Anthony Rusignuolo

CS 600WS – Advanced Algorithms

Reza Peyrovian

Homework 8

I pledge my honor that I have abided by the Stevens Honor System.

1. R-20.7 Consider an initially empty memory cache consisting of four pages. How many page misses does the LRU algorithm incur on the following page-request sequence?  
   (2, 3, 4, 1, 2, 5, 1, 3, 5, 4, 1, 2, 3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 3 | 5 | 4 | 1 | 2 | 3 |
|  |  | | | | | | | | | | | | | |
|  |  | Page Request Sequence | | | | | | | | | | | | |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Cache Page | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 4 |
| 2 |  | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 3 |
| 3 |  |  | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 2 | 2 |
| 4 |  |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  |  | m | m | m | m | h | m | h | m | h | m | h | m | m |

There are 9 misses and 4 hits.

1. C-20.1 Show how to implement a dictionary in external memory, using an unordered sequence so that insertions require only O(1) transfers and searches require O(n/B)transfers in the worst case, where n is the number of elements and B is the number of list nodes that can ﬁt into a disk block.
   1. The dictionary would look like a nested linked list. There would be a head node to start off the first linked list, called the super list, and each node of that list would be a linked list itself, called a sub list, containing B elements. When inserting items into this data structure, note that when the tail node of the super list is full a new tail node must be created and the item must be placed into it. Regardless, insertions still require O(1) transfers. Searches would be done by transferring each node of the super list (i.e. a sub list) to primary memory and iterating over the nodes in the sub list. Regardless of time, this would require O(n/B) transfers because the n nodes are put into blocks of size B, so 1/B becomes the unit size of the super list and the worst case of finding a given node in the super list would be finding the sub list of the tail node.
2. A-20.4 In the MapReduce framework, for performing a parallel computation, a crucial step involves an input that consists of a set of n key-value pairs, (k, v), for which we need to collect each subset of key-value pairs that have the same key, k, into a single ﬁle. Describe an efﬁcient external-memory algorithm for constructing all such ﬁles. How many disk transfers does your algorithm perform?
   1. Multi-way merge-sort  
      disk transfers come from sorting and scanning the sequence for duplicates  
      Perform multi-way merge-sort, then do a linear scan of the key value pairs placing all the pairs with the same key into their own file. The multi-way merge-sort would take O((n/B)log(n/B)/ log(M/B)), B being the size of a block and M being the size of internal memory. The linear scan would take O(n) time. Therefore, in total, the algorithm would run in O((n/B)log(n/B)/ log(M/B)+n)
3. R-23.12 Give an example of an input instance for lexicon matching problem, with just a single pattern in the lexicon, L, that forces the Karp-Rabin algorithm given in Algorithm 23.11 to run in Ω(nm) time.
   1. Executing the second for loop runs in O(nm) time, as described in the text. I believe, since you can guarantee it will compute that hash for all, not some, patterns, you can say that it is already Ω(nm). However, in case that’s not a solid justification, you can ensure that it will be Ω(nm) by making sure that all the words in the lexicon L match the words in text T. That ensures that it runs in nm2 time, which is Ω(nm).
4. C-23.4 Let T be a text of length n, and let P be a pattern of length m. Describe an O(n + m)-time method for ﬁnding the longest preﬁx of P that is a substring of T.
   1. Run the KMP algorithm with a longest prefix substring counter. For example, in algorithm 23.7, create a third variable, LP, and start it at 0. Then, in the else if case just above “j 🡨 f(j -1)” add a line that sets LP equal to j, therefore saving the longest prefix of P. Since KMP runs in O(n+m) time, so will this.
5. A-23.2 Search engines need a fast way to detect and ignore stop words, that is, words, such as prepositions, pronouns, and articles that are very common and carry no meaningful information content. Describe an efﬁcient method for storing and searching a set of stop words in a way that supports stop-word identiﬁcation in constant time for all constant-length stop words.
   1. This can simply be done creating a hash set of all the known stop words. If that word is in the hash set it will evaluate to being a stop word. This search will be done in constant time.