Jahidul Islam - 171001155 - PMII HW6

Problem 1

A) Separate Chaining with N=10
1093%10=3 which would be Index 3
1400%10=0 which would be Index 0 and so on...

0	1	2	3	4	5	6
1400 8980	3341 4321	7652	1093	5674		

B) Linear Probing with N=10

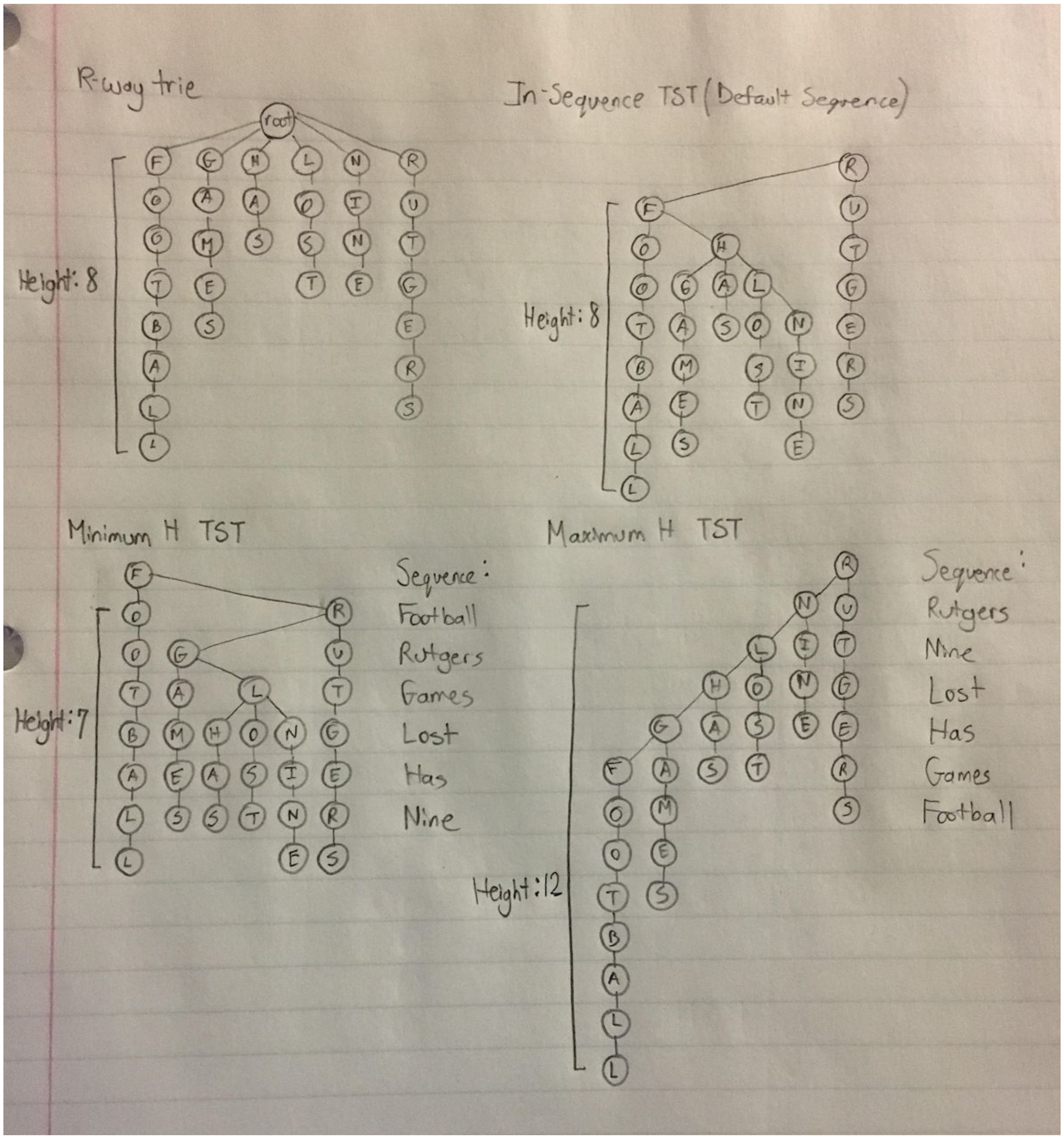
0	1	2	3	4	5	6
1400	3341	7652	1093	4321	5674	8980

When we get to 4321 it would have Index 1 but there is a value there so we add one and then modulo again but there is a value at Index 2 so add again. Value already at index 3 so add 1 again now we can place it at Index 4. Same process with 5674 and 8980.

Problem 2

In order to perform a check of whether an object exists in the set,we need to use multiple hash functions to hash each object. We then store a boolean value to indicate whether the object has been stored. For example, we start by using hash functions to convert objects in set but do not store the objects. Instead,we store whether the object is there with a boolean.Then, we see whether a selected object is in the set or not based on the value of the boolean.

Problem 3



Problem 4
If prefix free codes aren't used in Huffman coding, then it will cause problems in both encoding and decoding.

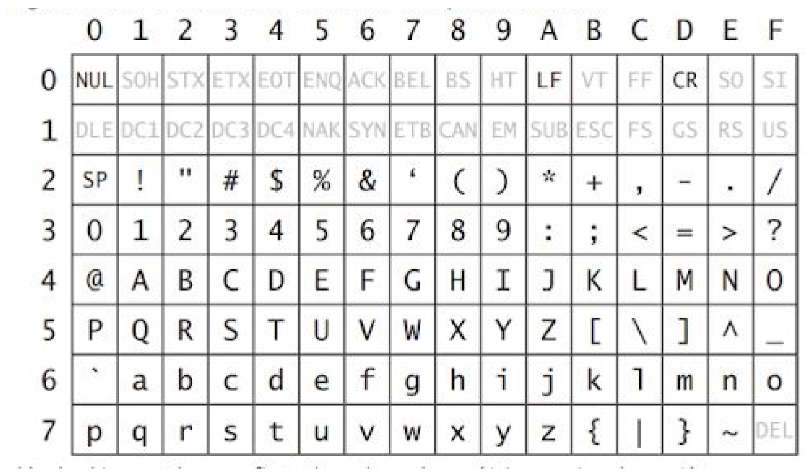
Huffman Codes are known as prefix free codes which means no code word can be prefix of any other code word. If for example the code word for "f" is 50 it means no other code word can begin with 50. Thus, if prefix free codes aren't used then problems arise in encoding and will create confusion. Prefix free codes allows the decoding process to follow a greedy approach. For example, in this, the user read the encoded string from left to right. When a code word is found, a character can be outputted. If we take the character 'F' and set it to 0000, now 0, 00 and 000 does not code anything and the user continues bit reading. When the code reaches this '0000' it encodes "F" as there can not be any other code word of this form (0000x) since it is prefix free. From our first example, if we have a 5, then it encodes nothing but a 50 encodes 'f'. Thus, if prefix free codes aren't used in Huffman coding then it will cause a problem while attempting to decode.

Problem 5
A) We are given B-bit counts for input N bits. First of all, let's say we have B=8 bit counts so the max run length is 2^8 or 256. N=2^B-1 so 2^8-1 =255. To get the best compression ratio you would need a sequence of 255 1's which can be represented with 8-bit count 11111111. This would have the best compression ratio of 8/255 meaning you would need a sequence of N bits of the same value for the best compression. The worst case would mean there is no length of the same value meaning the sequence could look like 010101010101…

B) For Huffman,the best case would happen when the input has only one unique character where each character would correspond with one digit representation for compression. The worst case is when the distribution of frequencies follows for example the Fibonacci numbers. The worst case occurs when an input has multiple instances of a character. This particular algorithm utilizes the frequency of appearance of characters in the input. In order to counter the fact that something that appears the most will have the shortest representation, we have to make the characters appear with the same frequency as present in the input which will make one of these characters retain the representation for a bad compression ratio.

C) Similar to before, LZW would be best when the input has one unique character since LZW searches for input patterns. With a unique character it would always find a new pattern and use it. Thus, the opposite, where the input is random with no patterns at all, would contribute to the worst case compression as the time to encode would increase as patterns decrease.

Problem 6



	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

We Need:

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

Breaking up by pattern detection we have:

A B A A C D F AB AC D

The values we need can be obtained from the following table::

key	value
A	41
B	42
C	43
D	44
F	46
AB	81
BA	82
AA	83
AC	84
CD	85

DF	86
FA	87
ABA	88
ACD	89

Thus we have:

Input	A	B	A	A	C	D	F	A	B	A	C	D	Termination
Matches	A	B	A	A	C	D	F	AB	-	AC	-	D	
Value	41	42	41	41	43	44	46	81		84		44	80