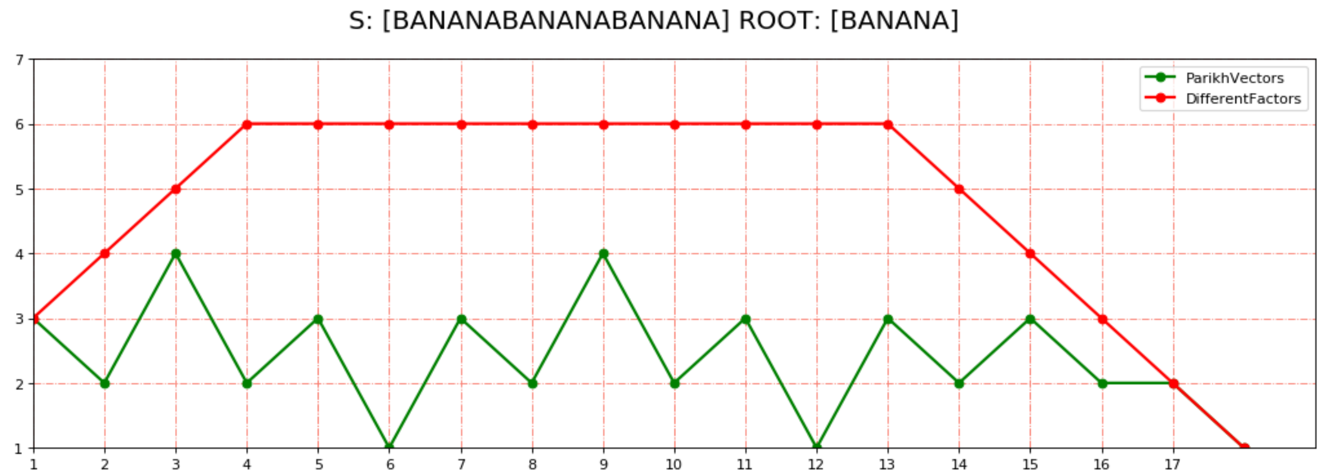
In [1]:

```python
from mylibrary import *
```

1° Topic - Basic Plotting

In [2]:

```python
myplot("BANANA"*3, "ABN")
```



2° Topic - Basic Conjectures

In [3]:

```python
#Inizializzazione variabili
root="abcbcef"
alphabet="abcef"
S=powerword(root, 4)
```

In [4]:

```
%%latex
Given a string S, s.t. S can be written as $U^k$ with $U$ primitive $$ \\ $$
(1) Show that the number of Parikh Vectors($ m \cdot k $) is 1 $\forall m \geq$ 1 )

$$ $$

It's trivial to show that if the string $S$ can be written as a concatenation of $U$ (with $U$ prim
itive
and $k$ power of the root ) if we move the window of $\mid U \mid$ size by one position we will st
ill get the same characters in a different order(but same PV), as the different-factors-PVs collap
se into one. $\textbf{(See example 1)}$
$$ $$ Following the same reasoning, if each time we move the sliding window by one position, we wi
ll get, at most, one change (increase or decrease) per PV. $$ $$
If none of the PV at $k+1$  collapse, then we will have  $\mid U \mid$ as number of distinct
Parikh vectors because the substring has been extended in all possible ways $\textbf{(See example
2)}$
```

Given a string S, s.t. S can be written as U^k with U primitive $$ \\ $$ (1) Show that the number of Parikh Vectors($ m \cdot k $) is 1 $\forall m \geq$ 1) $$ $$ It's trivial to show that if the string S can be written as a concatenation of U (with U primitive and k power of the root) if we move the window of $\mid U \mid$ size by one position we will still get the same characters in a different order(but same PV), as the different-factors-PVs collapse into one. $\textbf{(See example 1)}$ $$ $$ Following the same reasoning, if each time we move the sliding window by one position, we will get, at most, one change (increase or decrease) per PV. $$ $$ If none of the PV at $k+1$ collapse, then we will have $\mid U \mid$ as number of distinct Parikh vectors because the substring has been extended in all possible ways $\textbf{(See example 2)}$

In [5]:

```
FIRST(S, alphabet, len(root)).head()
```

```
The string is: [ abcbcefabcbcefabcbcefabcbcef ]
The length is: [ 28 ]
The root is: [ abcbcef ]
The period length is: [ 7 ]
For the values : [ 7 ] ... [ 14 ] ... [ 21 ] and so on we will always get just one type of PV as t
he different factors collapse
Number of distinct PV: [ 1 ]
```

Out[5]:

	S_Rotations	PV	index
0	cbcefab	[1, 2, 2, 1, 1]	[2, 9, 16]
1	bcefabc	[1, 2, 2, 1, 1]	[3, 10, 17]
2	abcbcef	[1, 2, 2, 1, 1]	[0, 7, 14, 21]
3	cefabcb	[1, 2, 2, 1, 1]	[4, 11, 18]
4	efabcbc	[1, 2, 2, 1, 1]	[5, 12, 19]

In [6]:

```
FIRST(S, alphabet, len(root)*2).head()
```

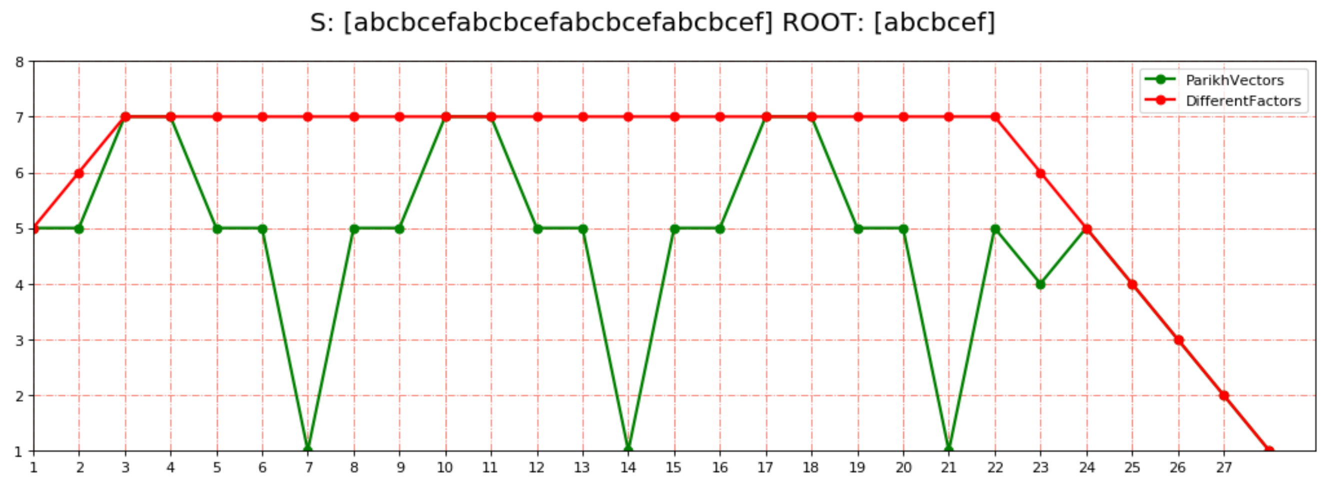
```
The string is: [ abcbcefabcbcefabcbcefabcbcef ]
The length is: [ 28 ]
The root is: [ abcbcef ]
The period length is: [ 7 ]
For the values : [ 7 ] ... [ 14 ] ... [ 21 ] and so on we will always get just one type of PV as t
he different factors collapse
Number of distinct PV: [ 1 ]
```

Out[6]:

	S_Rotations	PV	index
0	bcbcefabcbcefa	[2, 4, 4, 2, 2]	[1, 8]
1	abcbcefabcbcef	[2, 4, 4, 2, 2]	[0, 7, 14]
2	cefabcbcefabcb	[2, 4, 4, 2, 2]	[4, 11]
3	fabcbcefabcbce	[2, 4, 4, 2, 2]	[6, 13]
4	bcefabcbcefabc	[2, 4, 4, 2, 2]	[3, 10]

In [7]:

```
myplot(S, alphabet)
```



3° Topic - Extensibility Table

In [8]:

```
extensibilitytable(S, alphabet)
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	e	-	bc	+	cbc	-	cbce	-	cefab	-	bcbcef	-	cbcefab	-	efabcbce	-	cbcefabcb	-	abcbcefabc	-	bcbcefa
1	c	+	fa	-	cef	-	abcb	-	efabc	-	cefabc	-	bcefabc	-	bcbcefab	-	cefabcbce	-	bcbcefabcb	-	cefabcb
2	a	-	cb	-	abc	-	bcbc	-	bcbce	-	efabcb	-	abcbcef	-	cbcefabc	-	fabcbcefa	-	bcefabcbce	-	cbcefab
3	b	-	ab	-	efa	-	efab	-	abcbc	-	bcefab	-	cefabcb	-	abcbcefa	-	bcbcefabc	-	cefabcbcef	-	fabcbce
4	f	-	ce	-	fab	-	fabc	-	bcefa	-	cbcefa	-	efabcbc	-	cefabcbc	-	efabcbcef	-	cbcefabcbc	-	efabcbc
5	[redacted]		ef	-	bce	-	cefa	-	cbcef	-	abcbce	-	bcbcefa	-	bcefabcb	-	abcbcefab	-	efabcbcefa	-	abcbcef
6	[redacted]				bcb	-	bcef	-	fabcb	-	fabcbc	-	fabcbce	-	fabcbcef	-	bcefabcbc	-	fabcbcefab	-	bcefabc

4° Topic - Strings where #PVs is never equal to the #DFs between small m and big M

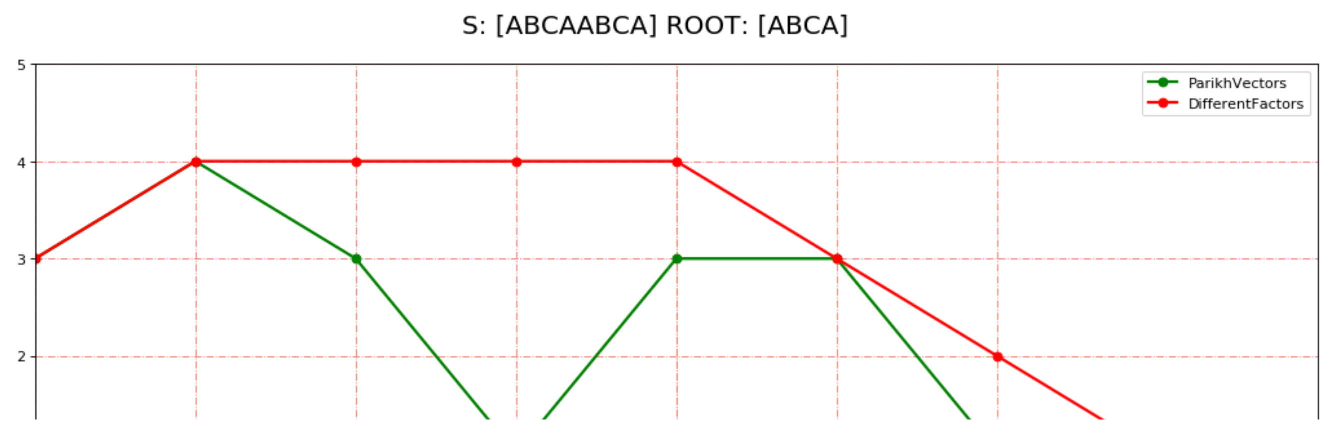
In [9]:

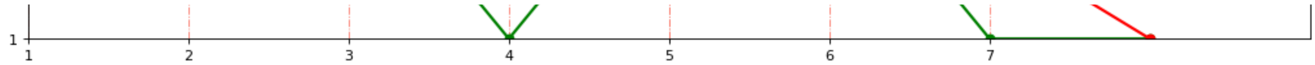
```
#First we generate all the words using all
#the permutations of the characters of the
#alphabet. Secondly, we do the calculation
#to cluster the strings accordingly
alphabet="ABC"
length=4
power=2

touch, nottouch=comparison(alphabet, length, power)

#total is length^|alphabet|
```

```
Words where #PVs is equal to #Dfs at some point

AAAA - AABC - AACB - ABAB - ABBC - ABCA - ABCC - ACAC - ACBA - ACBB - ACCB - BAAC - BABA - BACB -
BACC - BBAC - BBBB - BBCA - BCAA - BCAB - BCBC - BCCA - CAAB - CABB - CABC - CACA - CBAA - CBAC -
CBBA - CBCB - CCAB - CCBA - CCCC

-------------------------------------------------------------

Words where #PVs is NOT equal to #Dfs at any point

CABA - CCBC - ABAA - BABC - ACAB - CCCB - CBCC - BCCB - BCCC - ABAC - AACA - CAAC - AABA - CAAA -
BCBB - ABCB - BACA - CCBB - AAAB - BBAB - CBAB - CCAC - BAAB - AAAC - CBCA - CBBB - ACCC - AABB -
ABBA - CCCA - CACC - BABB - CBBC - BCBA - BCAC - BBAA - ACBC - BBCB - CCAA - BBCC - ACAA - BBBC -
ABBB - CACB - BAAA - ACCA - AACC - BBBA
```

In [17]:

```
myplot("ABCA"*2, alphabet)
```

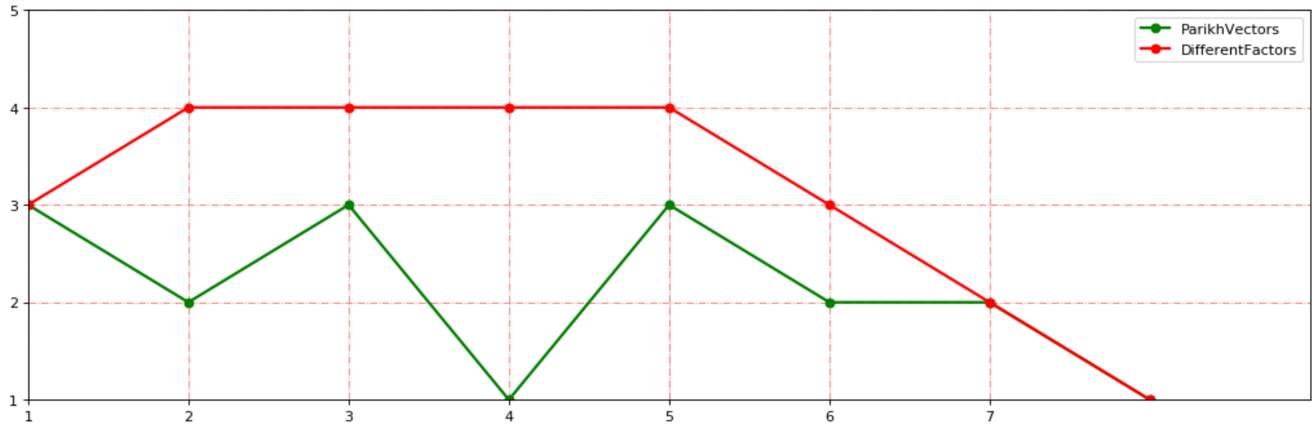




In [18]:

```
myplot("CABA"*2, alphabet)
```

S: [CABACABA] ROOT: [CABA]

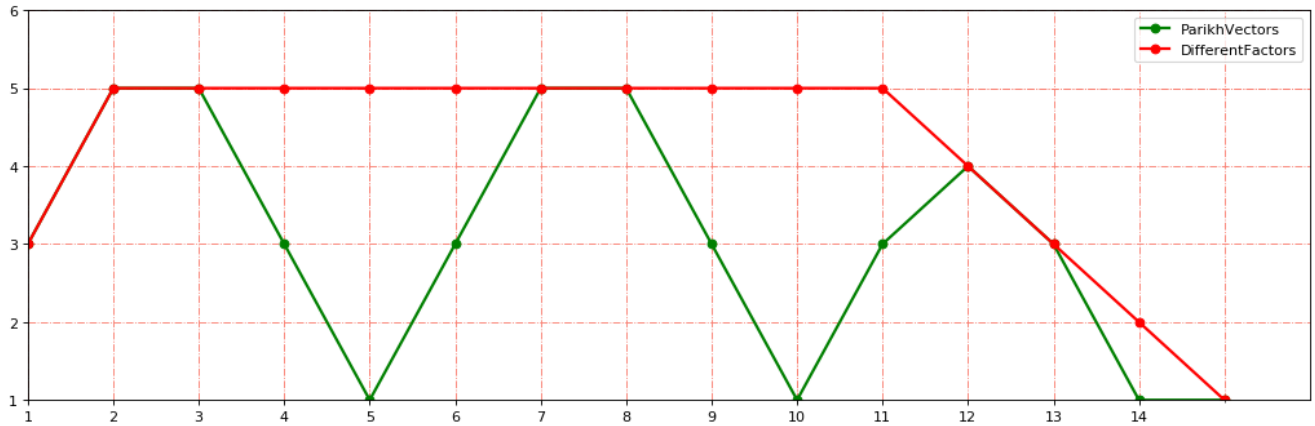


6° Topic - #PVs(mk+i) = |alph(root)| for each i between 1 and x (x is che #repeated char)

In [30]:

```
#REFUTED BY
myplot("ABCCA"*3, "ABC")
```

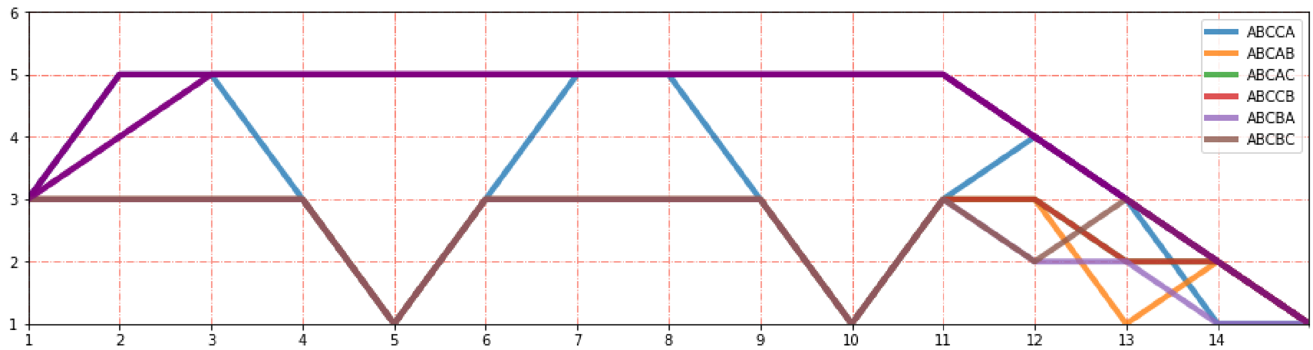
S: [ABCCAABCCAABCCA] ROOT: [ABCCA]



In [29]:

```
L=[ "ABCCA","ABCAB","ABCAC","ABCCB", "ABCBA", "ABCBC"]
myplot_list(L, "ABC", 3)
```

DFs in Purple, PVs other colours



7° Topic - Investigate the end

In [33]:

```
ALPHABET="ABC"
rootsize=4
power=3
bias=1
L=investigate(ALPHABET,rootsize, power, bias )
```

```
****************************************************

The size of the alphabet is [3]
The lengths of the strings is [12]
The lengths of the roots of the strings is [4]
We investigate position [9]

****************************************************

Words with [ 1 ] PVs
AAAA - BBBB - CCCC

------------------------------------------------------

Words with [ 2 ] PVs
AAAB - AAAC - AABA - AABB - AACA - AACC - ABAA - ABAB - ABBA - ABBB - ACAA - ACAC - ACCA - ACCC -
BAAA - BAAB - BABA - BABB - BBAA - BBAB - BBBA - BBBC - BBCB - BBCC - BCBB - BCBC - BCCB - BCCC -
CAAA - CAAC - CACA - CACC - CBBB - CBBC - CBCB - CBCC - CCAA - CCAC - CCBB - CCBC - CCCA - CCCB

------------------------------------------------------

Words with [ 3 ] PVs
AABC - AACB - ABAC - ABBC - ABCA - ABCB - ABCC - ACAB - ACBA - ACBB - ACBC - ACCB - BAAC - BABC -
BACA - BACB - BACC - BBAC - BBCA - BCAA - BCAB - BCAC - BCBA - BCCA - CAAB - CABA - CABB - CABC -
CACB - CBAA - CBAB - CBAC - CBBA - CBCA - CCAB - CCBA

------------------------------------------------------
```

In [ ]: