Shreya Patel RUID:177007543

Assignment 6

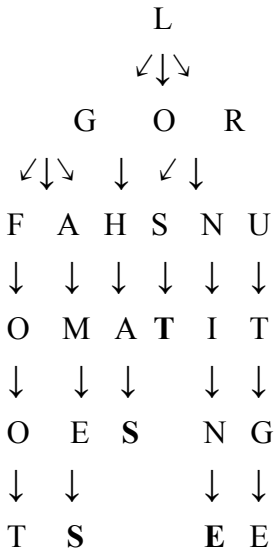
1. Hash-Tables
    a. Separate-Chaining

Index	0	1	2	3	4	5	6	7	8	9
Values	(B) 1400	(A) 1093, (D) 7652	(C) 3341, (E) 4321	-	(F) 5674	-	(G) 8980	-	-	-

    b. Linear Probing

Index	0	1	2	3	4	5	6	7	8	9
Values	1400	1093	3341	7652	4321	5674	8980	-	-	-

2. A solution to this problem would be to use multiple hash table functions in order to search if the value exists in the array but you don't need to save the value which means that it wouldn't be stored. So in order to make sure that all the numbers don't collide, you need 10 hash functions, for each numerical value of 0-9. Since all the numbers would fall into that range you need a hash function for each possibility. So every time a client asks to find a number you have to repeat the search since you can't save the values present since there is no way of storing them.
3. Tries
    a. The height of the trie is 9.
    b. The height of the Ternary Search Trie is 10.



↓ ↓
B R
↓ ↓
A **S**
↓
L
↓
L

c. In order to minimize the height of the tree you have to start with FOOTBALL, branching to the right from that should be LOST. Branching to the left of lost should be HAS and branching to the right of lost should be RUTGERS. Going to the left from HAS should be GAMES and going left of RUTGERS should be NINE. This TST gives a height of 9.
d. In order to maximize the height of the tree you have to start with RUTGERS, branching to the left should be HAS. Going to the left of HAS should be GAMES and going to the right should be NINE. Going to the left of GAMES should be FOOTBALL and going to the left of NINE should be LOST. This TST will give a height of 11.

4. If prefix free code isn't used in Huffman Coding there wouldn't be a problem in encoding but rather a problem in decoding. The reason for this is because while encoding you can follow the code assigned per input character without thinking twice about it because it is straight conversion. It would, however, create a problem in decoding because there is ambiguity as to which input character every code belongs to. Depending on the way each programmer views the code the decoded version could be different. For example, if we assign the characters (a=00, b=01, c=0 and d=1), the following code "0101" could have several different answers while decoding. The programmer could have meant to write "bb" but it could be written as one of these possibilities: ("bb", "cdcd", "bcd" or "cdb").

5. Data Compression worst and best cases
    a. Run Length coding of N bits:
        i. Best case: If all N bits of the code are the same throughout, then it can be converted to the number of times the character occurs and the character itself. So the compression ratio for the best case would be 2^{N-1}. For example, '000000000' can be converted to 9'0' which is the best case compression for the string.

ii. Worst case: If all the characters of the code that are next to each other keep on changing. This would result in the compression of the code being the same as the actual code because nothing can be shortened since the characters keep on changing that are next to each other. So the worst case compression ratio would be $(2^{N-1} + 1)$ bits. For example, '01010101' would be compressed to '01010101'. There is no change between the compression and the original code.

b. Huffman Coding of characters
   i. Best case: The best case for Huffman Coding would be when there is a normal distribution of frequency for the characters used. This would ensure that each character is used the average times which would be efficient.
   ii. Worst case: The worst case for Huffman Coding is when the distribution of the characters used follows the fibonacci numbers. This is due to the fact that distribution of frequencies increases quickly causing there to be an unbounded inefficiency.

c. LZW
   i. Best case:The best case would be if there are only 2 characters being used and they keep on alternating. This way you only introduce them individually once and then you can keep on combining them adding one more every time in order to get a different set as you go farther and farther into the code. The compression rate would be N for the best case.For example, 'ABABABA' would be '41 42 81 83 80'.
   ii. Worst case: The worst case for LZW is if all the characters in the code are the exact same. This way there would be no way of combing them in different ways since they are all the same. For example, if the code said 'AAAAAA' it would translate to '41 41 41 41 41 41 80'.

6. Sequence: ABAACDFABACD
   Combination: **ABAACDFABACD**
   LSW Compression: 41, 42, 41, 41, 43, 44, 46, 81, 83, 44