Project 3
Report

Hakan Eröztekin

Important Information About Compilation

Please use c++11 or above and use two steps when compiling the file.

1 - Compile:

```
g++ -std=c++11 main.cpp -o output.o
```

2- Execute:

```
./output.o vocab.txt search.txt
```

Screenshots

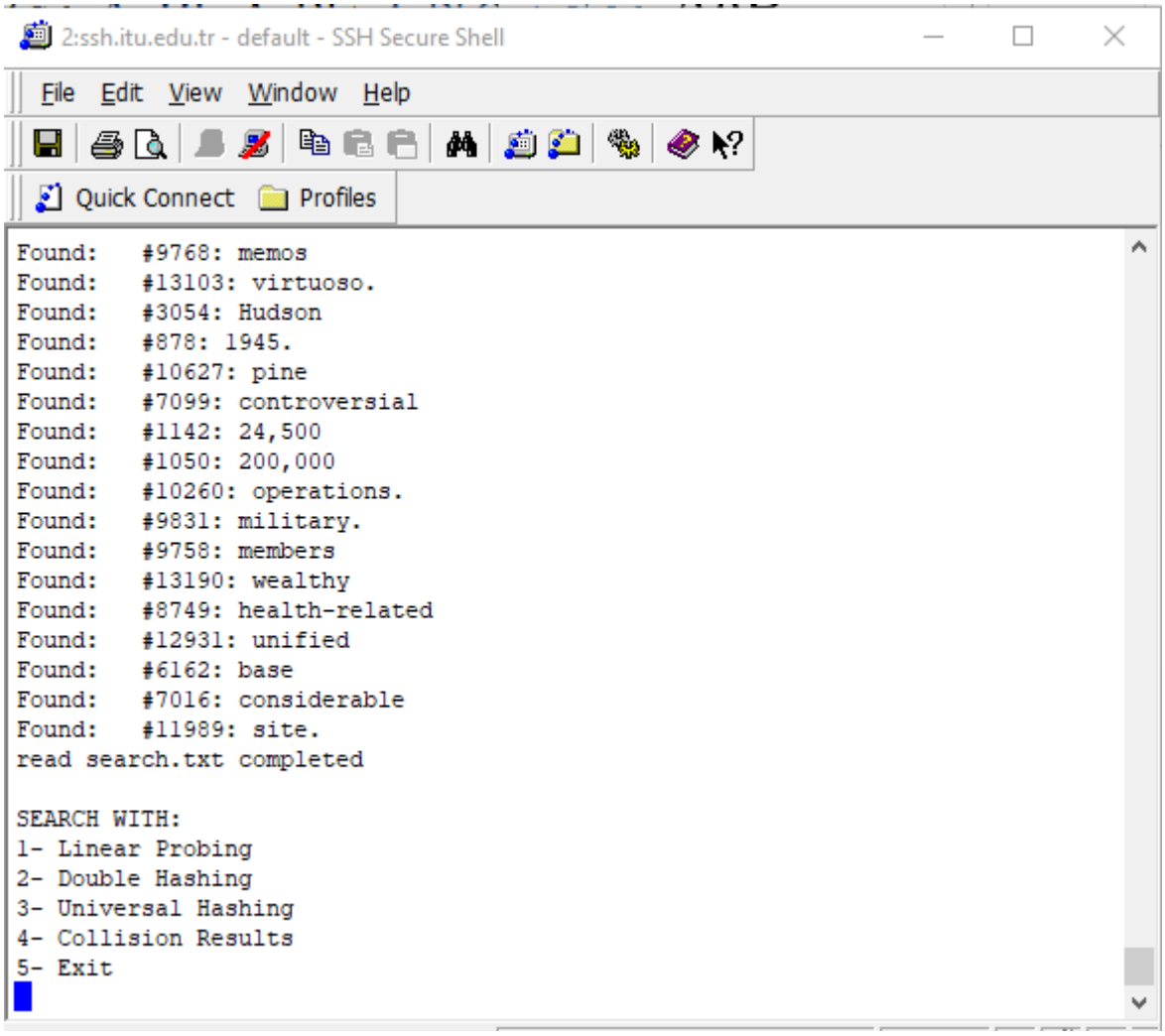

Figure 1 – Hashing

About part 5. in the homework;

m=17863	m=21929
Collusion Results. Collision On Insertion #1 Linear Probing: 0 #2 Double Hashing: 0 #3 Universal Hashing: 4986 Collision On Search #1 Linear Probing: 9965969 #2 Double Hashing: 9965969 #3 Universal Hashing: 20940603	Collusion Results. Collision On Insertion #1 Linear Probing: 0 #2 Double Hashing: 0 #3 Universal Hashing: 4043 Collision On Search #1 Linear Probing: 9965969 #2 Double Hashing: 9965969 #3 Universal Hashing: 19547466

Insertion

	Linear Probing	*Double Hashing*	*Universal Hashing*
m = 17863	0	0	4986
m=21929	0	0	4043

Search

	Linear Probing	*Double Hashing*	*Universal Hashing*
m = 17863	9965969	9965969	20940603
m=21929	9965969	9965969	19547466

Linear probing and Double hashing demonstrates the same results in this test. There are zero collision on insertion because the table size was always bigger than the text files' line number. They always find an empty cell in each insertion.

On the other hand, there are a lot of collisions for universal hashing because even though the random values are included, because there is not a very big gap between table size (m) and total line count in the text, when "modm" is calculated, there were a high probability for collision.

Similar thing applies for searching operation. Linear probing and double hashing iterates through the table so there are a lot of collision happened. About the universal hashing, its result is much greater (I had to use unsigned long long int) because hashed index value (i.e line number) was used in both insertion hash and searching hash. So that elevated the probability.

Thank you for your time.

Hakan Eröztekin