605.420 Foundations of Algorithms Lab 1

The following paper shows an application of hashing to Bioinformatics.

G. Rizk, D. Lavenier, R. Chikhi, ***DSK: k-mer counting with very low memory usage***, Bioinformatics (2013) [PDF] [Webpage]

ABSTRACT

Summary: Counting all the k-mers (substrings of length k) in DNA/RNA sequencing reads is the preliminary step of many bioinformatics applications. However, state of the art k-mer counting methods require that a large data structure resides in memory. Such structure typically grows with the number of distinct k-mers to count. We present a new streaming algorithm for k-mer counting, called DSK (**d**isk **s**treaming of k-mers), which only requires a fixed, user defined amount of memory and disk space. This approach realizes a memory, time and disk trade-off. The multi-set of all k-mers present in the reads is partitioned and partitions are saved to disk. Then, each partition is separately loaded in memory in a **temporary hash table**. The k-mer counts are returned by **traversing each hash table**. Low abundance k-mers are optionally filtered. DSK is the first approach that is able to count all the 27-mers of a human genome dataset using only 4.0 GB of memory and moderate disk space (160 GB), in 17.9 hours.

This lab is intended to study variations of hashing. Since the interest is primarily algorithmic efficiency, we will ignore physical device considerations. You will have 2 hashing schemes: mine and yours. Mine has three variations. You will have three collision handling strategies: linear probing, quadratic probing, and chaining. You will have two variations on the table structure: buckets of size one (120 addressable slots) or buckets of size three (120 spaces arranged as 40 addressable slots). The table below shows all the combinations that you need to handle.

	Hashing Scheme	Bucket size	Collision Handling Scheme	Print Data Items Across
1	Division modulo 120	bucket size = 1	Linear Probing	5 (120 buckets of size 1)
2	Division modulo 120	bucket size = 1	Quadratic Probing	5 (120 buckets of size 1)
3	Division modulo 120	bucket size = 1	Chaining	5 (120 buckets of size 1)
4	Division modulo 113	bucket size = 1	Linear Probing	5 (120 buckets of size 1)
5	Division modulo 113	bucket size = 1	Quadratic Probing	5 (120 buckets of size 1)
6	Division modulo 113	bucket size = 1	Chaining	5 (120 buckets of size 1)
7	Division modulo 41	bucket size = 3	Linear Probing	3 (40 buckets of size 3)
8	Division modulo 41	bucket size = 3	Quadratic Probing	3 (40 buckets of size 3)
9	Your hashing scheme	bucket size = 1	Linear Probing	5 (120 buckets of size 1)
10	Your hashing scheme	bucket size = 1	Quadratic Probing	5 (120 buckets of size 1)
11	Your hashing scheme	bucket size = 1	Chaining	5 (120 buckets of size 1)

- You need to write a routine which does hashing using division. Write it so that it accepts parameters on the modulo divisor and the bucket size so you can use it in schemes 1 through 8.
- You need write your own hashing method. It may NOT be a form of division. Consider multiplication, folding, mid-square, rehashing, universal hashing, etc.
- You need to write methods to handle collisions using linear probing, quadratic probing and chaining (using a linked list within the table). Chaining outside the table, as described in CLRS, will receive no credit.
  - Although it is not a requirement to do so, if you plan carefully, you can write one method with parameters to handle both linear and quadratic probing. For quadratic probing, use c1 = c2 = 0.5 values from the homework problem. Consider repeating schemes 2, 5, 8, and 10 with different values of c_1 and c_2.
  - **Use an open addressing scheme (which means to do the chaining within the table) and use a stack to keep track of free space.** Please note that when the book refers to chaining, it means outside the table, which is different than what is required here. You may use library list methods to handle the linked lists required for the chaining.
- There will be 120 table slots for all the schemes. They will just be grouped differently for schemes 7-8, which uses buckets of size three, meaning each bucket can hold three records. You may assume that the slots are large enough to also contain a pointer or dynamic reference to facilitate the chaining.
- For the required minimal input you are to use the data in the file LabHashingInput.txt. Read the entire file for each scheme. Your program may ignore text lines or blank lines in the input data. Supplement this as necessary with your own input files to exercise your program's features and demonstrate asymptotic cost. At a minimum create files of size 36, 84, and 108. Creating additional files with other characteristics/sizes is strongly encouraged.
- For each scheme, print the table after all the data values are stored, along with statistics on comparisons, primary collisions, secondary collisions, items unable to be stored, plus anything else you think is reasonable or useful. Be sure to handle all reasonable error situations.

ANALYSIS: The written analysis is extremely important. Discuss the ramifications of the different hashing and collision resolution techniques. Compare the schemes and figure out what is good and bad about each one. What would you do to address some of the problems you encountered?   What did you learn about load factors? What are the ramifications of deleting items from your hash table? Why is it useful and relevant to Bioinformatics? The analysis should include comments about what you learned, what you might do differently next time, justification of your design decision, and issues of efficiency with respect to time and space. Your analysis must include a table and a graph of the asymptotic costs that you observed. Be sure to consider your experiences with the problem and particularly consider the efficiency. Why is it useful and relevant to Bioinformatics?

Be sure to review the Programming Assignments Guidelines.

For the schemes that use buckets of size 1, print 5 across like this
xxxxx xxxxx xxxxx xxxxx xxxxx
xxxxx xxxxx xxxxx xxxxx xxxxx
xxxxx xxxxx xxxxx xxxxx xxxxx
etc for a total of 24 lines.

For the schemes that use buckets of size 3, print 3 across like this
1 XXXXX XXXXX XXXXX
2 XXXXX XXXXX XXXXX
...
38 XXXXX XXXXX XXXXX
39 xxxxx xxxxx xxxxx
with 1 line for each bucket.

With the buckets of size three, using 27 as an example,
27 AAAAA BBBBB ———
showing AAAAA as the first item to hash to location 27 and BBBBB as the second item hashing to location 27 and so far nothing else hashed to location 27. You must print something in each slot. **If there is no key, print a dashed line**.

Collect cost data by counting the number of comparisons needed to insert all the items from a file.