CZ 2001 Algorithm Lab Project 1

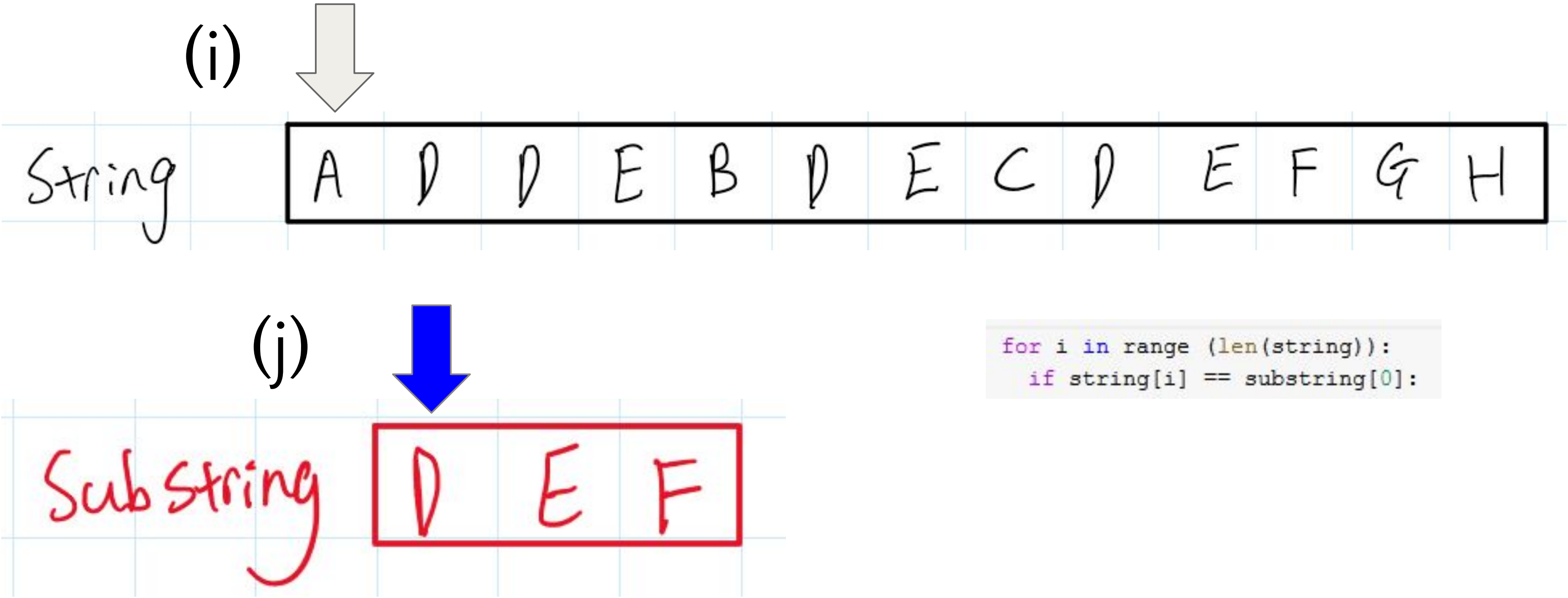
Done by:
Leong Kah Wai Alex,
Low Yu Benedict,
Ong Jing Hong Elliott,
Sunny Pek Yee Chong,
Loe Kit Leong Daniel

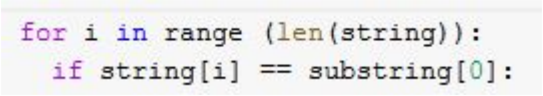
Brute Force Algorithm Analysis

How Brute Force Work

1. Searches from start of string to end
2. Once it matches, goes into inner loop to match substring
3. If mismatch, exits inner loop and continue search from i + 1
4. Algorithm continues until it finds a match

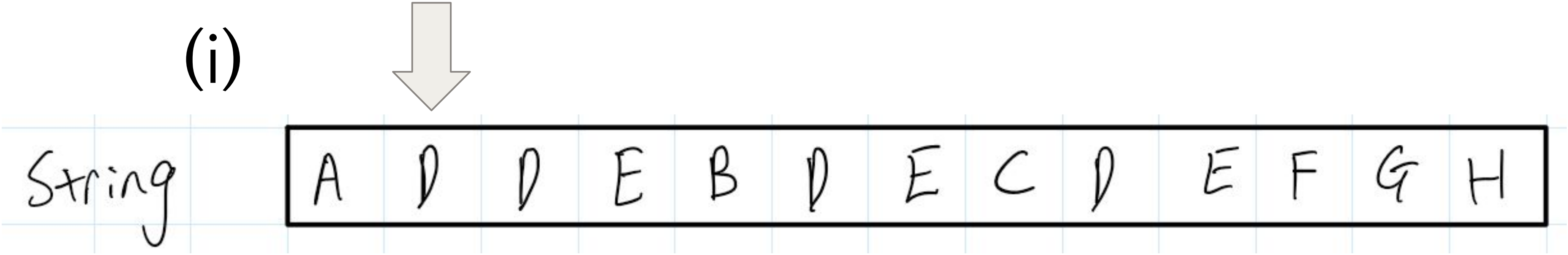
How Brute Force Work (Diagram)

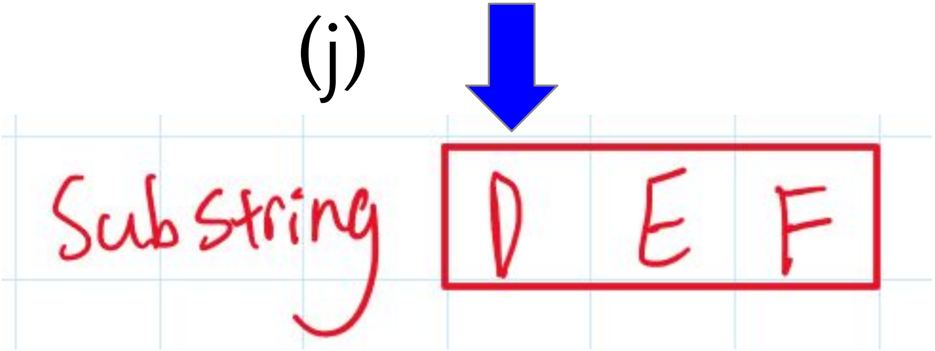


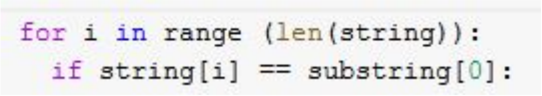


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

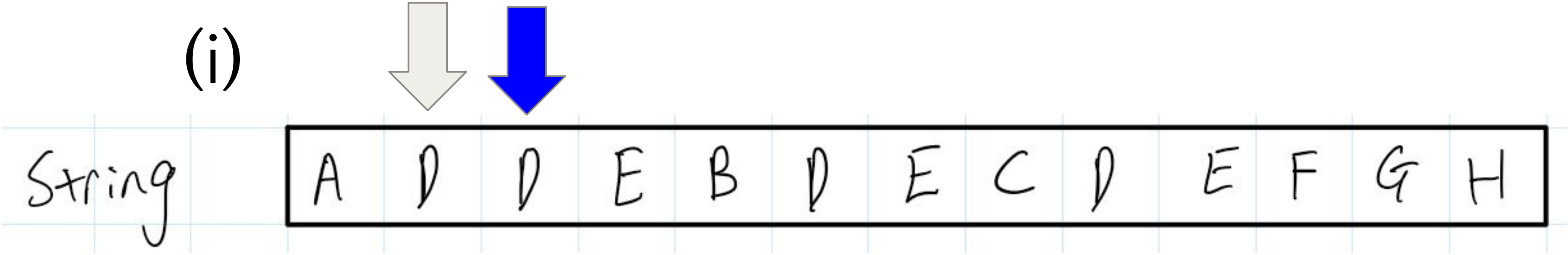


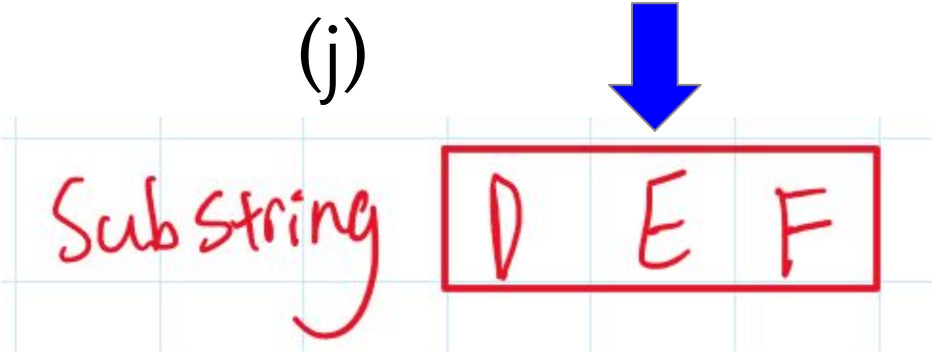


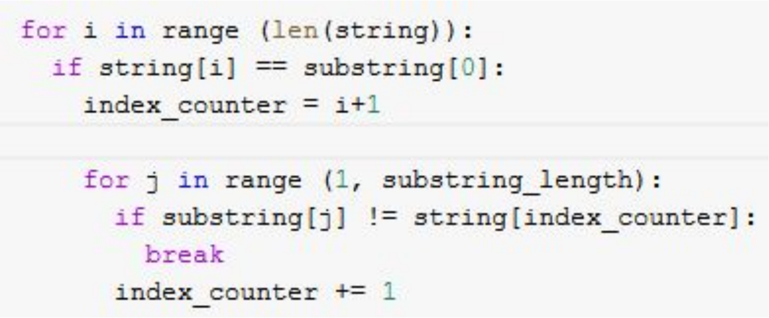


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

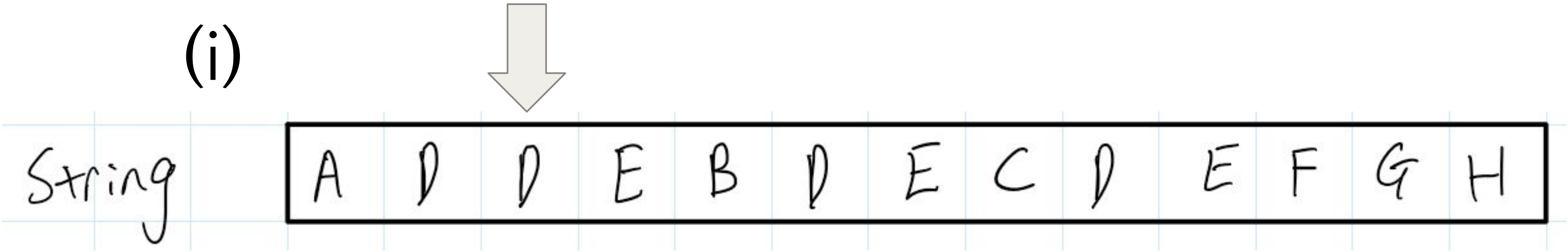






```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1
```

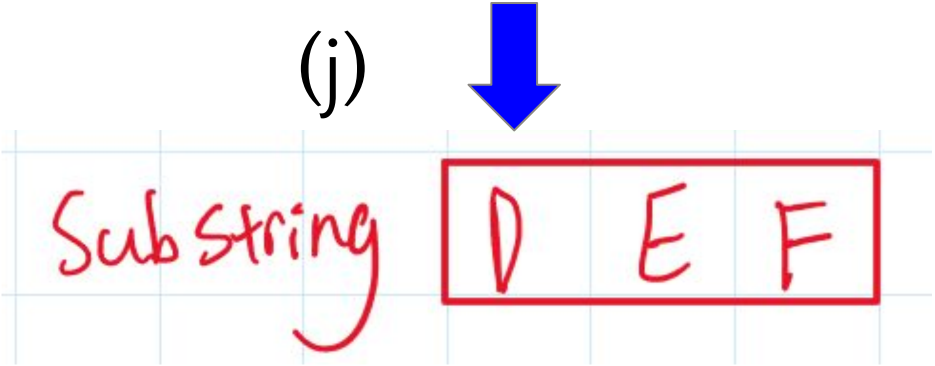
How Brute Force Work (Diagram)



(i)

String

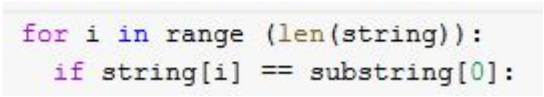
A	D	D	E	B	D	E	C	D	E	F	G	H



(j)

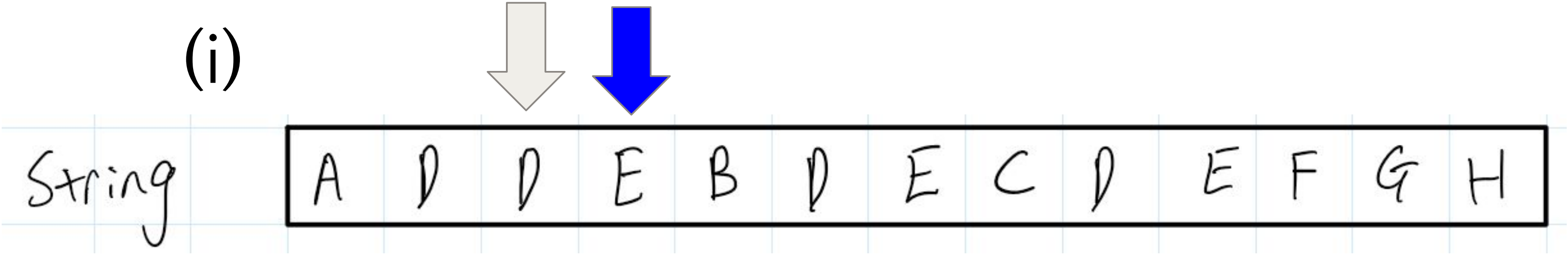
Sub String

D	E	F



```
for i in range (len(string)):
  if string[i] == substring[0]:
```

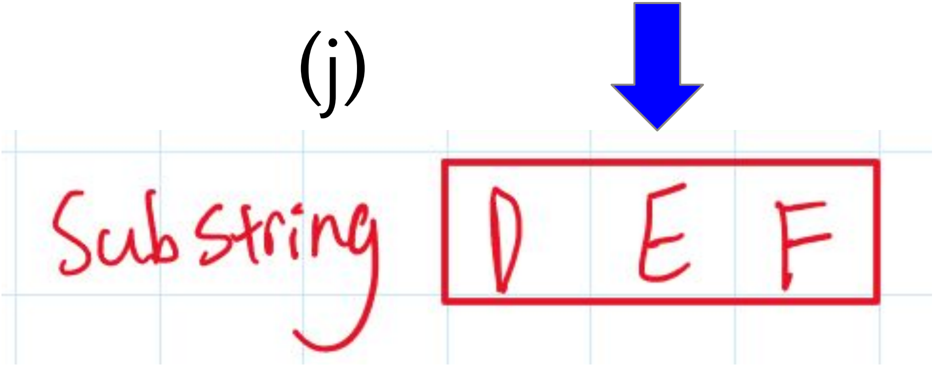
How Brute Force Work (Diagram)



(i)

String

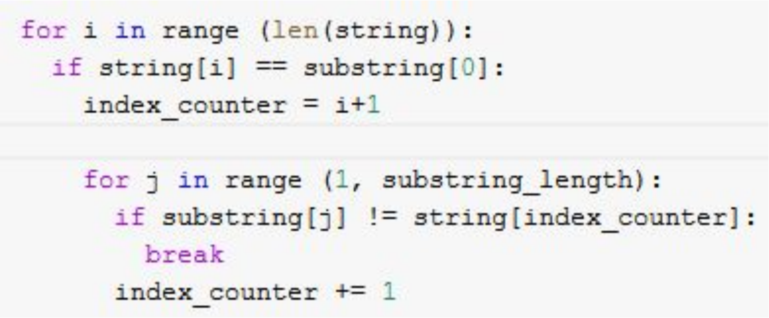
A	D	D	E	B	D	E	C	D	E	F	G	H



(j)

Sub String

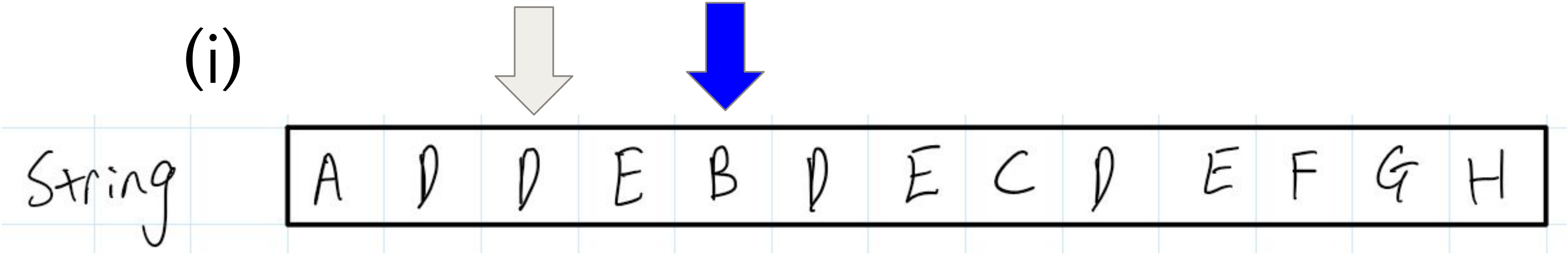
D	E	F



```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1
```

How Brute Force Work (Diagram)

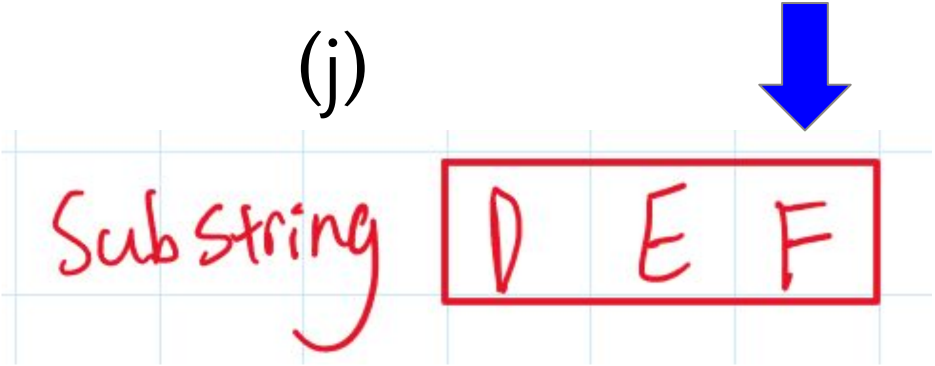
(i)



String

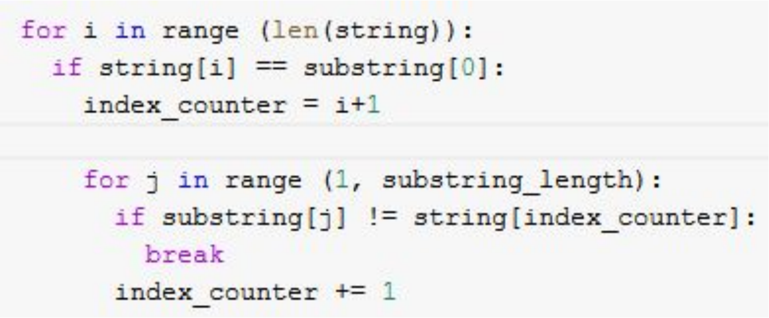
A	D	D	E	B	D	E	C	D	E	F	G	H

(j)



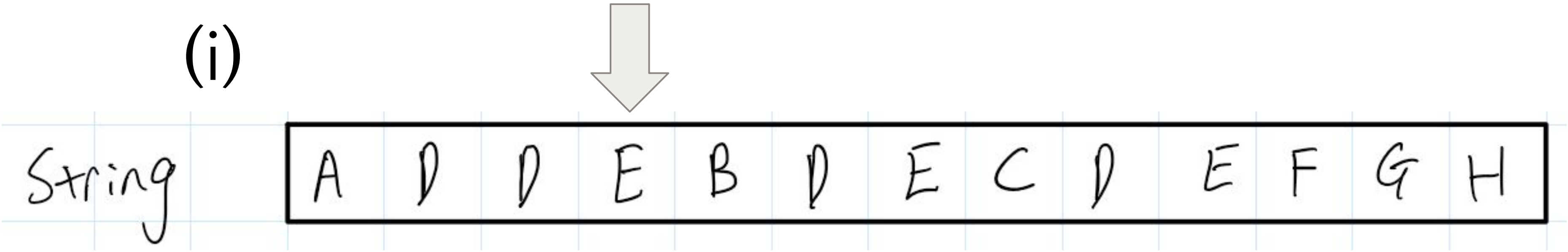
Sub String

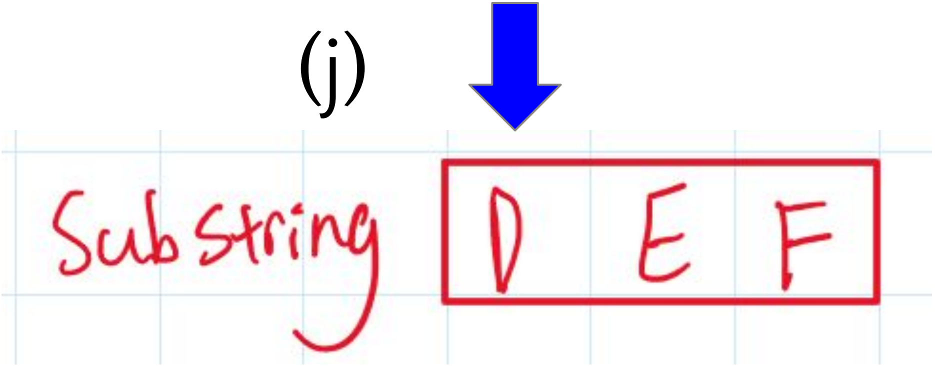
D	E	F

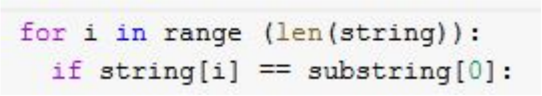


```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1
```

How Brute Force Work (Diagram)

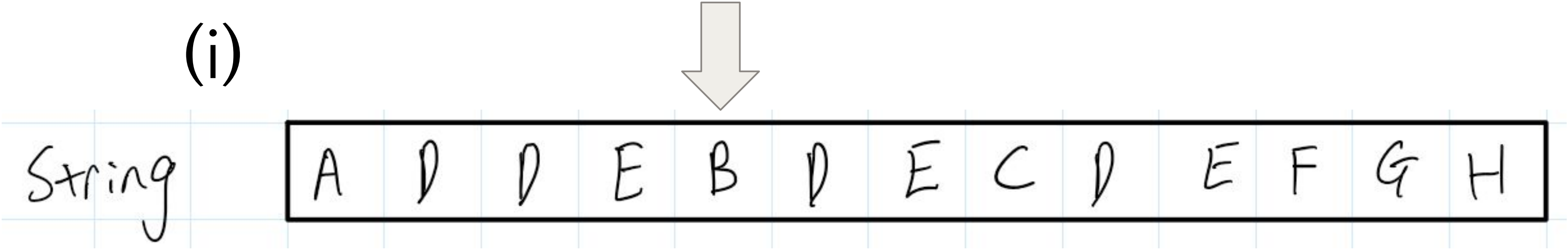


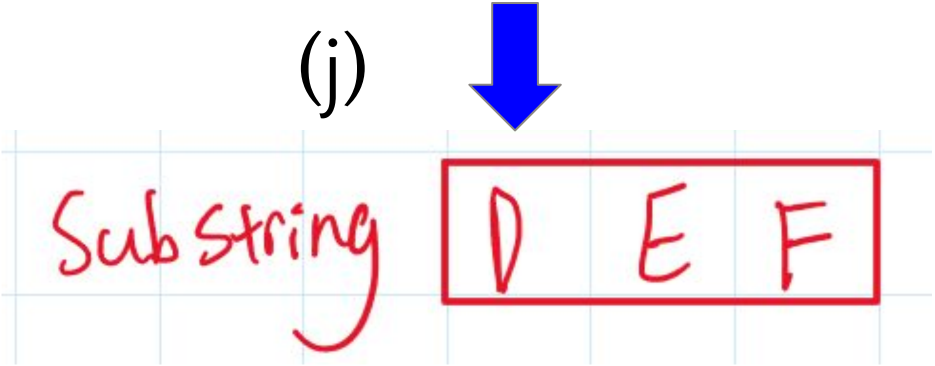


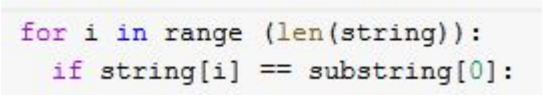


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

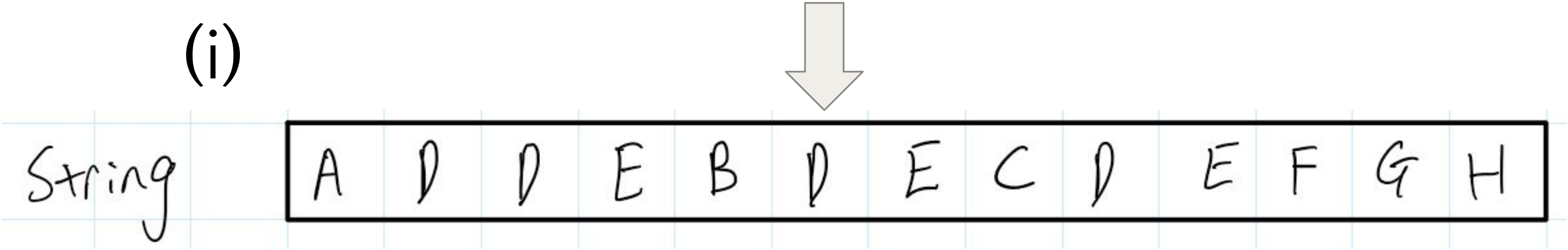


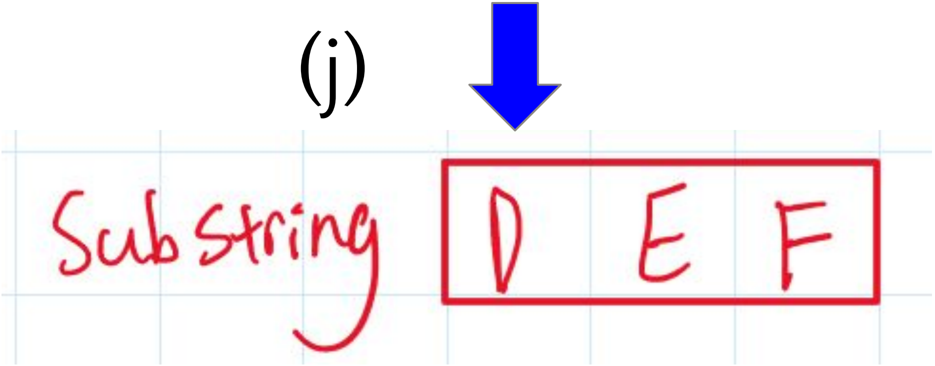


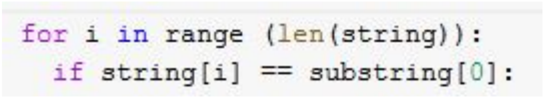


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

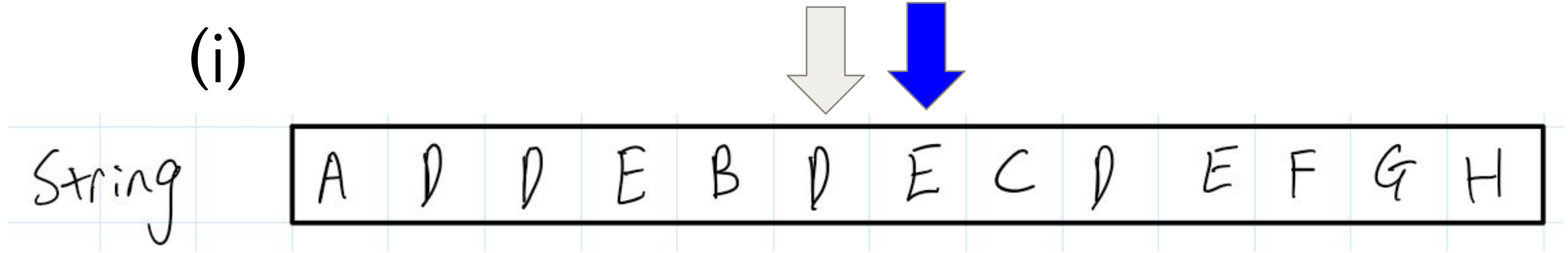


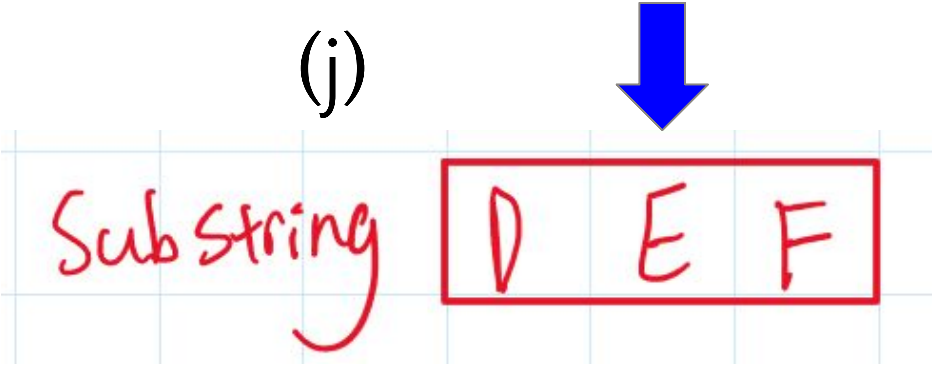


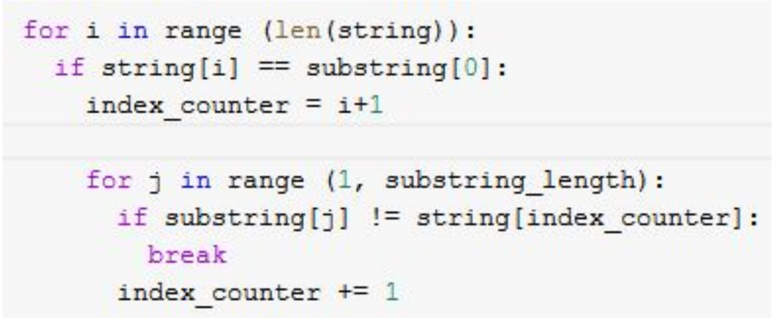


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

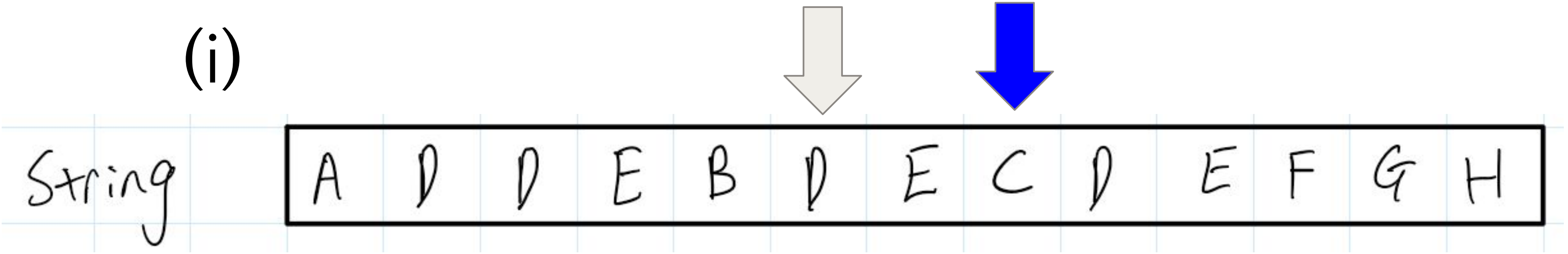






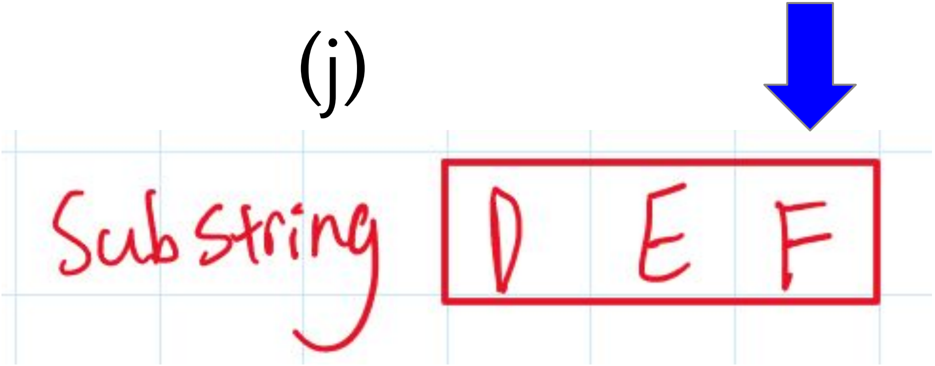
```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1
```

How Brute Force Work (Diagram)



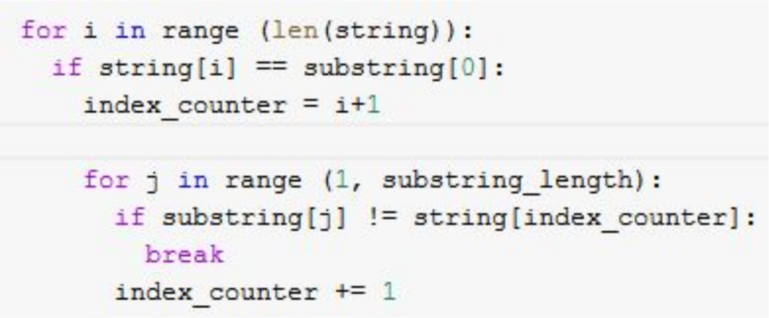
(i)

String	A	D	D	E	B	D	E	C	D	E	F	G	H



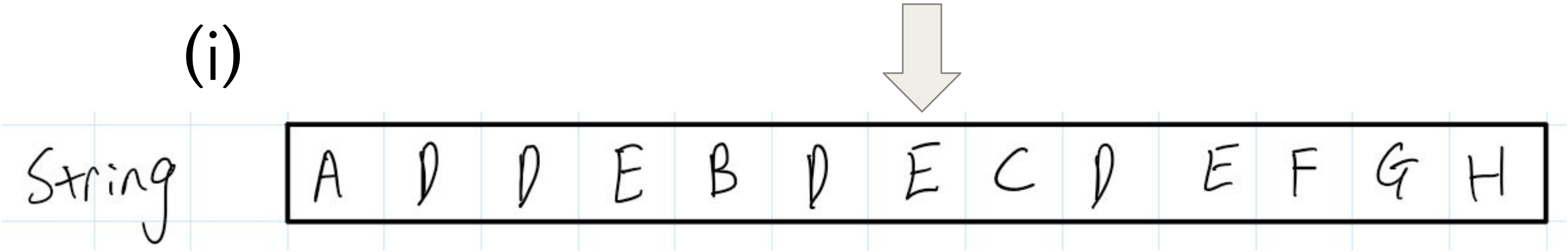
(j)

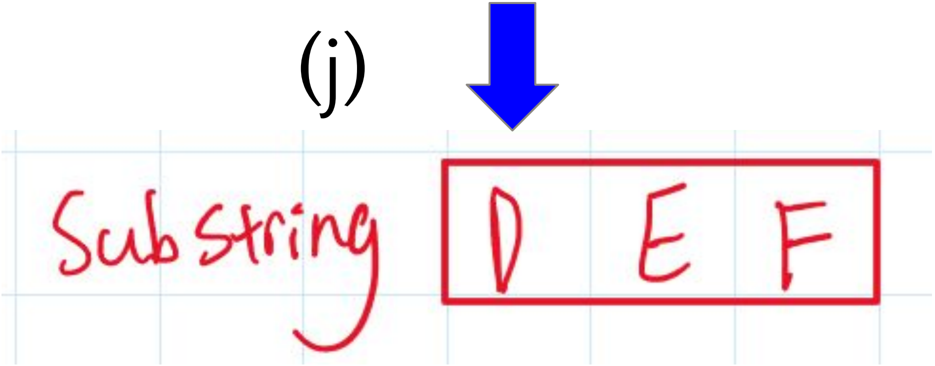
Sub String	D	E	F

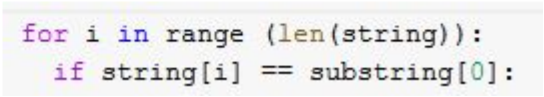


```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1
```

How Brute Force Work (Diagram)

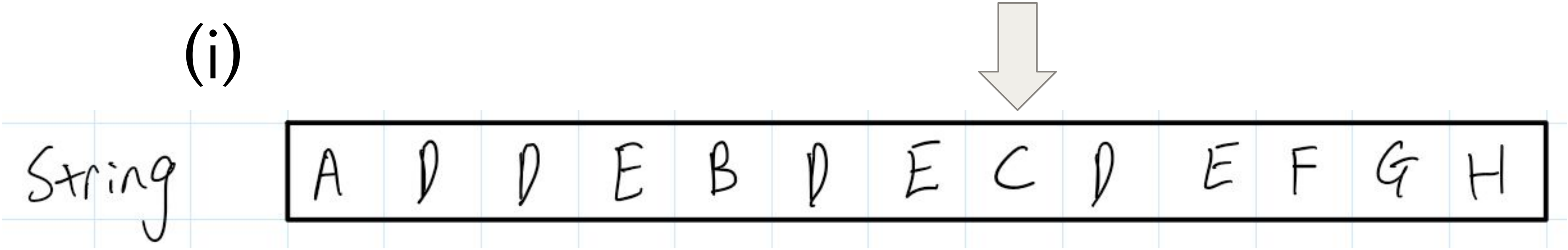


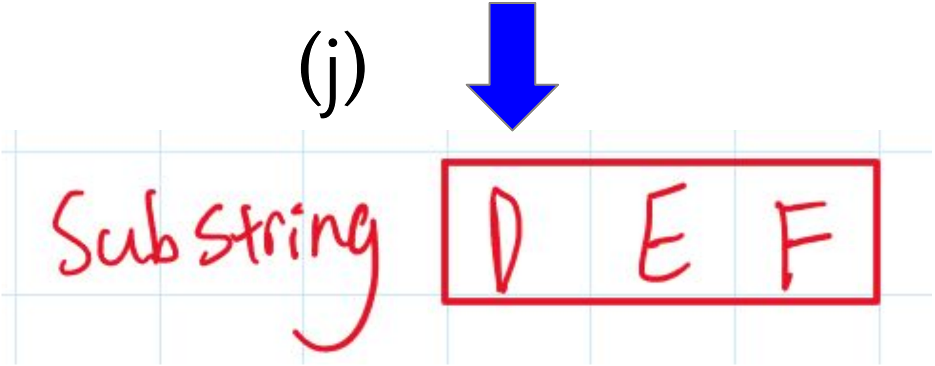


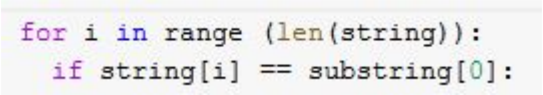


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

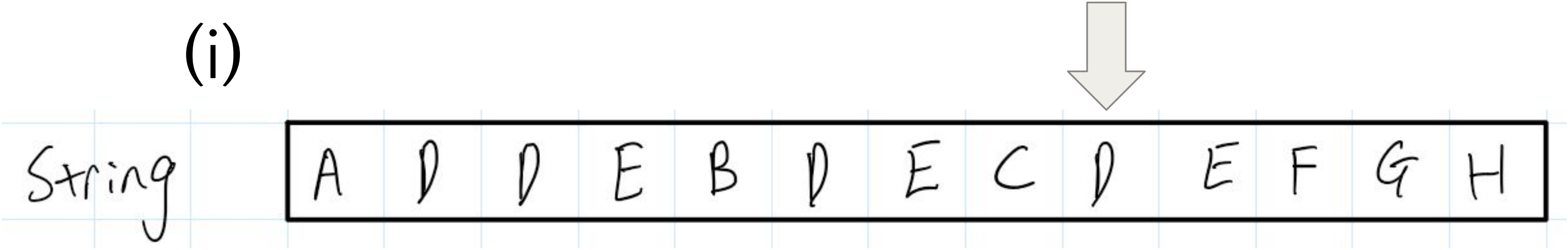




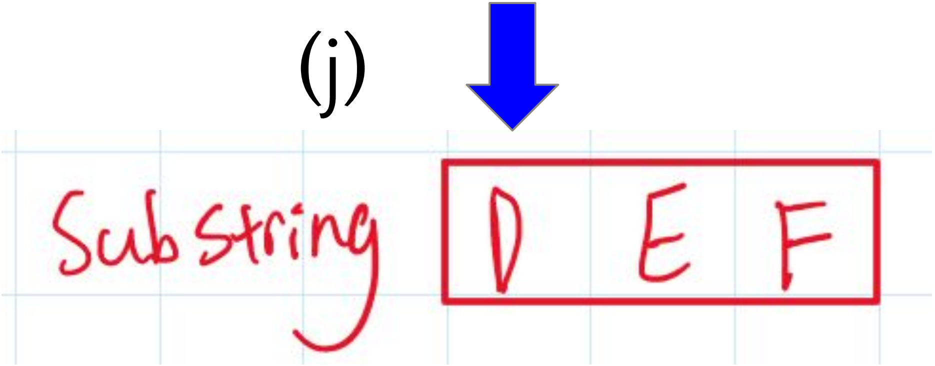


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

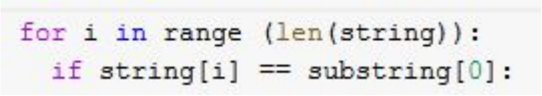
How Brute Force Work (Diagram)



String	A	D	D	E	B	D	E	C	D	E	F	G	H

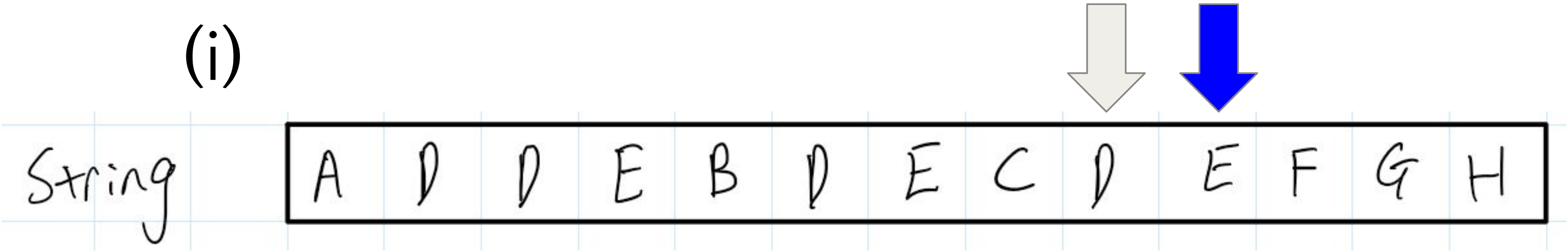


Sub String	D	E	F



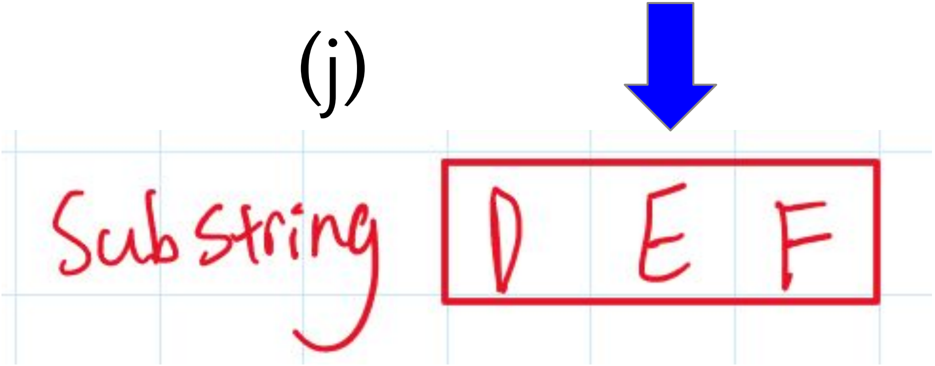
```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)



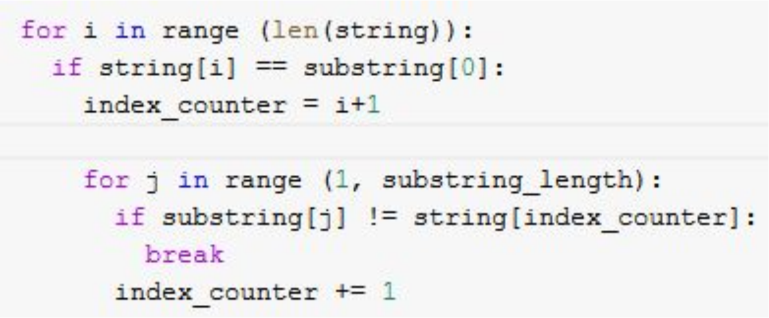
(i)

String	A	D	D	E	B	D	E	C	D	E	F	G	H



(j)

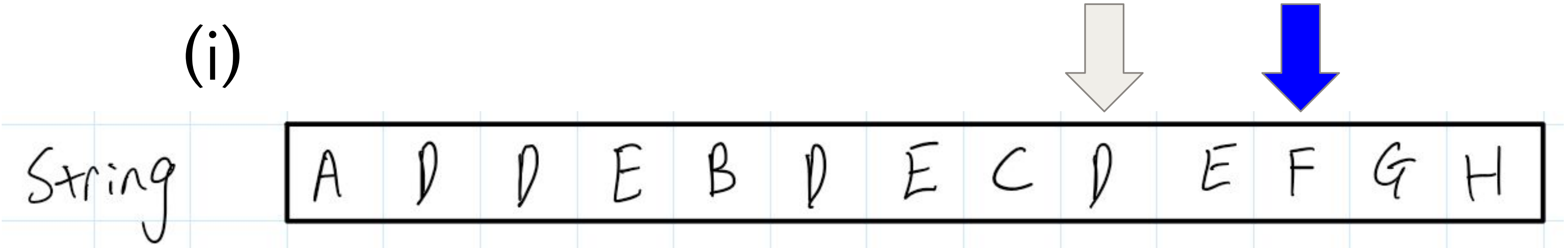
Sub String	D	E	F

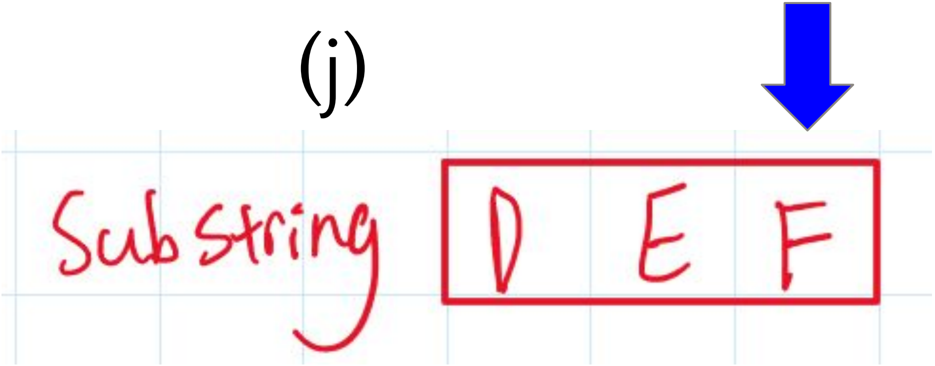


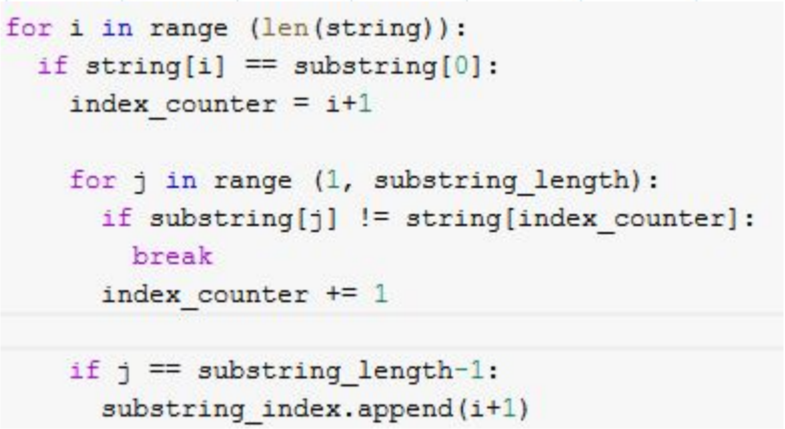
```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1
```

How Brute Force Work (Diagram)

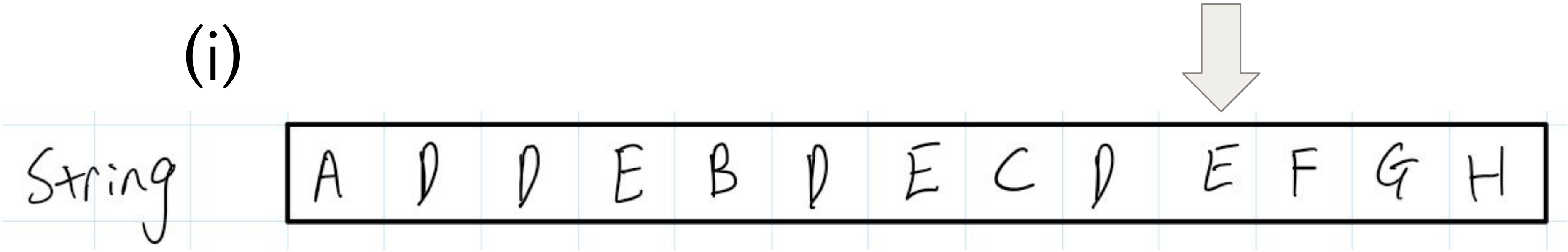
Match Found,
return index (i)

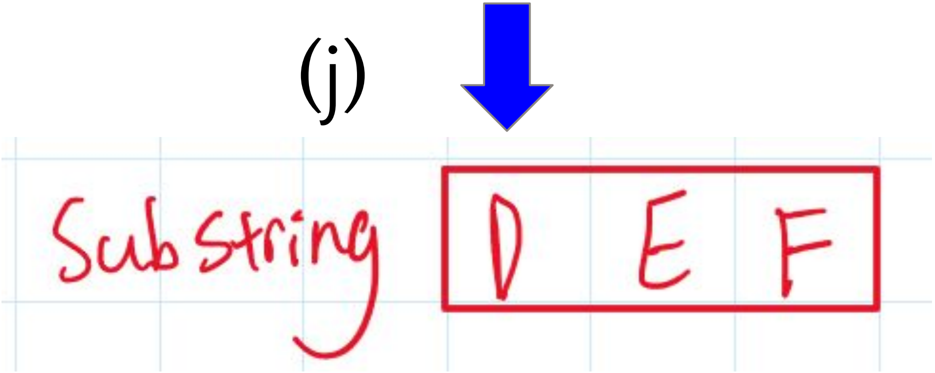


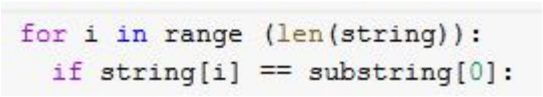



```
for i in range (len(string)):
  if string[i] == substring[0]:
    index_counter = i+1

    for j in range (1, substring_length):
      if substring[j] != string[index_counter]:
        break
      index_counter += 1

    if j == substring_length-1:
      substring_index.append(i+1)
```

How Brute Force Work (Diagram)

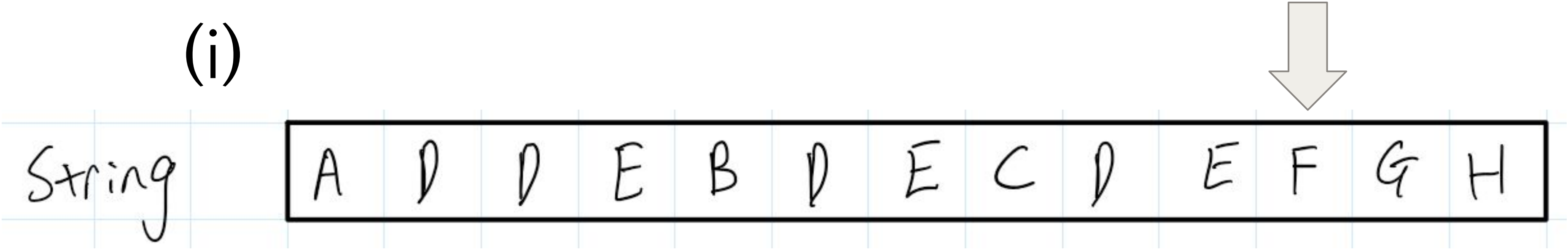


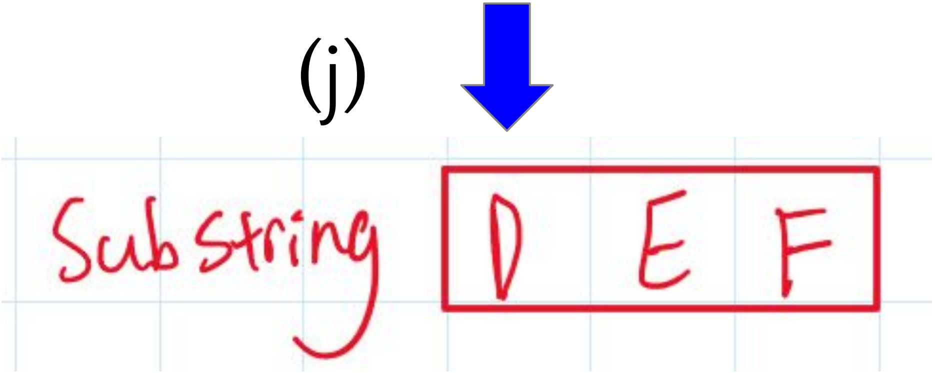


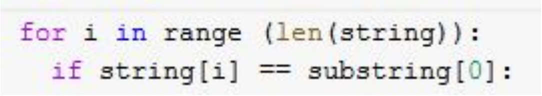


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

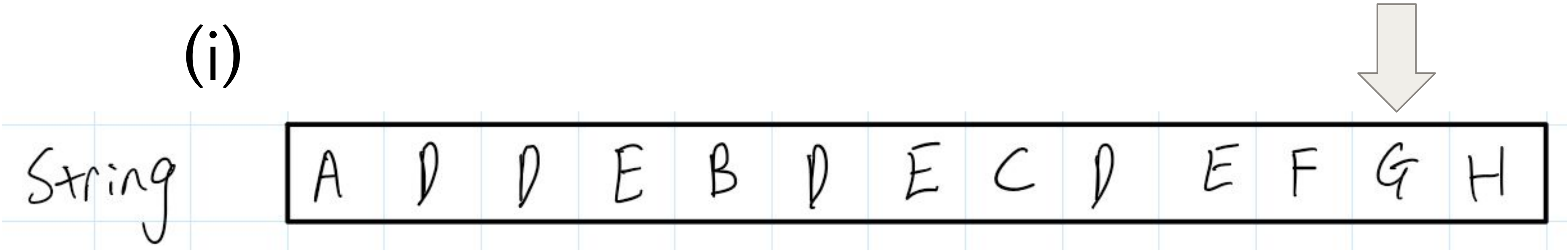


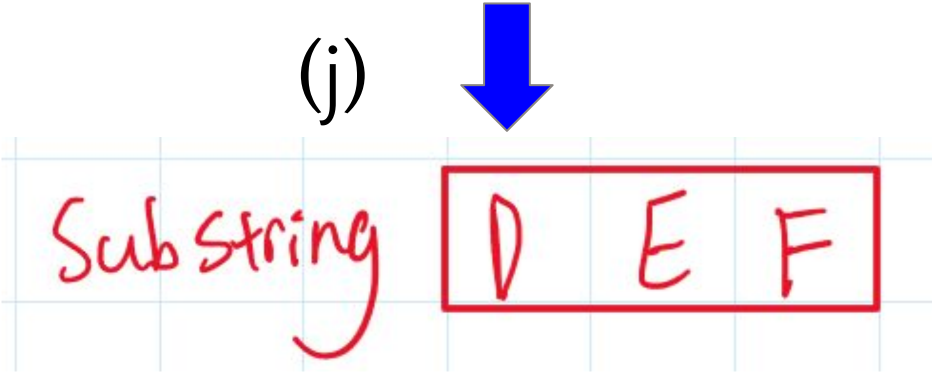


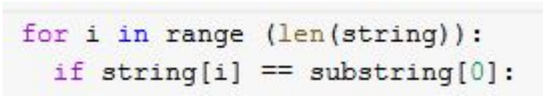


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

How Brute Force Work (Diagram)

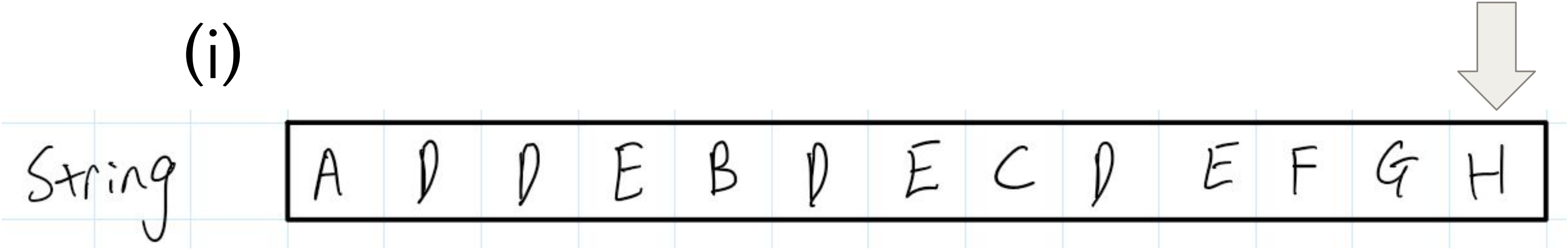




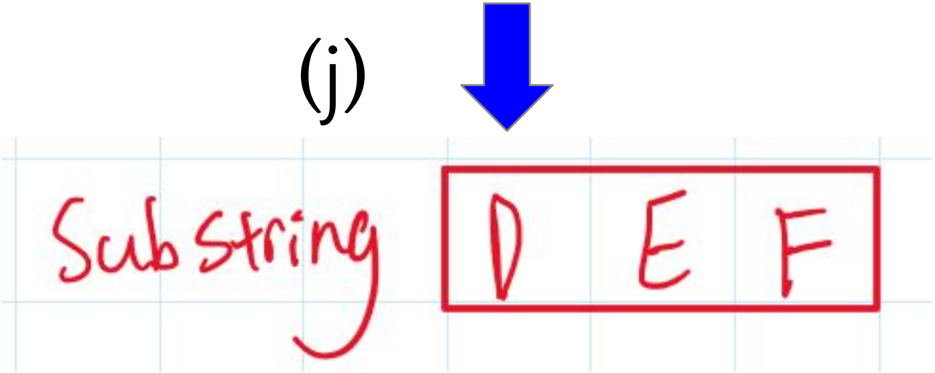


```
for i in range (len(string)):
  if string[i] == substring[0]:
```

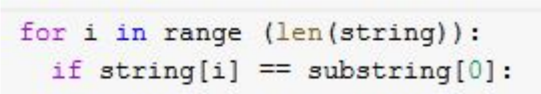
How Brute Force Work (Diagram)



String	A	D	D	E	B	D	E	C	D	E	F	G	H



Sub String	D	E	F



```
for i in range (len(string)):
  if string[i] == substring[0]:
```

Brute Force Time Complexity

If String = n elements

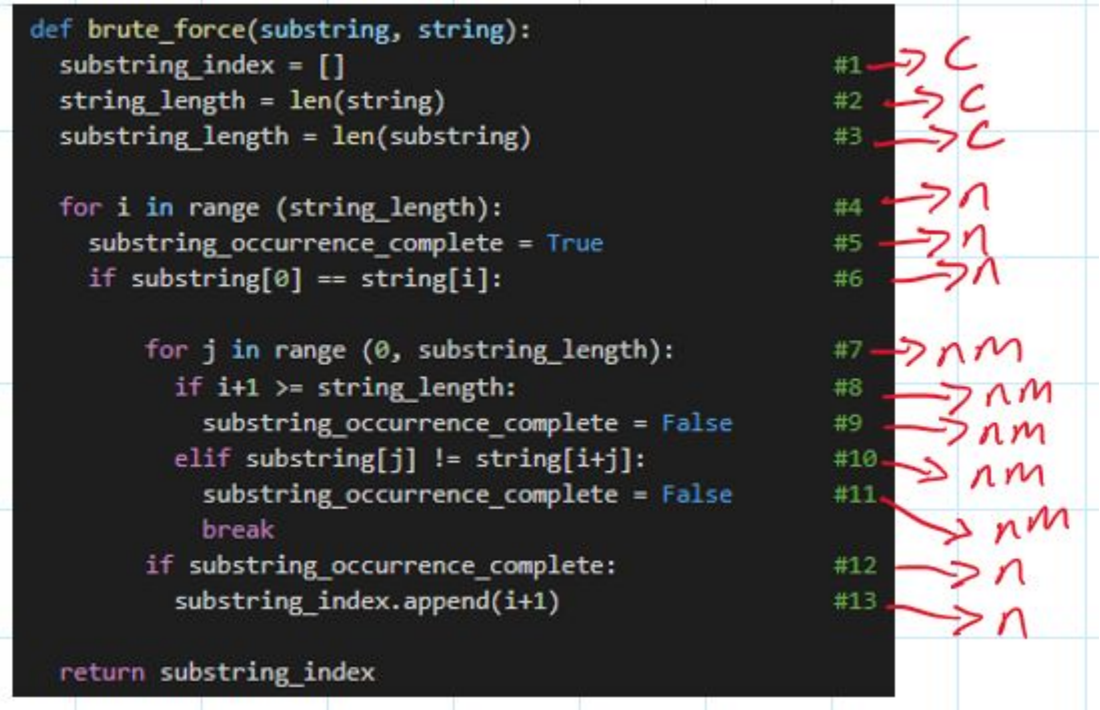
Substring = m elements

It will compare all elements in String with Substring hence

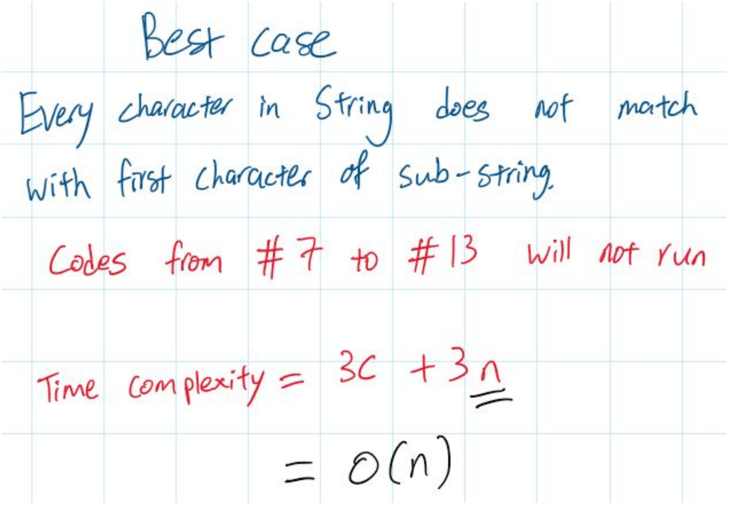
Best Case: O(n)

Worst Case:O(n*m)

Brute Force Time Complexity (Best Case)



```
def brute_force(substring, string):
  substring_index = []                                    #1 → c
  string_length = len(string)                             #2 → c
  substring_length = len(substring)                       #3 → c

  for i in range (string_length):                         #4 → n
    substring_occurrence_complete = True                  #5 → n
    if substring[0] == string[i]:                         #6 → n

        for j in range (0, substring_length):             #7 → nm
          if i+1 >= string_length:                        #8 → nm
            substring_occurrence_complete = False         #9 → nm
          elif substring[j] != string[i+j]:               #10 → nm
            substring_occurrence_complete = False         #11 → nm
            break
        if substring_occurrence_complete:                 #12 → n
          substring_index.append(i+1)                     #13 → n

  return substring_index
```



Best case

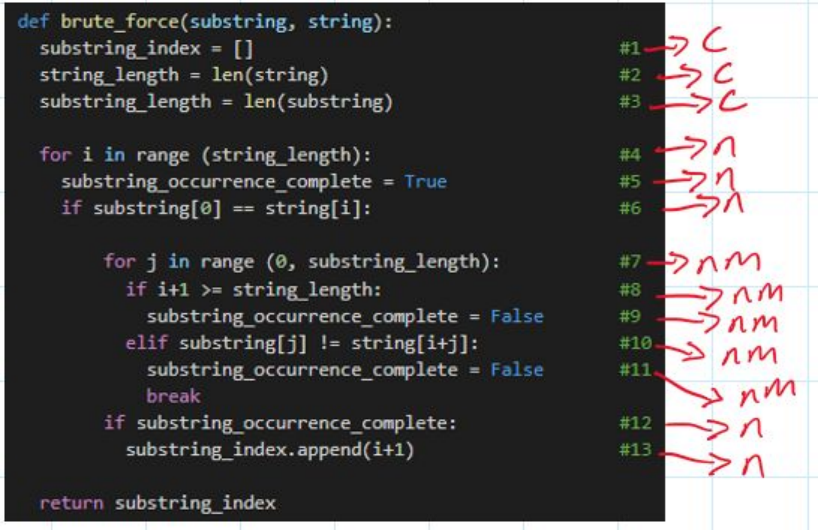
Every character in String does not match with first character of sub-string

Codes from #7 to #13 will not run

Time complexity = $3c + 3\underline{\underline{n}}$

$= O(n)$

Brute Force Time Complexity (Worst Case)



```
def brute_force(substring, string):
  substring_index = []                                  #1
  string_length = len(string)                           #2
  substring_length = len(substring)                     #3

  for i in range (string_length):                       #4
    substring_occurrence_complete = True                #5
    if substring[0] == string[i]:                       #6

        for j in range (0, substring_length):           #7
          if i+1 >= string_length:                      #8
            substring_occurrence_complete = False       #9
          elif substring[j] != string[i+j]:             #10
            substring_occurrence_complete = False       #11
            break
        if substring_occurrence_complete:               #12
          substring_index.append(i+1)                   #13

  return substring_index
```

Worst Case

$$\text{Time Complexity} = 3c + n + n + n + (nm) + (nm) + (nm) + (nm) + n + n$$

$$= 3c + 5n + \underline{\underline{5nm}}$$

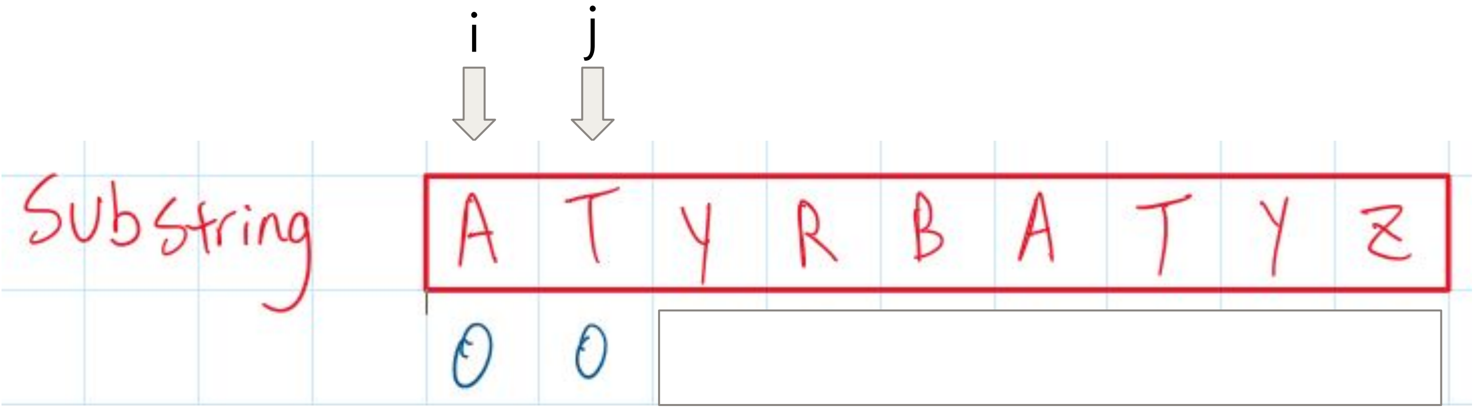
$$= O(nm)$$

Knuth–Morris–Pratt (KMP) Algorithm

How KMP Works

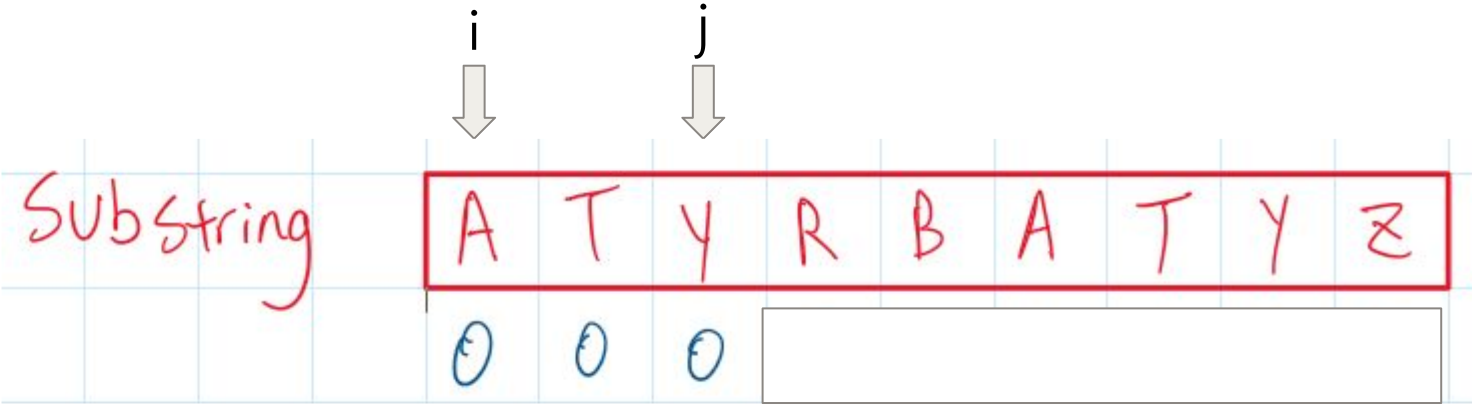
1. Uses the idea of prefix and suffix to skip unnecessary checks (LPS Table)
2. Still searches from start of string to end
3. When it encounters a mismatch, it will search the matched substring for a common prefix and suffix
4. Uses the prefix and suffix to skip checks

KMP Largest Prefix Suffix (LPS) Table



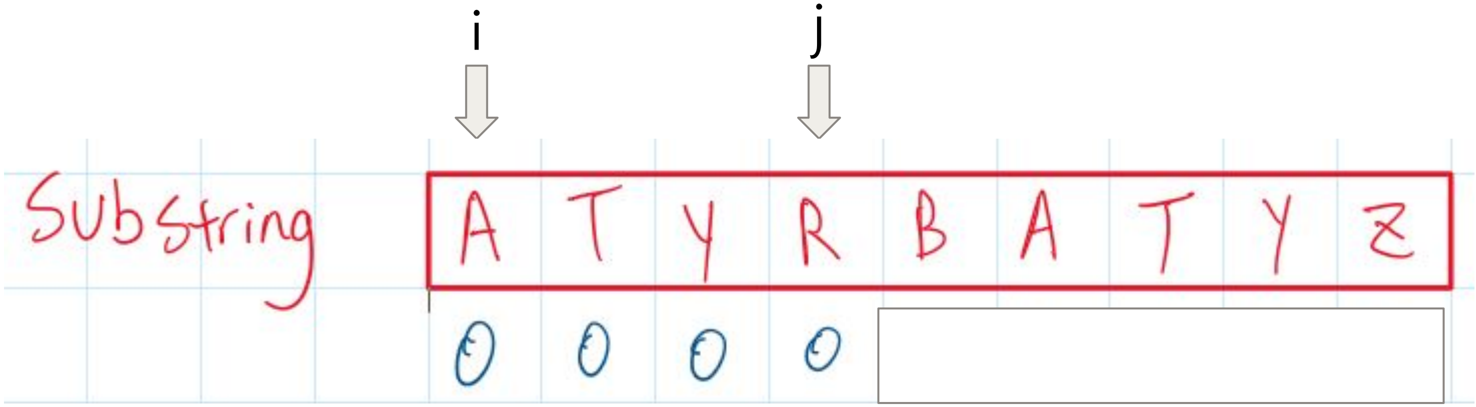
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0							

KMP Largest Prefix Suffix (LPS) Table



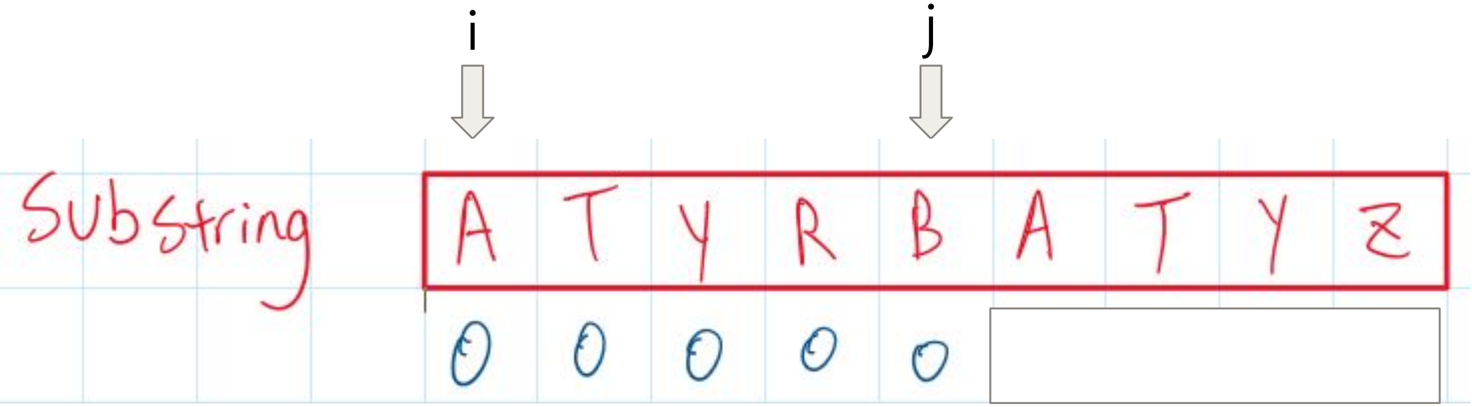
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0						

KMP Largest Prefix Suffix (LPS) Table



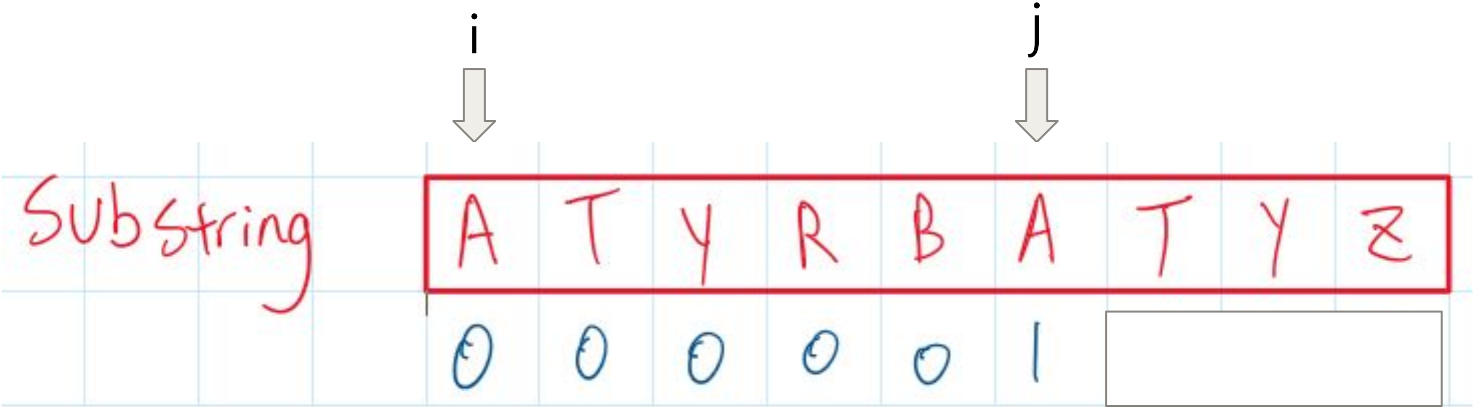
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0					

KMP Largest Prefix Suffix (LPS) Table



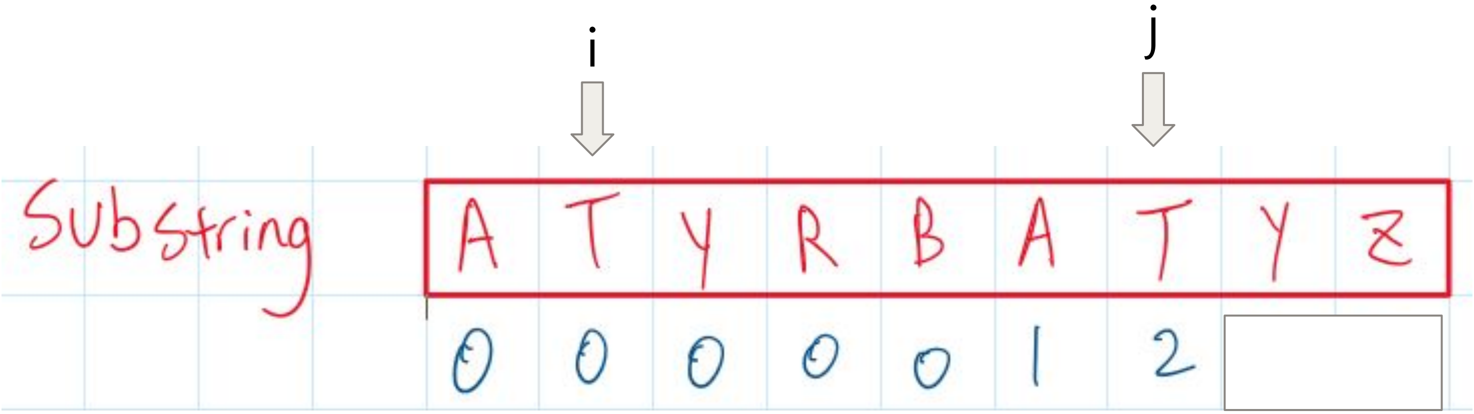
	i				j				
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0				

KMP Largest Prefix Suffix (LPS) Table



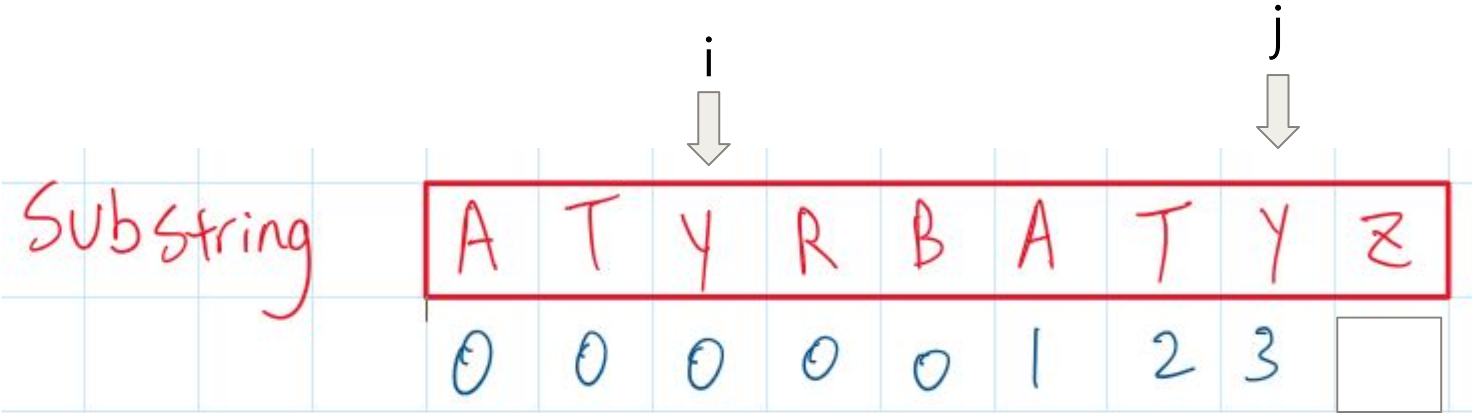
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1			

KMP Largest Prefix Suffix (LPS) Table



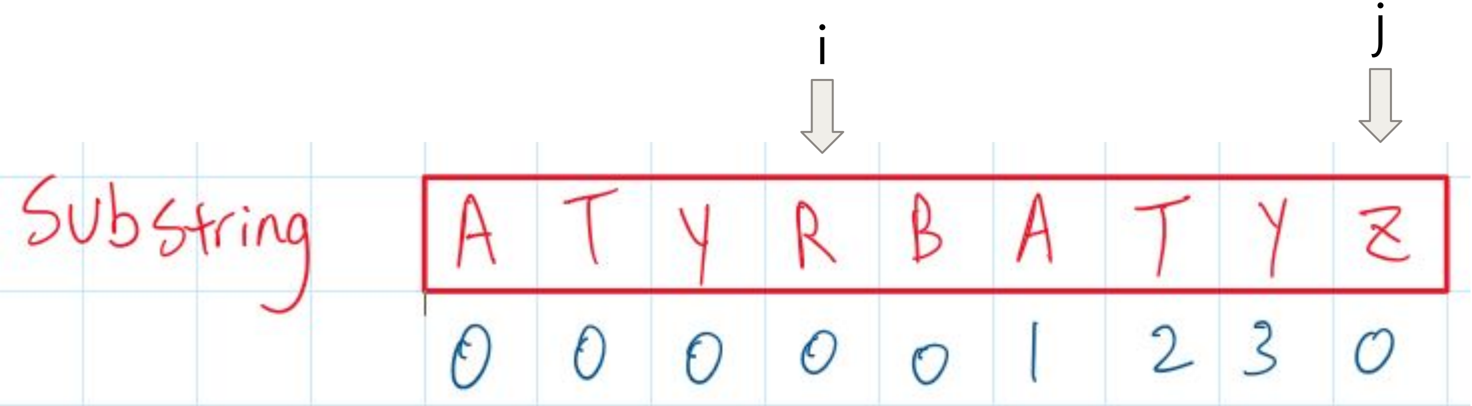
		i					j		
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2		

KMP Largest Prefix Suffix (LPS) Table



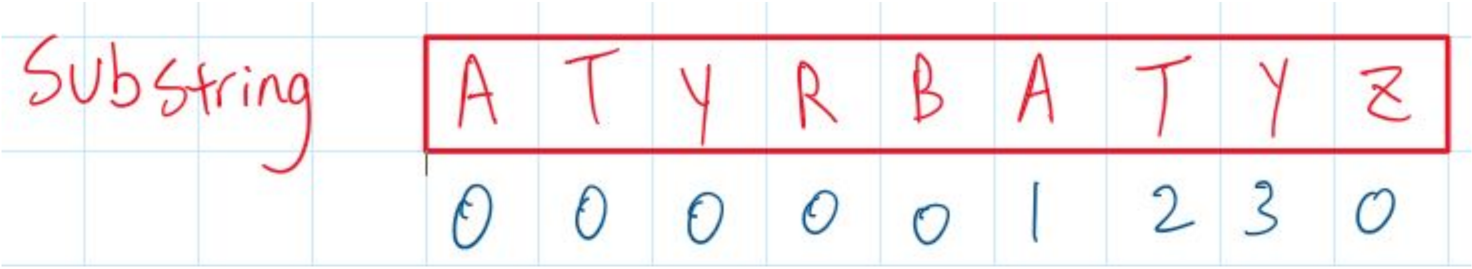
			i					j	
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	□

KMP Largest Prefix Suffix (LPS) Table



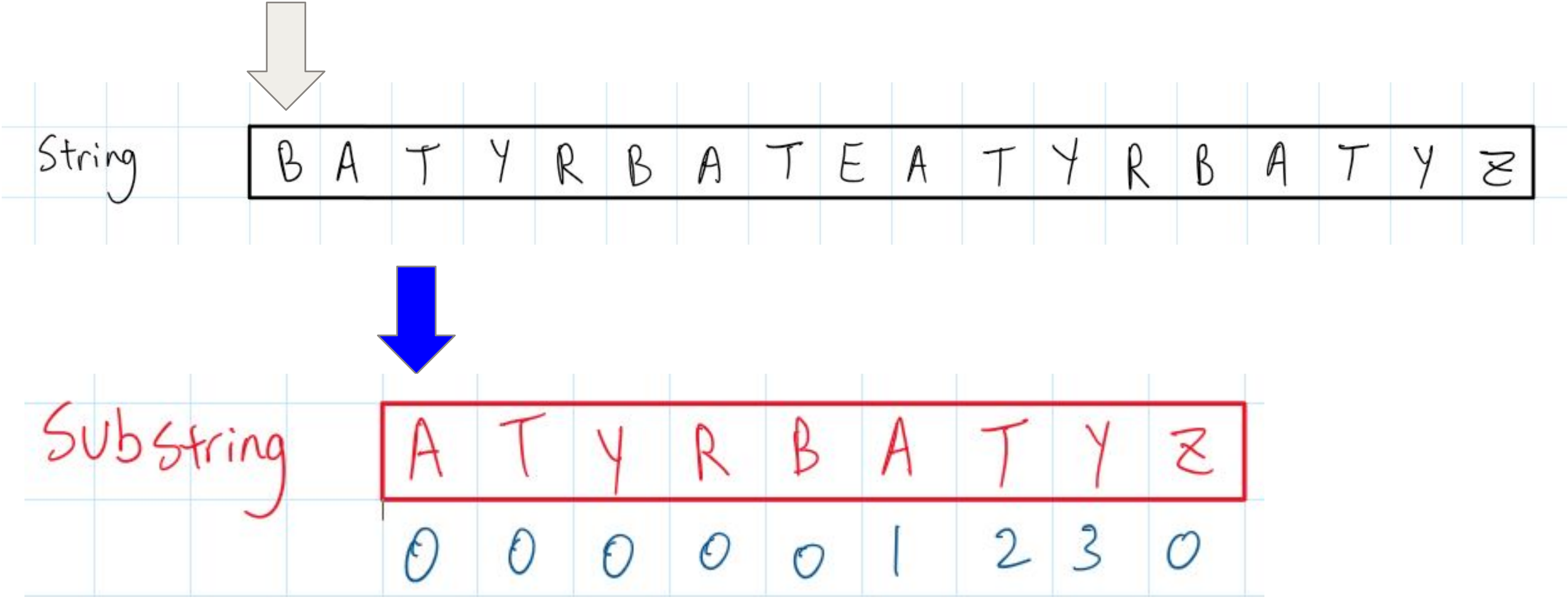
Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

KMP Largest Prefix Suffix (LPS) Table



Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

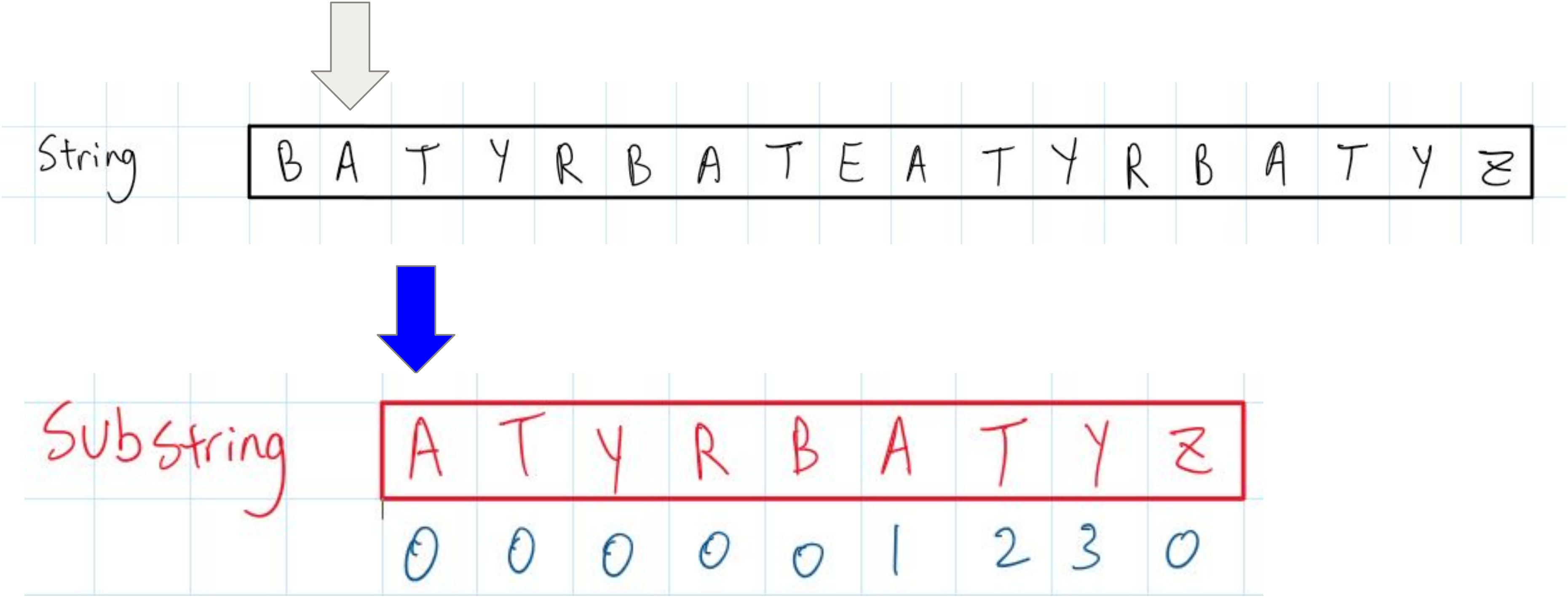
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

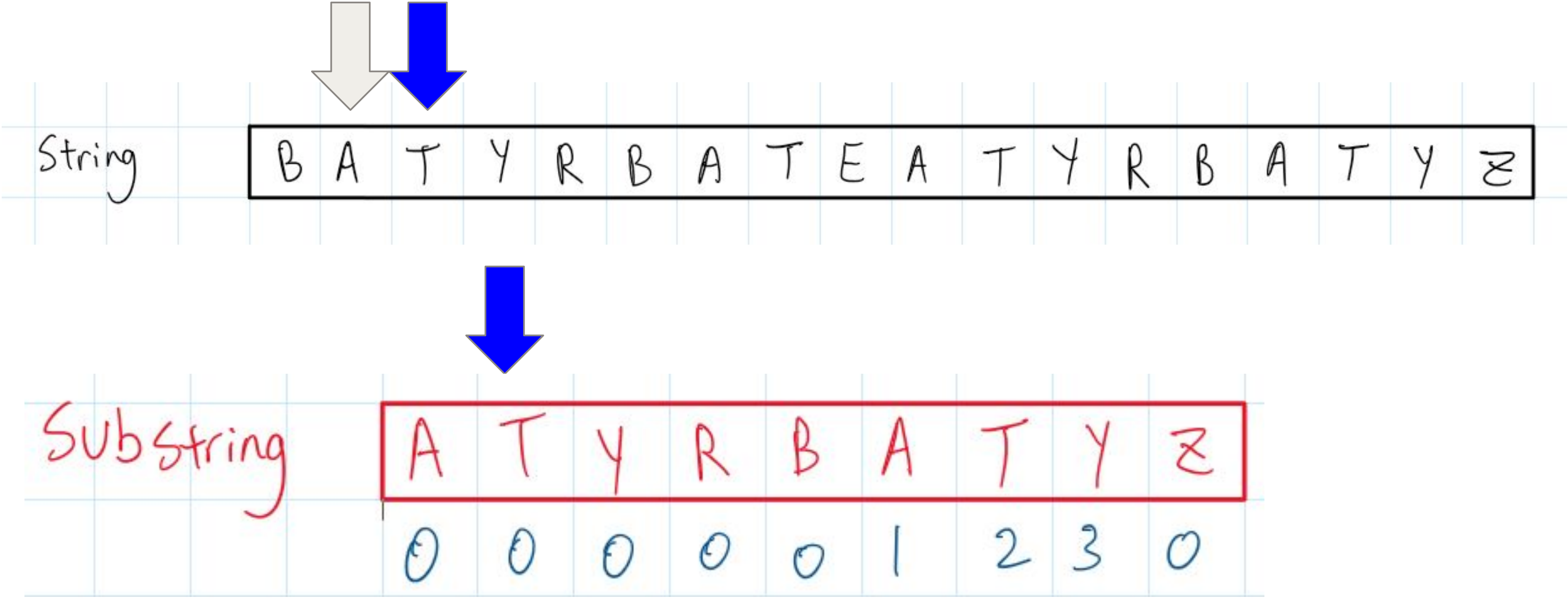
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

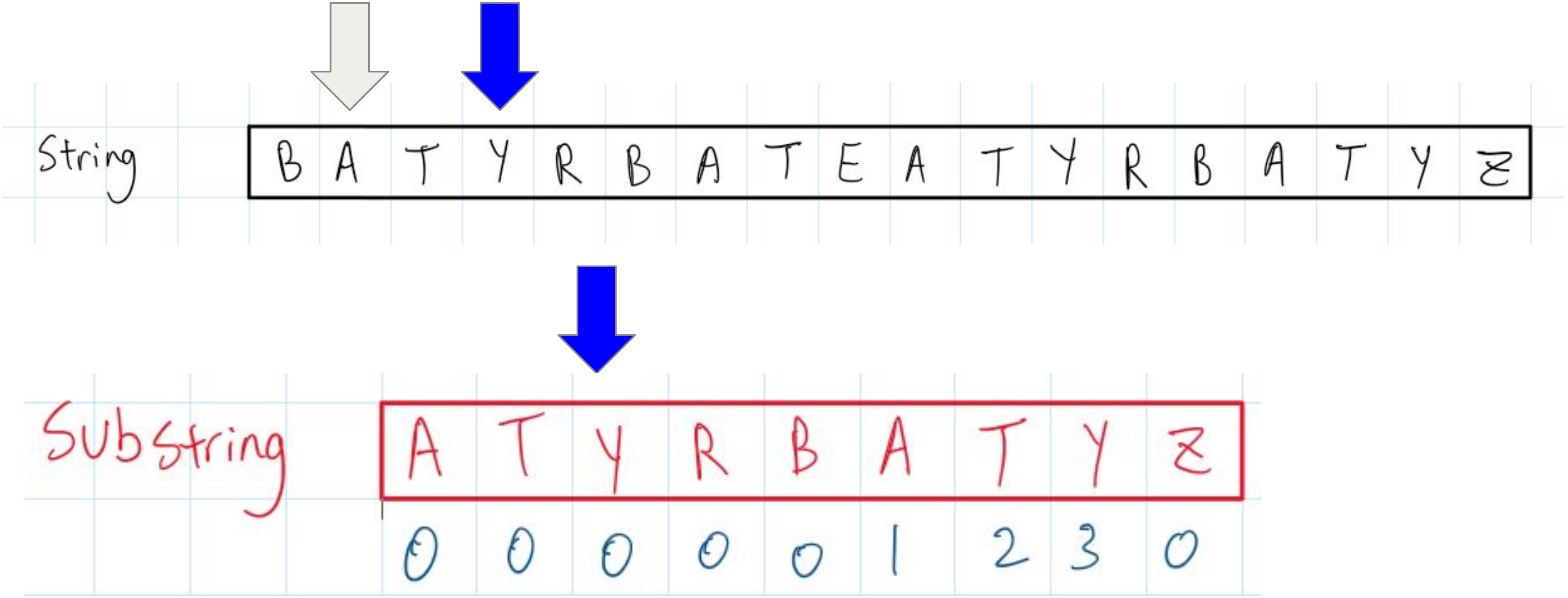
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

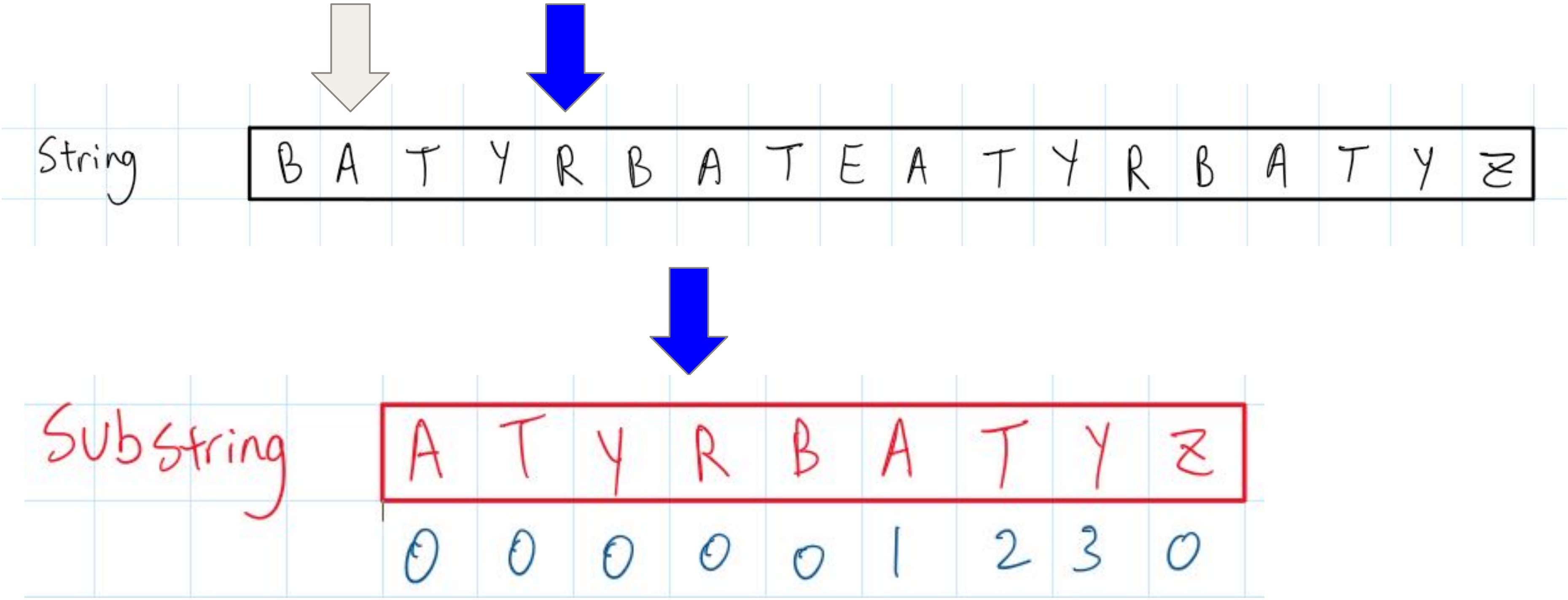
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

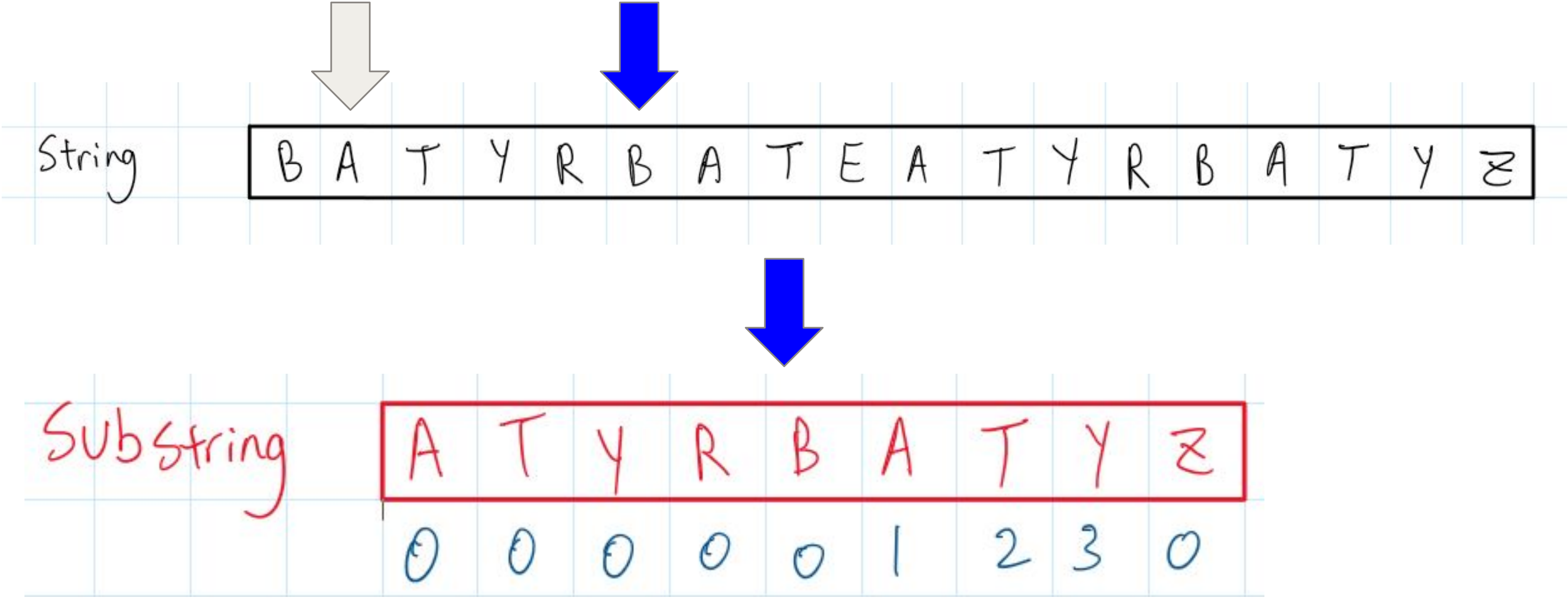
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

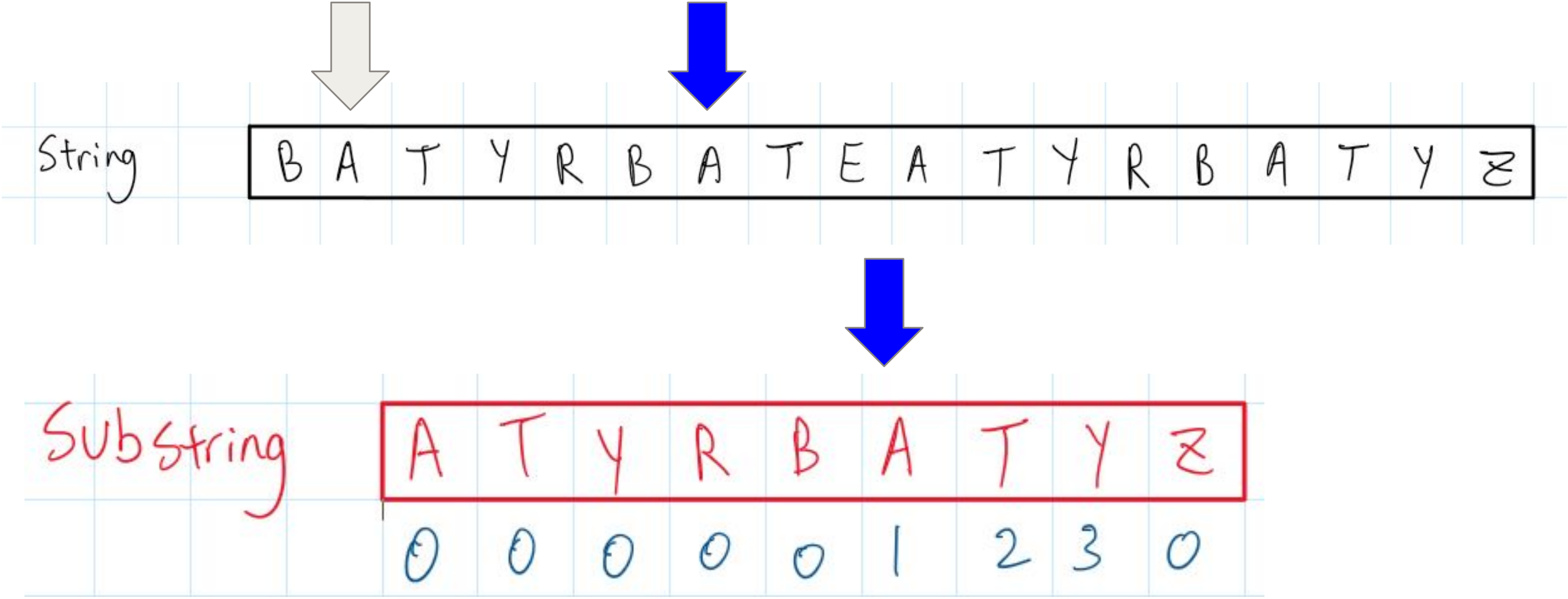
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

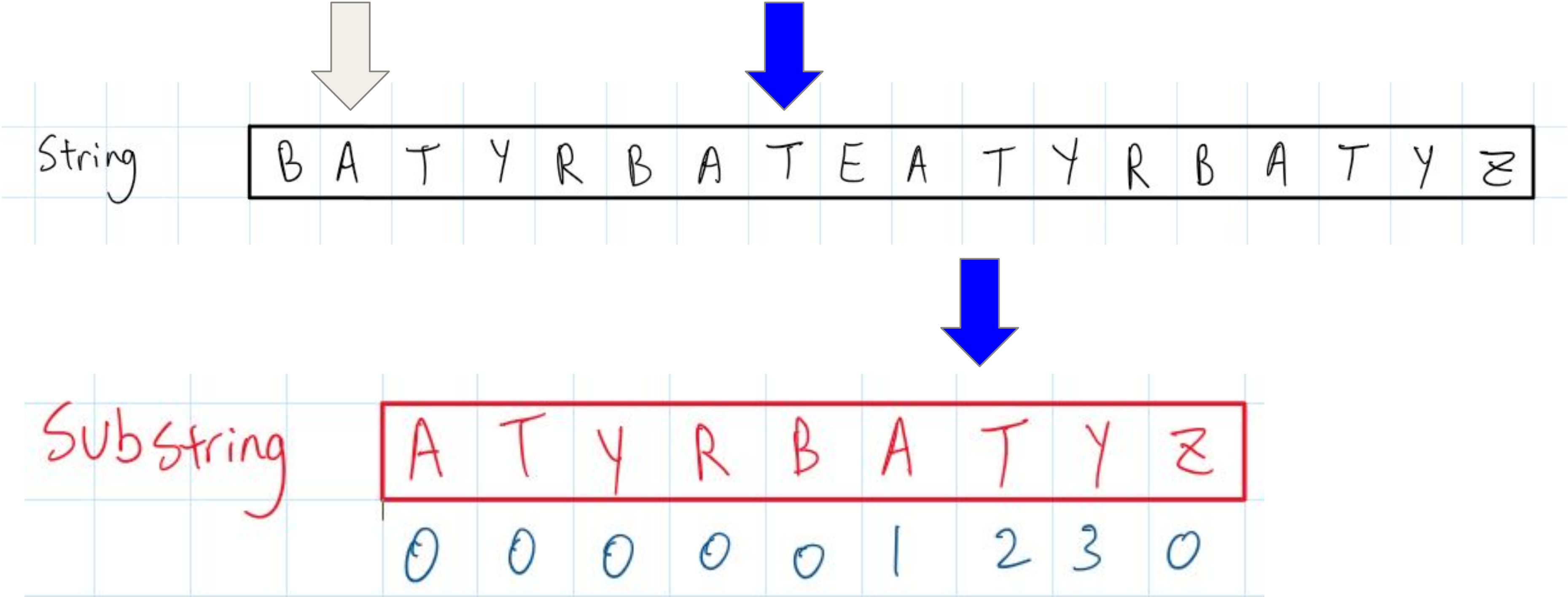
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

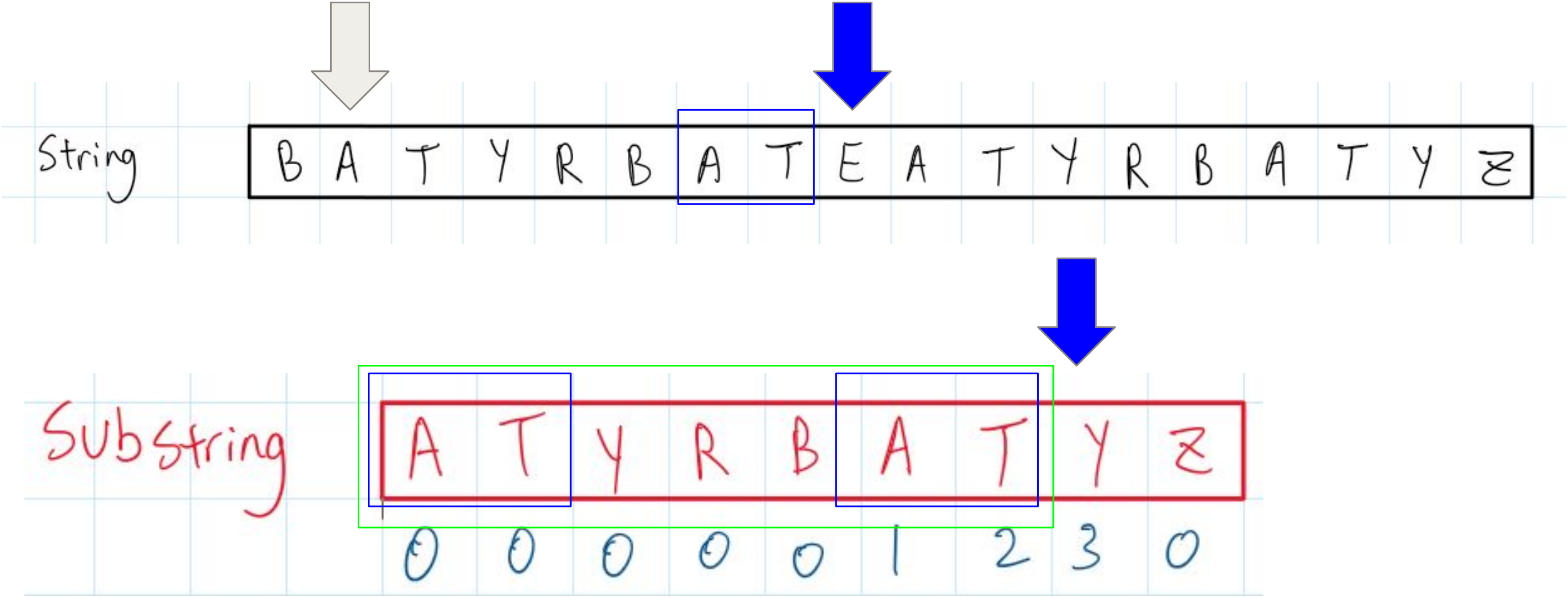
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

How KMP Works (Diagram)

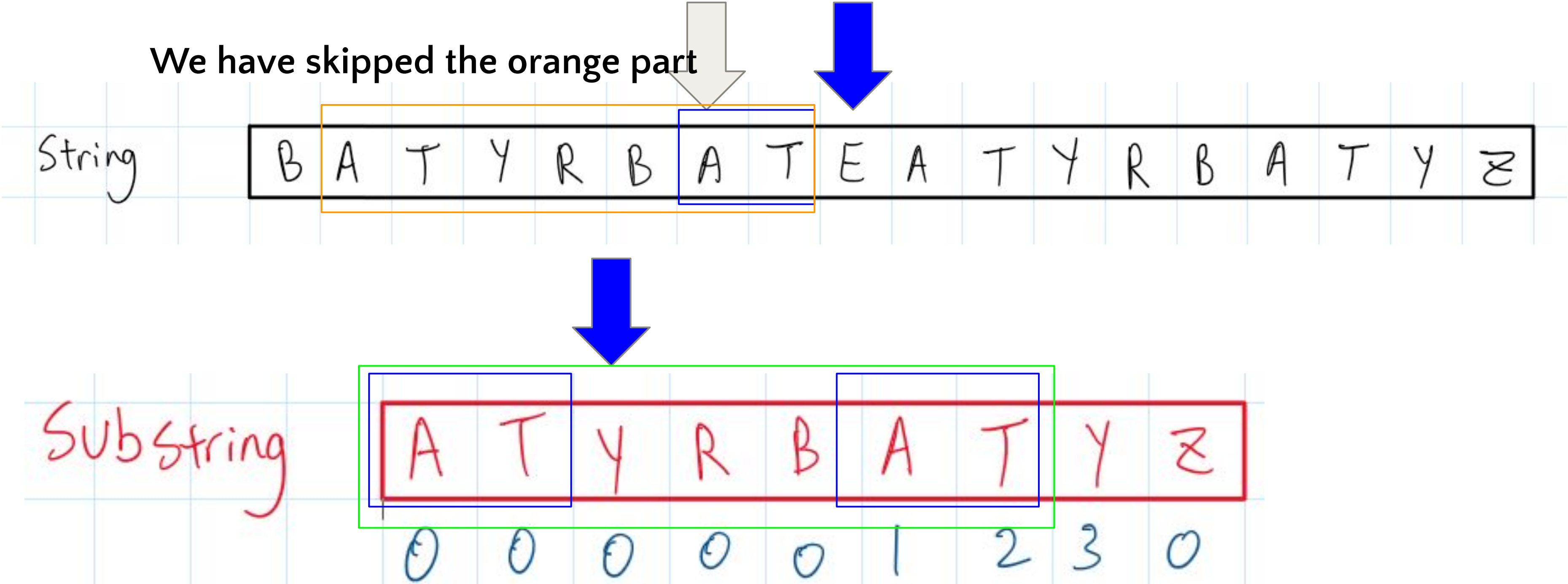


String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

How KMP Works (Diagram)

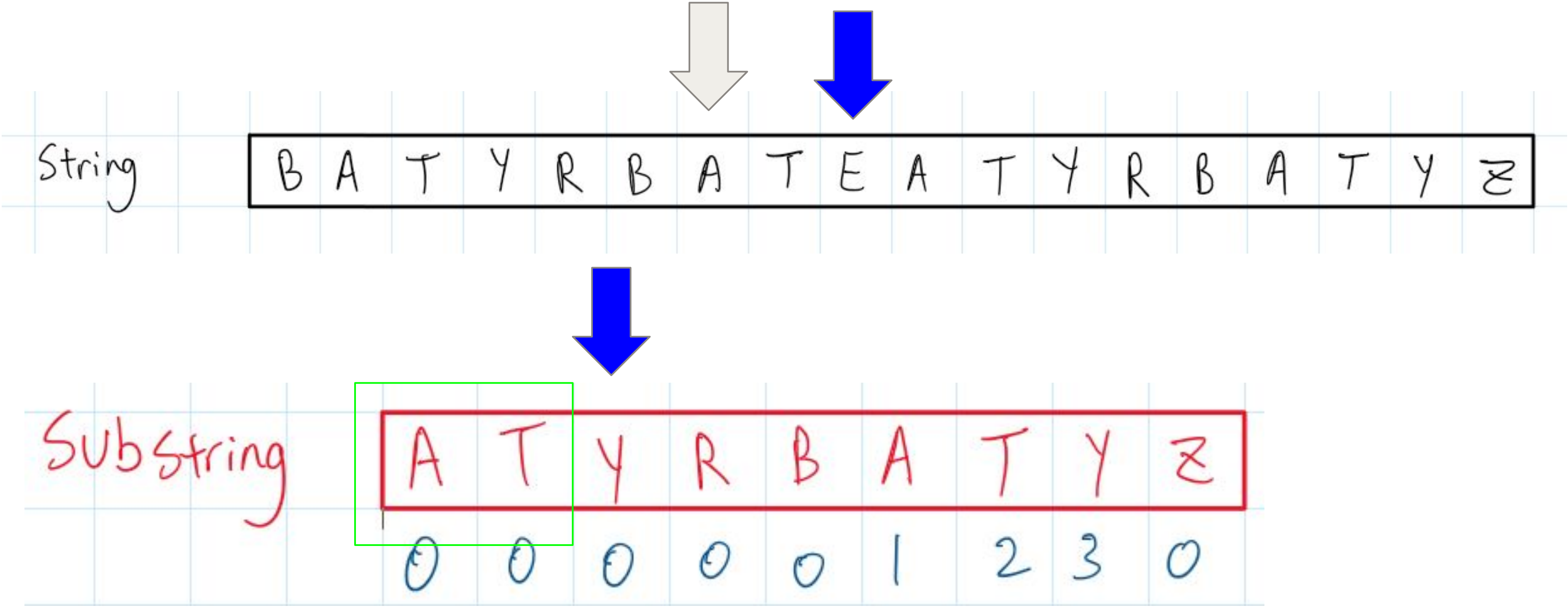
We have skipped the orange part



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

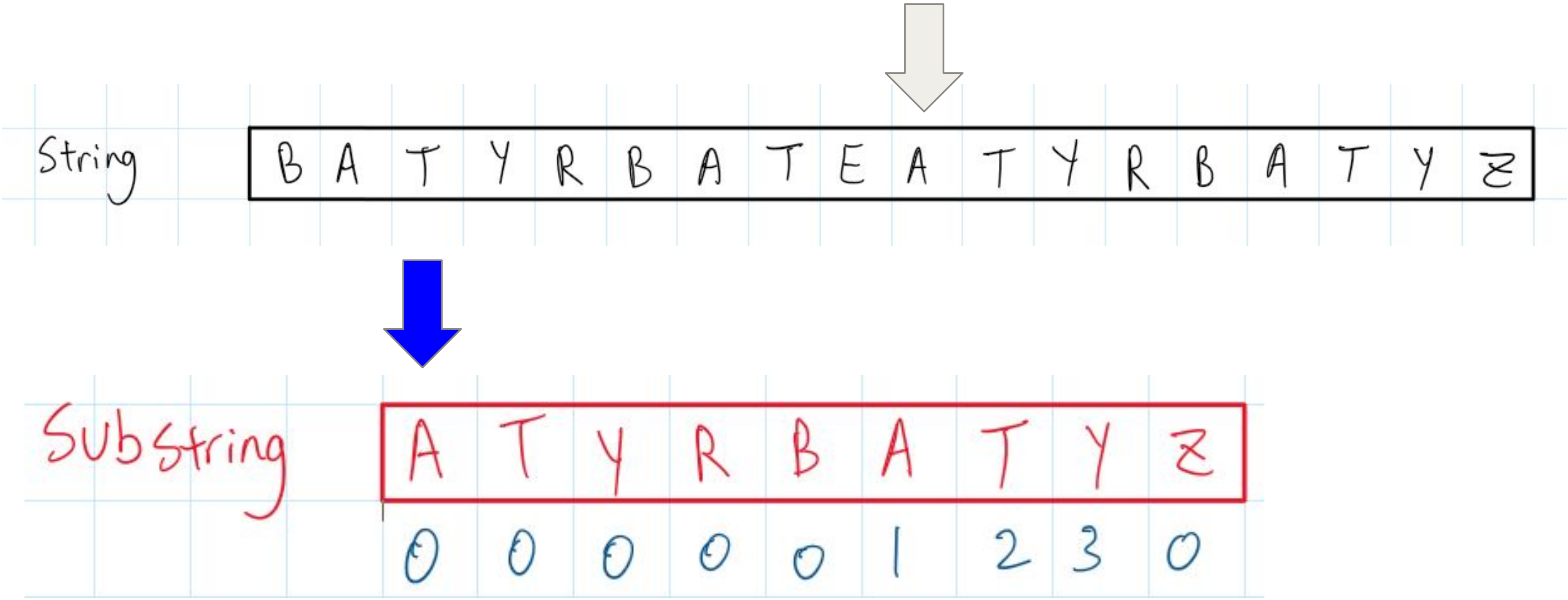
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

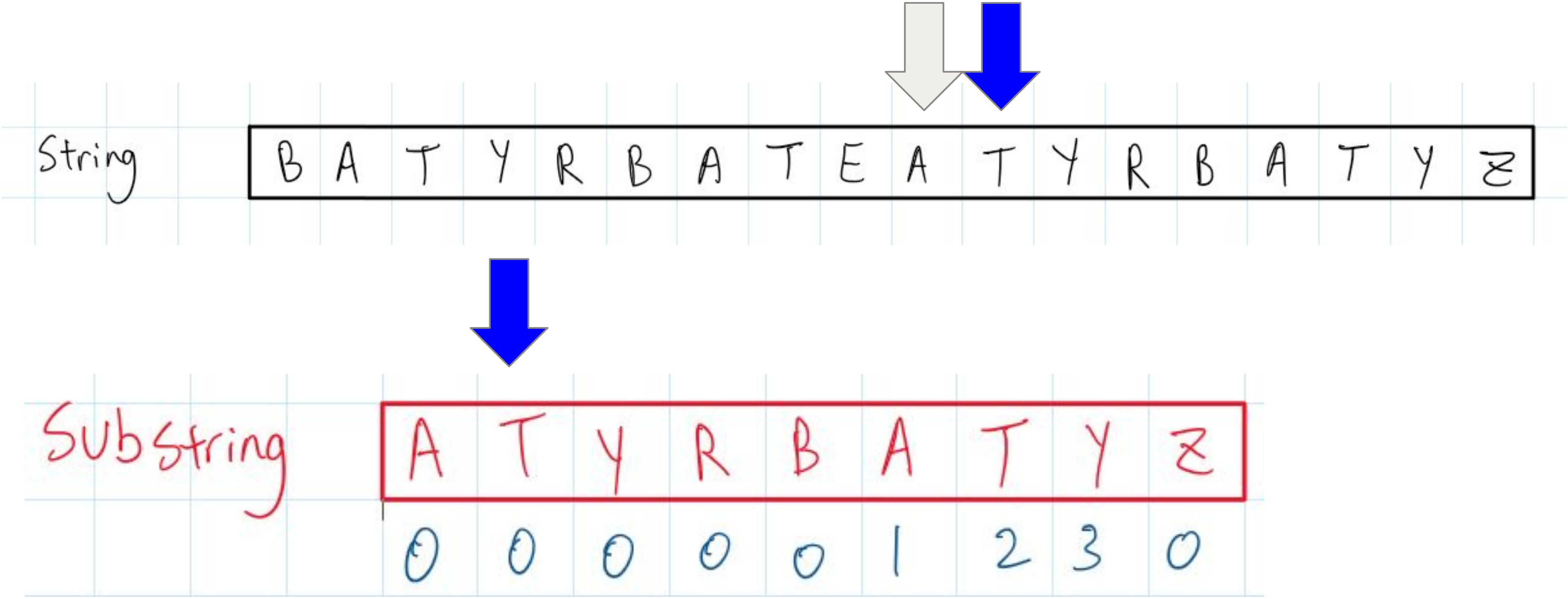
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

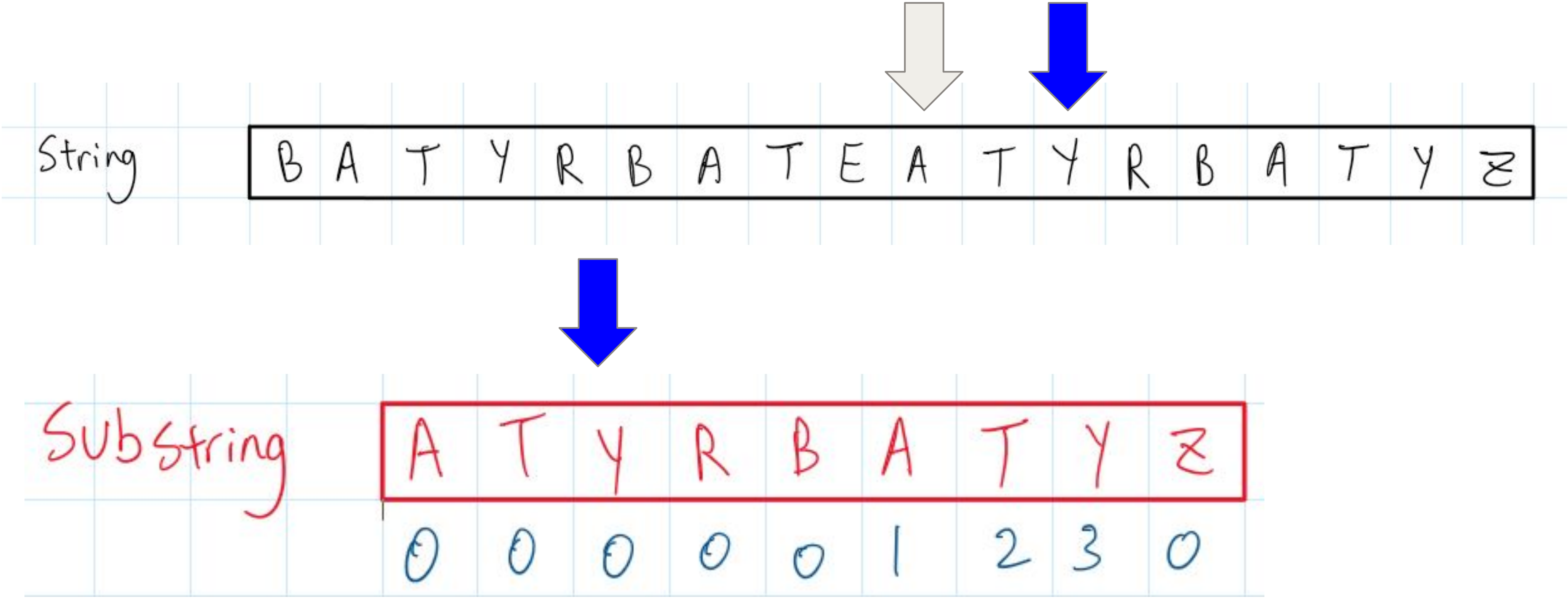
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

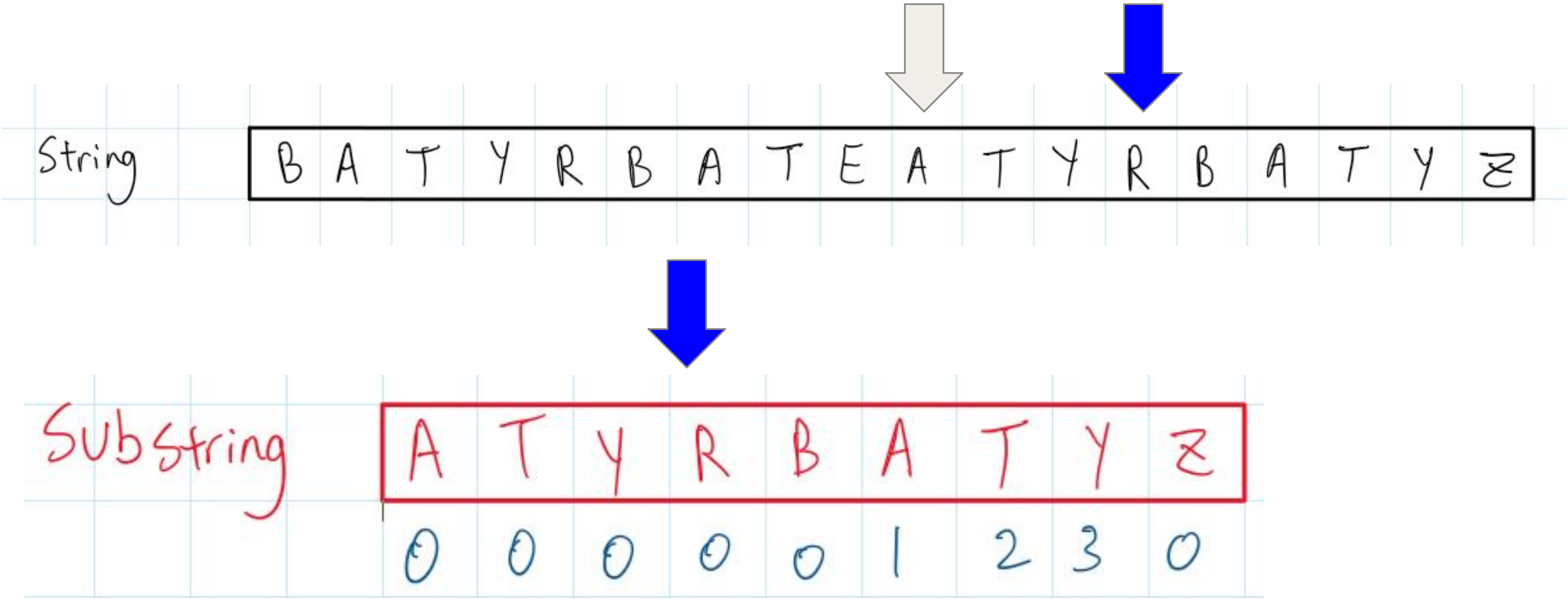
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

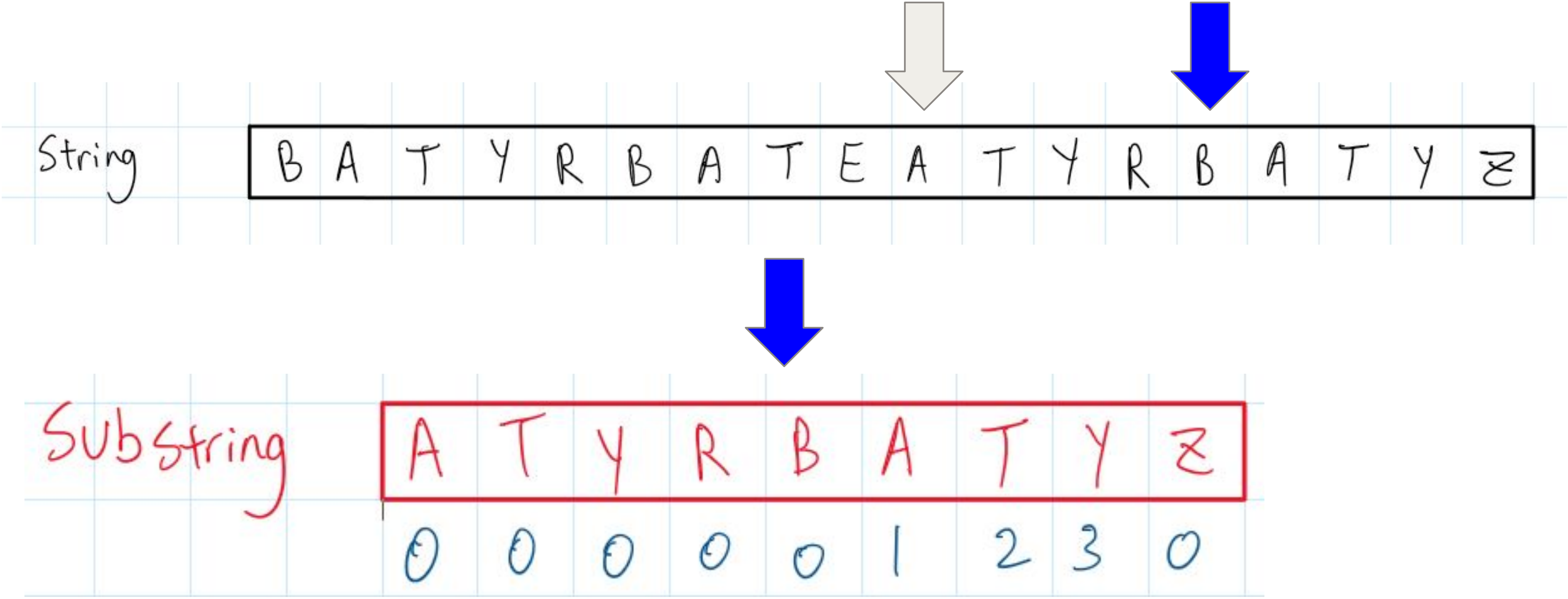
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

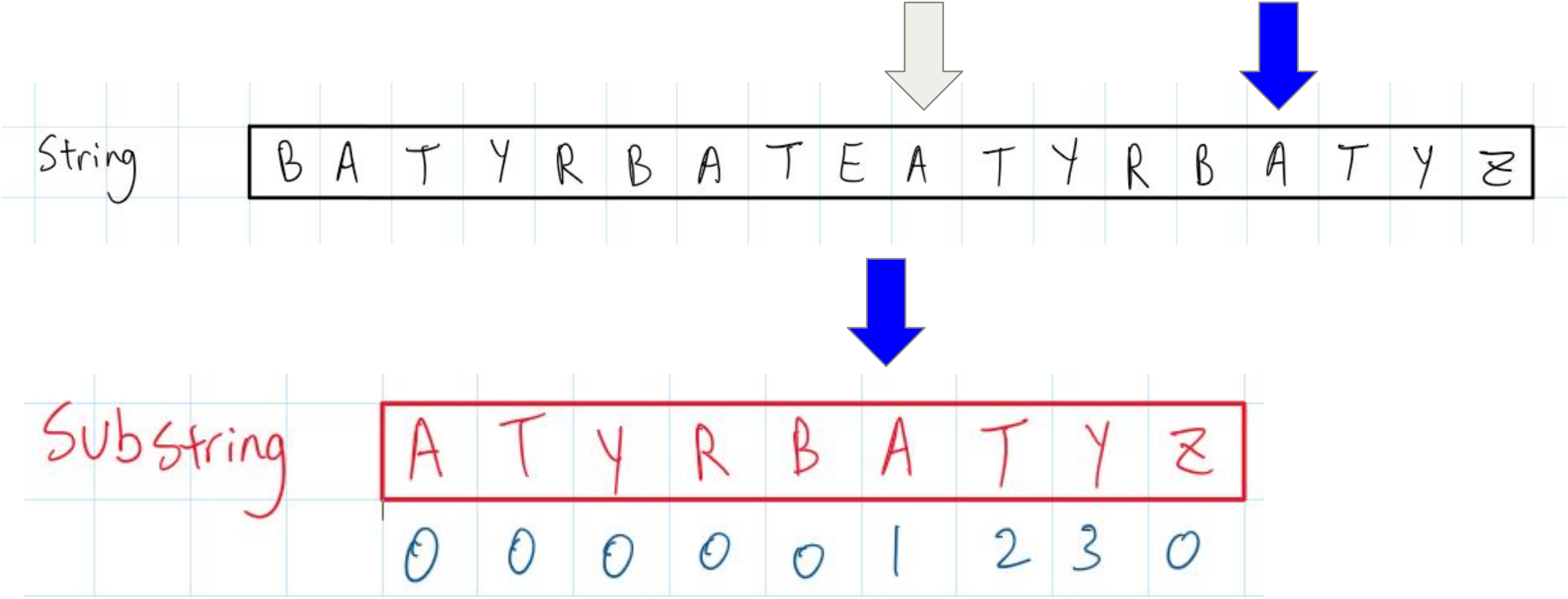
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

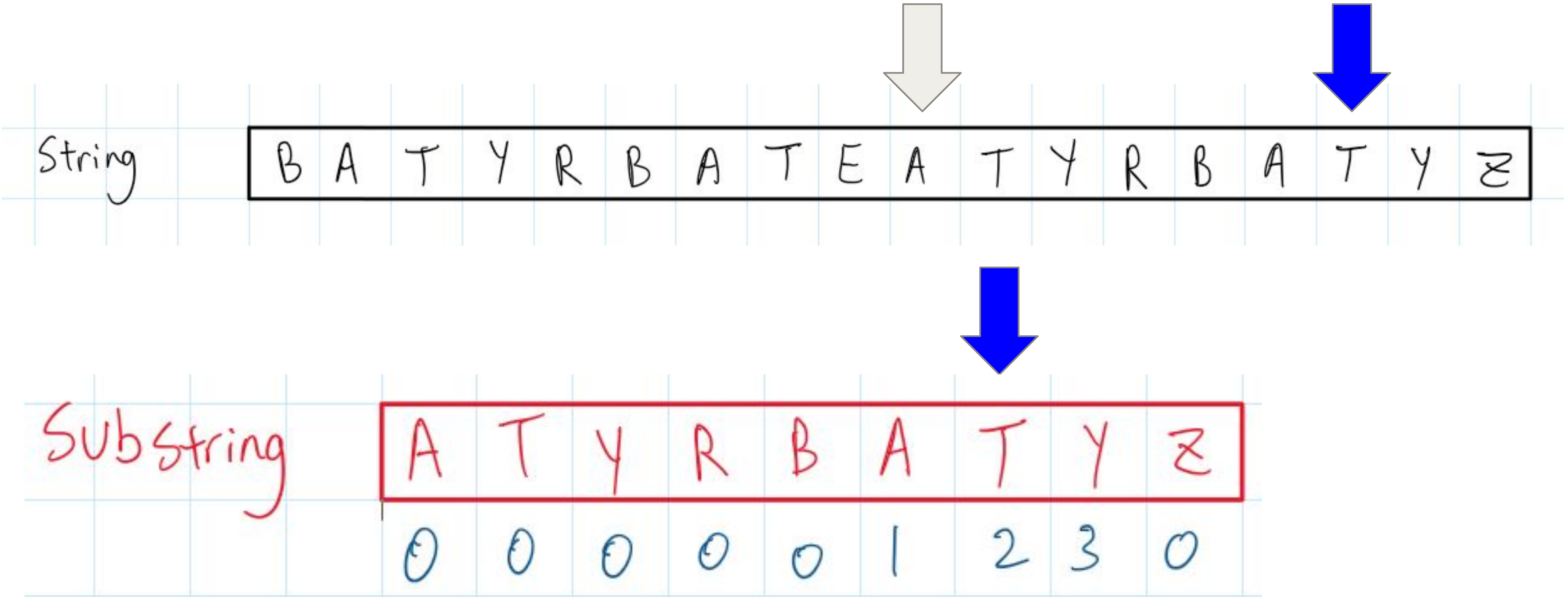
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

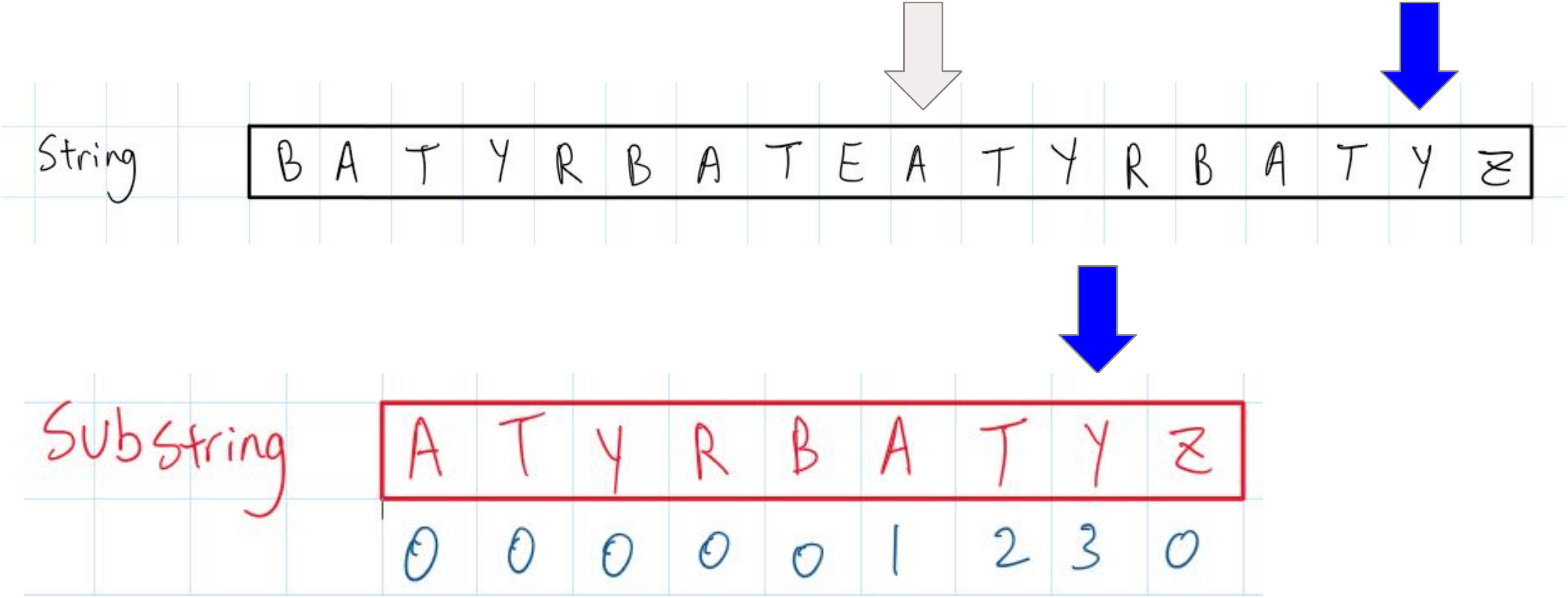
How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

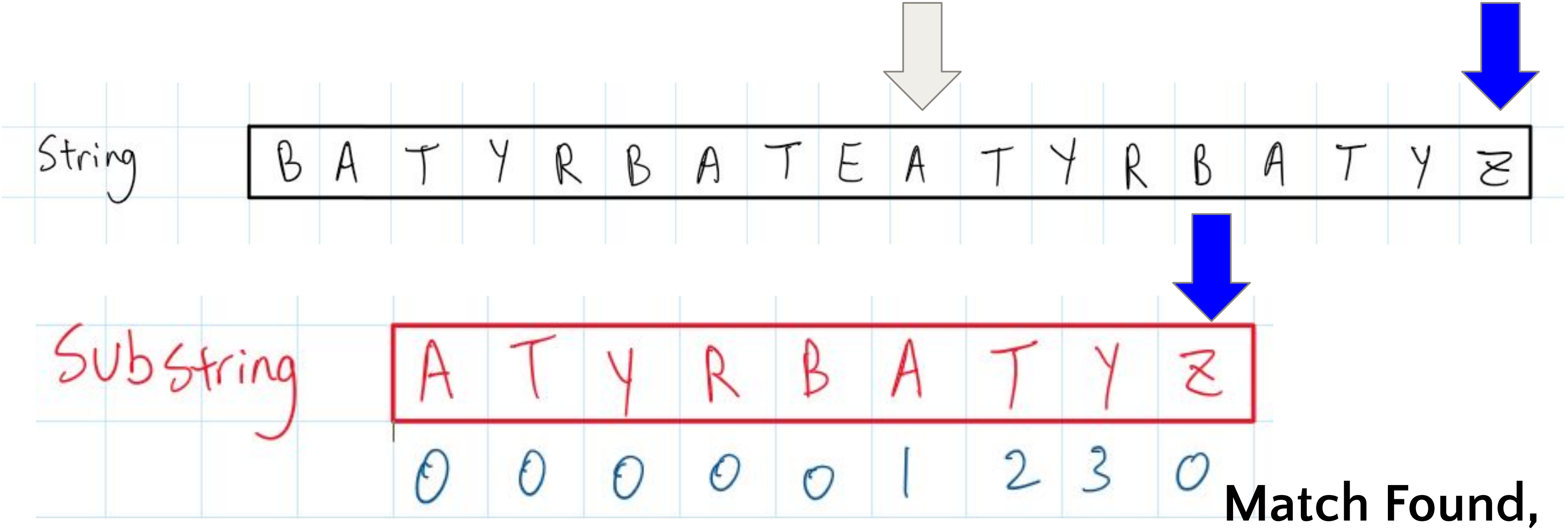
How KMP Works (Diagram)



String																		
	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

Substring									
	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

How KMP Works (Diagram)



String	B	A	T	Y	R	B	A	T	E	A	T	Y	R	B	A	T	Y	Z

SubString	A	T	Y	R	B	A	T	Y	Z
	0	0	0	0	0	1	2	3	0

Match Found, return index (i)

Analysis for KMP Time Complexity

If String = n elements

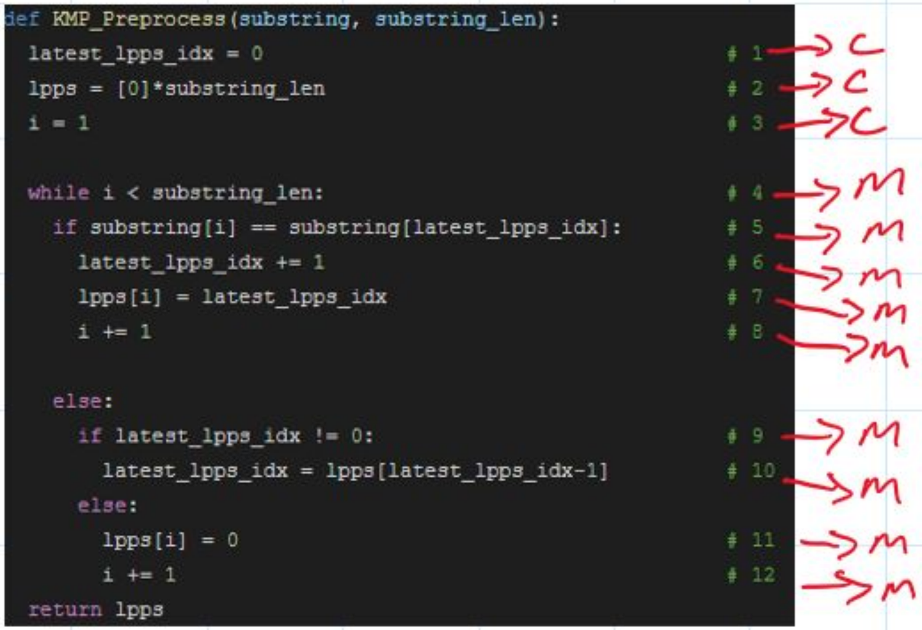
Substring = m elements

It will compare all elements in String with Substring hence

Best Case: O(n)

Worst Case: O(n+m)

KMP Preprocess (Generate LPS Table) Time Complexity

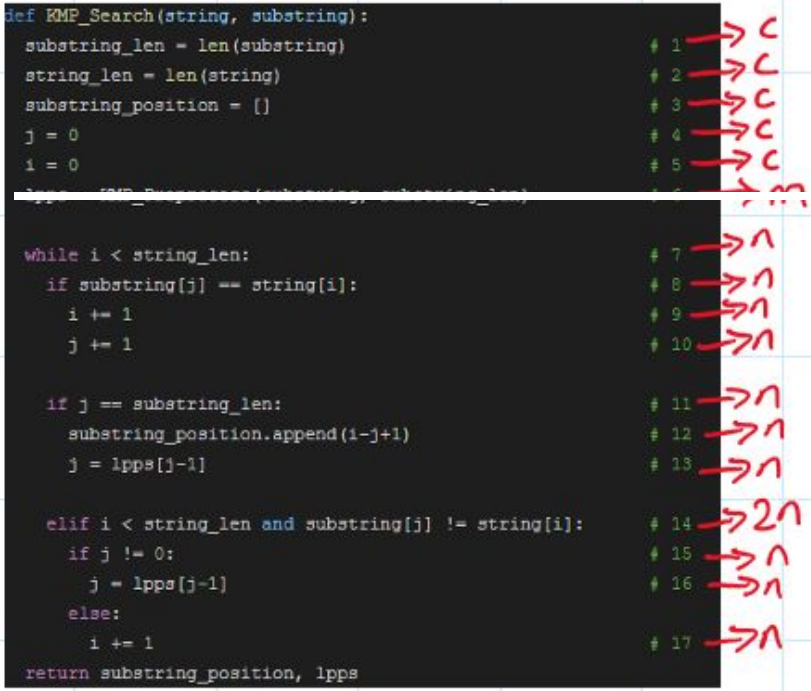


```
def KMP_Preprocess(substring, substring_len):
  latest_lpps_idx = 0                                    # 1  → c
  lpps = [0]*substring_len                               # 2  → c
  i = 1                                                  # 3  → c

  while i < substring_len:                               # 4  → m
    if substring[i] == substring[latest_lpps_idx]:       # 5  → m
      latest_lpps_idx += 1                               # 6  → m
      lpps[i] = latest_lpps_idx                          # 7  → m
      i += 1                                             # 8  → m

    else:
      if latest_lpps_idx != 0:                           # 9  → m
        latest_lpps_idx = lpps[latest_lpps_idx-1]        # 10 → m
      else:
        lpps[i] = 0                                      # 11 → m
        i += 1                                           # 12 → m
  return lpps
```

Time complexity $= 3c + 9m$

$= O(m)$

KMP Search Time Complexity (Best Case)



```
def KMP_Search(string, substring):
  substring_len = len(substring)                                # 1
  string_len = len(string)                                      # 2
  substring_position = []                                       # 3
  j = 0                                                         # 4
  i = 0                                                         # 5
  ~~lpps = KMP_Preprocess(substring, substring_len)~~              ~~# 6~~

  while i < string_len:                                         # 7
    if substring[j] == string[i]:                               # 8
      i += 1                                                    # 9
      j += 1                                                    # 10

    if j == substring_len:                                      # 11
      substring_position.append(i-j+1)                          # 12
      j = lpps[j-1]                                             # 13

    elif i < string_len and substring[j] != string[i]:          # 14
      if j != 0:                                                # 15
        j = lpps[j-1]                                           # 16
      else:
        i += 1                                                  # 17
  return substring_position, lpps
```

Best Case

Assuming that LPS Table already exists

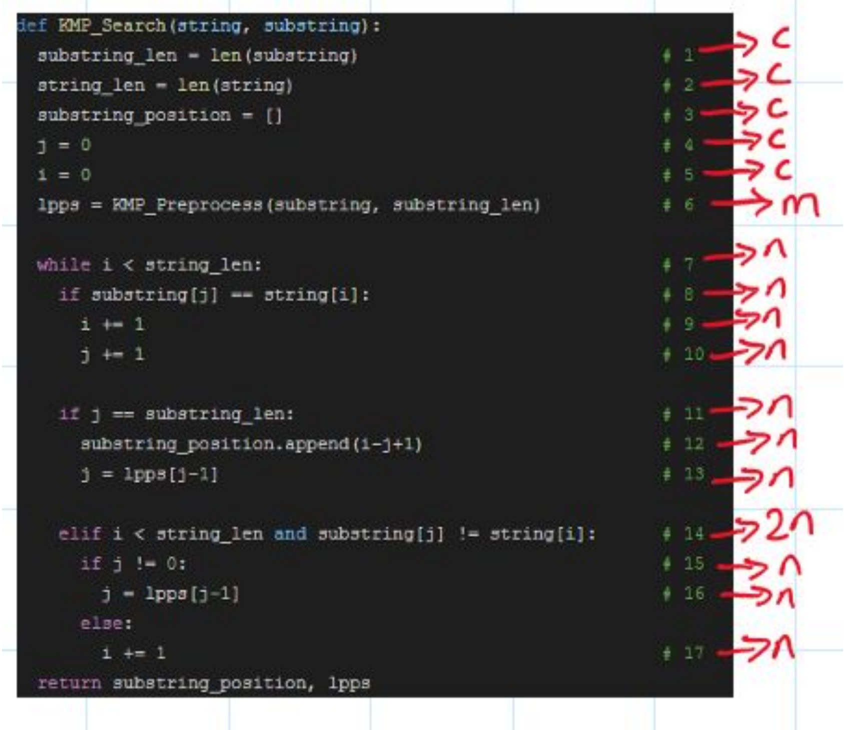
[Process to generate LPS table not required]

Code at #6 can be removed

$$\text{Time Complexity} = 5c + 12\underline{\underline{n}}$$

$$= O(n)$$

KMP Search Time Complexity (Worst Case)



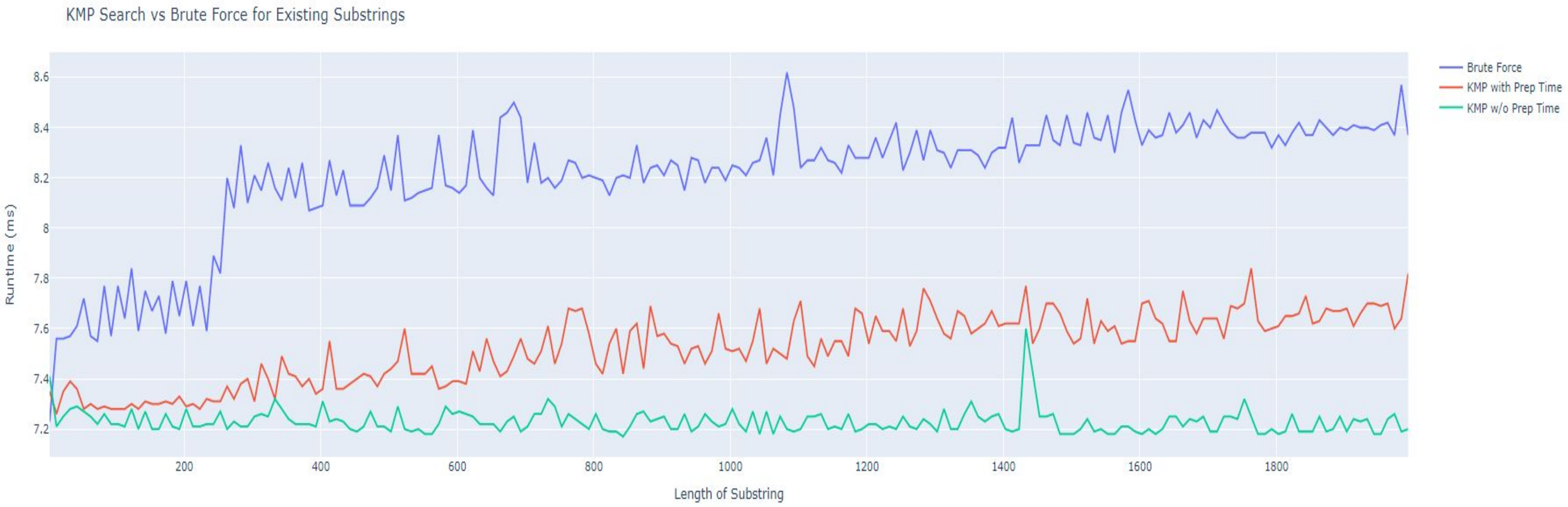
```
def KMP_Search(string, substring):
  substring_len = len(substring)                            # 1
  string_len = len(string)                                  # 2
  substring_position = []                                   # 3
  j = 0                                                     # 4
  i = 0                                                     # 5
  lpps = KMP_Preprocess(substring, substring_len)           # 6

  while i < string_len:                                     # 7
    if substring[j] == string[i]:                           # 8
      i += 1                                                # 9
      j += 1                                                # 10

    if j == substring_len:                                  # 11
      substring_position.append(i-j+1)                      # 12
      j = lpps[j-1]                                         # 13

    elif i < string_len and substring[j] != string[i]:      # 14
      if j != 0:                                            # 15
        j = lpps[j-1]                                       # 16
      else:
        i += 1                                              # 17
  return substring_position, lpps
```

Worst case

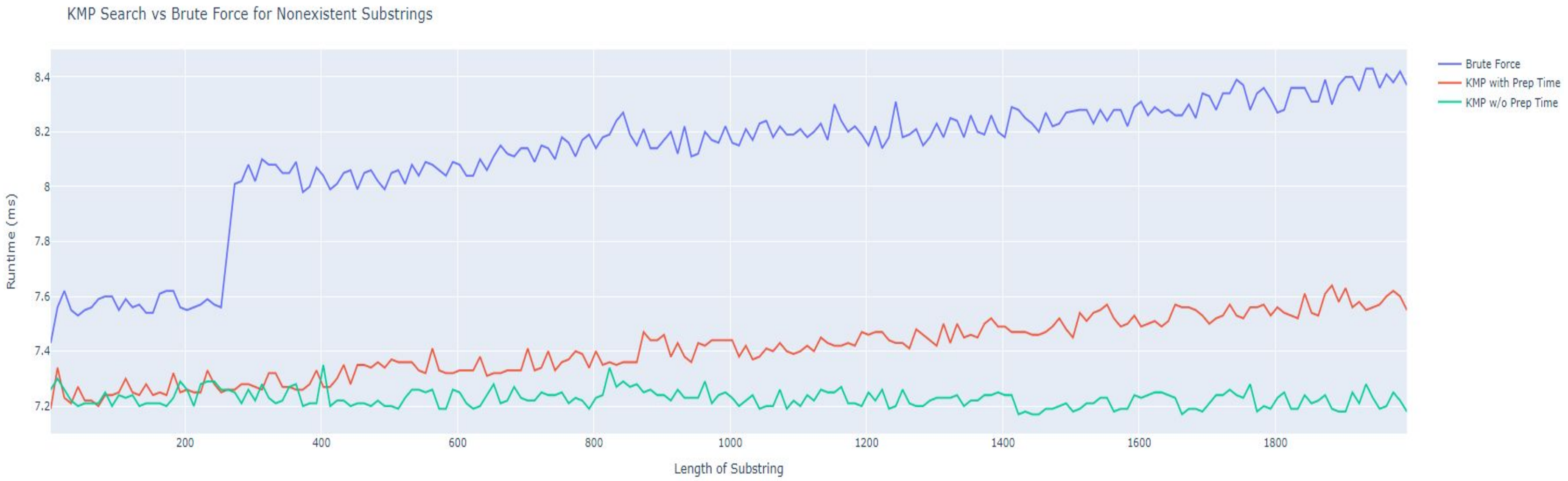
Time complexity $= 5c + m + 12n$

$= O(n+m)$

Comparisons



Comparisons



Live Demonstration