Page Replacement Algorithms

จัดทำโดย

นายสุริยา เตชะลือ 600610790

เสนอ

รศ.ดร.นริศรา เอี่ยมคณิตชาติ

สารบัญ

Concept & code of 3 algorithm

First In First Out (FIFO)

แนวคิดในการพัฒนา algorithm

สร้างฟังก์ชันที่รับตัวแปร DataList และ frame เพื่อรับ List ของ reference string และ จำนวนของ frame ตามลำดับ จากนั้นสร้าง frameList ขนาดเท่ากับ frame โดยสมาชิกทุกตัวเริ่มต้นที่ -1 และ pointFrameList เพื่อชี้ไปยังตำแหน่ง ของ frameList ที่เก่าที่สุด โดย มีการประมวลผลทุก ๆ ตัวที่รับเข้ามา และหากว่าไม่พบข้อมูลจาก dataList ใน frameList ให้เพิ่มจำนวน PageFaults ขึ้นอีก 1 จากนั้นเปลี่ยนข้อมูลใน frameList ที่ตำแหน่งเก่าที่สุด (pointFrameList) และเปลี่ยนตำแหน่งของข้อมูลที่เก่าที่สุดไปอีก 1 โดยฟังก์ชัน FIFO return จำนวนของ PageFaults

code

```
def FIFO (dataList, frame):
    pageFaults = 0
    frameList = [-1] * frame
    pointFrameList = 0
    for i in range(len(dataList)):
        pointFrameList = pointFrameList % frame
        if(frameList.count(dataList[i]) == 0):
            pageFaults += 1
            frameList[pointFrameList] = dataList[i]
            pointFrameList += 1

    return pageFaults
```

Optimal

แนวคิดในการพัฒนา algorithm

สร้างฟังก์ชันที่รับตัวแปร DataList และ frame เพื่อรับ List ของ reference string และ จำนวนของ frame ตามลำดับ จากนั้นสร้าง frameList ขนาดเท่ากับ frame โดยสมาชิกทุกตัวเริ่มต้นที่ -1 และเริ่มต้นโดยการนำข้อมูลช่วงต้นเท่ากับจำนวน frameList ไปใส่ยัง frameList และ pageFaults เท่ากับจำนวนของ frameList จากนั้นเช็คข้อมูลตัวอื่น ๆ ว่ามีอยู่ใน frameList หรือไม่ ถ้าหากไม่มี ให้ทำการเพิ่ม pageFaults ไปอีก 1 จากนั้นเช็คดูว่าข้อมูลที่อยู่ใน frameList ตัวใดจะถูกเรียกใช้ซ้ำหลังสุด ให้ทำการแทนที่ข้อมูลใหม่ในตำแหน่งที่ข้อมูลใน frameList ตัวใดจะถูกเรียกใช้ซ้ำหลังสุด โดยฟังก์ชัน Optimal return จำนวนของ PageFaults

code

```
def Optimal(dataList, frame):
    pageFaults = 0
    frameList = [-1] * frame
    point = 0
    for i in range(len(dataList)):
        if point < frame:
            frameList[i] = dataList[i]
            pageFaults += 1
            point += 1
        elif frameList.count(dataList[i]) == 0 :
            pageFaults += 1
            checkFar = [1] * frame
            for j in range(i+1, len(dataList)):
                if checkFar.count(1) <= 1:
                    break
                if frameList.count(dataList[j]) != 0:
                    checkFar[frameList.index(dataList[j])] = 0
            frameList[checkFar.index(1)] = dataList[i]

    return pageFaults
```

Least Recently Used (LRU)

แนวคิดในการพัฒนา algorithm

สร้างฟังก์ชันที่รับตัวแปร DataList และ frame เพื่อรับ List ของ reference string และ จำนวนของ frame ตามลำดับ จากนั้นสร้าง frameList ขนาดเท่ากับ frame โดยสมาชิกทุกตัวเริ่มต้นที่ -1 และเริ่มต้นโดยการนำข้อมูลช่วงต้นเท่ากับจำนวน frameList ไปใส่ยัง frameList และ pageFaults เท่ากับจำนวนของ frameList และการอัพเดทข้อมูลทุกครั้งจะเรียกฟังก์ชัน UpdateOlder เพื่อบอกเวลาแต่ละตัวใน frameList หากมีข้อมูลที่ไม่ตรงกับใน frameList เข้ามาให้ไปแทนที่ยังตำแหน่งที่เก่าที่สุดโดยอ้างอิงข้อมูลจาก countOlder จากนั้น ตำแหน่งที่แทนที่ countOlder ถูกอัพเดทเป็น 1 โดยฟังก์ชัน RLU return จำนวนของ PageFaults

code

```
def UpdateOlder(countOlder, frameList):
    for i in range(len(countOlder)):
        if frameList[i] != -1:
            countOlder[i] += 1
    return countOlder


def RLU (dataList, frame):
    pageFaults = 0
    frameList = [-1] * frame
    countOlder = [0] * frame

    for i in range(frame):
        frameList[i] = dataList[i]
        countOlder = UpdateOlder(countOlder, frameList)
        pageFaults += 1
    for j in range(frame, len(dataList)):
        if(frameList.count(dataList[j]) == 0):
            pageFaults += 1
            frameList[countOlder.index(max(countOlder))] = dataList[j]
            countOlder[countOlder.index(max(countOlder))] = 0
        else:
```

```
            countOlder[frameList.index(dataList[j])] = 0

        countOlder = UpdateOlder(countOlder, frameList)


    return pageFaults
```

การทดลอง

อธิบายสมมติฐานของแต่ละการทดลอง

ทำการทดลอง 15 ครั้ง โดยใช้ของมูล reference string 5 รูปแบบ กับ 3 อัลกอริทึม

- FIFO สามารถทำงานได้ดีเป็นอันดับสอง รองจาก optimal เนื่องจากมีการกำหนดคิวที่แน่นอนแต่เริ่มต้นที่ข้อมูลเข้ามาและจะออกไป
- Optimal ทำงานได้ดีที่สุดเนื่องจากสามารถคาดเดาข้อมูลที่จะใช้ได้ในอนาคต
- RLU จะทำงานได้ดีหากมีการเรียกใช้ข้อมูลตัวเดิมซ้ำ ๆ

อธิบายรูปแบบ reference string ที่ใช้ในแต่ละการทดลอง

ในการทดลองมี reference string 5 รูปแบบที่มีความยาวต่างกันคือ 10 20 30 40 และ 50 โดยมีการสร้างแบบสุ่มที่ทุกตัวมีโอกาสเกิดขึ้นเท่ากันทุกตัว

Data List 1

3	0	1	5	1	6	4	2	6	0

Data List 2

2	0	5	1	4	3	0	0	3	1
3	0	4	6	1	0	5	6	6	4

Data List 3

0	5	1	1	0	0	3	5	5	1
5	6	1	4	6	4	6	3	5	1
3	6	5	1	6	0	3	2	2	3

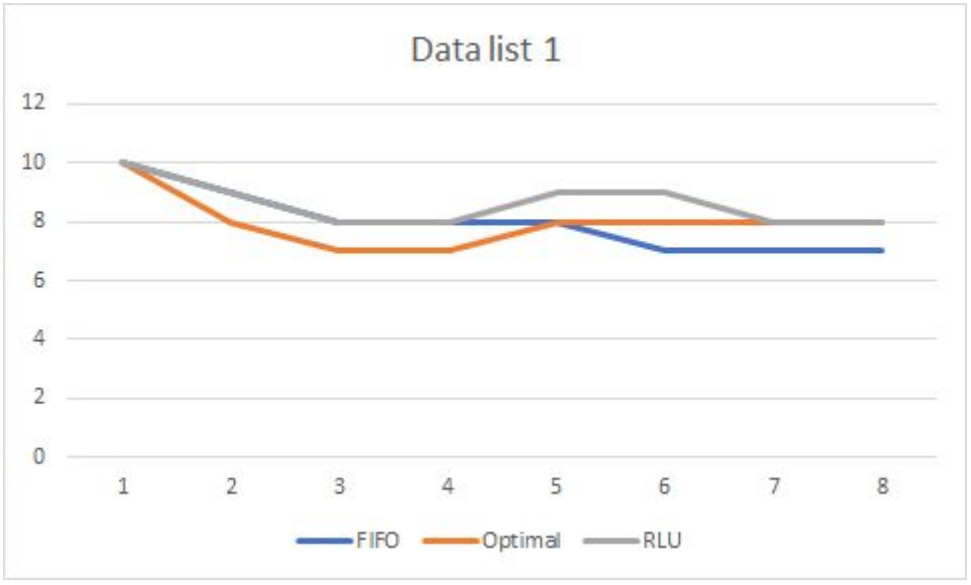
Data List 4

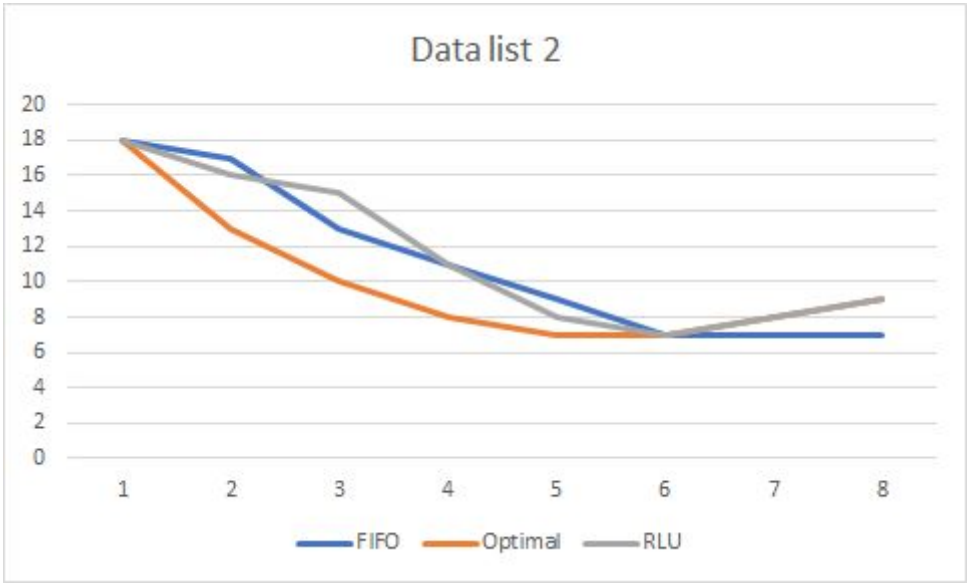
3	6	0	6	0	6	2	2	6	3
2	1	5	5	6	2	2	4	0	5
6	5	3	3	6	0	3	2	1	3
4	3	6	1	0	1	0	5	1	6

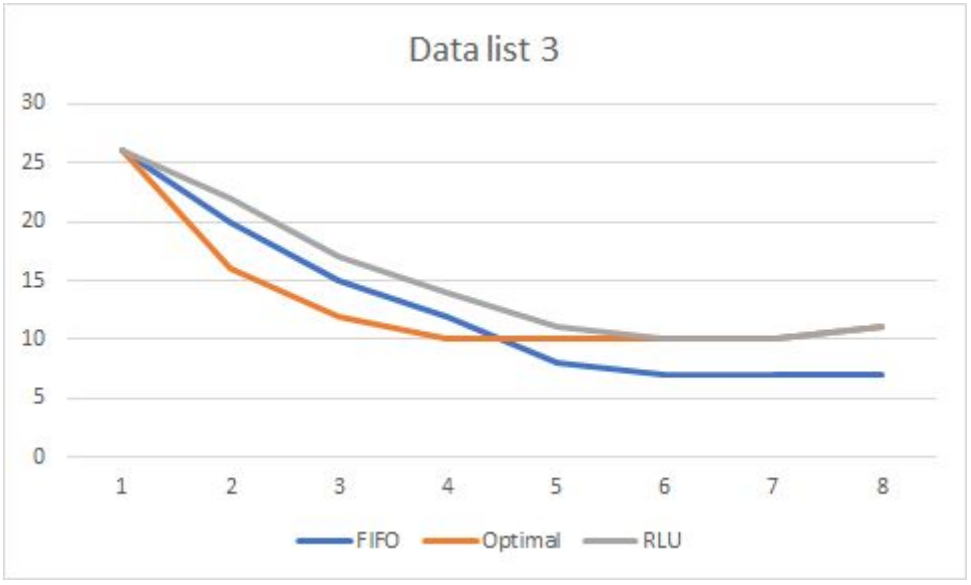
Data List 5

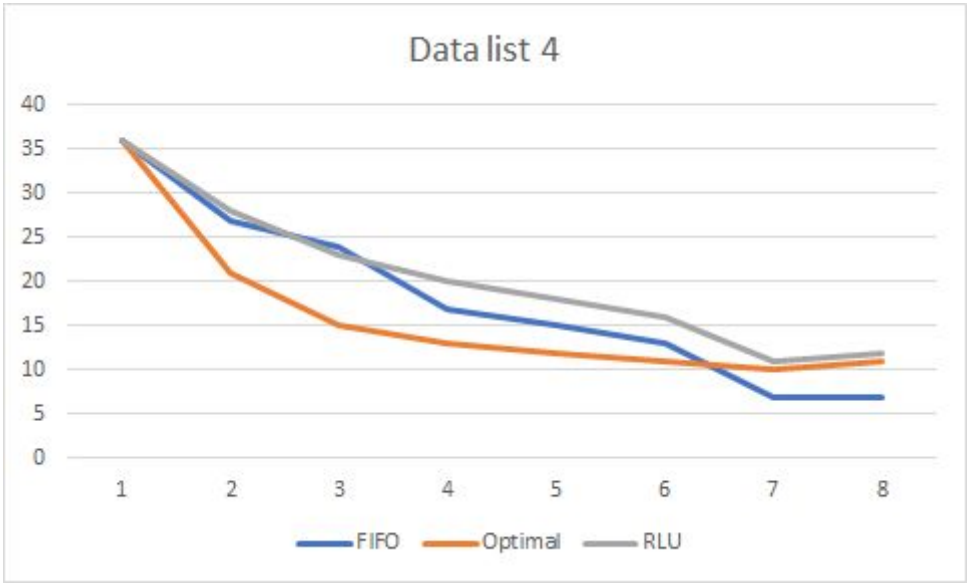
6	6	0	1	1	0	6	2	0	2
2	1	1	3	2	4	1	5	2	2
1	1	0	2	4	6	4	4	3	3
3	6	3	2	0	5	1	0	5	6
1	2	4	0	5	4	3	1	6	5

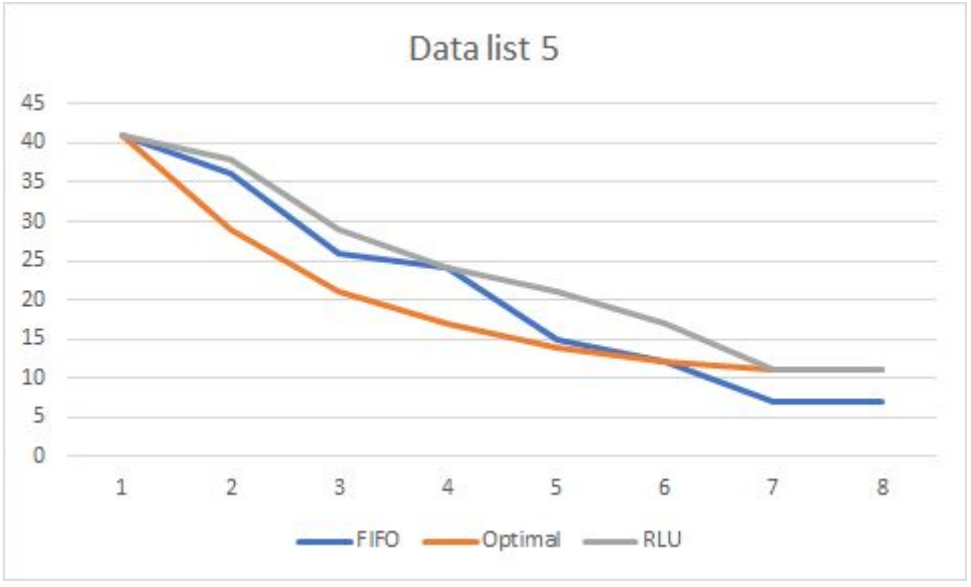
การประเมินผลการทดลองของแต่ละ algorithm






Data list 3
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
FIFO
Optimal
RLU


Data list 4
40
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
FIFO
Optimal
RLU


Data list 5
45
40
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
FIFO
Optimal
RLU

สรุปผลการทดลองของแต่ละการทดลอง

จากการทดลองพบว่า Optimal นั้นมีจำนวน pageFaults ที่น้อยเมื่อมีจำนวนจำนวน frame ที่รองรับน้อย แต่เมื่อมีจำนวน frame ที่มากขึ้นจะเห็นได้ว่า FIFO นั้นทำงานได้ดีกว่า อาจจะเป็นผลมาจาก reference string มีรูปแบบสุ่มที่ไม่เกิดขึ้นเป็นลำดับคล้าย ๆ กัน

อ้างอิง

https://github.com/fsuriya/OS/tree/master/HW02