Optimal caching problem

Bakh Khoussainov

Optimal caching

Caching is a process of storing a small amount of Data in a fast memory.

A fast memory is typically much quicker to access thAN slow memory.

A cashe maintenance Algorithm determines what to keep in the cache and what to evict.

Set up:

We have a set U of items stored in main memory.

We have a faster memory, cache, that stores at most k items of data, $k < n$.

Cache is assumed to have k items stored.

A sequence of data items is given

$$d_1, d_2, \ldots, d_m$$

all from U.

We need the sequence.

When we read d_i:

(1) If d_i is in the cache we access d_i quickly.

(2) If d_i is not in the cache we must bring d_i into cache by evicting an item. This is called a cache miss.

Note that items in cache are evicted.

Goal: Design an algorithm that minimizes the number of cache misses on input sequences

$$d_1, d_2, \ldots, d_m.$$

Example.

$U = \{a, b, c\}, \quad k = 2$

The sequence

$$a, b, c, b, c, a, b$$

Initial cache: a, b.

$\overset{\downarrow}{a}, b, c, b, c, a, b$

(1) cache: a, b

(2) $a, \overset{\downarrow}{b}, c, b, c, a, b$

cache: a, b

(3) $a, b, \overset{\downarrow}{c}, b, c, a, b$

cache: b, c

(6) $a, b, c, b, c, \overset{\downarrow}{a}, b$

cache: b, a

(7) $a, b, c, b, c, a, \overset{\downarrow}{b}$

cache: b, a

Farthest-in-Future algorithm

For $i = 1, 2, \ldots, m$

if d_i creates a cache miss

then evict the item that

is needed farthest into

the future

This is a natural algorithm.

Why is this algorithm

correct?

Example.

Input: a, b, c, d, a, d, e, a, d, b, c

Cache: a, b, c.

Step 4: a, b, c, d, a, d, e, a, d, b, c

cache: a, b, d

Step 7: a, b, c, d, a, d, e, a, d, b, c

cashe: a, e, d

We could have also evicted b on step 4 and c on Step 7.

A schedule is reduced if each time when it brings items x into cache there is a cache miss for x.

Note: (1) There are schedules that are not reduced.

(2) In reduced schedules

#(items brought) = #(cache misses)

FACT 1. Every schedule S can be transformed into a reduced schedule S_r such that S_r brings in at most as many items as S.

Indeed, suppose S is processing the sequence

$$d_1, d_2, \ldots, d_m .$$

Let t be the number of times S brings in items when no cache miss occurs.

We change S to a new schedule S' as follows.

Suppose S brings r in cache without a miss, and replaces x with r.

Idea. When S brings an item r that has not been requested, S_r "pretends" to do this but does not bring r into cache. S_r brings r into cache in a later stage at which r is requested.

In this way S_r has less cache misses than S.

Here is a bit more detailed explanation.

S: $d_1 \ldots \overset{\swarrow}{d_i} \ldots d_m$

Old-cache: $\ldots x \ldots d_i \ldots$

New-cache: $\ldots r \ldots d_i \ldots \leftarrow$ cache(S)

S': $d_1 \ldots \overset{\swarrow}{d_i} \ldots d_m$

Cache(S'): $\ldots x \ldots d_i \ldots$

Cache(S) and cache(S') differ at r and x. Now S' proceeds as follows:

(1) Continue on copying S.

(2) If S sees x, evicts r and brings x, then $cache(S) = cache(S')$.

S: $d_1 \ldots d_i \ldots \overset{\swarrow}{x} \ldots d_m$

old-cache(S): $\ldots r \ldots$

New-cache(S): $\ldots x \ldots$

S': $d_1 \ldots d_i \ldots \overset{\swarrow}{x} \ldots d_m$

cache(S'): $\ldots x \ldots$

(3) If S sees x, evicts $d \neq r$, and brings in x

S: $d_1 \ldots d_i \ldots \overset{\downarrow}{x} \ldots d_m$

old-cache(S): $\ldots r \ldots d \ldots$

new-cache(S): $\ldots r \ldots x \ldots$

then S' does nothing. So

cache(S'): $\ldots x \ldots d \ldots$

Now cache(S) and cache(S') differ on r and d.

So, we set $x := d$

(4) If S sees r then

(4a) S does not evict an item from its cache. In this case S' evicts x, and replaces it with r. So $\text{cache}(S) = \text{cache}(S')$.

(4b) S evicts an item x' AND brings in an item r'

S: $d_1 \ldots d_i \ldots r \ldots d_m$

old-cache (S): $\ldots r \ldots x' \ldots$

new-cache (S): $\ldots r \ldots r' \ldots$

In this case S' evicts x and replace it with r;

So, ;

S': $d_1 \ldots d_i \ldots r \ldots d_m$

old-cache(S'): $\ldots x \ldots x' \ldots$

new-cache(S'): $\ldots r \ldots x' \ldots$

So cache(S) and cache(S') differ on r' and x'.

Thus, S' has fewer cache misses than S.

We replace S with S', and continue this process.

This will eventually produces a reduced schedule S_r that brings in at most as many items as the original S.

Let S^* be the output of our algorithm.

We want to show that S^* is an optimal solution.

For this we need to show that S^* has no more cache misses than any other schedule S.
We might assume that S is reduced.

So, let

$$d_1, d_2, \ldots, d_m$$

be a sequence of items.

Assume that S^* AND S

agree on

$$d_1, d_2, \ldots, d_j.$$

We claim that there exists a reduced schedule S' with the following properties:

(1) S^* and S' agree on

$d_1, d_2, \ldots, d_j, d_{j+1}$.

(2) S' has no more cache misses than S does.

Indeed, we consider several cases. Note, both S^* AND S are reading $d = d_{j+1}$.
Also note that

$$\text{cache}(S^*) = \text{cache}(S)$$

by assumption on S and S'.

Case 1. $d \in Cache(S)$. We set $S' = S$. So in this case the claim is true.

Case 2. $d \notin Cache(S)$ AND both S and S^* evict the same item from their caches. We again set $S' = S$ which proves the claim.

Case 3. S evicts item f, S^* evicts item e, and $e \neq f$.

So,

Old-cache(S): $\ldots f \ldots e \ldots$

New-cache(S): $\ldots d \ldots e \ldots$

and

Old-cache(S^*): $\ldots f \ldots e \ldots$

New-cache(S^*): $\ldots f \ldots d \ldots e$

So, cache(S) has e but not f;
cache(S^*) has f but not e.

The rest of caches are equal.

We want to construct S' satisfying (1) and (2).

Satisfying (1) is easy. Schedule S' just copies S^* upto step $j+1$ inclusively.

So, afer reading d, we have

cache (S'): $\ldots f \ldots d \ldots$

with $\text{cache}(S') = \text{cache}(S^*)$.

Idea: Want to get S's cache back to the same state as S.

So, S' starts behaving the same as S from step $j+2$ onwards until one of the following happens.

(a) There is a request to item g such that $g \neq e, g \neq f$, $g \notin Cache(S)$ and S evicts e.

Note that in this case $g \notin Cache(S')$. So we make S' to evict f. From this point on

$$Cache(S') = Cache(S).$$

(b) There is a request to f, and S evicts e'.

If $e' = e$, then since $f \in Cache(S')$ we have $Cache(S') = Cache(S)$.

If $e' \neq e$, then we make S' to evict e' as well. This makes $cache(S') = cache(S)$.

But, S' is not reduced now.

We, however, can transform to a reduced schedule with no more cache misses than S' does.

Note that we used the defining property of the Farthest-in-Future algorithm.

Namely, one of the cases (a) or (b) will arise there is a request to e !

Now we show that S^* is optimal.
Let S_0 be an optimal solution.
Using the claim, construct S_1 that agrees with S^* on d_1 and no more cache misses than S_0.

Inductively for $j = 1, 2, 3, \ldots, m$ produce S_j such that

(1) S_j agrees with S^* on $d_1, \ldots, d_j$

(2) S_j has no more cache misses than S_{j-1}.

So, S_m is S^* and, hence, S^* is optimal.