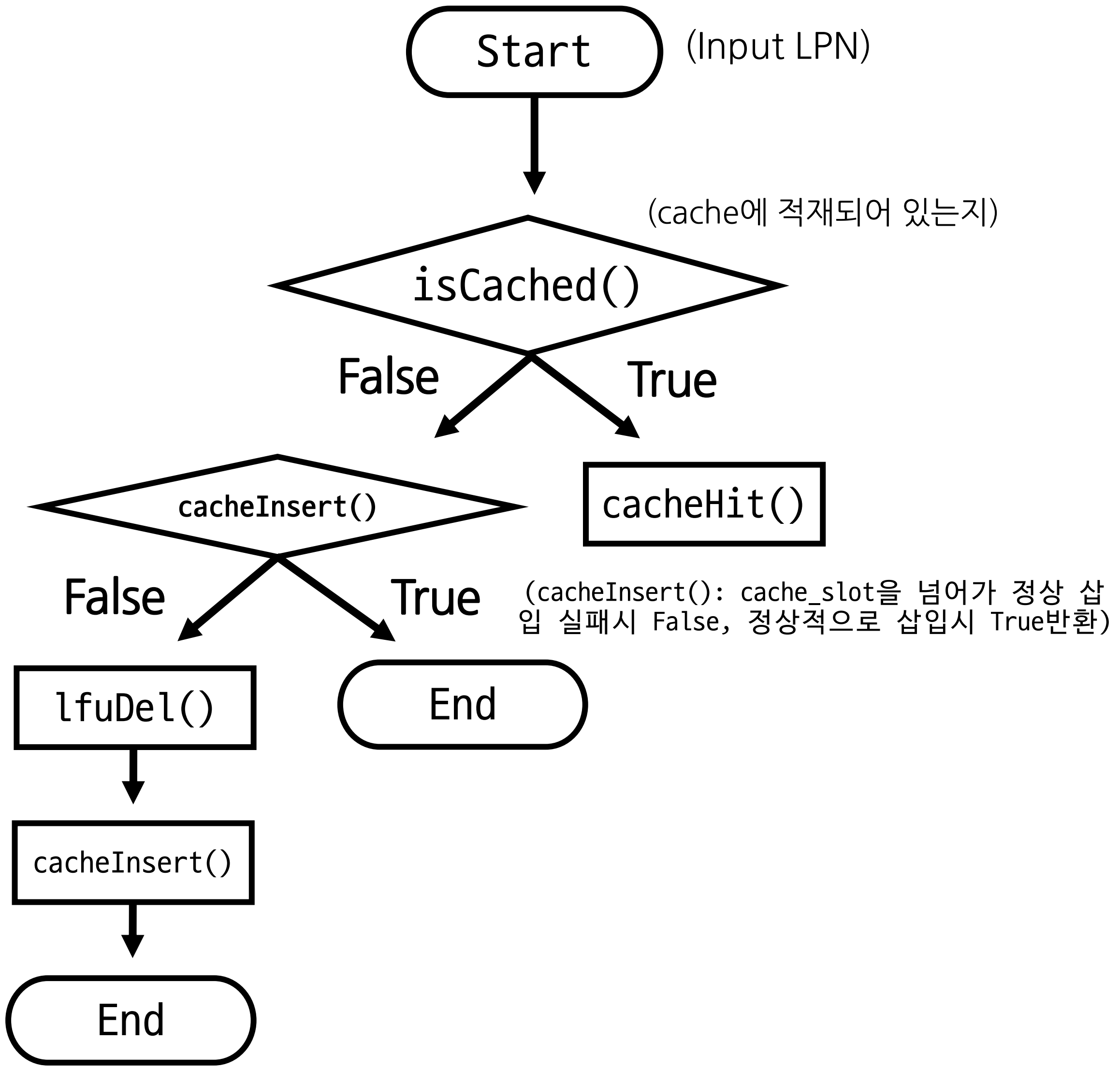
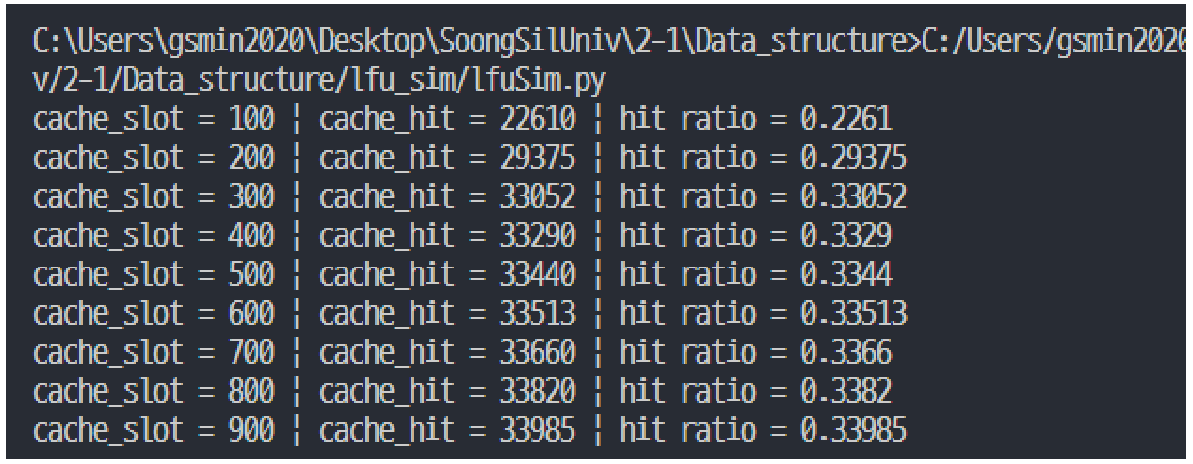
자료구조 LFU 시뮬레이터 (가)반 20203063 강수민

<Flow chart>

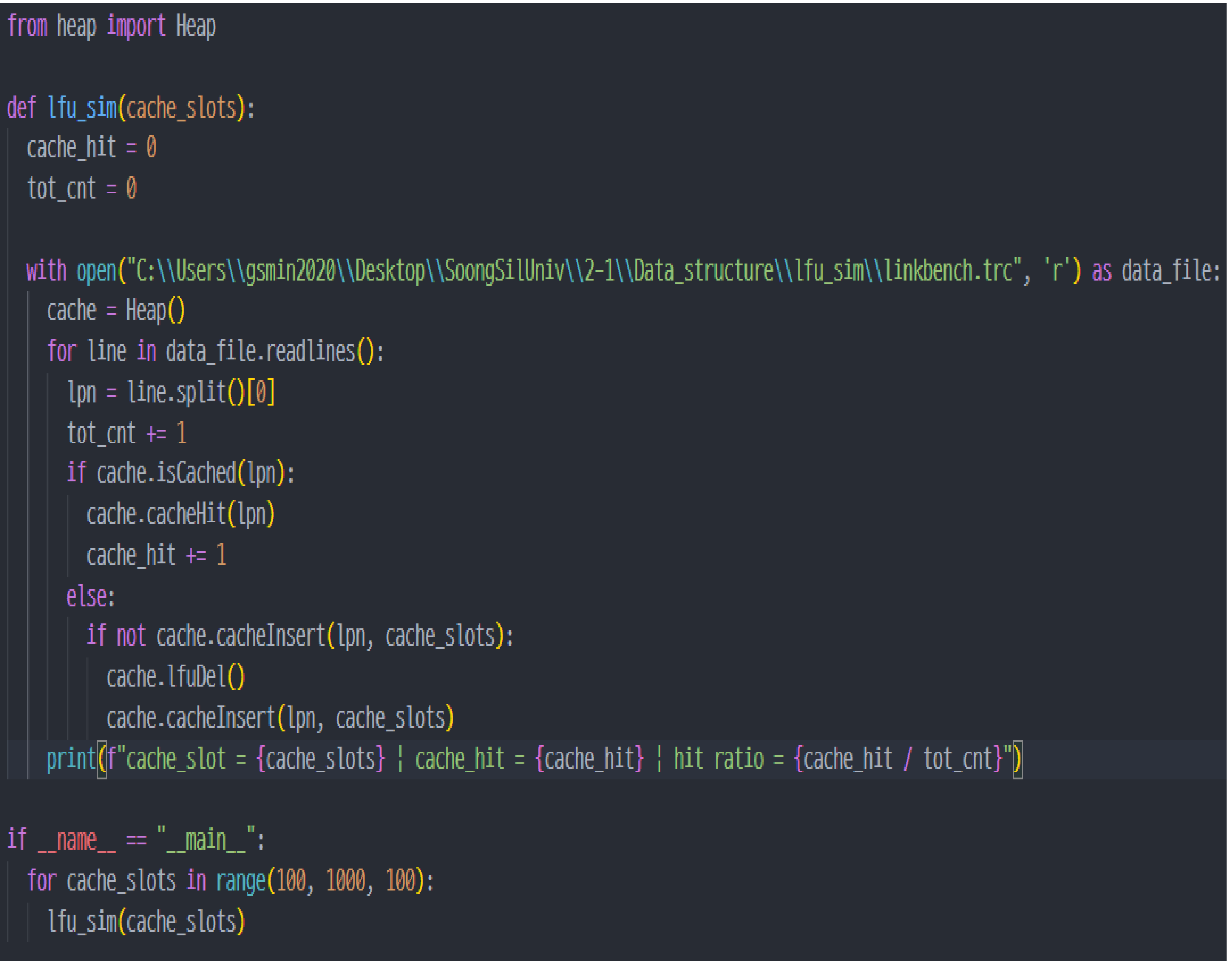

Start
(Input LPN)
(cache에 적재되어 있는지)
isCached()
False
True
cacheInsert()
cacheHit()
False
True
(cacheInsert(): cache_slot을 넘어가 정상 삽입 실패시 False, 정상적으로 삽입시 True반환)
lfuDel()
End
cacheInsert()
End

<Result>



```
C:\Users\gsmin2020\Desktop\SoongSilUniv\2-1\Data_structure>C:/Users/gsmin202
v/2-1/Data_structure/lfu_sim/lfuSim.py
cache_slot = 100 ¦ cache_hit = 22610 ¦ hit ratio = 0.2261
cache_slot = 200 ¦ cache_hit = 29375 ¦ hit ratio = 0.29375
cache_slot = 300 ¦ cache_hit = 33052 ¦ hit ratio = 0.33052
cache_slot = 400 ¦ cache_hit = 33290 ¦ hit ratio = 0.3329
cache_slot = 500 ¦ cache_hit = 33440 ¦ hit ratio = 0.3344
cache_slot = 600 ¦ cache_hit = 33513 ¦ hit ratio = 0.33513
cache_slot = 700 ¦ cache_hit = 33660 ¦ hit ratio = 0.3366
cache_slot = 800 ¦ cache_hit = 33820 ¦ hit ratio = 0.3382
cache_slot = 900 ¦ cache_hit = 33985 ¦ hit ratio = 0.33985
```

<lfuSim.py>



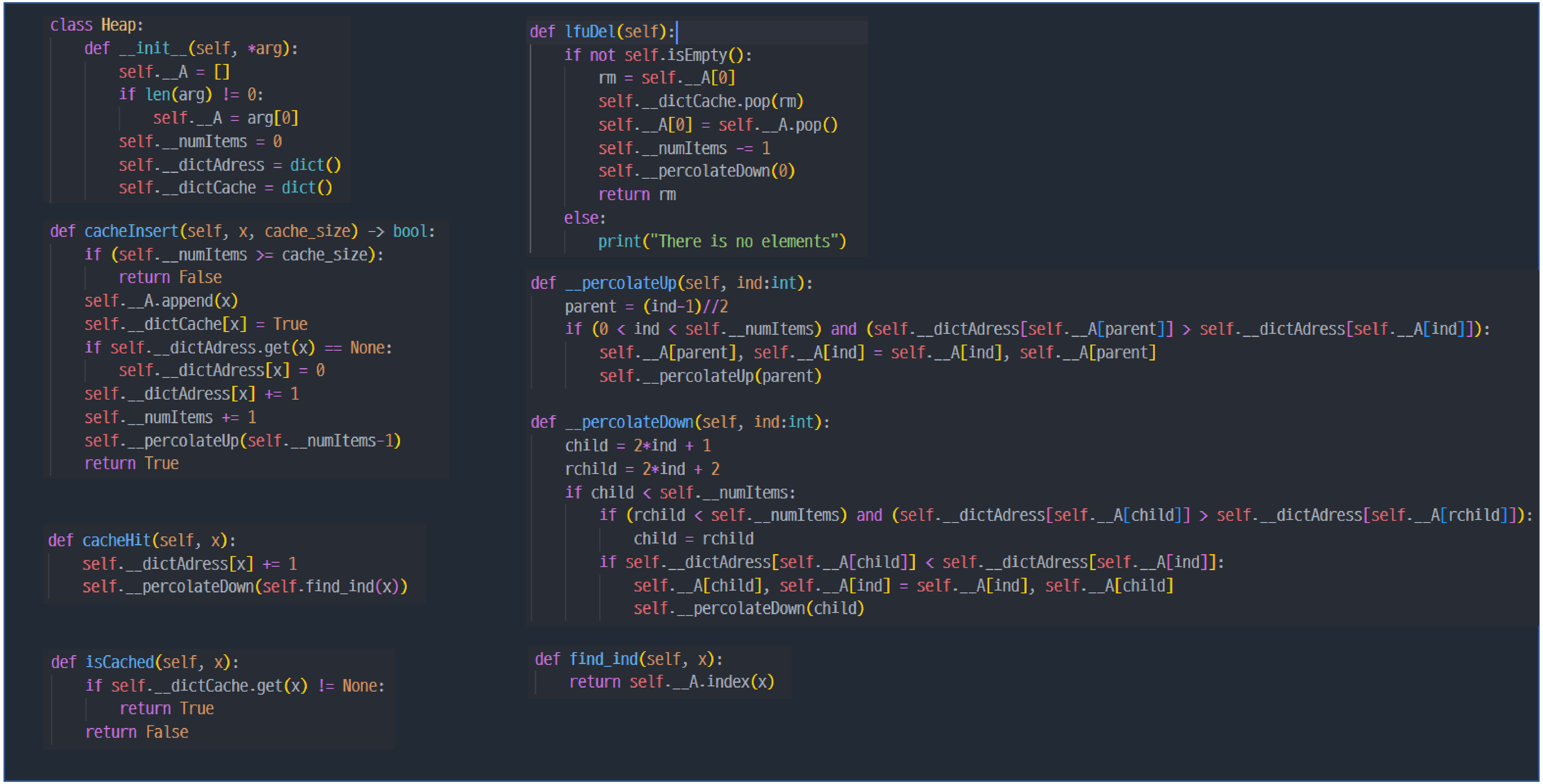
```python
from heap import Heap

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

  with open("C:\\Users\\gsmin2020\\Desktop\\SoongSilUniv\\2-1\\Data_structure\\lfu_sim\\linkbench.trc", 'r') as data_file:
    cache = Heap()
    for line in data_file.readlines():
      lpn = line.split()[0]
      tot_cnt += 1
      if cache.isCached(lpn):
        cache.cacheHit(lpn)
        cache_hit += 1
      else:
        if not cache.cacheInsert(lpn, cache_slots):
          cache.lfuDel()
          cache.cacheInsert(lpn, cache_slots)
    print(f"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)
```

<Class Heap 새로 추가/수정한 부분>

🟩 : 추가 🟨 : 수정

- **self.__dictAdress:** 프로그램 시작부터 종료까지 주소별 등장 횟수를 저장하는 딕셔너리
- **Self.__dictCache:** 현재 LPN에 저장된 주소를 저장하는 딕셔너리
- **cacheInsert(adress, size):** adress 삽입 성공시 True, 만약 size를 넘어가면 False반환
- **cacheHit(adress):** __dictAdress속 address의 value를 1증가 시킨 후, heap을 갱신해준다.
- **isCached(address):** dictCache key값 안에 해당하는 주소가 있으면 True 아니면, False 반환
- **lfuDel():** 가장 적게 호출된 adress를 반환하고, heap을 갱신 한다.
- **__percolateUp(index), __percolateDown(index):** 기존 함수는 호출 횟수가 아닌, 주소값을 대소비교. 이를 주소 등장 횟수로 대소비교하는 것으로 바꿈.
- **findInd(address):** 선형적으로 adress의 index를 찾는다.

<heap.py> 추가/수정한 부분 소스코드



```python
class Heap:
    def __init__(self, *arg):
        self.__A = []
        if len(arg) != 0:
            self.__A = arg[0]
        self.__numItems = 0
        self.__dictAdress = dict()
        self.__dictCache = dict()
```

```python
def cacheInsert(self, x, cache_size) -> bool:
    if (self.__numItems >= cache_size):
        return False
    self.__A.append(x)
    self.__dictCache[x] = True
    if self.__dictAdress.get(x) == None:
        self.__dictAdress[x] = 0
    self.__dictAdress[x] += 1
    self.__numItems += 1
    self.__percolateUp(self.__numItems-1)
    return True
```

```python
def cacheHit(self, x):
    self.__dictAdress[x] += 1
    self.__percolateDown(self.find_ind(x))
```

```python
def isCached(self, x):
    if self.__dictCache.get(x) != None:
        return True
    return False
```

```python
def lfuDel(self):
    if not self.isEmpty():
        rm = self.__A[0]
        self.__dictCache.pop(rm)
        self.__A[0] = self.__A.pop()
        self.__numItems -= 1
        self.__percolateDown(0)
        return rm
    else:
        print("There is no elements")
```

```python
def __percolateUp(self, ind:int):
    parent = (ind-1)//2
    if (0 < ind < self.__numItems) and (self.__dictAdress[self.__A[parent]] > self.__dictAdress[self.__A[ind]]):
        self.__A[parent], self.__A[ind] = self.__A[ind], self.__A[parent]
        self.__percolateUp(parent)

def __percolateDown(self, ind:int):
    child = 2*ind + 1
    rchild = 2*ind + 2
    if child < self.__numItems:
        if (rchild < self.__numItems) and (self.__dictAdress[self.__A[child]] > self.__dictAdress[self.__A[rchild]]):
            child = rchild
        if self.__dictAdress[self.__A[child]] < self.__dictAdress[self.__A[ind]]:
            self.__A[child], self.__A[ind] = self.__A[ind], self.__A[child]
            self.__percolateDown(child)
```

```python
def find_ind(self, x):
    return self.__A.index(x)
```