242. Valid Anagram

Easy

https://leetcode.com/problems/valid-anagram/

Agenda

- Concept Terminology
- Problem Statement
- Approach
  1. In Built methods
  2. Array
  3. Map
  4. Map
- Time & Space Complexity

Concept Terminology

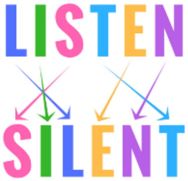
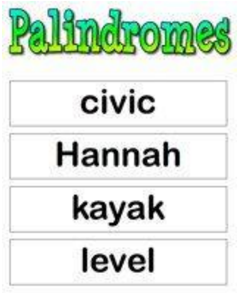
String is a sequence of characters and symbols.

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

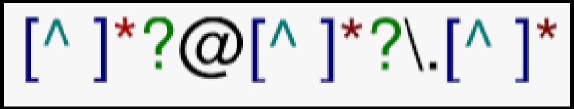
ASCII printable characters					
32	**space**	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	**nbsp**

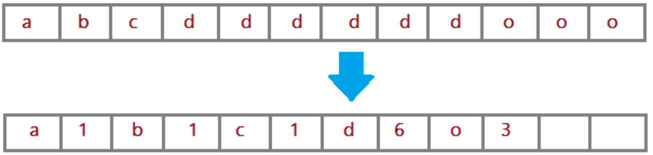
Common String – Words

Anagram	Isogram	Pangram	Palindrome
An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.	An Isogram is a word in which no letter occurs more than once.	A pangram is a unique sentence in which every letter of the alphabet is used at least once.	A word, phrase, or sequence that reads the same backwards as forwards
	**Machine** is Isogram. **Apple** is not isogram.		

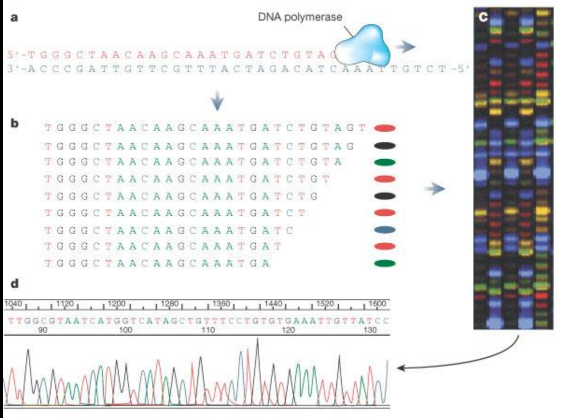
String Algorithms – Real life Application

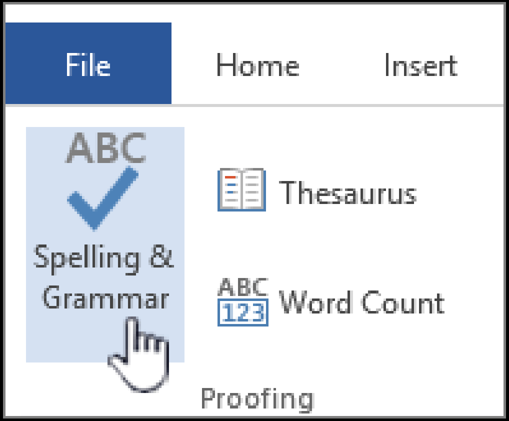


```
[^ ]*?@[^ ]*?\.[^ ]*
```



a	b	c	d	d	d	d	d	d	o	o	o
a	1	b	1	c	1	d	6	o	3		

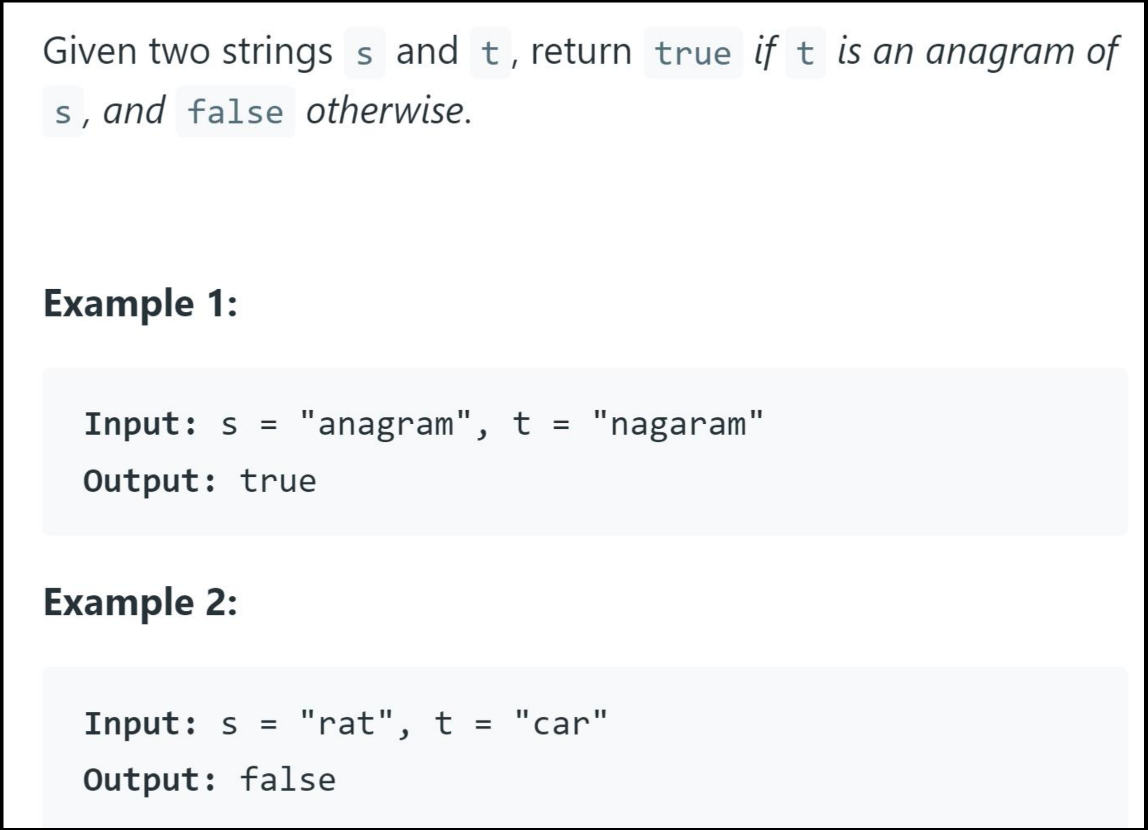




Common String Algorithms

Algorithms	Categories	Description	Pre- process the pattern	Time Complexity
Linear searching	**Brute Force Naïve**	Searching with all alphabets	No	O (m * n)
Heuristic	**Knuth-Morris Pratt**	KMP algorithm matches character from left to right and suits for small variables Prefix – left to right	Yes	*Worst case* O(n x m)
	Boyer-Moore	Boyer – Moore technique matches the character right to left and suits for long patterns. Suffix – right to left	Yes	*Worst case* O(n x m)
Hashing	**Rabin – Karp**	Rabin – Karp algorithm is used for finding any one of a set of pattern strings in a text. Prefix – left to right	Yes	*Worst case* O(n x m)

Problem Statement



Given two strings `s` and `t`, return `true` *if* `t` *is an anagram of* `s`*, and* `false` *otherwise.*

Example 1:

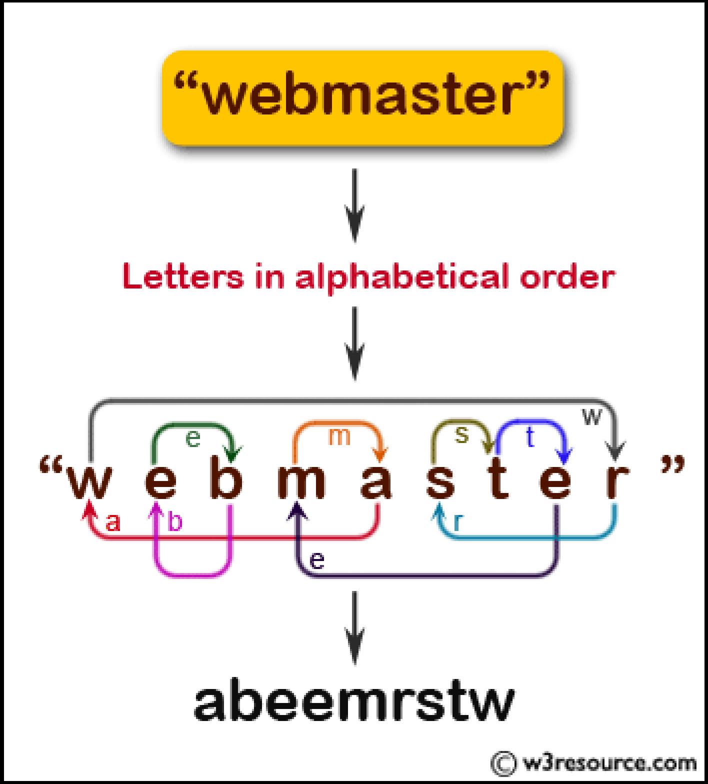
```
Input: s = "anagram", t = "nagaram"
Output: true
```

Example 2:

```
Input: s = "rat", t = "car"
Output: false
```

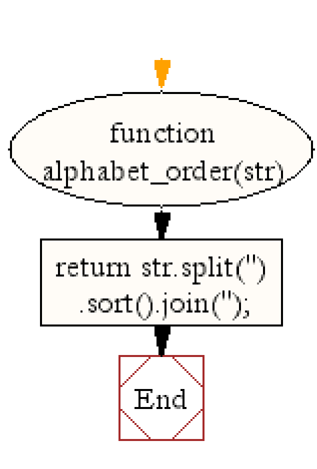
An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Approach 1 : Algorithm Steps

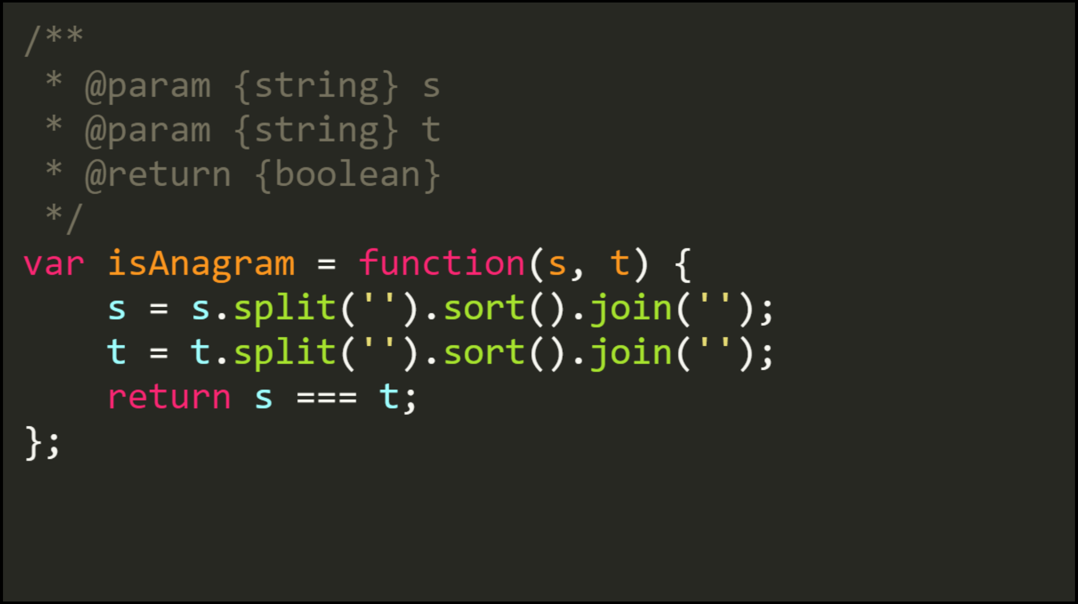


String Object

- split ( )
- sort ( )
- Join ( )


function alphabet_order(str)
return str.split('').sort().join('');
End

Approach 1 : Code



```
/**
 * @param {string} s
 * @param {string} t
 * @return {boolean}
 */
var isAnagram = function(s, t) {
    s = s.split('').sort().join('');
    t = t.split('').sort().join('');
    return s === t;
};
```

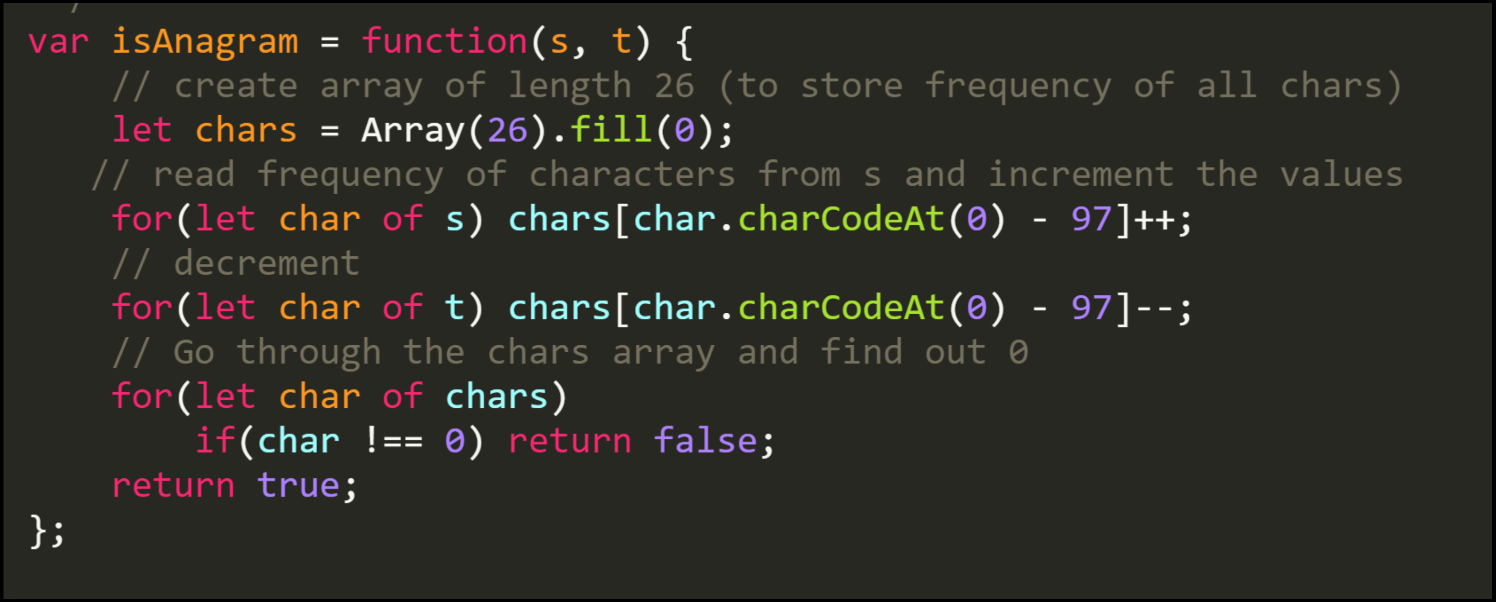
Time Complexity

- **O(n + m)**

Space Complexity

- **O(n + m)**

Approach 2 : Code



```
var isAnagram = function(s, t) {
    // create array of length 26 (to store frequency of all chars)
    let chars = Array(26).fill(0);
   // read frequency of characters from s and increment the values
    for(let char of s) chars[char.charCodeAt(0) - 97]++;
    // decrement
    for(let char of t) chars[char.charCodeAt(0) - 97]--;
    // Go through the chars array and find out 0
    for(let char of chars)
        if(char !== 0) return false;
    return true;
};
```

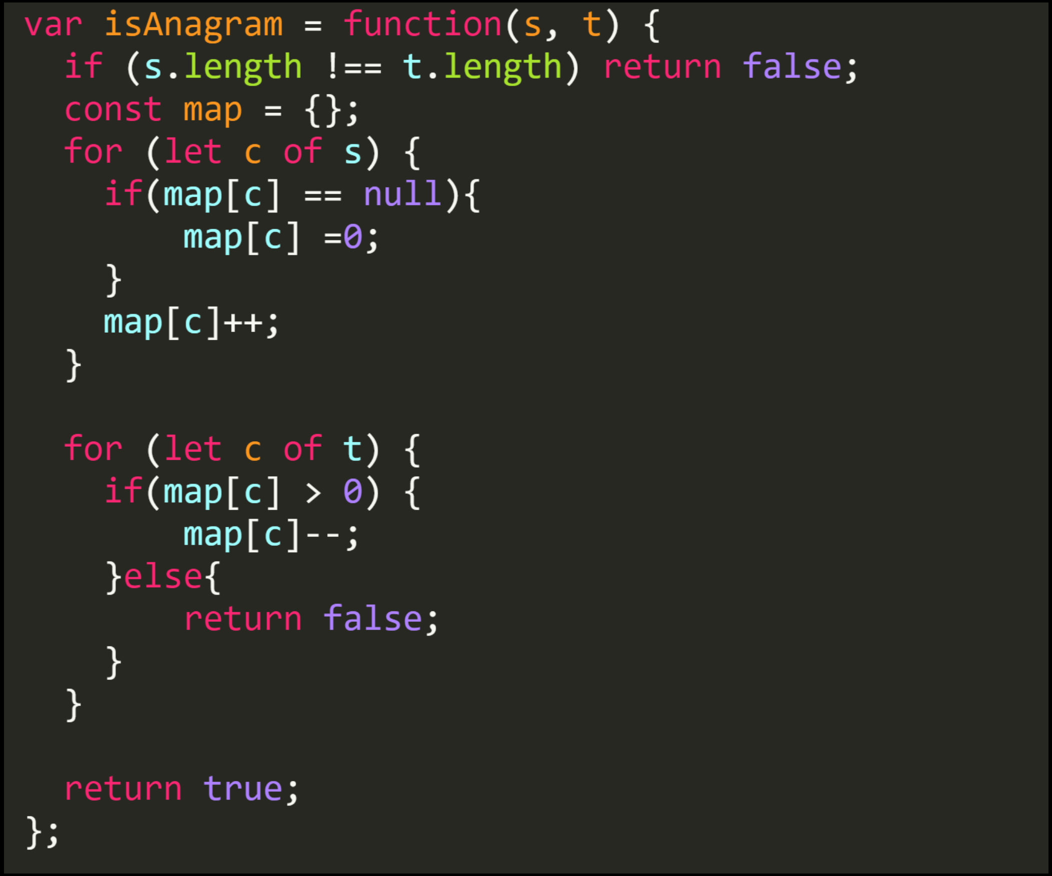
Time Complexity

- O(n + m)

Space Complexity

O(n)

Approach 3 : Code

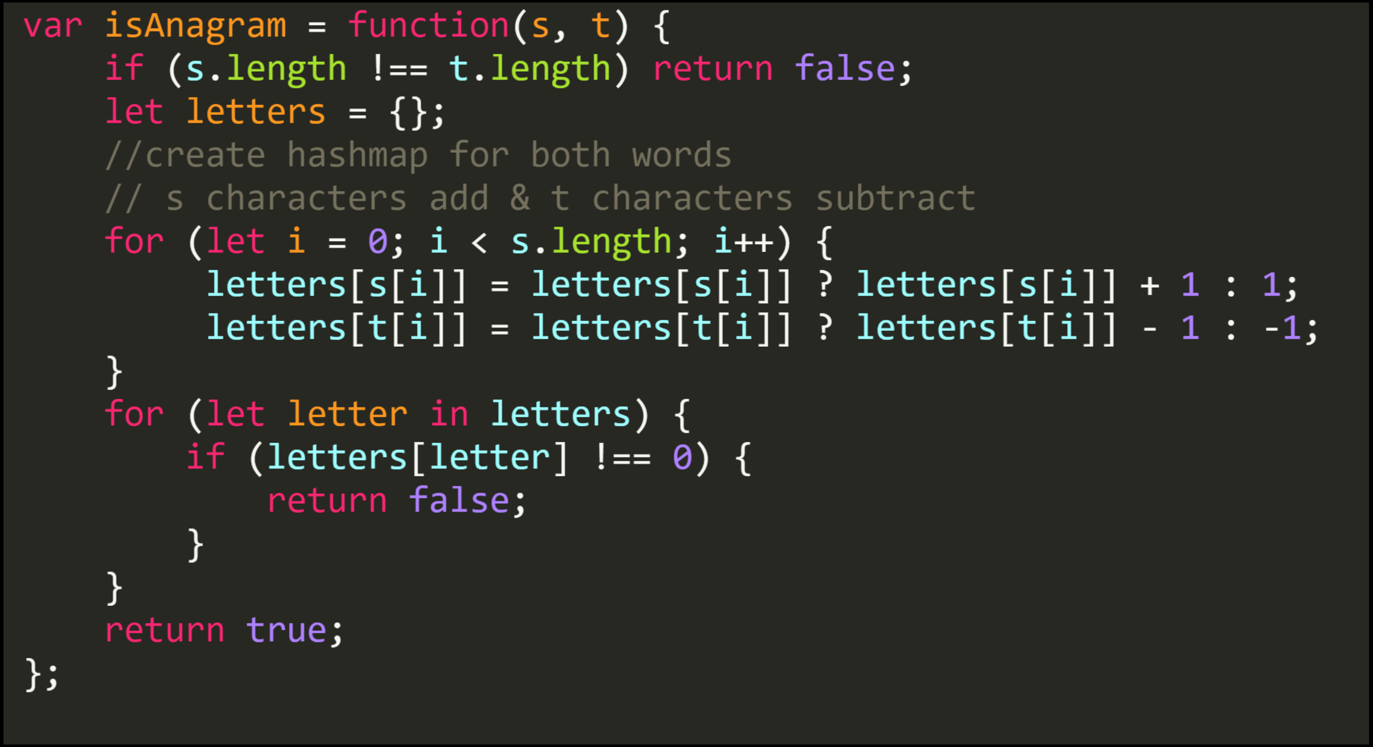


```
var isAnagram = function(s, t) {
  if (s.length !== t.length) return false;
  const map = {};
  for (let c of s) {
    if(map[c] == null){
        map[c] =0;
    }
    map[c]++;
  }

  for (let c of t) {
    if(map[c] > 0) {
        map[c]--;
    }else{
        return false;
    }
  }

  return true;
};
```

Approach 4 : Code



```
var isAnagram = function(s, t) {
    if (s.length !== t.length) return false;
    let letters = {};
    //create hashmap for both words
    // s characters add & t characters subtract
    for (let i = 0; i < s.length; i++) {
         letters[s[i]] = letters[s[i]] ? letters[s[i]] + 1 : 1;
         letters[t[i]] = letters[t[i]] ? letters[t[i]] - 1 : -1;
    }
    for (let letter in letters) {
        if (letters[letter] !== 0) {
            return false;
        }
    }
    return true;
};
```