程式作業三

- **開發環境**

  Visual Studio Code (使用語言: Python)

- **實作方法和流程**

1. 使用的資料結構

List 串列 : 使用 list 存放 page frame 資訊。

2. 實作方法與流程

首先將 input 中的頁置換法對應方法數字、page frame 以及 reference string 讀入，再依指定方法進行不同的頁置換法(Page Replacement)。

以下是五種 Page Replacement :

- **FIFO :**
  - ✧ 首先會先判斷 page frame list 是否已經滿了，當 page frame list 還有空間可以放 page，就判斷此 page 是否已經存在於 list 中，若該 page 尚未存在於 list 中就將它放入。
  - ✧ 反之，當 page frame list 已經滿了，就判斷此 page 是否已經存在於 list 中，若該 page 尚未存在於 list 中，就要進行置換，選擇 list 中最前端的 page，因為它是在 frame 中待最久的，因此先 pop 掉此 page 之後，再將目前被參考到的 page 加入到 list 當中。
- **LRU :**
  - ✧ 首先會先判斷 page frame list 是否已經滿了，當 page frame list 還有空間可以放 page，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就先將它 pop 掉，之後在 insert 入 list 中，這是因為 LRU 是需要更新 page 的 time，因此在此我會將它重新

放入 list 已表示更新此 page 的時間，若該 page 尚未存在於 list 中就將它放入。

- ✧ 反之，當 page frame list 已經滿了，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就先將它 pop 掉，之後在 insert 入 list 中，若該 page 尚未存在於 list 中，就要進行置換，選擇 list 中最前端的 page，因為它是在 frame 中待最久的，因此先 pop 掉此 page 之後，再將目前被參考到的 page 加入到 list 當中。

- ➢ **LFU + FIFO :**

  - ✧ 首先會先判斷 page frame list 是否已經滿了，當 page frame list 還有空間可以放 page，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就將此 page 的 count 加 1，若該 page 尚未存在於 list 中就將它放入，且初始此 page 的 count 為 1。
  - ✧ 反之，當 page frame list 已經滿了，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就將此 page 的 count 加 1，若該 page 尚未存在於 list 中，就要進行置換，選擇 list 中 page 的 count 最小的 pop 掉，如果 pages 的 count 值相等，會依照 FIFO 的規則，置換掉待在 list 中最久的 page，再將目前被參考到的 page 加入到 list 當中。

- ➢ **MFU + FIFO :**

  - ✧ 首先會先判斷 page frame list 是否已經滿了，當 page frame list 還有空間可以放 page，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就將此 page 的 count 加 1，若該 page 尚未存在於 list 中就將它放入，且初始此 page 的 count 為 1。
  - ✧ 反之，當 page frame list 已經滿了，就判斷此 page 是否已經存在

於 list 中，若該 page 存在於 list 中就將此 page 的 count 加 1，若該 page 尚未存在於 list 中，就要進行置換，選擇 list 中 page 的 count 最大的 pop 掉，如果 pages 的 count 值相等，會依照 FIFO 的規則，置換掉待在 list 中最久的 page，再將目前被參考到的 page 加入到 list 當中。

- **LFU + LRU :**
  - ✧ 首先會先判斷 page frame list 是否已經滿了，當 page frame list 還有空間可以放 page，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就將此 page 的 count 加 1，並將此 page pop 掉之後重新放入 list 中，此目的是為了更新 page 的時間，若該 page 尚未存在於 list 中就將它放入，且初始此 page 的 count 為 1。
  - ✧ 反之，當 page frame list 已經滿了，就判斷此 page 是否已經存在於 list 中，若該 page 存在於 list 中就將此 page 的 count 加 1，並將此 page pop 掉之後重新放入 list 中，若該 page 尚未存在於 list 中，就要進行置換，選擇 list 中 page 的 count 最小的 pop 掉，如果 pages 的 count 值相等，會依照 LRU 的規則，置換掉待在 list 中最久的 page，再將目前被參考到的 page 加入到 list 當中。

◆ 以上為各個方法的實作流程，在執行各個頁至換法時都會記錄 page falut 與 page replace 發生的次數，最後透過各個頁至換法回傳的 ans list 與得到計算完的 page falut、page replace 次數，將這結果寫入 output 檔中。

● 分析不同方法之間的比較

下方表格藉由 input1、input2 中的 Reference string 來進行五種不同置換法的比較。

Page frame number 皆為 3

Input1 Reference String : 123412512345

Input2 Reference String : 70120304230321201701

	FIFO	LRU	LFU+FIFO	MFU+FIFO	LFU+LRU
Input1	**9**	**10**	**10**	**9**	**10**
Input2	**15**	**12**	**13**	**15**	**11**

表 1 : Page Fault 次數比較表

	FIFO	LRU	LFU+FIFO	MFU+FIFO	LFU+LRU
Input1	**6**	**7**	**7**	**6**	**7**
Input2	**12**	**9**	**10**	**12**	**8**

表 2 : Page Replacement 次數比較表

從上方四個表格中的結果值可以明顯看到各個方法中發生 Page Fault 的次數會大於 Page Replacement 的次數，這是因為發生 Page Fault 時不一定會發生 Page Replacement。像是在一開始 page frame list 還有空間可以放 page，但目前參考的 page 尚未存在於 page frame list 當中，在此時就會發生 Page Fault，但因為 list 中仍有空間可以存放 page，因此不會發生 Page Replacement。

- **結果與討論**

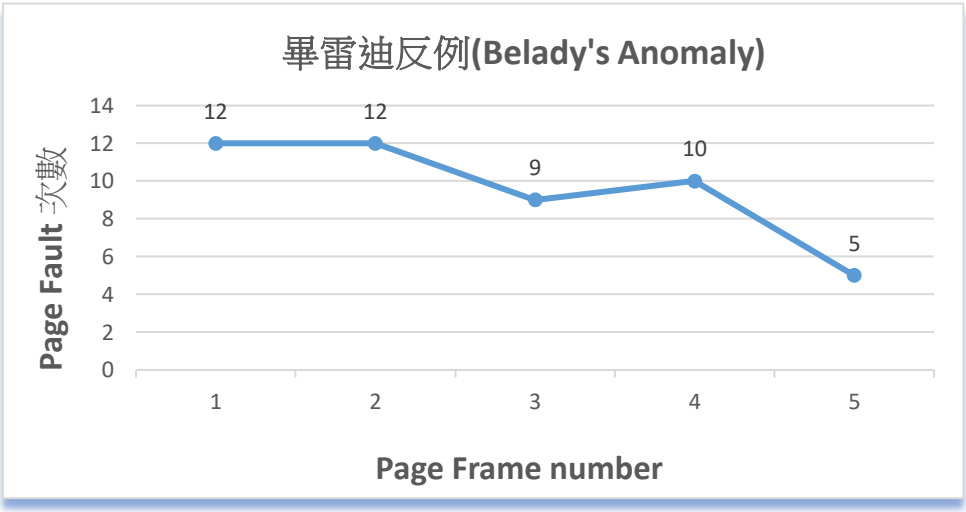
  - **畢雷迪反例(Belady's Anomaly)**

    頁置換法 : FIFO

Reference String : 234523623456

Page Frame number	1	2	3	4	5
Page Fault	12	12	9	10	5
Page Replacement	11	10	6	6	0

表 3 :畢雷迪反例的 Page Fault 次數與 Page Replacement 次數比較表



- ✧ 以上面例子為例，當 Page Frame number 增加到 4 時，Page Fault 發生的次數為 10，比 Page Frame number 為 3 時 Page Fault 發生的次數還多，因此可以得知當增加 Page Frame number，並不一定會減少 Page Fault 發生的次數!!!

➢ 實驗與發現

- ✧ 各個方法中發生 Page Fault 的次數會大於發生 Page Replacement 的次數。
- ✧ 發生 Page Fault 的次數與發生 Page Replacement 的次數之間的差距剛好會是 Page frame number 的大小。
- ✧ 對於每個方法的結果值好與壞取決於輸入的 Reference String 與 Page Frame。