PSO 13

Compression, Pattern Matching

Why compression

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.alpharithms.com

Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

Can you generalize your answer to find the Huffman codes when the frequencies are the first n Fibonacci numbers?

(2) A code is called **optimal** if it can be represented by a full binary tree, in which all of the nodes have either 0 or 2 children. Is the optimal code unique?

Question 2

(Trie & lexicographic sort)

Given two bit strings $a = a_0a_1 \dots a_p$ and $b = b_0b_1 \dots b_q$, we assume WLOG that $p \leq q$. Recall that a is said to be **lexicographically less** than b if one of the following happens:

- there exists an integer $j \leq p$ such that $a_i = b_i$ for all $0 \leq i < j$ and $a_j < b_j$.
- $p < q$ and $a_i = b_i$ for all $0 \leq i \leq p$.

Given a set S of distinct bit strings whose lengths sum to n, show how to use a radix tree (a.k.a. trie for bit strings) to sort S lexicographically in $O(n)$ time. For example, if $S = \{1011, 10, 011, 100, 0\}$, then the output should be the sequence 0, 011, 10, 100, 1011.

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

2. Is there any other pattern matching algorihtm that works better in this scenario?

Question 4

(Forward pattern matching)

Another efficient pattern matching algorithm, named the Knuth-Morris-Pratt (KMP) algorithm, is based upon forward pattern matching, in which a failure function (also named as "suffix function") is calculated to determine the most distance we can shift the pattern to avoid redundant comparisons. Specifically, for a pattern P, its corresponding failure function $F_P(j)$, or $F(j)$ for short, is defined as

$$F(j) := \max_k \{k \leq j-1 : P[0:k] = P[j-k:j]\}.$$

In other words, $F(j)$ represents the size of the largest prefix of $P[0:j]$ that is also a suffix of $P[1:j]$.

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Answer the following questions:

(1) Apply the KMP algorithm to the pattern matching problem in Question 1. Does it perform much better than Boyer-Moore?

(2) What is the failure function for the pattern $P :=$ "mamagama"?

(3) Let $T :=$ "rahrahahahahromaromamagagaoohlala", run the KMP pattern matching algorithm for the pattern P in (2).

Question 1

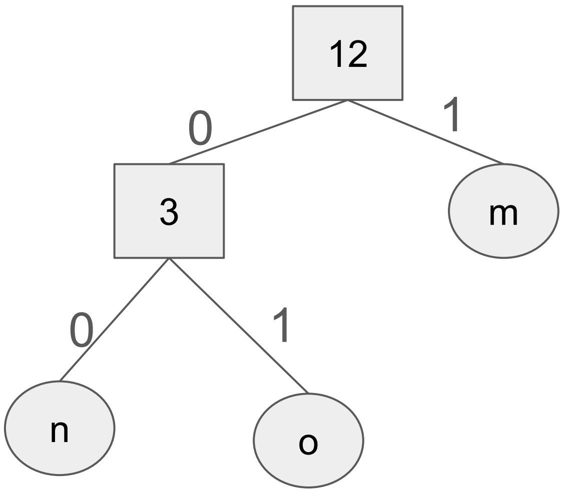
(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

Huffman Idea: Compress the most frequent letters to be shortest, an example..



Inner-nodes: freqs
Leaves: letters

What is the most freq. letter? What's the encoding of 'o'? 'n'? 'm'?

Question 1

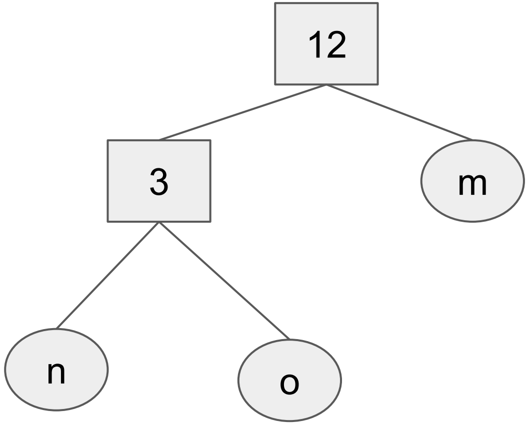
(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

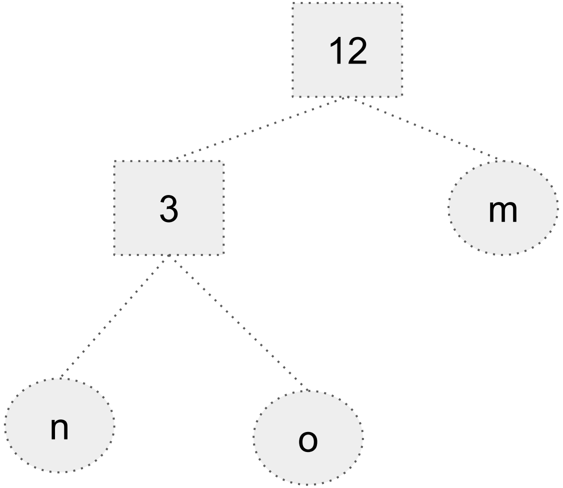
Huffman Idea: Compress the most frequent letters to be shortest



Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

Quick example: start off with freqs

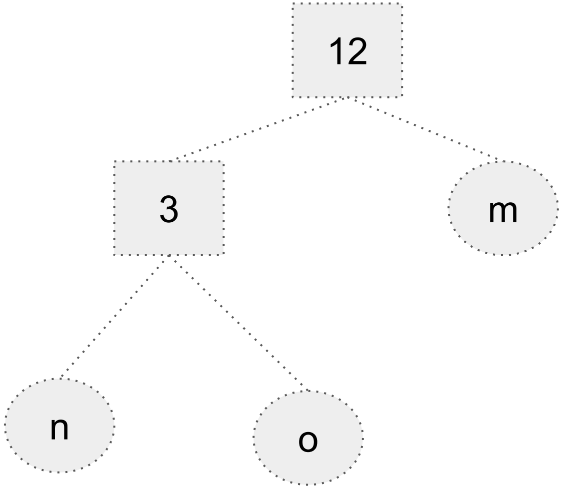


Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Quick example

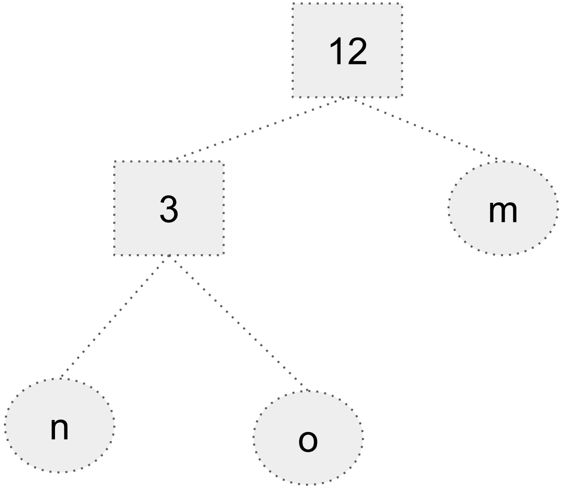


Steps:

1. **Add all letters to minHeap by their frequencies**
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Quick example



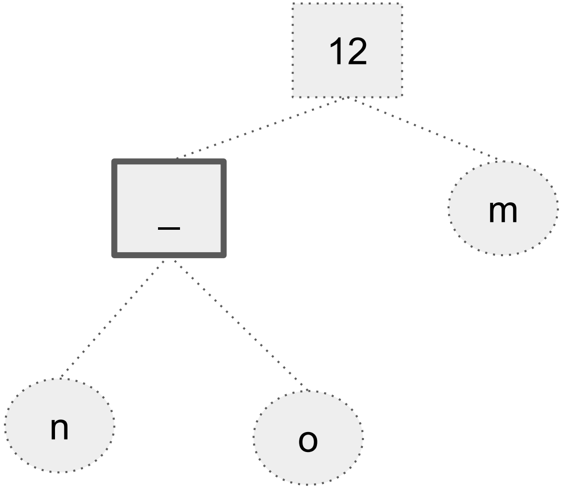
Steps:

1. **Add all letters to minHeap by their frequencies**
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q
(1,n)
(2,n)
(9,m)

Quick example: Step 2



Steps:

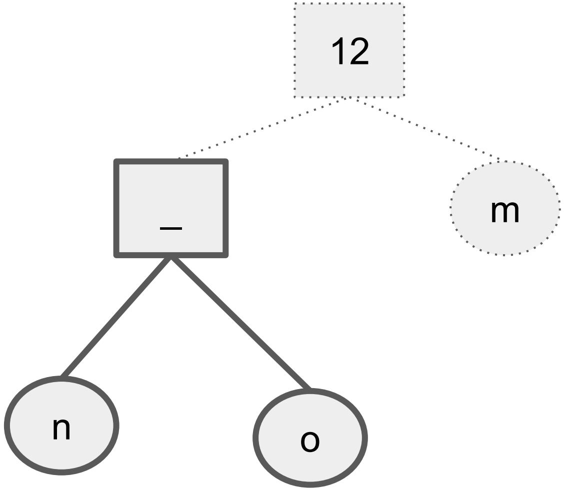
1. **Add all letters to minHeap by their frequencies**
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q
(1,n)
(2,n)
(9,m)

Step 2 in-depth:
2a. Initialize node curr

Quick example: Step 2



Steps:

1. **Add all letters to minHeap by their frequencies**
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

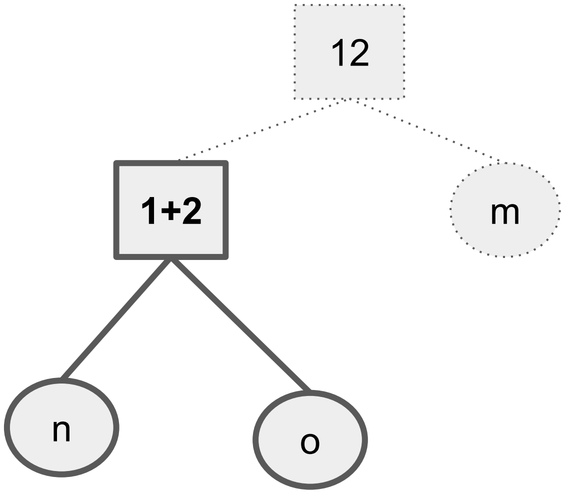
Q
~~(1,n)~~
~~(2,n)~~
(9,m)

Step 2 in-depth:

2a. Initialize node curr

2b. Set children to be next two minHeap elts

Quick example: Step 2



Steps:

1. **Add all letters to minHeap by their frequencies**
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

~~(1,n)~~

~~(2,n)~~

(9,m)

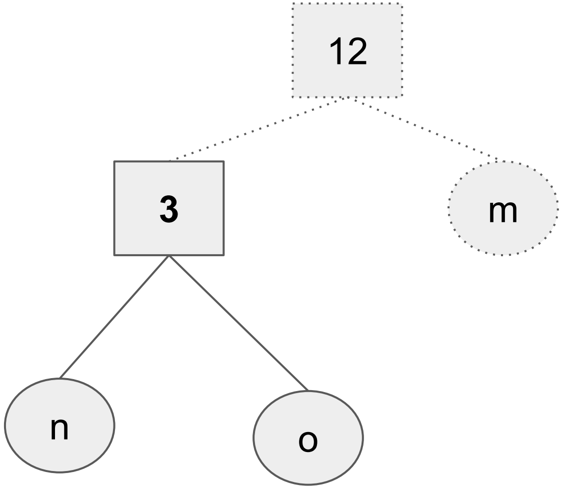
Step 2 in-depth:

2a. Initialize node curr

2b. Set children to be next two minHeap elts

2c. curr.freq = Add up freq of children

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat**

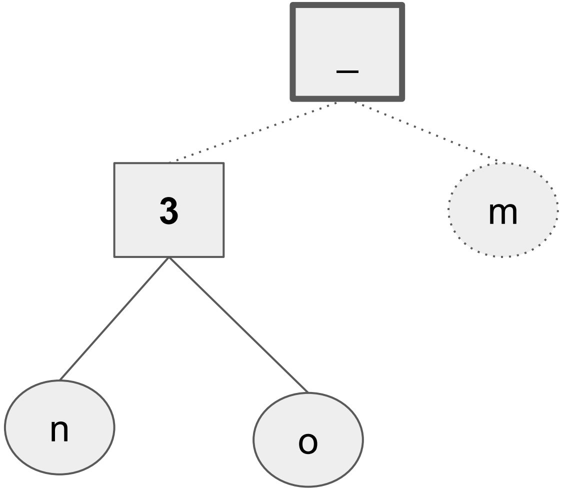
m	n	o
9	1	2

<u>Q</u>

(3,no)

(9,m)

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

(3,no)

(9,m)

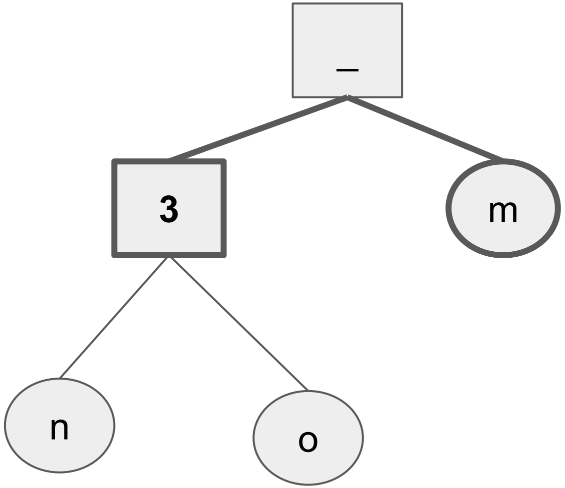
Step 2 in-depth:

2a. **Initialize node curr**

2b. Set children to be next two minHeap elts

2c. curr.freq = Add up freq of children

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

<u>Q</u>

~~**(3,no)**~~

~~(9,m)~~

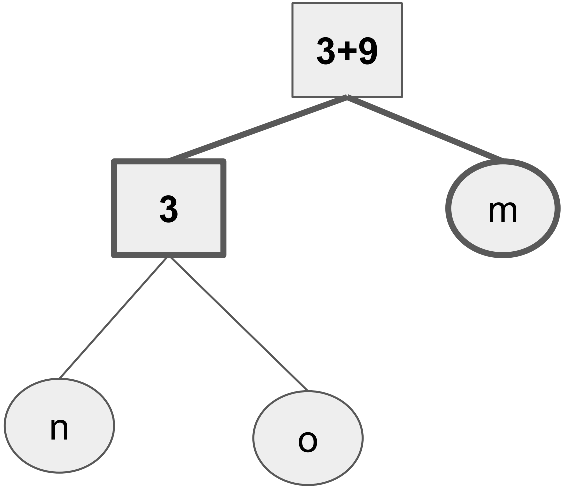
Step 2 in-depth:

2a. Initialize node curr

2b. **Set children to be next two minHeap elts**

2c. curr.freq = Add up freq of children

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

~~**(3,no)**~~

~~(9,m)~~

Step 2 in-depth:

2a. Initialize node curr

2b. Set children to be next two minHeap elts

2c. curr.freq = Add up freq of children

Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

Q

(1,a)
(1,b)
(2,c)
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. **Add all letters to minHeap by their frequencies**
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

Question 1

(Huffman codes)

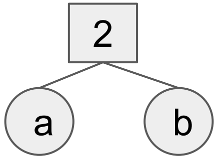
Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

Q

~~(1,a)~~
~~(1,b)~~
(2,c)
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

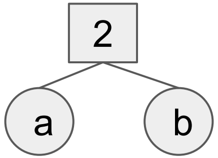
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(2,ab)
(2,c)
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

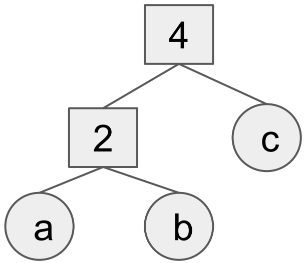
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(2,ab)~~
~~(2,c)~~
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

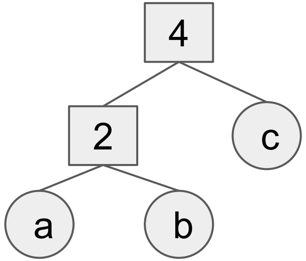
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q
(3,d)
(4,abc)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

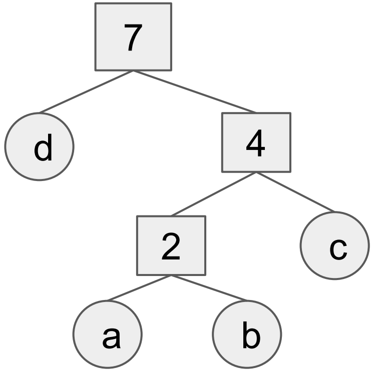
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$



Q

~~(3,d)~~

~~(4,abc)~~

(5,e)

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

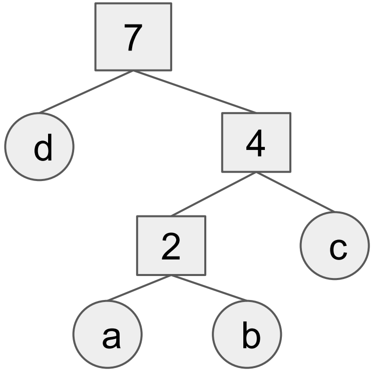
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(5,e)

(7,abcd)

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

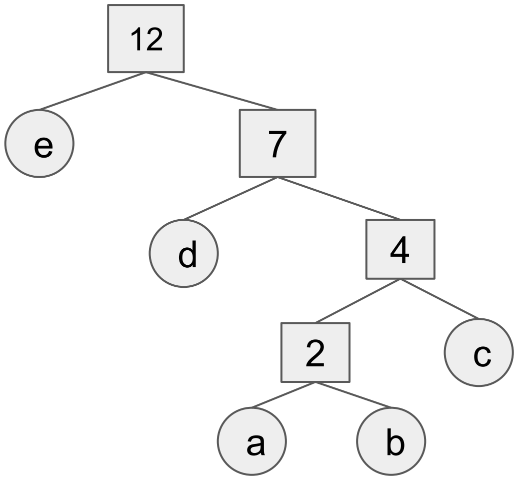
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(5,e)~~

~~(7,abcd)~~

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

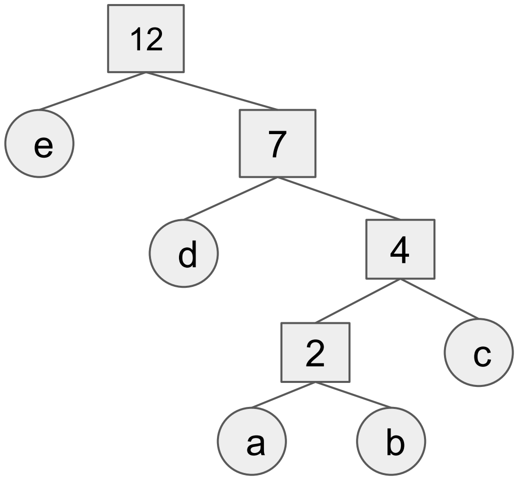
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(8,f)

(12,abcde)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

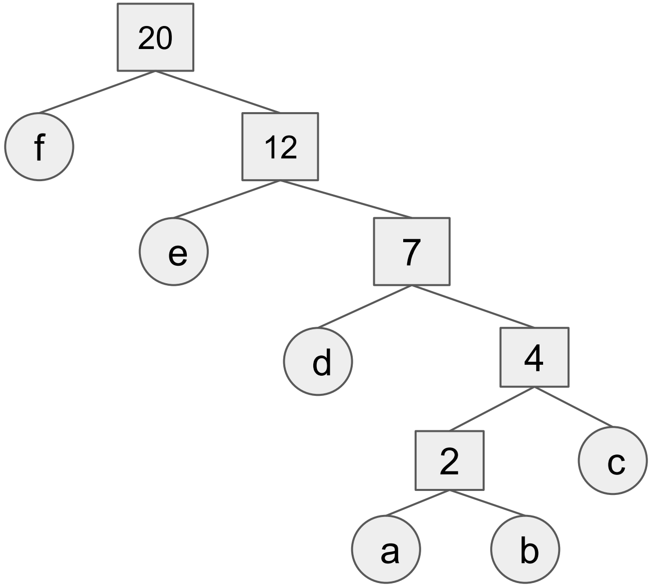
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(8,f)~~

~~(12,abcde)~~

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

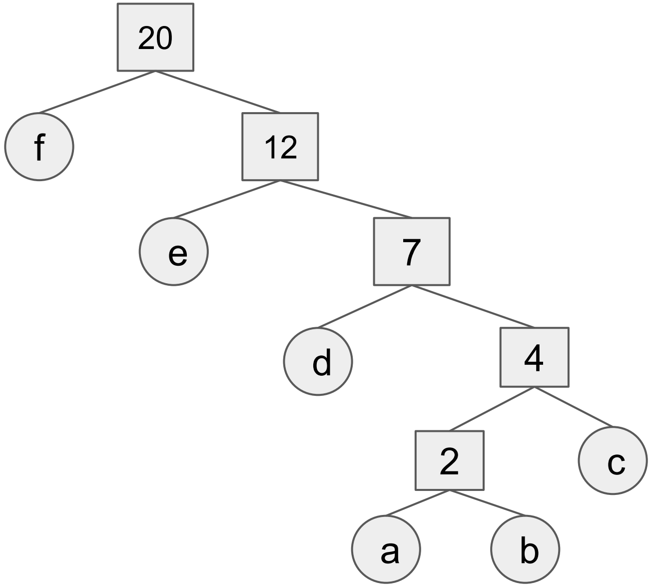
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(13,g)

(20,abcdef)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

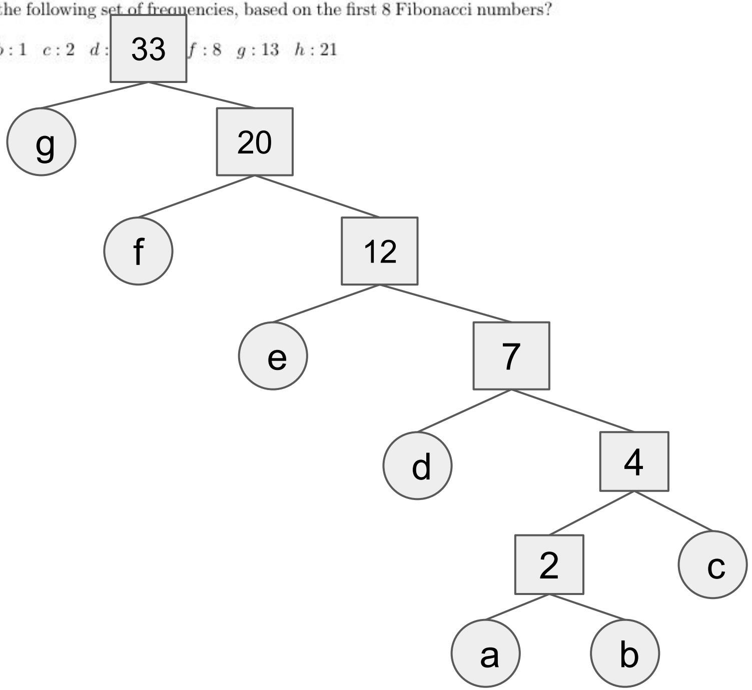
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1 \quad b:1 \quad c:2 \quad d:$ ... $f:8 \quad g:13 \quad h:21$



Q

~~(13,g)~~

~~(20,abcdef)~~

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

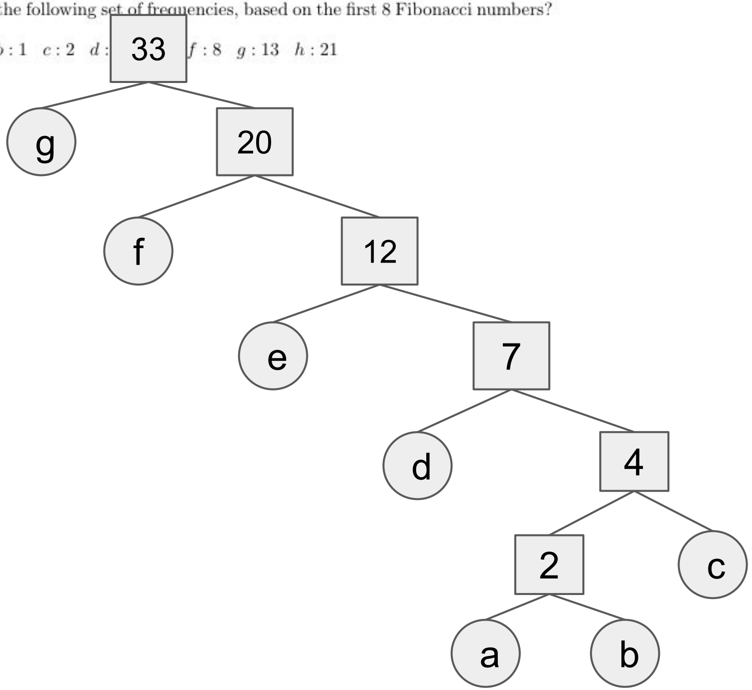
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1 \quad b:1 \quad c:2 \quad d:$ … $f:8 \quad g:13 \quad h:21$



Q

(21,h)

(33,abcdefg)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

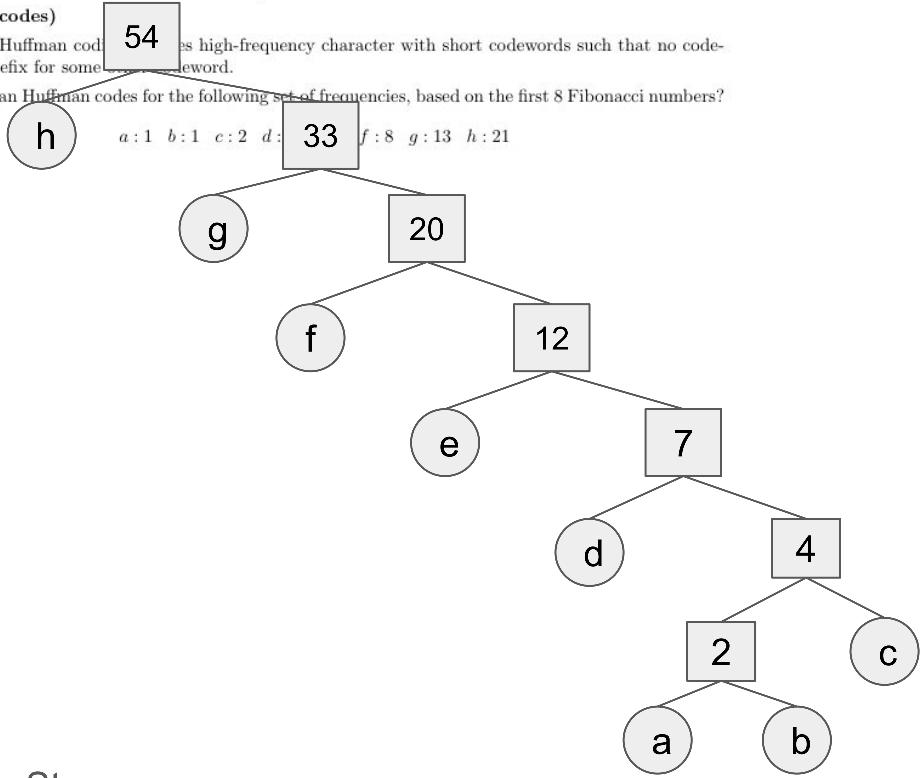
Question 1

(Huffman codes)

Recall that Huffman cod[...]es high-frequency character with short codewords such that no code-word is a prefix for some [...]eword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1 \quad b:1 \quad c:2 \quad d:$ [...] $f:8 \quad g:13 \quad h:21$



<u>Q</u>

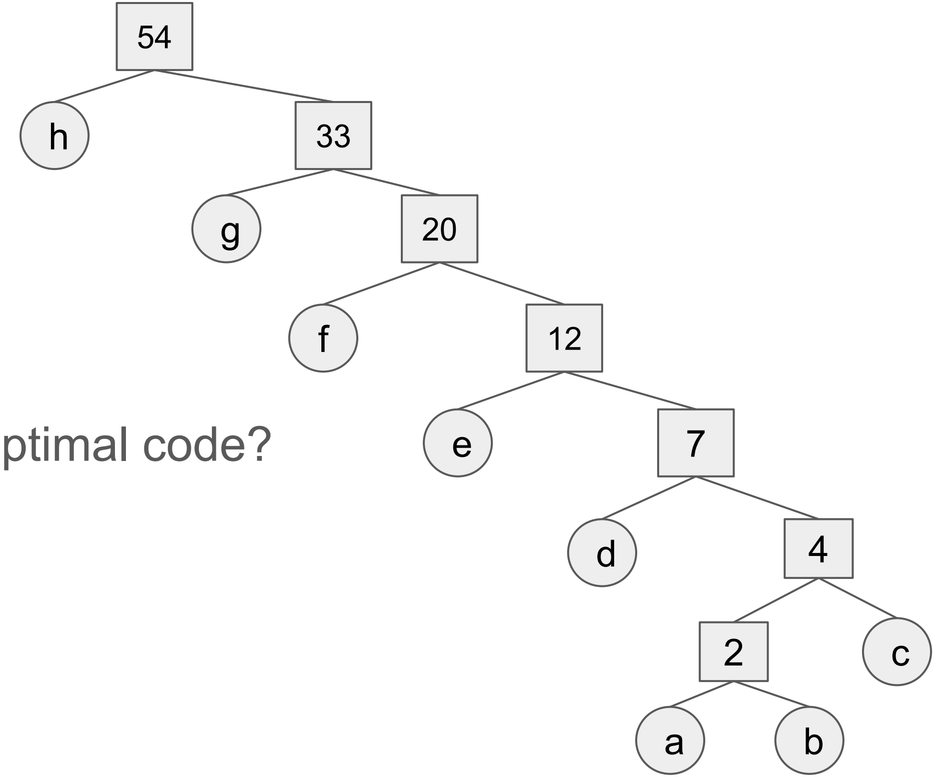
~~(21,h)~~

~~(33,abcdefg)~~

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

(2) A code is called **optimal** if it can be represented by a full binary tree, in which all of the nodes have either 0 or 2 children. Is the optimal code unique?



Can I get another optimal code?

Question 2

(Trie & lexicographic sort)

Given two bit strings $a = a_0a_1 \ldots a_p$ and $b = b_0b_1 \ldots b_q$, we assume WLOG that $p \leq q$. Recall that a is said to be **lexicographically less** than b if one of the following happens:

- there exists an integer $j \leq p$ such that $a_i = b_i$ for all $0 \leq i < j$ and $a_j < b_j$.
- $p < q$ and $a_i = b_i$ for all $0 \leq i \leq p$.

Given a set S of distinct bit strings whose lengths sum to n, show how to use a radix tree (a.k.a. trie for bit strings) to sort S lexicographically in $O(n)$ time. For example, if $S = \{1011, 10, 011, 100, 0\}$, then the output should be the sequence 0, 011, 10, 100, 1011.

Lexigraphic Ordering Practice:

- ‘c’ vs ‘ab’
- ‘abc’ vs ‘abca’
- ‘abbbbb’ vs ‘baaaaa’

Question 2

(Trie & lexicographic sort)

Given two bit strings $a = a_0 a_1 \ldots a_p$ and $b = b_0 b_1 \ldots b_q$, we assume WLOG that $p \leq q$. Recall that a is said to be **lexicographically less** than b if one of the following happens:

- there exists an integer $j \leq p$ such that $a_i = b_i$ for all $0 \leq i < j$ and $a_j < b_j$.
- $p < q$ and $a_i = b_i$ for all $0 \leq i \leq p$.

Given a set S of distinct bit strings whose lengths sum to n, show how to use a radix tree (a.k.a. trie for bit strings) to sort S lexicographically in $O(n)$ time. For example, if $S = \{1011, 10, 011, 100, 0\}$, then the output should be the sequence 0, 011, 10, 100, 1011.

Form the trie for S

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	**a**	a	a	a
P	b	a	a	a	a	**a**			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	**a**	a	a	a	a
P	b	a	a	a	**a**	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	**a**	a	a	a	a	a
P	b	a	a	**a**	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	**a**	a	a	a	a	a	a
P	b	a	**a**	a	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	**a**	a	a	a	a	a	a	a
P	b	**a**	a	a	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	**a**	a	a	a	a	a	a	a	a
P	**b**	a	a	a	a	a			

T[0] does not equal P[0]! Next steps..

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

T[0] does not equal P[0]! Next steps.. We mismatched on target a

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

T[0] does not equal P[0]! Next steps.. We mismatched on target a

The last occurrence of pattern a

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

Move P (to align target a with pattern a) OR (one after target mismatch)

Whichever moves P the *least* amount – in this ex. We move one after mismatch

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P		b	a	a	a	a	a		

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

<u>Boyer-Moore</u>: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	**a**	a	a
P		b	a	a	a	a	**a**		

Fast forward..

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	**a**	a	a	a	a	a	a	a
P		**b**	a	a	a	a	a		

Fast forward.. Same mismatch, jump 1

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P			b	a	a	a	a	a	

Same thing will happen 1 more time

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

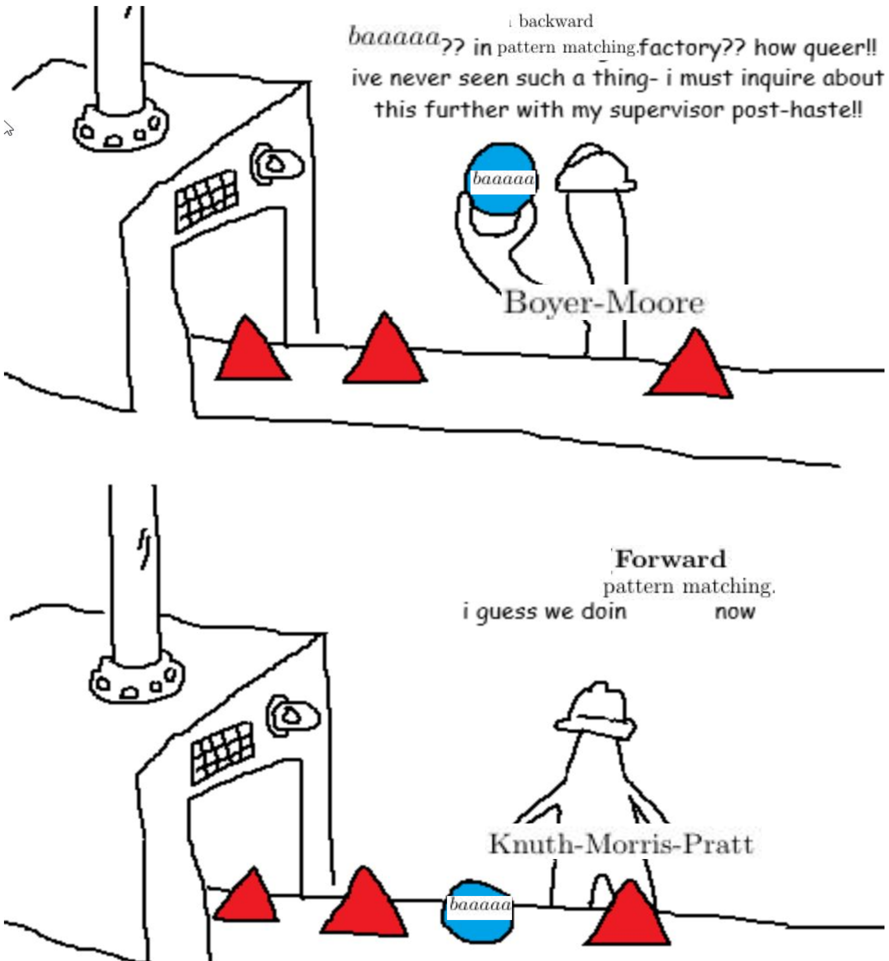
Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P				b	a	a	a	a	a

Total compares:

Same thing will happen 1 more time, and conclude no match

2. Is there any other pattern matching algorihtm that works better in this scenario?



Question 4

(Forward pattern matching)

Another efficient pattern matching algorithm, named the Knuth-Morris-Pratt (KMP) algorithm, is based upon forward pattern matching, in which a failure function (also named as "suffix function") is calculated to determine the most distance we can shift the pattern to avoid redundant comparisons. Specifically, for a pattern P, its corresponding failure function $F_P(j)$, or $F(j)$ for short, is defined as

$$F(j) := \max_k \{k \leq j - 1 : P[0 : k] = P[j - k : j]\}.$$

In other words, $F(j)$ represents the size of the largest prefix of $P[0 : j]$ that is also a suffix of $P[1 : j]$.

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j - 1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i + 1]$.

Answer the following questions:

(1) Apply the KMP algorithm to the pattern matching problem in Question 1. Does it perform much better than Boyer-Moore?

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

Question 4

(Forward pattern matching)

Another efficient pattern matching algorithm, named the Knuth-Morris-Pratt (KMP) algorithm, is based upon forward pattern matching, in which a failure function (also named as "suffix function") is calculated to determine the most distance we can shift the pattern to avoid redundant comparisons. Specifically, for a pattern P, its corresponding failure function $F_P(j)$, or $F(j)$ for short, is defined as

$$F(j) := \max_k \{k \leq j-1 : P[0:k] = P[j-k:j]\}.$$

In other words, $F(j)$ represents the size of the largest prefix of $P[0:j]$ that is also a suffix of $P[1:j]$.

(2) What is the failure function for the pattern $P :=$ "mamagama"?

m a m a g a m a

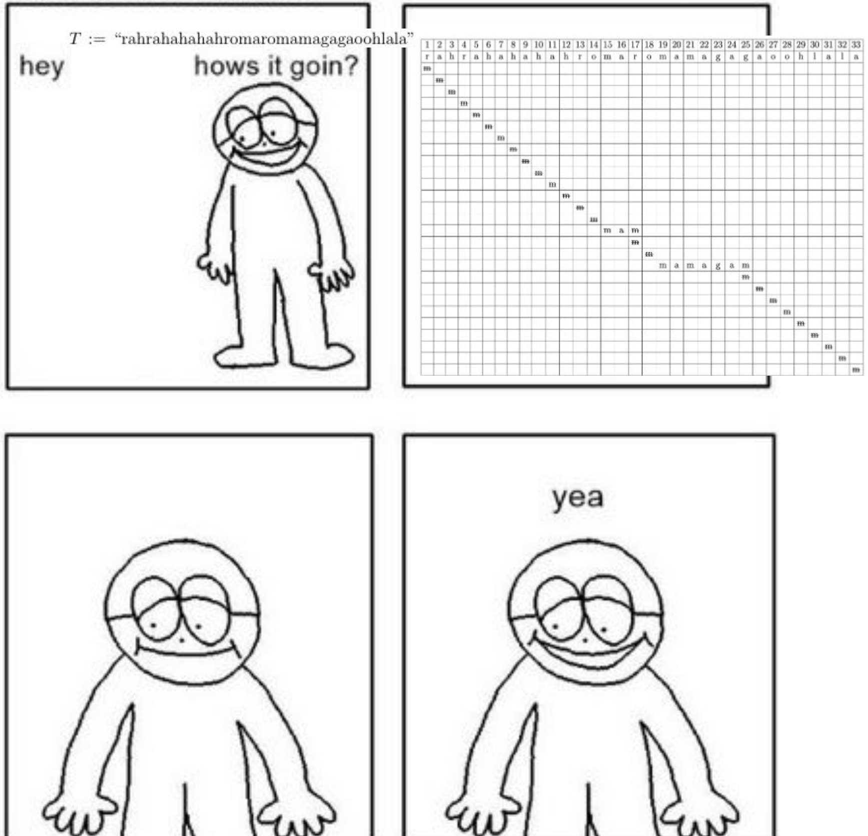
j	1	2	3	4	5	6	7
f(j)							

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

(3) Let $T :=$ “rahrahahahahromaromamagagaoohlala”, run the KMP pattern matching algorithm for the pattern P in (2).

This example is a bit long..



In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	**r**	a	h	m	a	m	a	m	a	m	a	m	a
P	**m**	a	m	a	g	a	m	a					

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	**a**	h	m	a	m	a	m	a	m	a	m	a
P		**m**	a	m	a	g	a	m	a				

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	**h**	m	a	m	a	m	a	m	a	m	a
P			**m**	a	m	a	g	a	m	a			

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = "rahmamamamagama"

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	**m**	a	m	a	m	a	m	a	m	a
P				**m**	a	m	a	g	a	m	a		

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P				m	a	m	a	**g**	a	m	a		

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = "rahmamamamagama"

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P				m	a	m	a	**g**	a	m	a		

Mismatch at P[4]

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P				m	a	m	a	**g**	a	m	a		

Mismatch at P[4], align P[2] with T[7]

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P						m	a	m	a	**g**	a	m	a
				m	a	m	a	**g**	a	m	a		

Mismatch at P[4], align P[2] with T[7] **Why?**

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P				m	a	m	a	**g**	a	m	a		

Mismatch at P[4], align P[2] with T[7] **Why?** **f(3) says these are equal**

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P				m	a	m	a	**g**	a	m	a		

Mismatch at P[4], align P[2] with T[7] **Why?**

Mismatch at P[4] → No mismatch before P[4]

In brief, the KMP algorithm can be described as: When a mismatch occurs at $T[i]$, if you are

- currently at $P[j]$ with some $j > 0$, then shift P to align $P[F(j-1)]$ with $T[i]$.
- currently at $P[0]$, then shift $P[0]$ to align with $T[i+1]$.

Let T = “rahmamamamagama”

j	1	2	3	4	5	6	7
f(j)	0	1	2	0	0	1	2

T	r	a	h	m	a	m	a	**m**	a	m	a	m	a
P						m	a	m	a	**g**	a	m	a
				m	a	m	a	**g**	a	m	a		

Mismatch at P[4], align P[2] with T[7] **Why?**

No mismatch before P[4] → I can move pattern two spaces