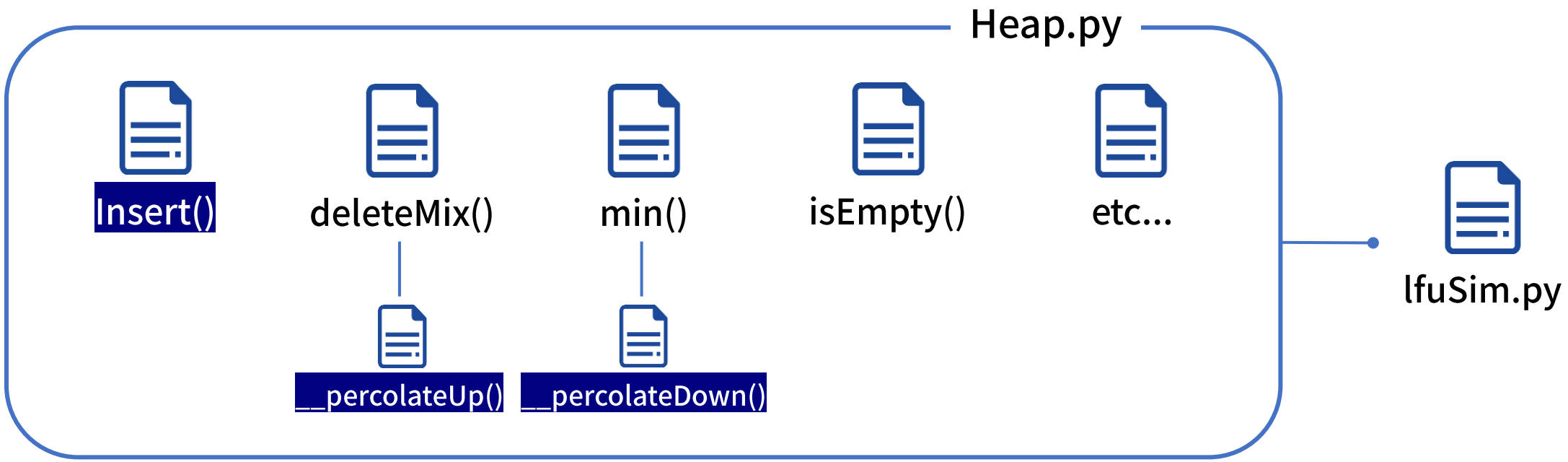
프로그램 구조



- 최소힙, lpn-frequency 관리를 위해 기존 Heap에서 재구성한 부분

Heap.py 코드

```python
class Heap:
    def __init__(self, *args):
        if len(args) != 0:
            self.__A = args[0]
        else:
            self.__A = []

    def insert(self, lpn, frequency):
        lpn_list = [node[0] for node in self.__A] #lpn 값만 리스트로 받아오기
        is_hitted = False

        if lpn not in lpn_list:
            self.__A.append([lpn,frequency])
            is_hitted = False
        else:
            lpn_index = lpn_list.index(lpn)
            self.__A[lpn_index][1] += 1
            frequency = self.__A[lpn_index][1]
            is_hitted = True

        self.__percolateUp(len(self.__A)-1)

        return is_hitted

    def __percolateUp(self, i:int):
        parent = (i - 1) // 2
        if i > 0 and self.__A[i][1] < self.__A[parent][1]: #frequency를 기준으로 동작하도록 변경
            self.__A[i], self.__A[parent] = self.__A[parent], self.__A[i]
            self.__percolateUp(parent)
```

Heap을 리스트에 lpn과 frequency를 [lpn, frequency] 형식의 리스트로 묶어서 이중 리스트로 저장하도록 설계함

lpn_list에 Heap에 저장된 lpn들만 담고
lpn_list에 입력받은 lpn이 이미 있는지 확인함

cache안에서 lpn이 변하면 hit이므로

lpn_list 안에 lpn이 없으면 is_hitted를 False로 설정하고 [lpn, frequency]를 새로 저장

lpn_list 안에 lpn이 들어있으면 is_hitted를 True로 설정하고 해당 lpn의 frequency만 1 증가시킴

스며오르기의 비교가 원소들의 frequency를 기준으로 동작하도록 변경하고 최소힙을 만들기 위해 작은 값이 위로 올라가도록 설정함

- 최소힙, lpn-frequency 관리를 위해 기존 Heap에서 재구성한 부분

Heap.py 코드

```python
def deleteMin(self):
    # heap is in self.__A[0...len(self.__A)-1]
    if (not self.isEmpty()):
        min = self.__A[0]
        self.__A[0] = self.__A.pop()
        self.__percolateDown(0)
        return min
    else:
        return None
```

deleteMax 함수가 아니라 최솟값을 삭제하도록
deleteMin 함수 사용

```python
def __percolateDown(self, i:int):
    # Percolate down w/ self.__A[i] as the root
    child = 2 * i + 1
    right = 2 * i + 2  # right child
    if (child <= len(self.__A)-1):
        if (right <= len(self.__A)-1 and self.__A[child][1] > self.__A[right][1]):
            child = right  # index of smaller child
        if self.__A[i][1] >= self.__A[child][1]:
            self.__A[i], self.__A[child] = self.__A[child], self.__A[i]
            self.__percolateDown(child)
```

스며오르기의 비교가 원소들의 frequency를 기준으로 동작하도록 변경하고 최소힙을 만들기 위해 큰 값이 밑으로 내려오도록 설정함

```python
def isHitted(self):
    return self.is_hitted

def min(self):
    return self.__A[0]

def isEmpty(self) -> bool:
    return len(self.__A) == 0
```

max 함수가 아니라 최솟값을 리턴하도록
min 함수 사용

- 최소힙, lpn-frequency 관리를 위해 기존 Heap에서 재구성한 부분

lfuSim.py 코드

```python
from Heap import *

def lfu_sim(cache_slots):
  cache_hit = 0
  tot_cnt = 0

  data_file = open("src\ds_2023-main\lfu_sim\linkbench.trc")
  memory = {}
  cache = Heap()

  for line in data_file.readlines():
    lpn = line.split()[0]

    # Program here
    if lpn in memory: memory[lpn] += 1
    else: memory[lpn] = 1

    frequency = memory[lpn]

    if cache.insert(lpn, frequency): cache_hit += 1 #cache.insert()는 ishitted를 리턴
    elif cache.size() > cache_slots: cache.deleteMin() #캐쉬가 가득차면 freqency가 최소인 노드를 삭제

    tot_cnt +=1

  print("cache_slot = ", cache_slots, "cache_hit = ", cache_hit, "hit ratio = ", cache_hit / tot_cnt)

if __name__ == "__main__":
  for cache_slots in range(100, 1000, 100):
    lfu_sim(cache_slots)
```

Frequency를 초기화하지 않고 전역적으로 동작하기 위해서 모든 데이터를 저장해놓는 {lpn:frequency} 형식의 memory 딕셔너리를 생성함

memory에 lpn이 이미 존재하면 (이전의 접근 기록이 있으면) frequency에 1을 더하고 lpn이 없으면 새로운 데이터를 (접근 기록을) 생성

memory에서 해당 lpn의 frequency를 받아오고 cache에 lpn과 frequency을 추가

cache안에서 lpn이 변하면 hit이므로 hit일 때는 cache size가 변하지 않음

만약 cache_slots이 가득찬 상태에서 새로운 데이터가 들어오면(메모리를 접근하면) deleteMin()을 통해 frequency가 가장 적은 (접근 횟수가 가장 적은) 데이터를 삭제

실행결과

```
C:\Users\Owner\OneDrive\바탕 화면\대학 과제\2학년_1학기\자료구조>C:/Python/Python310/python.exe "c:/Users/Owner/Onedrive/바탕 화면/대학 과제/2학년_1학기/자료구조/src/
ds_2023-main/lfu_sim/lfuSim.py"
cache_slot =  100 cache_hit =  21288 hit ratio =  0.21288
cache_slot =  200 cache_hit =  27206 hit ratio =  0.27206
cache_slot =  300 cache_hit =  32448 hit ratio =  0.32448
cache_slot =  400 cache_hit =  32688 hit ratio =  0.32688
cache_slot =  500 cache_hit =  32907 hit ratio =  0.32907
cache_slot =  600 cache_hit =  33142 hit ratio =  0.33142
cache_slot =  700 cache_hit =  33143 hit ratio =  0.33143
cache_slot =  800 cache_hit =  33387 hit ratio =  0.33387
cache_slot =  900 cache_hit =  33651 hit ratio =  0.33651
```

감점하시진 않겠죠? 왜 값이 다른진 모르겠는데 사랑해요 교수님 🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍🤍