Problem 13

Least Recently Used

12 points

JAVA: program name must be prob13.class
C /C++ program name must be: prob13.exe

Task Description

We all want our computers faster, cheaper, and with more storage. The Least Recently Used algorithm provides computer designers with a way to increase system performance with little additional cost. It does this by allowing commonly accessed pages to be read from fast system memory instead of slow disk drives. Understanding how it works requires some explanation.

People commonly express memory and disk storage capacity in bytes or gigabytes, but memory is often organized into pages of 4KB each. Hard drives can permanently store far more pages of data than RAM, with a much lower cost per GB. Disk access, however, is significantly slower than RAM. If only we could have hard disk drives as fast as main memory! For now, at least, it is not so.

Let's consider a way to improve performance with the hardware we have. Studies have shown that computers often access some pages repeatedly from the disk, while most of the disk's pages are very rarely accessed. This access pattern has led computer designers to store commonly accessed pages in otherwise unused portions of memory called a cache. A cache requires an effective algorithm to achieve good results. When a program accesses a page from the disk drive, the system first looks to see if the page is in the cache. If it is, this is called a cache hit and the page is transferred from the cache to the program's memory. Very fast! If not, this is called a cache miss and the cache system must read the page from the disk. In addition, the cache system must decide what to do with this newly read page. A key feature of any caching algorithm is its page replacement policy. In our case, on every cache miss we will replace the least recently used page in the cache with the currently accessed page. Hit or miss, the currently accessed page becomes the most recently used page.

For this program we will consider a very simple storage and cache system. The disk capacity is 1,024 pages (numbered from zero) and there are 16 pages of memory available for the cache.

There's more than one way to implement an LRU algorithm. Write one.

Program Input

The input (from file prob13.in) will consist of exactly five lines. Each line will contain twenty page numbers separated by spaces. The program must interpret the numbers as page requests in order from left to right.

```
390 444 390 591 556 635 259 960 701 398 344 396 339 690 874 960 418 635 875 390
390 581 960 14 390 960 500 390 731 398 623 344 398 960 417 390 390 355 344 134
875 89 701 146 290 635 635 777 217 146 722 960 960 390 134 390 623 960 793 264
134 417 282 138 406 558 390 740 134 390 979 453 43 390 217 344 697 601 398 390
417 453 89 398 396 970 398 398 259 720 390 89 259 993 89 390 418 14 86 398
```

Program Output

After every 20 page accesses (i.e. after each line of input) the program must print the cumulative cache hit rate as a percentage of the total number of pages accessed. The program begins with an empty cache.

```
cache hit rate: 20.00%
cache hit rate: 40.00%
cache hit rate: 41.67%
cache hit rate: 38.75%
cache hit rate: 42.00%
```