OS Homework3

資訊三乙 11027209 巫年巨

開發環境 : python , 版本3.10.7

實作方法與流程

這次有作業要實做5個Page Replacement與1個呼叫全部的方法，由於有大量的資料需要紀錄，故使用多維list把全部的資料(記錄起來，除此之外多設了一個一維的list叫page\_frame來記錄現在page\_frame的狀況，而處理有LFU跟MFU的狀況時設了一個dict字典來記錄某個字被reference等的次數，每個placement方法大致上都會跑page\_reference\_string長度大小的迴圈，依三個狀況去做判斷，分別是在該字不在page\_frame且page\_frame沒滿或是有滿，或是該字在page\_frame的三個情況，第一個情況和第三個情況在每個placement的演算法都類似，一的話都是增加page\_fault次數並記錄F和把該字放入page\_frame就好了而三的話就是紀錄空字串就好了，每個演算法的不同主要是處理該字不在page\_frame裡且page\_frame是滿的狀況，但是共通點都是會記錄F和page\_fault和page\_replace會加1。

那麼首先就來講第一個方法FIFO，這個就用queue的概念把最一開始加入的給pop掉，在append新的字進page\_frame就好了。

第二個方法是LRU的話，我的方法是創一個index\_list大小同page\_frame且都設成false，然後依照現在的字(如果現在找page\_reference\_string是i是5的話，就從4往前找到 i = 0)為止的字，如果字出現在page\_frame裡的話，就把page\_frame字同index的地方設為True，就這樣直到只剩一個false的時候，就知道那個字是最久的˙所以要被替換掉，所以把page\_frame的那個false的index位置給pop掉後再append新的字進來就好了。

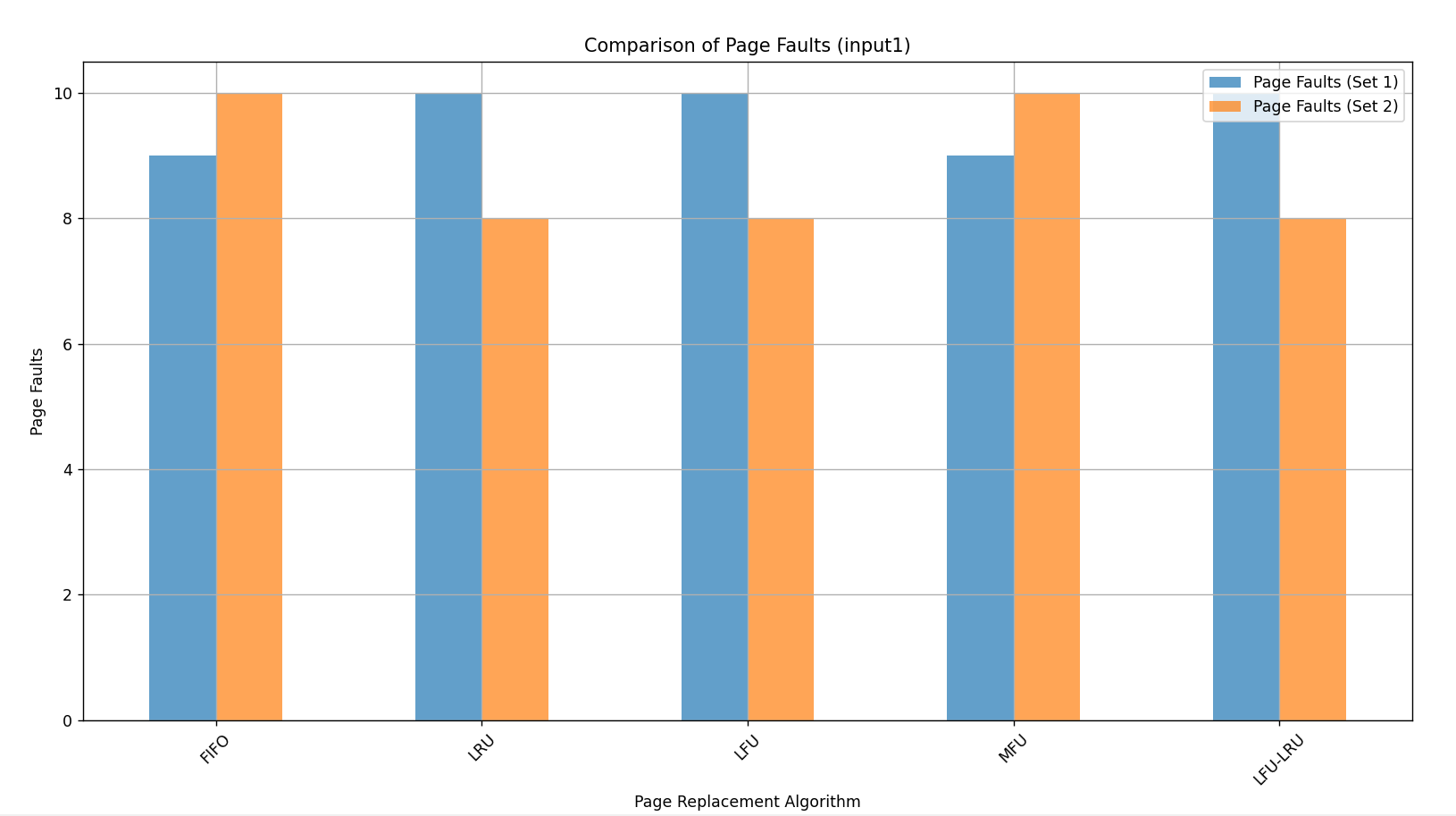
第三個LFU+FIFO的話，跟上者一樣創一個index\_list，然後放的是無限大，用意是為了找出最小的字出來，首先把page\_frame的字放到字典中看次數，去更新index\_list的內容，並且去記錄最小的次數和他的index，做完後要多做一個判斷就是最小的次數是不是有多個跟他一樣的，有的話就做FIFO抓最小的index的給pop掉，沒有的話就是把那個最小的給pop掉在append新的字就好，要記得在pop掉前要先把被pop掉的字，在字典的次數做reset就好，而之後紀錄新的字不再字典的話就紀錄並且次數為1，在的話就次數加1就可以結束了。

第四個MFU+FIFO，大致上跟第三個相同，只要把無限大變成負無限大，然後找小的變成找大的，流程上差不了多少就結束了。

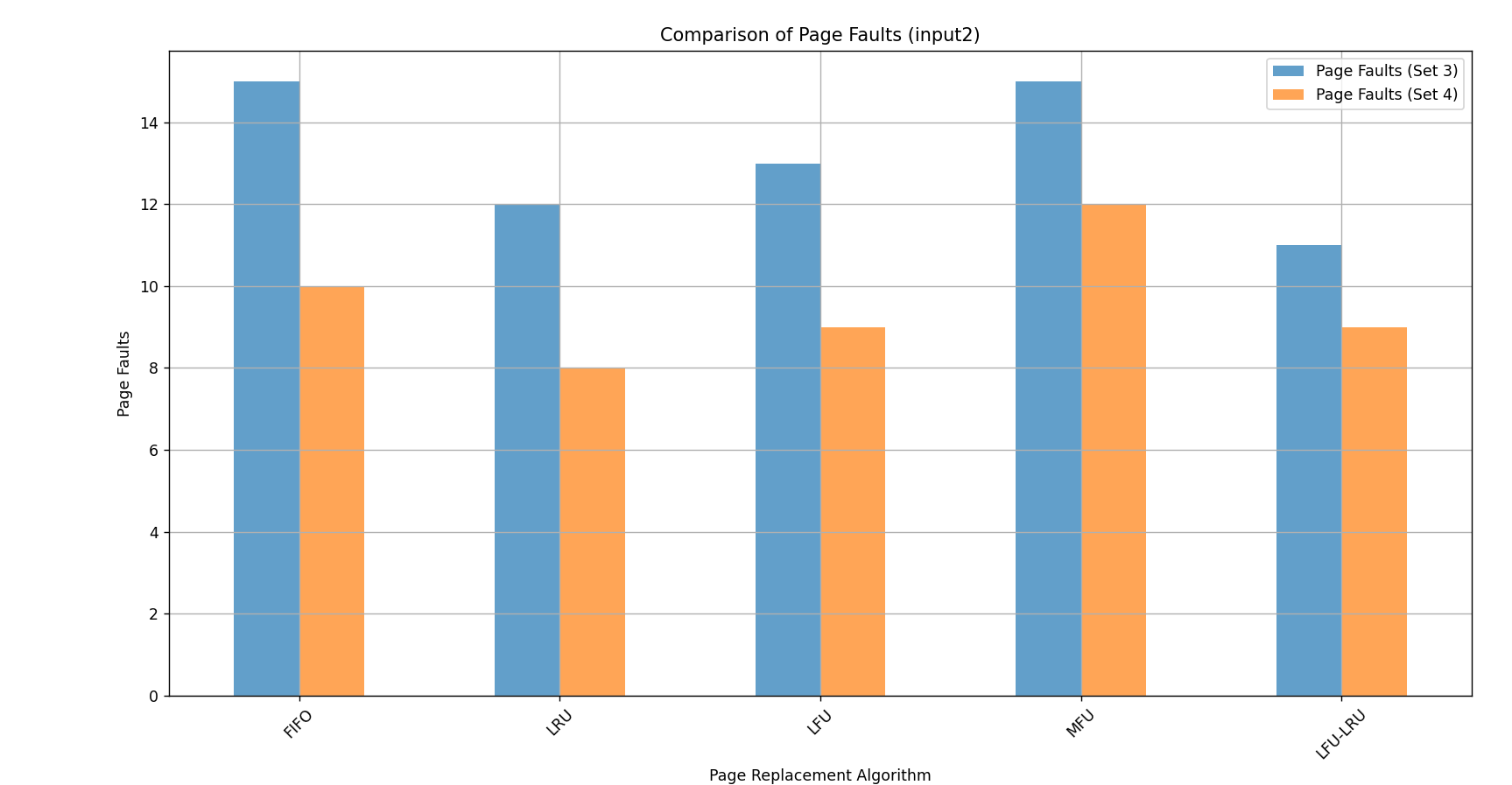
第五個LFU+LRU，其實也就是把二跟三抓一點抓一點部分近來，抓的分別是一開始用LRU的index\_list找次數最小的部分，如果次數最小的有兩個或以上，就變成另一個LFU的那個另一個index\_list找剩下的那個false的部分，找完後的那個index，pop掉後就剩下更新字典之後就結束了。

不同方法的比較

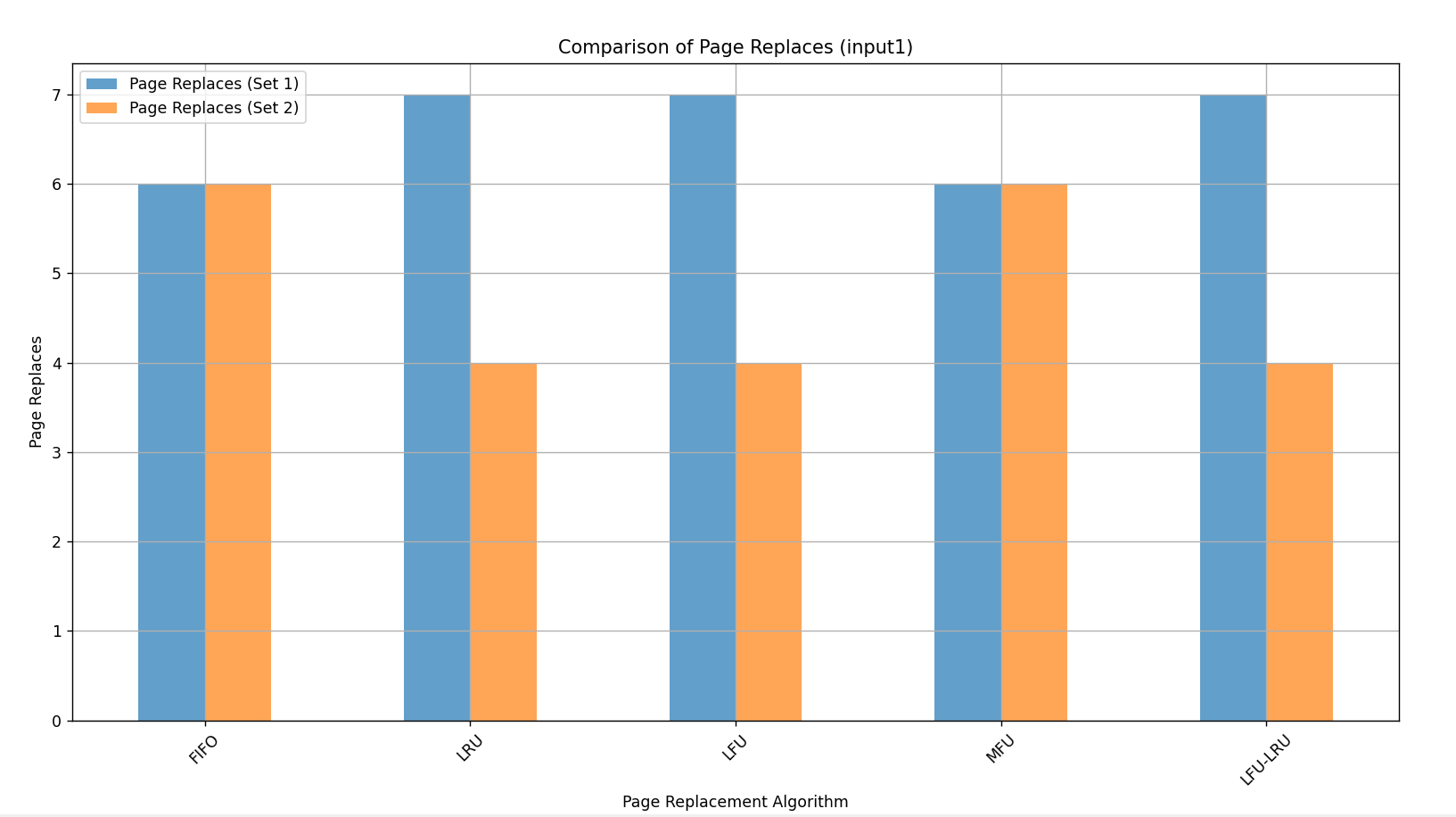
以下是Page\_Fault的比較，分別是input1的資料藍色長條為放入page\_frame大小為3，而橘色是大小為4的狀況，正常來講Page\_Fault次數隨著page\_frame越大應該越小才對，但是可以從圖中可看出畢雷迪反例在FIFO和MFU的時候，而其他的狀況大致上可看出LRU和LFU在處理page\_fault有著較好的表現



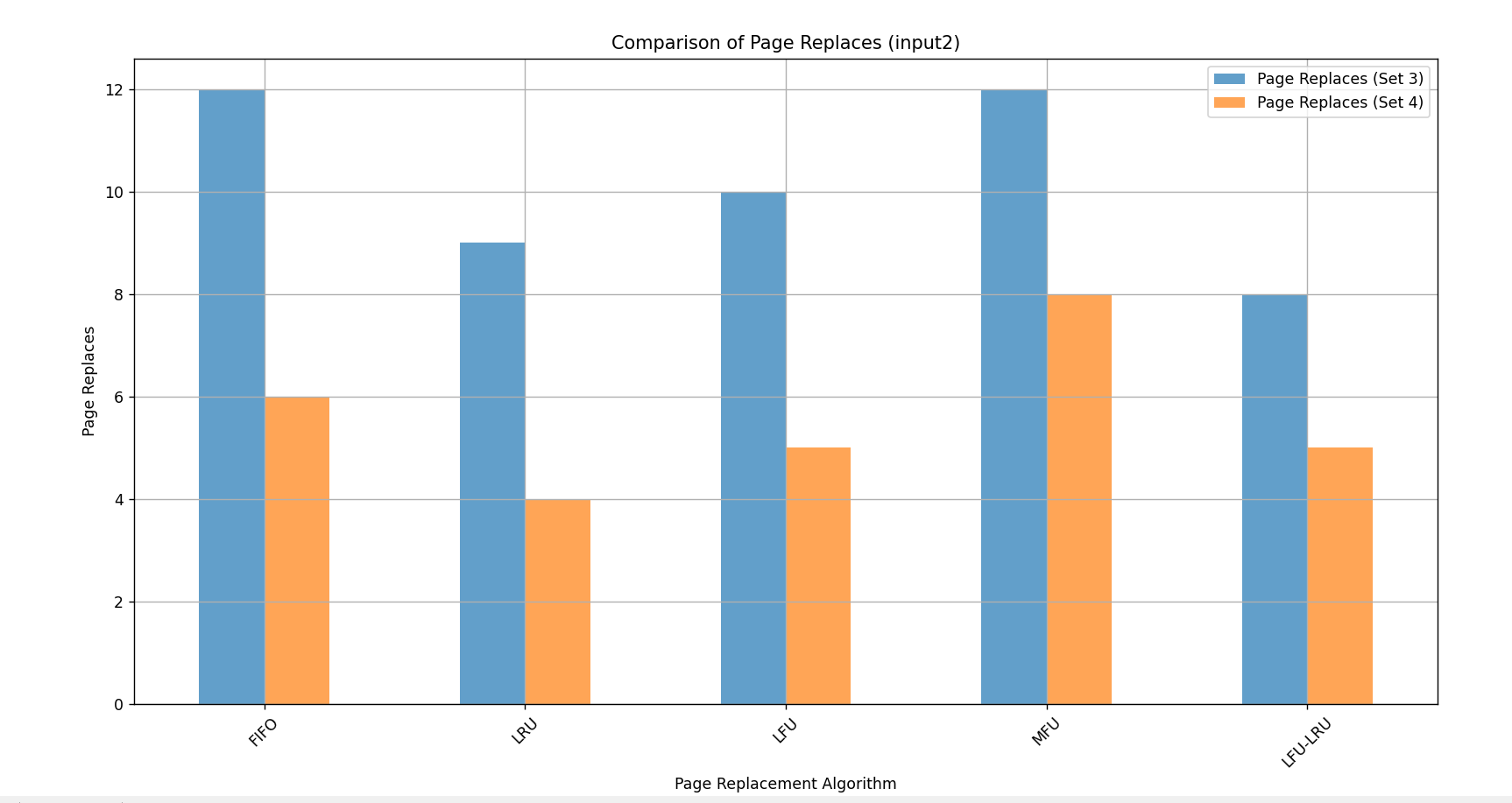
而在input2的時候也可看出，LRU和LFU在處理page\_fault有著較好的表現，FIFO在所有表現中也未必最差，MFU在Page\_frame大小為4的時候，是比FIFO還要差的。LFU大致上都在中規中局的位置，而LFU+LRU未必每次加起來就會有比較好的效果，用圖表可看出單純在PAGE\_FRAME大小為4時比+了LRU的效果還好。



以下是Page\_Replace的比較，分別是input1的資料藍色長條為放入page\_frame大小為3，而橘色是大小為4的狀況，Page\_Replace次數確實如page\_frame大小便得越來越小，最糟也是相同，可看出在input1的狀況中，在藍條(3)的FIFO跟MFU的狀況是比其他LRU和LFU好的，但是橘條下(4)FIFO和MFU大小上去後發生replace次數相同，而其他的replace的次數就下降至兩者之下了。



而在input2的時候也可看出，LRU和LFU在處理Page\_Replace大致上有著較好的表現，FIFO和MFU的表現就差了點，Page\_frame大小為4的時候， LFU大致上也還是在中規中局的位置，LRU在4的時後效果最後，而LFU+LRU在3的時後效果最好，所以可歸納出每個方法對於Page Replace來說都有一定的優勢存在。



結果與討論

這次的任務中從效能上來看各種方法的話，我覺得比較來說綜合評分整體下來看選LRU是比較好的選擇，在大部分的情況下的Page Fault和Page Replacement較低，而應用上也屬於最佳化的演算法不會有畢雷迪反例。

至於畢雷迪反例的狀況的話，用input1來看，增加了page\_frame大小但是FIFO和MFU的page\_fault卻上升了，跟上課說的是一模一樣，可以發現在魔幻數字123412512345下觀察到這個狀況，所以知道了不是一直增加page\_frame大小就可以使page\_fault減少，還是會有特例出現的。

我比較訝異的是MFU的大致效能盡然會跟FIFO差不多，可是應用上我覺得還是會有用到MFU的地方，不然只有LFU的話感覺會有餓死或是某一個霸佔太久的狀況會發生，而LRU加上LFU聽起來把兩個優化的東西加起來要更快，但實際上卻只有少少的例子是1+1=2大於前面的LRU或LFU，這次的總結是不是把任何最好的方法放在一起就是最好的方法，每個方法還是有它的優點存在的。