Computer Science 413

Greedy Algorithms — Minimizing Cache Misses for Offline Caching

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #18

Goals for Today

- Presentation and analysis of another ***greedy algorithm*** — one that solves a a (simplified) problem motivated by a (more complex) problem concerning ***memory management***.
- The algorithm presented in this lecture is sometimes called "Bélády's Algorithm" (naming it after its inventor) or the "Clairvoyant Replacement Algorithm". It uses a greedy strategy that is sometimes called LONGEST FORWARD DISTANCE.

The Problem To Be Solved: Offline Caching

Modern computers use a ***cache*** to store a small amount of data in fast memory.

- Even though main memory may be much larger — and much slower — the use of a cache may significantly decrease processing time.
- Unfortunately, processing is slowed down again if information is not in the cache when needed: It has to be copied from main memory into the cache — and something else must be removed from the cache to make room for it.

The Problem To Be Solved: Offline Caching

- In practice it is necessary to decide what must be removed from the cache as soon as something else must be brought in. You do now have — and, therefore, cannot use — any information about ***future*** memory requests.
- Algorithms that deal with this kind of problem are called ***online algorithms*** — and the problem described, here, is called ***online hashing***.

The Problem To Be Solved: Offline Caching

- Another problem that is less realistic (for modern computation) but is still of some interest is one in which the entire sequence of memory requests is made available before it is necessary to figure out how to serve the *first* (or any later) one.
- This version of the problem — which is considered in this lecture — is called ***offline caching***.

The Problem To Be Solved: Offline Caching

In order to model this more formally, consider a problem whose instances include the following information:

(a) A positive integer M: the size of (i.e., number of machine words in) ***main memory***.

(b) A positive integer k such that $k < M$: The size of the ***cache***.

(c) A set $\mathcal{C} \subseteq \{0, 1, \dots, M-1\}$ such that $|\mathcal{C}| = k$: The indices (in main memory) of the ***initial contents of the cache***.

(d) A sequence $\langle r_1, r_2, \dots, r_n \rangle$ of elements of $\{0, 1, \dots, M-1\}$: The ***addresses of pages in main memory*** that will be requested.

The Problem To Be Solved: Offline Caching

Given the above, one can define a ***valid sequence of changes*** $\langle \alpha_1, \alpha_2, \dots, \alpha_n$, with ***corresponding caches*** $\langle \mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n \rangle$, to be any pair of sequences that satisfy the following:

(a) $\mathcal{C}_0 = \mathcal{C}$.

(b) For $1 \le i \le n$, if $r_i \in \mathcal{C}_{i-1}$ then $\alpha_i = \mathtt{null}$ and $\mathcal{C}_i = \mathcal{C}_{i-1}$. We say that there has been a ***cache hit*** in this case.

(c) For $1 \le i \le n$, if $r_i \notin \mathcal{C}_{i-1}$, then $\alpha_i \in \mathcal{C}_{i-1}$ and

$$\mathcal{C}_i = (\mathcal{C}_{i-1} \setminus \{\alpha_i\}) \cup \{r_i\}.$$

Thus α_i is (the index of) the page being removed from the class in order to include r_i. We say that there has been a ***cache miss*** in this case.

The Problem To Be Solved: Offline Caching

The problem to be solved can now be formally defined as follows.

Offline Caching

Precondition: Positive integers M and k, a set $\mathcal{C}$ and sequence $\langle r_1, r_2, \ldots, r_n \rangle$, as described above, are given as output.

Postcondition: A valid sequence $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ such that ***the number of cache misses is as small as possible*** is returned as output.

Solving the Problem: Proving that Every Instance Has a Solution

Claim #1: Every instance of the “Offline Caching” problem has a solution.

Proof: Consider an instance of the problem as described above. Then the set $\mathcal{F}$ of all ***feasible solutions*** is the set of all valid sequences of changes, and it is easily proved (by induction on n) that $1 \leq |\mathcal{F}| \leq k^n$, so that this set is both nonempty and finite.

The number of caches misses is a well-defined (and total) function from $\mathcal{F}$ to $\mathbb{N}$.

The claim now follows. $\square$

Solving the Problem: Identifying and Solving Trivial Instances

Consider an instance of this problem to be ***trivial*** if $n = 1$.

- If $r_1 \in \mathcal{C}$ then $\langle \texttt{null} \rangle$ must be returned, because it is the only feasible solution (and there are no cache misses at all).
- If $r_1 \notin \mathcal{C}$ then α_1 can be set to be any element of the cache $\mathcal{C}$: Then $\langle \alpha_1 \rangle$ is a feasible solution and, since r_1 is the final request, the number of cache misses will be one, regardless of the choice that is made.

Solving the Problem: A Greedy Strategy for Nontrivial Instances

Consider an instance of this problem such that $n \geq 2$.

- If $r_1 \in \mathcal{C}$ then, once again, `null` must be chosen as the first entry of the sequence of changes to be returned, so this must be the "greedy choice" in this case.
- If $r_1 \notin \mathcal{C}$ and there exists at least one element α of $\mathcal{C}$ such that $\alpha \notin \{r_2, r_3, \ldots, r_n\}$ then one can set α_1 to be α — because this will never be needed again.

  If there is more than one such value then it does not matter which one is picked.

Solving the Problem: A Greedy Strategy for Nontrivial Instances

- If $r_1 \notin \mathcal{C}$ and $\mathcal{C} \subseteq \{r_2, r_3, \ldots, r_n\}$, then α_1 should be chosen to be the unique element of $\mathcal{C}$ that is ***not needed for as long as possible*** — that is, the unique element α of $\mathcal{C}$ such that if $2 \le i \le n$ and $\alpha = r_i$ (and $\alpha \neq r_k$ for any integer k such that $2 \le k \le i-1$) then, for every other element β of $\mathcal{C}$, there exists an integer j such that $2 \le j \le i-1$ and $\beta = r_j$.

  This is sometimes called the use of a ***longest forward distance*** strategy.

Solving the Problem: Proving Correctness of the Greedy Strategy

Lemma #2: Let $\langle \widehat{\alpha}_1, \widehat{\alpha}_2, \dots, \widehat{\alpha}_n \rangle$ be any feasible solution for this instance of the "Offline Caching" problem. Then there exists a feasible solution $\langle \alpha_1, \alpha_2, \dots, \alpha_n \rangle$ for this instance of the problem such that α_1 is the "greedy choice", described, above, and the number of cache misses for the sequence $\langle \alpha_1, \alpha_2, \dots, \alpha_n \rangle$ is less than or equal to the number of cache misses for the sequence $\langle \widehat{\alpha}_1, \widehat{\alpha}_2, \dots, \widehat{\alpha}_n \rangle$.

Proof: Note that the result is easily proved if $\widehat{\alpha}_1 = \alpha_1$, because all we need to do is to set α_i to be $\widehat{\alpha}_i$, for $2 \leq i \leq n$, in order to establish the claim.

Suppose, therefore, that $\widehat{\alpha}_1 \neq \alpha_1$. Then there must have been a cache miss right away!

Solving the Problem: Proving Correctness of the Greedy Strategy

Let $\widehat{\mathcal{C}}_0 = \mathcal{C}, \widehat{\mathcal{C}}_1, \dots, \widehat{\mathcal{C}}_n$ be the corresponding sequence of caches for the sequence of changes $\widehat{\alpha}_1, \widehat{\alpha}_2, \dots, \widehat{\alpha}_n$, and let $\mathcal{C}_0 = \mathcal{C}, \mathcal{C}_1, \dots, \mathcal{C}_n$ be the corresponding sequence of caches for the sequence of changes $\alpha_1, \alpha_2, \dots, \alpha_n$ now being described.

Subclaim: For every integer i such that $1 \leq i \leq n$ it is possible to choose $\alpha_2, \alpha_3, \dots, \alpha_i$ in a way that ensures that one of the following ***situations*** arises after the first i requests, $r_1, r_2, \dots, r_i$, have been served:

Solving the Problem: Proving Correctness of the Greedy Strategy

1. There exists a set $\mathcal{S} \subseteq \{0, 1, \ldots, M-1\}$ such that $|\mathcal{S}| = k-1$, and elements $\beta, \gamma \in \{0, 1, \ldots, M-1\}$ such that $\beta, \gamma \notin \mathcal{S}$, $\beta \neq \gamma$, $\mathcal{C}_i = \mathcal{S} \cup \{\beta\}$, and $\widehat{\mathcal{C}}_i = \mathcal{S} \cup \{\gamma\}$.

   Furthermore, the number of cache misses caused by handling requests $r_1, r_2, \ldots, r_i$ using changes $\alpha_1, \alpha_2, \ldots, \alpha_i$ is less than or equal to the number of cache misses used by handling these requests using changes $\widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_i$.

   Finally, γ is not needed again until β is: If there exists an integer j such that $i+1 \leq j \leq n$ and $\gamma = r_j$ (and $\gamma \neq r_k$ for any integer k such that $i+1 \leq k \leq j-1$) then there exists an integer h such that $i+1 \leq h \leq j-1$ and $\beta = r_h$.

Solving the Problem: Proving Correctness of the Greedy Strategy

2. There exists a set $\mathcal{S} \subseteq \{0, 1, \ldots, M-1\}$ such that $|\mathcal{S}| = k-1$, and elements $\beta, \gamma \in \{0, 1, \ldots, M-1\}$ such that $\beta, \gamma \notin \mathcal{S}$, $\beta \neq \gamma$, $\mathcal{C}_i = \mathcal{S} \cup \{\beta\}$, and $\widehat{\mathcal{C}}_i = \mathcal{S} \cup \{\gamma\}$.

   Furthermore, the number of cache misses caused by handling requests $r_1, r_2, \ldots, r_i$ using changes $\alpha_1, \alpha_2, \ldots, \alpha_i$ is strictly less than the number of cache misses used by handling these requests using changes $\widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_i$.
3. $\widehat{\mathcal{C}}_i = \mathcal{C}_i$. Furthermore, the number of cache misses caused by handling requests $r_1, r_2, \ldots, r_i$ using changes $\alpha_1, \alpha_2, \ldots, \alpha_i$ is less than or equal to the number of cache misses used by handling these requests using changes $\widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_i$.

Solving the Problem: Proving Correctness of the Greedy Strategy

Proof of Subclaim: By induction on *i*. The standard form of mathematical induction can be used.

Basis: If $i = 1$ then it follows by the ***greedy choice:*** that the first situation must hold: $\mathcal{S} = (\mathcal{C} \setminus \{\alpha_1, \widehat{\alpha}_1\}) \cup \{r_1\}$, $\beta = \widehat{\alpha}_1$, $\gamma = \alpha_1$, and *one* cache miss occurred when handling the first request, r_1, using either the initial change α_1 or the initial change $\widehat{\alpha}_1$. It follows by the “greedy choice” that $\widehat{\alpha}_1$ will be needed before α_1 is, if α_1 is ever needed at all.

Solving the Problem: Proving Correctness of the Greedy Strategy

Inductive Step: Let k be an integer such that $k \geq 1$. It is necessary to use the following

> Inductive Hypothesis: The above is true when $i = k$.

to prove the following

> Inductive Claim: The above is true when $i = k + 1$.

With that noted, suppose that $i = k \geq 1$ and that the Inductive Hypothesis is satisfied. There is nothing to prove if $k \geq n$, since $i = k + 1 \geq n + 1$ in this case and the claim is empty. It may therefore be assumed that $k \leq n - 1$.

It is useful to consider what can, and should be done, when we are in each of the three situations mentioned in the subclaim after the first $i = k$ requests have been served.

Solving the Problem: Proving Correctness of the Greedy Strategy

Situation #1:

Subcase: $r_{k+1} \in \mathcal{S}$. We must set $\alpha_{k+1} = \widehat{\alpha}_{k+1} - \texttt{null}$ — and remain in Situation #1 after the $k+1^{\text{st}}$ request has been served.

Subcase: $r_{k+1} \notin \mathcal{S} \cup \{\beta, \gamma\}$. Cache misses occur in each case. It suffices to set $\alpha_{k+1} = \widehat{\alpha}_{k+1}$ in order to make sure that we remain in Situation #1.

Subcase: $r_{k+1} = \beta$, so that there was a cache miss when serving the $k+1^{\text{st}}$ request using $\widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{k+1}$, but not when using $\alpha_1, \alpha_2, \ldots, \alpha_{k+1}$ (and α_{k+1} must be set to be $\texttt{null}$). In this case either $\widehat{\alpha}_{k+1} \neq \gamma$, and we have moved to Situation #2 (with β replaced by $\widehat{\alpha}_{k+1}$), or $\widehat{\alpha}_{k+1} = \gamma$ and we have moved to Situation #3.

Subcase: $r_{k+1} = \gamma$. This case is ***impossible***, because γ will not be needed again before β is if we are in Situation #1.

Solving the Problem: Proving Correctness of the Greedy Strategy

Situation #2:

The *subcases* $r_{k+1} \in \mathcal{S}$, $r_{k+1} \in \mathcal{S} \cup \{\beta, \gamma\}$, and $r_{k+1} = \beta$ should be handled as described for Situation #1. This ensures that one either remains in Situation #2 (instead of Situation #1, now) or moves to Situation #3.

Subcase: $r_{k+1} = \gamma$. This case is now ***possible***. However, $\widehat{\alpha}_{k+1}$ must be `null`, and we are free to choose $\alpha_{k+1} = \beta$. The number of cache misses that arise when $\alpha_1, \alpha_2, \ldots, \alpha_{k+1}$ is used to serve the first $k+1$ requests is still less than or equal to the number that arise when $\widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{k+1}$ are used to serve these requests, instead — and $\widehat{\mathcal{C}}_{k+1} = \mathcal{C}_{k+1}$, so that we have now moved to Situation #3.

Solving the Problem: Proving Correctness of the Greedy Strategy

Situation #3:

Since $\widehat{\mathcal{C}}_k = \mathcal{C}_k$ it is always possible (or necessary) to choose $\alpha_{k+1} = \widehat{\alpha}_{k+1}$, in order to ensure that $\widehat{\mathcal{C}}_{k+1} = \mathcal{C}_{k+1}$. Either another cache miss has occurred when each sequence of changes is being used to serve the requests, or no cache miss has occurred in either one of them — keeping us in Situation #3.

Since all cases (and subcases) have now been considered this establishes the Inductive Claim, as needed to complete the Inductive Step and complete the proof of the Subclaim.

Lemma #2 is a straightforward consequence of the subclaim, considering the case that $i = n$, so its proof is also now complete. $\square$

Solving the Problem: Proving Correctness of the Greedy Strategy

Claim #3: Every nontrivial instance of the “Offline Caching” problem has a solution $\langle \alpha_1, \alpha_2, \dots, \alpha_n \rangle$ such that α_1 is the ***greedy choice*** that is described above.

Proof: This is now a straightforward consequence of Claims #1 and #2. $\square$

Solving the Problem: How to Continue

Once a greedy choice α_1 has been obtained, one can form a smaller instance of this problem (including a sequence of $n-1$ requests instead of a sequence of n requests) as follows.

- Leave M and k unchanged.
- Replace $\mathcal{C}$ with $\widehat{\mathcal{C}} = \mathcal{C}_1$, the cache obtained by serving request r_1 with change α_1.
- Replace the sequence of requests $\langle r_1, r_2, \ldots, r_n \rangle$ with the sequence

$$\langle \widehat{r}_1, \widehat{r}_2, \ldots, \widehat{r}_{n-1} \rangle = \langle r_2, r_3, \ldots, r_n \rangle.$$

Solving the Problem: How to Continue

After recursively solving this instance of the problem, to obtain a sequence $\langle \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$ of changes, one should return

$$\langle \alpha_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$$

as a solution for the originally given instance of the problem.

Solving the Problem: Correctness of the Continuation

Claim #4: The sequence $\langle \alpha_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \dots, \widehat{\alpha}_{n-1} \rangle$, given above, is a correct solution for the originally given (nontrivial) instance of the “Offline Caching” problem.

Proof: The sequence $\langle \alpha_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \dots, \widehat{\alpha}_{n-1} \rangle$ is certainly a ***feasible solution*** for this instance of the problem, so it is necessary and sufficient to show that it minimizes the number of cache misses that can arise.

- By Claim #3, there exists a solution $\langle \alpha_1, \widetilde{\alpha}_2, \widetilde{\alpha}_3, \dots, \widetilde{\alpha}_n \rangle$ for this instance of the problem that also begins with the greedy choice, α_1.

Solving the Problem: Correctness of the Continuation

- Let h be the number of cache misses that arise when the sequence of changes $\langle \alpha_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$ is used to serve the sequence of requests $\langle r_1, r_2, \ldots, r_n \rangle$, and let ℓ be the number of cache misses that arise when the sequence of changes $\langle \alpha_1, \widetilde{\alpha}_2, \widetilde{\alpha}_3, \ldots, \widetilde{n} \rangle$ is used to serve this sequence of requests instead.

  Since $\langle \alpha_1, \widetilde{\alpha}_2, \widetilde{\alpha}_2, \ldots, \widetilde{\alpha}_n \rangle$ is a solution for this instance of the problem, it is now necessary and sufficient to argue that $h \leq \ell$.

Solving the Problem: Correctness of the Continuation

- Now consider the smaller instance or the problem (including M, k, $\widehat{\mathcal{C}}$ and $\langle \widehat{r}_1, \widehat{r}_2, \dots, \widehat{r}_{n-1} \rangle$) as described above.

  The sequences of changes $\langle \widetilde{\alpha}_2, \widetilde{\alpha}_3, \dots, \widetilde{\alpha}_n \rangle$ is certainly a feasible solution for this instance of the problem.

Solving the Problem: Correctness of the Continuation

- Let $\widehat{h}$ be the number of cache misses that arise when the sequence of changes $\langle \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$ is used to serve the sequence of requests $\langle \widehat{r}_1, \widehat{r}_2, \ldots, \widehat{r}_{n-1} \rangle$ and let $\widehat{\ell}$ be the number of cache misses that arise when the sequence of changes $\langle \widetilde{\alpha}_2, \widetilde{\alpha}_3, \ldots, \widetilde{\alpha}_n \rangle$ is used to serve the requests instead.
- Since $\langle \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$ is a (correct) solution for the smaller instance of the problem and $\langle \widetilde{\alpha}_2, \widetilde{\alpha}_3, \ldots, \widetilde{\alpha}_n \rangle$ is a feasible solution for it, $\widehat{h} \leq \widehat{\ell}$.

Solving the Problem: Correctness of the Continuation

- However, either $\alpha_1 = \texttt{null}$, in which case $h = \widehat{h}$ and $\ell = \widehat{\ell}$, or $\alpha_1 \neq \texttt{null}$, in which case $h = \widehat{h} + 1$ and $\ell = \widehat{\ell} + 1$.
- It either case it follows that $h \leq \ell$, as needed to establish the claim. $\square$

Solving the Problem: The Algorithm

Th following algorithm can be used to solve a ***trivial instance*** of this problem by the method described above.

```
solve_trivial($\mathcal{C}$, $r_1$) {
1.  if ($r_1 \in \mathcal{C}$) {
2.      return $\langle$null$\rangle$
    } else {
3.      Let $\alpha_1$ be an element of $\mathcal{C}$
4.      return $\langle \alpha_1 \rangle$
    }
}
```

This returns a solution for a trivial instance of this problem using at most three steps.

Solving the Problem: The Algorithm

The following algorithm computes and returns a greedy choice when $n \geq 2$.

```
find_greedy($\mathcal{C}$, $\langle r_1, r_2, \ldots, r_n \rangle$) {
 1.  if ($r_1 \in \mathcal{C}$) {
 2.    return null
     } else {
 3.  set $\mathcal{S}$ := $\mathcal{C}$
 4.  integer i := 2
```

Solving the Problem: The Algorithm

```
5.      while $((|\mathcal{S}| > 1)$ and $(\mathtt{i} \leq n))$ {
6.        if $(r_{\mathtt{i}} \in \mathcal{S})$ {
7.          $\mathcal{S}$ := $\mathcal{S} \setminus \{r_{\mathtt{i}}\}$
          }
8.        i := $\mathtt{i} + 1$
        }
9.      if $(|\mathcal{S} > 1)$     // $\mathtt{i} = n + 1$
10.       return any element of $\mathcal{S}$
        } else {
11.       return the unique remaining element of $\mathcal{S}$
        }
      }
}
```

Solving the Problem: The Algorithm

Exercises:

1. Confirm that this algorithm always terminates — if $\mathcal{C}$ and $\langle r_1, r_2, \ldots, r_n \rangle$ are as described in the precondition for the "Offline Caching" problem and $n \geq 2$ — and that it returns a greedy choice as described above.
2. Assuming the Uniform Cost Criterion, prove that this algorithm always uses a number of steps that is at most linear in n. Indeed, it uses at most $4n + 2$ steps in the worst case.

Solving the Problem: The Algorithm

A main method can now be given:

```
greedy_cache($M, k, \mathcal{C}, \langle r_1, r_2, \ldots, r_n \rangle$) {
1.  if ($n$ == 1) {
2.    return solve_trivial($\mathcal{C}, r_1$)
    } else {
3.    $\alpha_1$ := find_greedy($\mathcal{C}, \langle r_1, r_2, \ldots, r_n \rangle$)
4.    if ($\alpha_1$ == null) {
5.      $\widehat{\mathcal{C}}$ := $\mathcal{C}$
      } else {
6.      $\widehat{\mathcal{C}}$ := $(\mathcal{C} \setminus \{\alpha_1\}) \cup \{r_1\}$
      }
```

Solving the Problem: The Algorithm

```
7.   $\langle \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$
            := greedy_cache($M, k, \widehat{\mathcal{C}}, \langle r_2, r_3, \ldots, r_n \rangle$)
8.   return $\langle \alpha_1, \widehat{\alpha}_1, \widehat{\alpha}_2, \ldots, \widehat{\alpha}_{n-1} \rangle$
   }
}
```

Solving the Problem: Correctness

Another Exercise:

3. Write a proof (by induction on *n*, using the standard form of mathematical induction) that this algorithm correctly solves the “Offline Hashing” problem.

   This should be easy, using Claims #1, #3, and #4.

Solving the Problem: Efficiency

Let $T(n)$ be the number of steps used by the above in the worst case when it is executed with the precondition for the “Offline Caching” problem satisfied, as a function of the number n of requests included in the input.

More Exercises:

4. Assuming the Uniform Cost Criterion, confirm that (for $n \in \mathbb{N}$ and $n \geq 1$)

$$T(n) \leq \begin{cases} 5 & \text{if } n = 1, \\ T(n+1) + 4 & \text{if } n \geq 2. \end{cases}$$

5. Use this to prove that $T(n) \leq 2n^2 + 2n + 1 \in O(n^2)$ for every integer $n \geq 1$.

A More Realistic Problem

A variety of heuristics are used to try to solve the “Online Caching”, including LEAST RECENTLY USED: When there is a cache miss, removed from the cache the item that was last requested (or some item in the cache that has never been requested at all, if such an item exists).

Consider, now, another parameter — the size h of a ***larger*** cache. It is possible to prove the following: For any sequence $\langle r_1, r_2, \dots, r_n \rangle$ of requests, if

- $m_{\mathrm{LFD}}(k)$ is the number of cache misses that arise when the above greedy algorithm is used to serve these requests, with a cache of size k, and
- $m_{\mathrm{LRU}}(h)$ is the number of cache misses that arise when the above greedy algorithm is used to serve these requests, with a larger cache of size h, then

$$m_{\mathrm{LRU}}(h) \leq \frac{h}{h-k+1} \cdot m_{\mathrm{LFD}}(k) + h.$$

A More Realistic Problem

For a proof of this, and similar results relating the performance of online algorithms to offline algorithms, see

> Allan Borodin an Ran El-Yaniv,
> *Online Computation and Competitive Analysis,*
> Cambridge University Press, 1998.