The following is an abstraction of data caching, which you may have learned about in the ICS 51 and 53. However, you do not need knowledge from those classes for this problem. The particular description of the problem comes from *Algorithm Design* by Kleinberg and Tardos.

Consider a set U of n pieces of data stored in *main memory*. We also have a faster memory, the cache, that can hold $k < n$ pieces of data at any one time. We will assume that the cache initially holds some set of k items. A sequence of data items $D = d_1, d_2, \ldots d_m$ drawn from U is presented to us : this is the sequence of memory references we must process – and in processing them we must decide at all times which k items to keep in the cache. When item d_i is presented, we can access it very quickly if it is already in the cache; otherwise, we are required to bring it from main memory into the cache and, if the cache is full, to *evict* some other piece of data that is currently in the cache to make room for d_i. This is called a *cache miss* and we want to have as few of these as possible.

Thus, on a particular sequence of memory references, a cache maintenance algorithm determines an *eviction schedule* – specifying which items should be evicted from the cache at which points in the sequence – and this determines the contents of the cache and the number of misses over time.

Example: if cache size $k = 2$ and the requests are a, b, c, b, c, a, b, we have items a, b to start in cache, then at the third request, we evict a for c, then at the sixth request, we evict c for a. Thus we have two misses for this input.

Show that the policy of evicting whichever item in the cache will be next requested *farthest in the future* (among those in the cache currently) will minimize evictions. Obviously a real cache couldn't implement this, but that's not the point of this exercise.

1. We call an eviction schedule *reduced* if the only time it performs evictions is when a data element is requested, but not already in the cache. This means, for example, that it doesn't preemptively evict a data element. Prove that for any (possibly not reduced) schedule S, there is a schedule S' that has no more evictions.
   This means we don't have to investigate any algorithms for this problem that do anything other than evictions made to bring into the cache the currently-requested data element.

2. Let S_{FF} be the schedule produced by the farthest in future algorithm described in the problem statement and let OPT be a schedule has the fewest cache misses. Show that we can convert any OPT into S_{FF} without increasing the number of cache misses. *This demonstrates that no schedule can have fewer misses than S_{FF}, because we can convert any optimal into S_{FF} without making it worse.*