(http://www.uff.br)
(http://www.ic.uff.br)

Disciplina: Sistemas de Computação
Alunos: Eduardo Vieira Queiroga e Rodrigo Alves Prado da Silva

Simulador de políticas de substituição de páginas de memória cache

1. Descrição do projeto e implementação

O projeto consiste da implementação de um simulador das seguintes políticas de substituição de páginas de memória cache: First In First Out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU) e Random. O projeto foi implementado na linguagem Python (versão 3.6.0). Basicamente, uma classe foi criada para cada política e um parâmetro de chamada inserido pelo usuário determina qual classe (política) será instanciada e seus métodos utilizados para buscar e/ou adicionar/substituir a página requisitada em cada linha do arquivo de requisições (passado como parâmetro). Segue abaixo o código-fonte do simulador:

simulator.py

In [ ]:

```
# coding: iso-8859-1
import sys
import math
import random

class FIFO:

    def __init__(self, cacheSize):
        self.size = 0 # current cache memory size
        self.cache = [None] * cacheSize # create the virtual cache
        self.tail = 0 # pointer to the end of queue
        self.count = 0 # number of requests
        self.cacheHit = 0 # number of cache hits

    def search(self, obj): # Check if the page is already in memory
        self.count += 1
        for i in range(self.size):
            if obj == self.cache[i]:
                self.cacheHit += 1
                print("cache =",self.cache)
                return False # Cache Hit
        return True # Cache Miss
```

In [ ]:

```
    def update(self, obj): # Add or replace the request content
        self.cache[self.tail] = obj;
        self.tail = (self.tail + 1) % len(self.cache)
        if self.size < len(self.cache):
            self.size += 1
        print("cache =",self.cache)

    def printLog(self): # print simulation stats
        print("Total Requests: " + str(self.count) + \
              ", Total Cache Hit: " + str(self.cacheHit) + \
              ", Total Cache Miss: " + str(self.count - self.cacheHit) + \
              ", Cache Hit Ratio: " + str(round(self.cacheHit/self.count,2)) + \
              ", Cache Miss Ratio: " + \
              str( round( (self.count - self.cacheHit)/self.count, 2 ) ) )

class LRU:

    def __init__(self, cacheSize):
        self.size = 0 # current cache memory size
        self.cache = [None] * cacheSize # create the virtual cache
        self.age = [0] * cacheSize # array that stores the age of the pages
        self.count = 0  # number of requests
        self.cacheHit = 0 # number of cache hits

    def search(self, obj): # Check if the page is already in memory
        self.count += 1
        for i in range(self.size):
            if obj == self.cache[i]:
                self.cacheHit += 1
                self.age[i] = self.count

                print("cache =",self.cache, ", age =", self.age )

                return False
        return True

    def update(self, obj): # Add or replace the request content
        # while the cache is not full, add the new item at the end
        if self.size < len(self.cache):
            self.cache[self.size] = obj
            self.age[self.size] = self.count

            print("cache =",self.cache, ", age =", self.age )
            self.size += 1
        else: # if the cache is full, replace items through the LRU policy
            ageMin, ageMinPos = self.age[0], 0
            for i in range(1, self.size):
                if self.age[i] < ageMin:
                    ageMin = self.age[i]
                    ageMinPos = i

            self.cache[ageMinPos] = obj
            self.age[ageMinPos] = self.count

            print("cache =",self.cache, ", age =", self.age )

    def printLog(self): # print simulation stats
        print("Total Requests: " + str(self.count) + \
              ", Total Cache Hit: " + str(self.cacheHit) + \
```

```
            ", Total Cache Miss: " + str(self.count - self.cacheHit) + \

            ", Cache Hit Ratio: " + str(round(self.cacheHit/self.count,2)) + \
            ", Cache Miss Ratio: " + \
            str( round( (self.count - self.cacheHit)/self.count, 2 ) ) )

class LFU:

    def __init__(self, cacheSize):
        self.size = 0
        self.cache = [None] * cacheSize
        self.access = [0] * cacheSize
        self.count = 0
        self.cacheHit = 0

    def search(self, obj): # Check if the page is already in memory
        self.count += 1
        for i in range(self.size):
            if obj == self.cache[i]:
                self.cacheHit += 1
                self.access[i] += 1
                print("cache =",self.cache, ", access =", self.access )
                return False
        return True

    def update(self, obj): # Add or replace the request content
        # while the cache is not full, add the new item at the end
        if self.size < len(self.cache):
            self.cache[self.size] = obj
            self.access[self.size] = 1
            print("cache =",self.cache, ", access =", self.access )
            self.size += 1
        else: # if the cache is full, replace items through the LFU policy
            accessMin, accessMinPos = self.access[0], 0
            for i in range(1, self.size):
                if self.access[i] < accessMin:
                    accessMin = self.access[i]
                    accessMinPos = i

            self.cache[accessMinPos] = obj
            self.access[accessMinPos] = 1

            print("cache =",self.cache, ", access =", self.access )

    def printLog(self): # print simulation stats
        print("Total Requests: " + str(self.count) + \
              ", Total Cache Hit: " + str(self.cacheHit) + \
              ", Total Cache Miss: " + str(self.count - self.cacheHit) + \
              ", Cache Hit Ratio: " + str(round(self.cacheHit/self.count,2)) + \
              ", Cache Miss Ratio: " + \
              str( round( (self.count - self.cacheHit)/self.count, 2 ) ) )

class Random:

    def __init__(self, cacheSize):
        self.size = 0
        self.cache = [None] * cacheSize
        self.count = 0
        self.cacheHit = 0

    def search(self, obj): # Check if the page is already in memory
        self.count += 1
```

```
        for i in range(self.size):
            if obj == self.cache[i]:
                self.cacheHit += 1
                print("cache =",self.cache )
                return False # Cache Hit
        return True # Cache Miss

    def update(self, obj): # Add or replace the request content
        # while the cache is not full, add the new item at the end
        if self.size < len(self.cache):
            self.cache[self.size] = obj
            print("cache =",self.cache )
            self.size += 1
        else:
            rVal = random.randint(0, len(self.cache)-1 )
            self.cache[rVal] = obj
            print("cache =",self.cache )

    def printLog(self): # print simulation stats
        print("Total Requests: " + str(self.count) + \
              ", Total Cache Hit: " + str(self.cacheHit) + \
              ", Total Cache Miss: " + str(self.count - self.cacheHit) + \
              ", Cache Hit Ratio: " + str(round(self.cacheHit/self.count,2)) + \
              ", Cache Miss Ratio: " + \
              str( round( (self.count - self.cacheHit)/self.count, 2 ) ) )

def main(argv):

    requestFilePath = argv[1]
    cacheSize = int(argv[2])
    replacementPolicy = argv[3]

    Manager = []

    if replacementPolicy.lower() == "fifo":
        Manager = FIFO(cacheSize);
    elif replacementPolicy.lower() == "lru":
        Manager = LRU(cacheSize);
    elif replacementPolicy.lower() == "lfu":
        Manager = LFU(cacheSize);
    elif replacementPolicy.lower() == "random":
        Manager = Random(cacheSize);
    else:
        print("The replacement policy was not chosen correctly!")
        return

    requestFile = open(requestFilePath, "r")

    for request in requestFile:
        request = request.strip()
        cacheMiss = Manager.search(request)

        if cacheMiss:
            Manager.update(request)

    Manager.printLog()

    requestFile.close()

if __name__ == "__main__":
    main(sys.argv)
```

2. Instalação e execução

Sistemas operacionais modernos tipo Unix (e.g. Ubuntu, Linux Mint, MAC OS X) já possuem o interpretador da linguagem Python pré-instalado. Para instalar o simulador deve-se extrair o conteúdo do arquivo *Cache-Simulator.zip*, presente no CD de instalação, na pasta *home* do usuário.

Para executar o simulator, deve-se acessar o diretório da aplicação via terminal e executar o seguinte comando:

```
python ./simulator.py <requestFile> <cacheSize> [fifo|lru|lfu|random]
```

onde,

- <*requestFile*>: caminho com arquivo de requisições onde cada linha do arquivo mantém o conteúdo da página a ser armazenada na memória simulada.
- <*cacheSize*>: tamanho da memória cache simulada.
- [fifo | lru | lfu | random]: política de substituição de página escolhida. O usuário deve escolher uma das políticas.

Exemplo de execução com um arquivo de 14 requisições:

example.txt

```
ORANGE
GRAPE
APPLE
BANANA
ORANGE
ORANGE
APPLE
GRAPE
ORANGE
APPLE
APPLE
GRAPE
GRAPE
GRAPE
```

Execução:

```
python ./simulator.py example.txt 3 lru
```

Saída:

```
1 cache = ['ORANGE', None, None] , age = [1, 0, 0]
2 cache = ['ORANGE', 'GRAPE', None] , age = [1, 2, 0]
3 cache = ['ORANGE', 'GRAPE', 'APPLE'] , age = [1, 2, 3]
4 cache = ['BANANA', 'GRAPE', 'APPLE'] , age = [4, 2, 3]
5 cache = ['BANANA', 'ORANGE', 'APPLE'] , age = [4, 5, 3]
6 cache = ['BANANA', 'ORANGE', 'APPLE'] , age = [4, 6, 3]
```

```
cache = ['BANANA', 'ORANGE', 'APPLE'] , age = [4, 6, 7]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [8, 6, 7]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [8, 9, 7]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [8, 9, 10]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [8, 9, 11]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [12, 9, 11]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [13, 9, 11]
cache = ['GRAPE', 'ORANGE', 'APPLE'] , age = [14, 9, 11]
Total Requests: 14, Total Cache Hit: 8, Total Cache Miss: 6, Cache Hit Ratio:
0.57, Cache Miss Ratio: 0.43
```