605.420: Algorithms for Bioinformatics

Jack Zhan

Project 2 Analysis

04/12/2016

Collision Analysis

The analysis below was used to determine which probing method was most efficient in reducing the number of collisions. For Quadratic probing, the values listed for the variables are: C1=1 c3=3 for Quadratic Probing 1, C1=2 c3=2 for Quadratic Probing 2, and C1=3 c3=1 for Quadratic Probing 3.

The first series of test tried to determine if changing the modulo had any effect on the number of collision encountered when hashing the function.

The table bellows shows the Collisions for the 3 methods when the modulo is 120.

	Number of Collision	Total Number of Collision
Linear Probing	19	30
Quadratic Probing	38	93
Quadratic Probing 2	41	310
Quadratic Probing 3	40	211
Chaining	17	23

Based on the results above the chaining method has the least amount collisions while quadratic probing was the worst in all 3 cases.

The table bellows shows the Collisions for the 3 methods when the modulo is 113.

	Number of Collision	Total Number of Collision
Linear Probing	17	30
Quadratic Probing	32	53
Quadratic Probing 2	33	55
Quadratic Probing 3	33	55
Chaining	14	14

The table bellows shows the Collisions for the 3 methods when the modulo is 110.

	Number of Collision	Total Number of Collision
Linear Probing	20	37
Quadratic Probing	43	101
Quadratic Probing 2	43	200
Quadratic Probing 3	43	138
Chaining	18	27

Based on the results above, selecting a prime number for the modulo slightly reduced the amount the amount of collisions when compared to a non-prime number.

The next test was determining whether having a bucket size increased the amount of collisions.

The table bellows shows the Collisions for the 2 methods with 3 buckets when the modulo is 41.

	Number of Collision	Total Number of Collision
Linear Probing	28	45
Quadratic Probing	56	97
Quadratic Probing 2	56	97
Quadratic Probing 3	56	97

The results showed that reducing the size of the table and adding buckets, greatly increased the amount of collision when hashing the list of values.

Finally, the last test was used to determine if my custom hashing function Key^2 % modulo had any effect on the collision count.

The table bellows shows the Collisions for the 3 methods when the modulo is 120 and custom function.

	Number of Collision	Total Number of Collision
Linear Probing	19	26
Quadratic Probing	40	128
Quadratic Probing 2	41	306
Quadratic Probing 3	39	82
Chaining	17	23

The result showed that my custom function greatly increased the collision count for quadratic probing but did effect the collision count much for linear probing and chaining.

What I Learned

The results from above confirmed that linear probing consistently had the least amount of collision when compared to linear and quadratic probing. Selecting a prime number was beneficial in reducing the number of collision. Finally the use of buckets actually increase the total number of collisions. In future test, I would like to test with an increased load factor and add deletion functionality.

Efficiency:

I believe that my algorithm was efficiency in storing the data into the hash table. The problem I was running into was there was no easy way to properly implement chaining with a table, so I to create my own class similar to a linked list.

Justification for Design Decision:

It was easier to code the file reader to directly read the data of the input file into array lists. This allowed me to split reading the file and running the algorithm into separate functions. In addition I had the program directly print the result so I could check if the algorithm was running correctly. In addition, I printed the collision counts for further analysis. I did not implement code to delete values from the table as this was not required. This would be easy as I would be the same as inserting the value and going through multiple collisions until the value is reached. For the chaining method, the pointers for the linked list also have to be updated.

Relevance to Bioinformatics:

This type of analysis is important in bioinformatics because the amount of sequences one analyzes can easily become very large. Storing these sequences and accessing them becomes extremely important and optimizing the hashing method and probing method to reduce runtime.