Concordia University
Dept. of Computer Science and Software Engineering
COMP 691 – Online Algorithms and Competitive Analysis
Winter 2024

$\mathcal{P}$roject

1 Description

Recall, the Paging problem. Let k denote the size of the cache, N denote the size of the large memory, and n the length of the request sequence. The input is a sequence $p_1, \ldots, p_n$ of page requests, where $p_i \in [N]$. Initially, an algorithm starts with an empty cache. When request to page p_i arrives, if the page is not in the cache, then the algorithm experiences a page fault (also called a cache miss). If the cache is full, then a page is chosen to be evicted from the cache, and p_i is brought into the cache. The goal is to minimize the total number of page faults incurred on the request sequence.

Recall, that an optimal offline algorithm OPT evicts a page whose next request appears furthest in the future. We define h_i, corresponding to p_i, to denote the step after i when the next request to p_i occurs. We let h_i be $n+1$ if p_i is never requested after step i. Formally:

$$h_i = \begin{cases} \min\{j : j > i \text{ and } p_j = p_i\} & \text{if } p_i \text{ is requested again in } p_{i+1}, p_{i+2}, \ldots, p_n \\ n+1 & \text{otherwise} \end{cases}$$

If an online algorithm had knowledge of h_i at the time of arrival of p_i then it could simulate OPT. In this project, you will explore the version of the problem where an online algorithm receives predicted values $\hat{h}_i$ alongside p_i. We shall refer to this problem as the Paging problem with predictions (or, PPP, for short).

That is, in PPP, an algorithm receives as input a sequence of pairs $(p_1, \hat{h}_1), (p_2, \hat{h}_2), \ldots, (p_n, \hat{h}_n)$, where p_i is a page request as above and $\hat{h}_i$ is a predicted value of h_i. We do not worry about where the predictions come from, but it may be helpful to think of these predictions as coming from some machine-learning-based algorithm that has been trained on many input sequences, so it can provide $\hat{h}_i$ values with some accuracy.

Once the input is observed, one can compute the true values of h_i, and measure the error e of predictions as follows:

$$e = \sum_{i=1}^{n} |h_i - \hat{h}_i|.$$

The algorithm *BlindOracle* behaves similarly to OPT but using predicted values $\hat{h}_i$ instead of the true values h_i. You will discover that when the prediction error e is small, *BlindOracle* converges to OPT. When the prediction error e is large, *BlindOracle* can perform significantly worse than *LRU*. You will design a *Combined* algorithm (out of *BlindOracle* and *LRU*) that would have the following property: when the prediction error e is small, *Combined* should behave similar to OPT, but when the prediction error e is large, *Combined* should behave similar to LRU. Thus, no matter what the prediction error actually is, the algorithm has performance in between OPT and LRU.

In this project, you will write a program that generates random structured input sequences with predictions, implement 3 standard algorithms (OPT, *BlindOracle*, *LRU*), evaluate performance of these algorithms on the generated inputs, propose a new algorithm *Combined*, evaluate its performance, and summarize your findings in a report. The project consists of 3 phases, each phase building on a previous phase, and each phase having a different deadline. The total weight of the project is 10% of the overall grade in the course: 3% for phase 1, 3% for phase 2, and 4% for phase 3. This project is to be completed individually, and you can use one of the following programming languages to complete programming components: C, C++, Java, Python3.

2 Phase 1: code base

Deadline: submission on Moodle by 23:55PM on March 10, 2024

What to submit: a single .zip file with source code + readme.txt file

Weight: 3% of the overall course grade

Description of the phase:

In your chosen programming language, submit a single file containing the following functions:

1. $generateRandomSequence(k, N, n, \epsilon)$. The input consists of parameters k, N, n having their meanings as defined in the previous section, and $\epsilon \in (0, 1)$ is an extra parameter which controls the amount of locality in a sequence to be generated. The output of this function should be a sequence (could be an array, vector, list, etc) of page requests $p_1, p_2, \dots, p_n$ generated as follows: the first k requests are to pages $1, 2, \dots, k$. Let $L = \{1, 2, \dots, k\}$. For each step $i > k$ do the following: choose page $x \in L$ uniformly at random, and choose page $y \in [N] \setminus L$ uniformly at random. With probability ϵ set $p_i = x$ and move to the next step. With the remaining probability $1 - \epsilon$ set $p_i = y$ and update $L \leftarrow (L \setminus \{x\}) \cup \{y\}$. One can think of L as a set of local pages. With probability ϵ, we generate the next request from a local page, and with probability $1 - \epsilon$ we request a non-local page, and update the local set.

2. $generateH(seq)$. The input seq is a sequence of page requests $(p_1, p_2, \dots, p_n)$. The output is the sequence $(h_1, h_2, \dots, h_n)$, where h_i has been defined in the previous section.

3. $addNoise(hseq, \tau, w)$. The input is the sequence $hseq$ of the values $(h_1, h_2, \dots, h_n)$, $\tau \in (0, 1)$ is a noise parameter, and $w \in \mathbb{N}$ is another noise parameter. This function takes real values of the h_i and adds noise to them producing "predicted values" $(\hat{h}_1, \hat{h}_2, \dots, \hat{h}_n)$. The noise is constructed as follows: for each i, with probability $1 - \tau$ set $\hat{h}_i = h_i$, and with probability τ let $\hat{h}_i$ be a number chosen uniformly between $\ell = \max(i + 1, h_i - \lfloor w/2 \rfloor)$ and $\ell + w$ (inclusive).

4. $blindOracle(k, seq, hseq)$. The input is parameter k - cache size, seq is the sequence of page requests $p_1, \ldots, p_n$, $hseq$ is the sequence of predictions $\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_n$. The output is the number of page faults incurred by $BlindOracle$ algorithm. As discussed in the previous section, this algorithm evicts furthest in the future page using predicted values $\hat{h}_i$ instead of true values h_i. That is, suppose that the cache is full containing pages $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$ and new request p_i arrives. If p_i is not in the cache (page fault), then the algorithm evicts p_{i_j} that has the highest value of $\hat{h}_{i_j}$. Then p_i is brought in, and $\hat{h}_i$ is recorded as the predicted next time p_i is requested. If p_i is present in the cache, say, it is p_{i_1} then there is no page fault, but the old value $\hat{h}_{i_1}$ needs to be updated with the new predicted value $\hat{h}_i$.

5. $test1(), test2(), test3(), \ldots$. These are functions that you can use to test the functionality of the above functions.

6. $main()$. This function runs a bunch of test functions, and reports if everything is working as expected.

Important notes:

- You can use standard libraries that come with the language. You are not allowed to use third-party libraries.
- Document your code well. Documentation is part of the grade. If I cannot read and quickly understand what is happening because of the lack of documentation, you may lose points.
- Design good test functions. Provide additional logic and documentation in the readme.txt file. For example, you may wish to check special cases $N = k + 1$, $N \gg k$, etc. Why should this test function give confidence to correctness of implementation?
- State precisely the command-line command that you use to compile your code in the readme.txt file. Also state what version of the compiler/interpreter you used to compile/interpret and run your code, and what is the expected output of your program.
- You can discuss the project with your peers, but do not plagiarise the code. Plagiarised code is easy to detect, and it would lead to an incident report, academic code of conduct hearings, tribunals, etc. Sometimes it may cause delays in graduation, registration for other courses, visa problems. In short, it's not worth it. Please, don't do it. Write your own code, and if you have discussed it with others, wait a sufficient time period before writing it, so that you forget the details of discussion, and reconstruct them in your own way.

3 Phase 2: extension of code base

Deadline: submission on Moodle by 23:55PM on March 31, 2024

What to submit: a single .zip file with source code + readme.txt

Weight: 3% of the overall course grade

Description of the phase: TBA

4 Phase 3: experiments and report

Deadline: submission on Moodle by 23:55PM on April 15, 2024

What to submit: a single .zip file with source code + readme.txt + report.pdf

Weight: 4% of the overall course grade

Description of the phase: TBA