자료구조 과제#1

리스트를 활용하여 LRU 시뮬레이터 구현하기

20231765 박종승

1) 파이썬 라이브러리인 배열을 활용한 시뮬레이터 코드 – 파일명: LRU_sim1

```
class CacheSimulator:
    def __init__(self, cache_slots):
        self.cache_slots = cache_slots
        self.cache = []
        self.cache_hit = 0
        self.tot_cnt = 1

    def do_sim(self, page):
        if page in self.cache:
            self.cache.remove(page)
            self.cache.append(page)
            self.cache_hit += 1
        else:
            if len(self.cache) == self.cache_slots:
                self.cache.pop(0)
            self.cache.append(page)
        self.tot_cnt += 1

    def print_stats(self):
        print("cache_slot = ", self.cache_slots, "cache_hit = ",
self.cache_hit, "hit ratio = ", self.cache_hit / self.tot_cnt)


if __name__ == "__main__":
    data_file = open("./linkbench.trc")
    lines = data_file.readlines()
    for cache_slots in range(100, 1001, 100):
        cache_sim = CacheSimulator(cache_slots)
        for line in lines:
            page = line.split()[0]
            cache_sim.do_sim(page)

        cache_sim.print_stats()
```

캐시 시뮬레이션을 수행하는 함수를 기존 코드에서 추가적으로 작성하였다.

page 의 캐시 여부를 확인한 후 존재할 시 해당 페이지에서 캐시를 제거한 후 다시 캐시에 추가함으로서 캐시 히트 발생시킨다. Page에 캐시가 존재하지 않을 경우, 캐시를 추가한다. 이때 캐시가 가득 차게 될 경우에 가장 오래된 페이지를 캐시에서 제거한다. 그리고 총 요청횟수를 증가하여 기록한다.

실행결과 화면:

```
jo\list\LRU_sim1.py"
cache_slot =  100 cache_hit =  14554 hit ratio =  0.14553854461455384
cache_slot =  200 cache_hit =  15377 hit ratio =  0.15376846231537686
cache_slot =  300 cache_hit =  16001 hit ratio =  0.16000839991600083
cache_slot =  400 cache_hit =  16665 hit ratio =  0.16664833351666483
cache_slot =  500 cache_hit =  17628 hit ratio =  0.17627823721762784
cache_slot =  600 cache_hit =  18796 hit ratio =  0.1879581204187958
cache_slot =  700 cache_hit =  20387 hit ratio =  0.2038679613203868
cache_slot =  800 cache_hit =  23947 hit ratio =  0.23946760532394676
cache_slot =  900 cache_hit =  26340 hit ratio =  0.26339736602633973
cache_slot =  1000 cache_hit =  28110 hit ratio =  0.2810971890281097
PS C:\Users\parkj\OneDrive\바탕 화면\숭실대\2-1\자료구조>
```

2) 파이썬의 연결리스트 CircularLinkedlist를 이용한 시뮬레이터 코드 – 파일명: LRU_sim2

```python
from circularLinkedList import CircularLinkedList

class CacheSimulator:
    def __init__(self, cache_slots):
        self.cache_slots = cache_slots
        self.cache = CircularLinkedList()
        self.cache_hit = 0
        self.tot_cnt = 1

    def do_sim(self, page):
        if self.cache.size() < self.cache_slots:
            if self.cache.index(page) != -2:
                self.cache_hit += 1
            else:
                self.cache.append(page)
        else:
            if self.cache.index(page) != -2:
                self.cache.remove(page)
                self.cache.append(page)
                self.cache_hit += 1
            else:
                self.cache.remove(self.cache.getNode(0).item)
                self.cache.append(page)
        self.tot_cnt += 1

    def print_stats(self):
        print("cache_slot = ", self.cache_slots, "cache_hit = ",
self.cache_hit, "hit ratio = ", self.cache_hit / self.tot_cnt)
```

```
if __name__ == "__main__":
    data_file = open("./linkbench.trc")
    lines = data_file.readlines()
    for cache_slots in range(100, 1001, 100):
        cache_sim = CacheSimulator(cache_slots)
        for line in lines:
            page = line.split()[0]
            cache_sim.do_sim(page)

        cache_sim.print_stats()
```

우선 기존 실습시간에 했던 CircularLinkedlist를 추가적으로 수정하였다.

(git에 수정된 CircularLinkedlist 다시 업로드함.)

CircularLinkedlist파일을 import해주고 기존의 시뮬레이터 코드에서 새롭게 추가 및 작성하였다.

실행결과 화면:

```
' '--' 'c:\Users\parkj\OneDrive\바탕 화면\숭실대\2-1\자료구조\jaryogujo\list\LRU_sim2.py'
cache_slot =  100 cache_hit =  14553 hit ratio =  0.14552854471455284
cache_slot =  200 cache_hit =  15376 hit ratio =  0.15375846241537586
cache_slot =  300 cache_hit =  16000 hit ratio =  0.15999840001599985
cache_slot =  400 cache_hit =  16664 hit ratio =  0.16663833361666383
cache_slot =  500 cache_hit =  17628 hit ratio =  0.17627823721762784
cache_slot =  600 cache_hit =  18795 hit ratio =  0.1879481205187948
cache_slot =  700 cache_hit =  20387 hit ratio =  0.2038679613203868
cache_slot =  800 cache_hit =  23947 hit ratio =  0.23946760532394676
cache_slot =  900 cache_hit =  26341 hit ratio =  0.26340736592634073
cache_slot =  1000 cache_hit =  28110 hit ratio =  0.2810971890281097
PS C:\Users\parkj\OneDrive\바탕 화면\숭실대\2-1\자료구조>
```

배열을 사용한 코드보다 일부 캐시 히트 값이 1씩 작게 결과값이 도출되었다. 처음엔 호출 카운트가 하나 배제돼서 발생한 문제라 생각하고 코드를 검토하였지만 일부 캐시 히트만 값이 다른것을 보아 다른 부분에서 발생된 오류라 판단하였다. 또한 hit ratio의 값을 포함하여 소수점 몇 자리의 오차범위내의 오차기에 감안하고 실습을 진행하였다.

3) 2번의 내용을 C언어로 구현하시오 - 파일명: LRU_sim3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {
    char *item;
    struct Node *next;
```

```c
} Node;

typedef struct CircularLinkedList {
    Node *head;
    int size;
} CircularLinkedList;
//circular를 초기화 시키는 함수
CircularLinkedList *initializeList() {
    CircularLinkedList *list = (CircularLinkedList
*)malloc(sizeof(CircularLinkedList));
    list->head = NULL;
    list->size = 0;
    return list;
}
// 리스트에 새로운 노드를 추가시킴
void append(CircularLinkedList *list, char *item) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->item = strdup(item);
    newNode->next = NULL;

    if (list->head == NULL) {
        list->head = newNode;
        newNode->next = list->head;
    } else {
        Node *temp = list->head;
        while (temp->next != list->head)
            temp = temp->next;
        temp->next = newNode;
        newNode->next = list->head;
    }
    list->size++;
}
// 리스트에서 특정 항목을 제거시킴
void removeItem(CircularLinkedList *list, char *item) {
    if (list->head == NULL)
        return;

    Node *current = list->head;
    Node *prev = NULL;

    do {
        if (strcmp(current->item, item) == 0) {
            if (prev == NULL) { // head에 있는 항목을 삭제하는 경우
                Node *last = list->head;
                while (last->next != list->head)
                    last = last->next;
                if (list->size == 1) {
                    free(list->head);
```

```
                    list->head = NULL;
                } else {
                    last->next = current->next;
                    free(current->item);
                    free(current);
                    list->head = last->next;
                }
            } else { // head를 제외한(중간이나 끝)에 있는 항목을 삭제하는 경우
                prev->next = current->next;
                free(current->item);
                free(current);
            }
            list->size--;
            return;
        }
        prev = current;
        current = current->next;
    } while (current != list->head);
}
//리스트에서 특정 항목속에 있는 인덱스를 찾는 함수
int indexOf(CircularLinkedList *list, char *item) {
    if (list->head == NULL)
        return -2;

    Node *current = list->head;
    int index = 0;

    do {
        if (strcmp(current->item, item) == 0)
            return index;
        index++;
        current = current->next;
    } while (current != list->head);

    return -2;
}

int size(CircularLinkedList *list) {
    return list->size;
}
//리스트 삭제
void destroyList(CircularLinkedList *list) {
    Node *current = list->head;
    Node *temp = NULL;

    if (list->head != NULL) {
        do {
            temp = current->next;
```

```
            free(current->item);
            free(current);
            current = temp;
        } while (current != list->head);
    }

    free(list);
}

typedef struct CacheSimulator {
    CircularLinkedList *cache;
    int cache_slots;
    int cache_hit;
    int tot_cnt;
} CacheSimulator;
//CacheSimulator 를 초기화 시킴
CacheSimulator *initializeCacheSimulator(int cache_slots) {
    CacheSimulator *cacheSimulator = (CacheSimulator
*)malloc(sizeof(CacheSimulator));
    cacheSimulator->cache = initializeList();
    cacheSimulator->cache_slots = cache_slots;
    cacheSimulator->cache_hit = 0;
    cacheSimulator->tot_cnt = 1;
    return cacheSimulator;
}
//CacheSimulator 수행함수
void doSim(CacheSimulator *cacheSimulator, char *page) {
    if (size(cacheSimulator->cache) < cacheSimulator->cache_slots) {
        if (indexOf(cacheSimulator->cache, page) != -2) {
            cacheSimulator->cache_hit++;
        } else {
            append(cacheSimulator->cache, page);
        }
    } else {
        if (indexOf(cacheSimulator->cache, page) != -2) {
            removeItem(cacheSimulator->cache, page);
            append(cacheSimulator->cache, page);
            cacheSimulator->cache_hit++;
        } else {
            removeItem(cacheSimulator->cache, cacheSimulator->cache->head-
>item);
            append(cacheSimulator->cache, page);
        }
    }
    cacheSimulator->tot_cnt++;
}
//최종적인 통계값을 출력하는 함수
void printStats(CacheSimulator *cacheSimulator) {
```

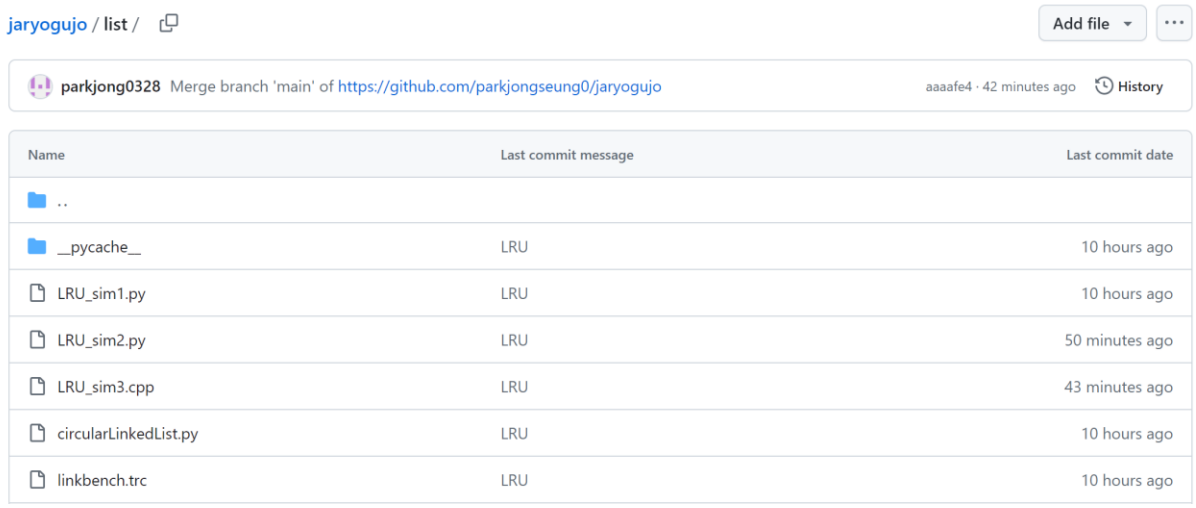
```
    printf("cache_slot = %d, cache_hit = %d, hit ratio = %f\n",
           cacheSimulator->cache_slots,
           cacheSimulator->cache_hit,
           (float)cacheSimulator->cache_hit / cacheSimulator->tot_cnt);
}

void destroyCacheSimulator(CacheSimulator *cacheSimulator) {
    destroyList(cacheSimulator->cache);
    free(cacheSimulator);
}
//메인 함수
int main() {
    FILE *data_file = fopen("./linkbench.trc", "r");
    if (data_file == NULL) {
        perror("Error opening file");
        return -1;
    }

    char line[256];
// 캐시 슬롯 수를 100 부터 1000 까지 100 씩 증가하면서 테스트함
    for (int cache_slots = 100; cache_slots <= 1000; cache_slots += 100) {
        rewind(data_file);
        CacheSimulator *cache_sim = initializeCacheSimulator(cache_slots);
        while (fgets(line, sizeof(line), data_file)) {

            line[strcspn(line, "\n")] = 0;
            char *page;
            char *token = strtok(line, " ");
            while (token != NULL) {
                page = strdup(token);
                doSim(cache_sim, page);
                free(page);
                token = strtok(NULL, " ");
            }
        }
        printStats(cache_sim);
        destroyCacheSimulator(cache_sim);
    }

    fclose(data_file);
    return 0;
}
```

파이썬으로 작성한 코드를 C로 새롭게 작성하는 과정에서 파이썬에선 기본적으로 가능한 파일 불러오기나 내장함수, 메모리 할당 및 문자열 처리등에 대해 파이썬보다 더욱 정교하고 많은 함수를 요구하여 코드의 길이가 매우 길어졌다. 두번째 실습과 동일한 목적을 가지고 같은 파일을 다루는 코드기에 결과값이 동일하게 나옴을 확인할 수 있다.

실행결과 화면:

```
cache_slot = 100, cache_hit = 14553, hit ratio = 0.145529
cache_slot = 200, cache_hit = 15376, hit ratio = 0.153758
cache_slot = 300, cache_hit = 16000, hit ratio = 0.159998
cache_slot = 400, cache_hit = 16664, hit ratio = 0.166638
cache_slot = 500, cache_hit = 17628, hit ratio = 0.176278
cache_slot = 600, cache_hit = 18795, hit ratio = 0.187948
cache_slot = 700, cache_hit = 20387, hit ratio = 0.203868
cache_slot = 800, cache_hit = 23947, hit ratio = 0.239468
cache_slot = 900, cache_hit = 26341, hit ratio = 0.263407
cache_slot = 1000, cache_hit = 28110, hit ratio = 0.281097
PS C:\Users\parkj\OneDrive\바탕 화면\숭실대\2-1\자료구조\jaryogujo\list>
```

코드 업로드한 Git 링크 : https://github.com/parkjongseung0/jaryogujo/tree/main/list



-Git push 화면(LRU_sim1,2,3,circularlinkedlist)