Assignment 6: Hash Tables, Data Compression, Tries

In this assignment, you will solve problems on directed and undirected graphs and submit the solution as a single PDF through Github.

You will push all of your solutions to the Assignment 4 repository through the Github Classroom. Organize your repository in such a way that it contains:

Root – Assignment 6 directory

- Assignment 6.PDF

Note: You do not have to submit any runnable code.

There is no restriction on the programming language that you use.

For the problems, you will be implementing your code as an API through which the graders will make function calls.

Include a README.md file for the GitHub repository that has your Name as the title and a description of what is inside the repository, generally.

!! INVITATION LINK https://classroom.github.com/a/35yN32h7 !!

For anyone new to Github, we encourage you to use **Github Desktop** as a simple GUI for interacting: https://help.github.com/desktop/guides/getting-started-with-github-desktop/

For people who want to be more advanced and up to industry standards, you can use the Git command line:

https://git-scm.com/downloads

A very barebones introduction to the command line (very readable, I recommend it):

http://rogerdudler.github.io/git-gu ide/

1. An array {1093, 1400, 3341, 7652, 4321, 5674, 8980} is given. Assuming the hashcodes for the integers is the same as the value of those integers, use the hash function $h(x) = x \% N$, where N is the size of the hash table to solve these problems:
    a. Separate Chaining, N = 10
        i. Represent your answer as a two row table, the first row contains the index and the row contains a comma separated list of all values hashed to that index. The list should in order of first inserted to last inserted.
    b. Linear Probing, N = 10
        i. Represent your answer as a two row table, the first row contains the index and the second row contains the value hashed to that index.
2. Your client has one million objects in a set, and wants to support queries on the set to check whether or not an object exists in the set. Your client is averse to hashing the information since the hash solutions in Problem 1 requires storing the hashed values, thus costing too much space (for example, reference space requirements of hashing in the hash tables, tries, and data compression slides). The client is fine saying an object is in the set when it is not, but only if the possibility of that occurring is very small. Explain how to solve this problem using hashing without storing the actual values. Your description does not require any pseudocode, but should follow logically and be explained fully. Give an example of how your solution might work in your explanation.
3. Suppose the sequence of strings: {RUTGERS, FOOTBALL, HAS, LOST, NINE, GAMES } are inserted into an R-way trie, with values all equal to '1'.
    a. What is the height of the trie? Consider the root to be level '0'.
    b. Insert the values in-sequence into a TST. What is the height of the trie?
    c. What insertion order minimizes the height of the TST, and what is the height?
    d. What insertion order maximizes the height of the TST, and what is the height?
4. What happens if prefix free codes aren't used in Huffman coding – is it a problem with encoding or decoding or both?
5. Explain the sequences that generate the worst and best case compression ratios occur for the below compression algorithms, where the input is
    a. Run-length coding of B-bit counts for inputs of N bits
    b. Huffman Coding of B-bit characters for inputs of N characters
    c. LZW compression of B-bit characters for inputs of N characters with L-bit codewords

    List the compression ratios for each scenario as a function of these parameters, depending on compression:
    - **Based on the example in the slides, the best case scenario of RLC is when we have an input sequence N = 2^(B)-1, which can be encoded by (B) bits, for $\frac{B}{2^B-1}$.**

6. In the lecture for LZW Compression, I said to take it for granted that ‘41’ maps to A, ‘42’ maps to B, and so forth. The reason is because the authors did not include the codeword dictionary for converting between hexadecimal and ASCII. This is pictured below:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

This table is used by looking up the row first, then the column (A is row 4, column 1).
Using this table for referencing the base characters, expand the dictionary and encode the below sequence using LZW compression:

A B A A C D F A B A C D