习，但是这次是集合关联缓存，并且需要考虑替换策略（如最近最少使用LRU）。
    - **集合关联缓存模拟**：需要实现一个简单的集合关联缓存模拟器。需要实现的函数与直接映射缓存类似，但是需要处理多个集合。
    - **验证**：验证通过Python缓存模拟器得到的命中率是否与热身练习中计算的一致。

作业的提交物包括：

- 两个Python文件：`cache_dm.py`（直接映射缓存）和`cache_sa.py`

（集合关联缓存）。

- 一份报告，包含：
  - 代码的输出结果。
  - 对代码逻辑的解释和输出的澄清。
  - 热身练习和验证部分的工作。

这份作业的目的是让学生通过实践来更深入地理解课堂上学到的缓存概念。通过编写和测试缓存模拟器，学生可以更好地理解缓存的工作原理，如地址映射、命中和未命中的处理，以及不同的缓存替换策略。

说明

在此作业中，您将使用 Python 创建直接映射缓存和集相联缓存。每个内存地址（字节或字地址）指向内存中的某些数据。此缓存模拟器本质上仅适用于地址。在此缓存模拟器项目中，我们对数据的实际值不感兴趣。

1. 您需要在您使用的机器/笔记本电脑上安装 Python3。我们已经在 Ubuntu 系统上测试了该程序，我们建议学生使用相同的系统。拥有 CS-Portal 帐户的 CS/CpE 学生应该能够通过 SSH 在门户服务器上运行他们的实验。确保您使用 python3。如果尚未安装，您可能还需要使用 pip3 安装 numpy。
2. 要模拟直接映射缓存，请转到包含 sim dm.py 文件的目录并使用命令 python3 sim dm.py 。将 sim dm.py 替换为 sim sa.py。如果需要，您可以将标准输出重定向到文件。

可交付成果

1. 2 个 Python 文件，缓存 dm.py 用于直接映射缓存，缓存 sa.py 用于组相联缓存。请勿更改 Python 文件的名称。
2. 1 份报告，包含：
   (a) 代码输出
   (b) 代码说明（使用的逻辑、输出说明）
   (c) 您对直接映射缓存和组相联缓存的热身部分和验证部分的工作。

展示您的工作不是可选的，您的最终得分将高度依赖于报告的质量。

1 直接映射缓存 (50)

1.1 热身

下面是 12 个 32 位内存地址引用序列，以字地址形式给出。1、18、2、3、4、20、5、21、33、34、1、4。请注意，字地址和字节地址不同。字地址加上偏移量（两个零）可得出字节地址。换句话说，字地址是 4 的倍数。例如，如果字地址为“1”，则等效字节地址为“4”（100）。

1.假设直接映射缓存具有一个字块，总大小为 16 个块，假设缓存最初填充了字地址 0、1、2、... 15 个内存数据，列出每次引用是命中还是未命中。显示最后一次引用后缓存的状态。

2.计算此引用字符串的命中率。 （正确答案：命中率 = 4/12）
现在，假设一个直接映射缓存有两个字块，总大小为 8 个块，假设缓存最初为空，列出每次引用是命中还是未命中。显示最后一次引用后的缓存状态。计算此引用字符串的命中率。（正确答案：命中率 = 1/12）

1.2 直接映射缓存 (40 分)

我们将逐步用 Python 构建一个简单的缓存模拟器。目标是更深入地理解课堂上学习的概念。我们将从构建一个简单的直接映射缓存模拟器开始。资源文件夹中有三个文件夹：traces、code 和 reference output。在代码目录中，我们有两个 Python 文件：a) sim dm.py 和 cache dm.py。仔细阅读 Python 文件。sim dm.py 是直接映射缓存模拟器文件。cache dm.py 是类文件，其中包含缓存类定义和空函数。这些函数在 sim dm.py 中被适当调用。您需要实现这些函数 - 无需更改代码中的任何其他内容。总之，您需要编写以下四个 Python 函数。

• find set 函数接受一个地址并返回设置值（注意：对于直接映射缓存，方式数 = 1，集合数 = 缓存中的块数）。
• find tag 函数接受一个地址并返回标签值。
• find 函数接受一个地址并返回一个布尔值（缓存未命中时返回 false，缓存命中时返回 true）。您可以重用 find set 和 find tag 函数来实现此 find 函数。
• load 函数接受一个地址并将适当的数据加载到缓存中。请注意，只有当缓存未命中时，才会在 sim dm.py 中调用 load 函数。如果需要，您可以重用上述实现的函数。

与 traces 文件夹相关的文档可在 sim dm.py 中找到。traces 文件夹本质上包含引用字符串（第二列）。这些引用字符串是十进制的字节地址。

1.2.1 示例命令行

python3 sim dm.py test . trace 16 1 8 其中
• test.trace 是存储在 ../traces 中的跟踪文件
• 16 是缓存大小（以字节为单位）（大小 = 块大小 x 集 x 方式；所有单位均为字节）• 1 是方式数
• 8 是每个块的字节数

1.3 验证（10 分）

验证从 Python 缓存模拟器获得的命中率是否与您在预热部分执行的计算中获得的命中率相同。相关的跟踪文件是 dm a.trace 和 dm b.trace。请注意，在 dm a.trace 中，前 16 行只是为了根据预热要求填充缓存；不要将它们考虑用于计算未命中率。

2 组相联缓存 (50 分) 2.1 热身

以下是作为字地址给出的 12 个 32 位内存地址引用序列。1、18、2、3、4、20、5、21、33、34、1、4。请注意，字地址和字节地址不同。字地址加上偏移量（两个零）可得出字节地址。换句话说，字地址是 4 的倍数。例如，如果字地址为“1”，则等效字节地址为“4”（100）。
假设组相联缓存有两条路径、两个字块，总大小为 8 个块，假设缓存最初为空（假设 LRU 替换），列出每个引用是命中还是未命中。显示最后一次引用后的缓存状态。计算此引用字符串的命中率。（正确答案：命中率 = 5/12）

2.2 集合关联缓存 (40 分)

我们将逐步用 Python 构建一个简单的缓存模拟器。我们将构建一个集合关联缓存。资源文件夹中有三个文件夹：traces、code 和 reference output。在代码目录中，我们有两个 Python 文件：a) sim sa.py 和 cache sa.py。仔细阅读 Python 文件。sim sa.py 是集合关联缓存模拟器文件。cache sa.py 是类文件，其中包含缓存类定义和空函数。这些函数在 sim sa.py 中被适当调用。您需要实现这些函数 - 无需更改代码中的任何其他内容。总之，您需要编写以下四个 Python 函数。

• find set 函数接受一个地址并返回设置值（注意：对于集合关联缓存，路径数大于 1，并且集合数 = 缓存中的块/路径数）。
• find tag 函数接受一个地址并返回标签值。
• find 函数接受一个地址并返回一个布尔值（缓存未命中时返回 false，缓存命中时返回 true）。您可以重用 find set 和 find tag 函数来实现此 find 函数。
• load 函数接受一个地址并将适当的数据加载到缓存中。请注意，只有当缓存未命中时，才会在 sim dm.py 中调用 load 函数。如果需要，您可以重用上述实现的函数。

与 traces 文件夹相关的文档可在 sim sa.py 中找到。traces 文件夹本质上包含引用字符串（第二列）。这些引用字符串是十进制的字节地址。

2.2.1 示例命令行

python3 sim sa .py test . trace 16 2 8 其中
• test.trace 是存储在 ../traces 中的跟踪文件
• 16 是缓存大小（以字节为单位）（大小 = 块大小 x 集 x 方式；所有单位均为字节）• 1 是方式数
• 8 是每个块的字节数

2.3 验证（10 分）

验证从 Python 缓存模拟器获得的命中率是否与热身部分计算的结果相同。相关的跟踪文件是 sa.trace。