

## Greedy Algorithms: Off-line caching (Textbook Problem 16-5)

Modern computers use a cache to store a small amount of data in a fast memory. When a computer program executes, it makes a sequence  $\langle r_1, r_2, \dots, r_n \rangle$  of  $n$  memory requests, where each request is for a particular data element. For example, a program that accesses 4 distinct elements  $\{a, b, c, d\}$  might make the sequence of requests  $\langle e, b, d, b, d, a, c, e, b, a, c, b \rangle$ . Let  $k$  be the size of the cache. When the cache contains  $k$  elements and the program requests the  $(k + 1)$ st element, the system must decide, for this and each subsequent request, which  $k$  elements to keep in the cache. In other words, **cache hit** and **cache miss** will happen. The cache replacement algorithm evicts data with the goal of *minimizing the number of cache misses* over the entire sequence of requests.

Typically, caching is an on-line problem. That is, we have to make decisions about which data to keep in the cache without knowing the future requests. Here, however, we consider the off-line version of this problem, in which we are given in advance the entire sequence of  $n$  requests and the cache size  $k$ .

We can solve this off-line problem by a greedy strategy called **furthest-in-future**, which chooses to evict the item in the cache whose next access in the request sequence comes furthest in the future.

- (a) Write pseudocode for a cache manager that uses furthest-in-future strategy. Assume that request sequence is of size  $n$  and cache size is  $k$  and  $n > k$ .
- (b) What is the running time of this algorithm in terms of  $n, k$ .
- (c) Justify its optimality.  
*Hint:* Greedy choice property and optimal substructure property

---

FURTHEST-IN-FUTURE( $R, C$ )

```
1  for  $i = 1$  to  $R.length$ 
2      if  $R[i] \in C$ 
3          Cache Hit
4      else
5          Cache Miss
6          if Cache is not full
7              add  $R[i]$  to Cache
8      else
9          //  $tmp$  is used to keep track of the latest
10         // appearance of cache item in the sequence
11          $tmp = i$ 
12         //  $pos$  is used to keep track of
13         // the position in cache to be replaced
14          $pos = 1$ 
15         for  $j = 1$  to  $C.length$ 
16              $p = i$ 
17             // find the first appearance of the specified
18             // cache item in the sequence
19             while _____
20                  $p = p + 1$ 
21             // if  $p$  reaches the end of sequence without
22             // finding the specified cache item
23             if _____
24                 //  $p = Infinity$ 
25                  $pos = j$ 
26                 break from the inner for loop
27             // update  $tmp$ 
28             if _____
29                 _____
30                 _____
31          $C[pos] = R[i]$ 
```

See slides 48 - 51 for more detailed analysis of optimality at: <http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf>