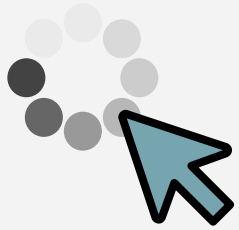
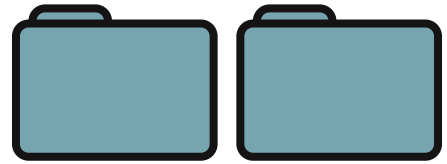
CZ2001 Lab Project 1

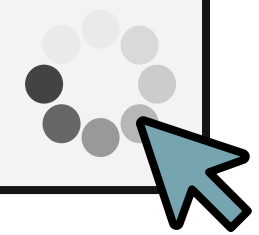


Group 4



Team Members

Chang Heen Sunn	U1920383H
Luo Wenyu	U1922009G
Ooi Wei Chern	U1920504J
Saw William Joseph	U1921246F
Tan Ching Fhen	U1920787D

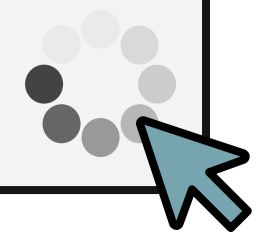


String -Matching

Matching genomic pattern with COVID 19 genome

Presentation Flow

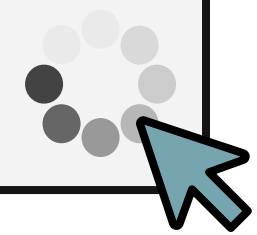
- ➔ Knuth-Morris-Pratt (KMP) Algorithm
- ➔ Boyer-Moore-Horspool (BMH) Algorithm
- ➔ Comparison and Conclusion



Knuth-Morris-Pratt (KMP)

Knuth-Morris-Pratt

- String-searching algorithm by the means of repeated prefix which is also a suffix of the pattern



Pattern: AAGGAAAAG

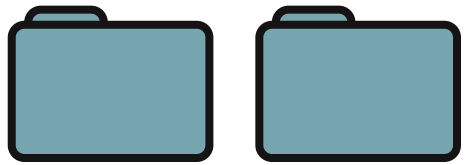
Prefix ->	Suffix <-
A	G
AA	AG
AAG	AAG
...	...
AAGGAAAA	AGGAAAAG

Pi_table / LPS array

Longest proper

Prefix which is also a

Suffix array



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

ANALYSIS

A	A	B	A	A	A	C
0	1	2	3	4	5	6

Pattern

0						

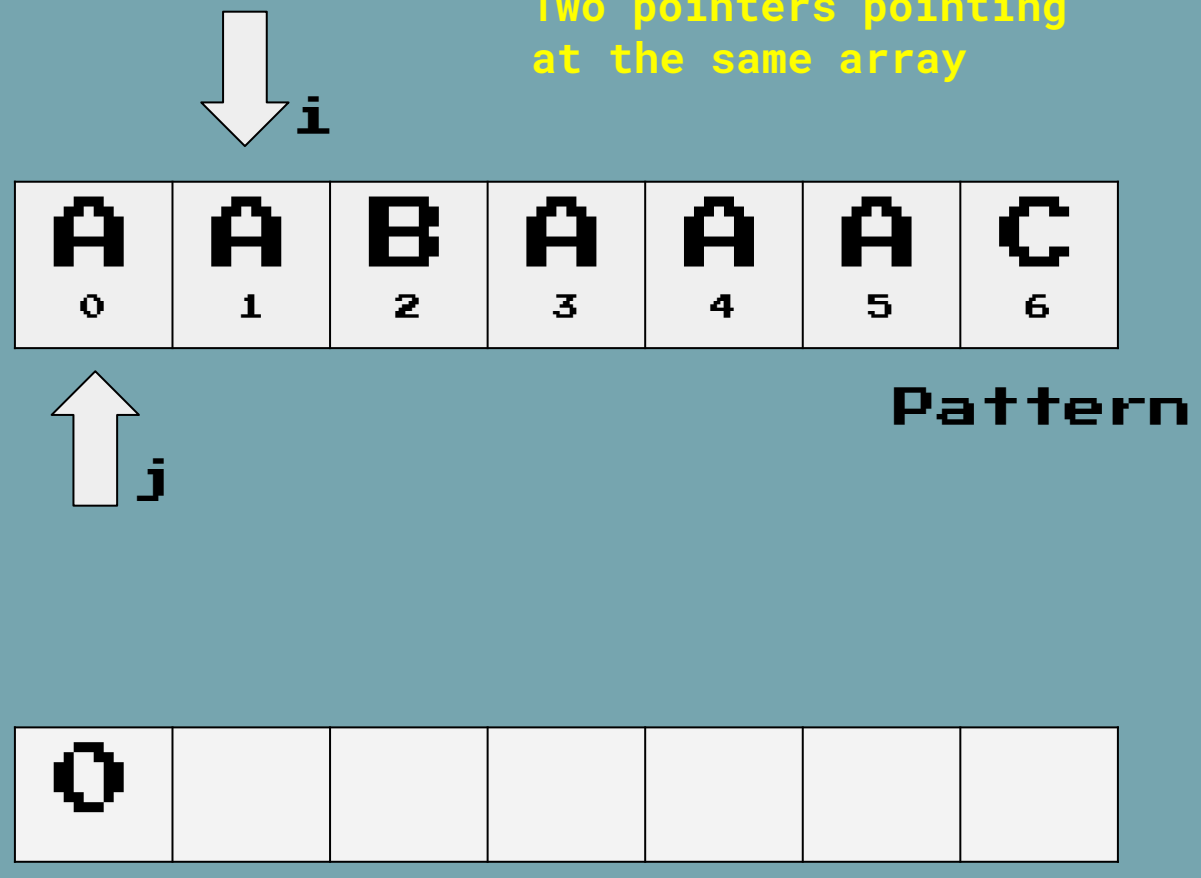
LPS

CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

ANALYSIS

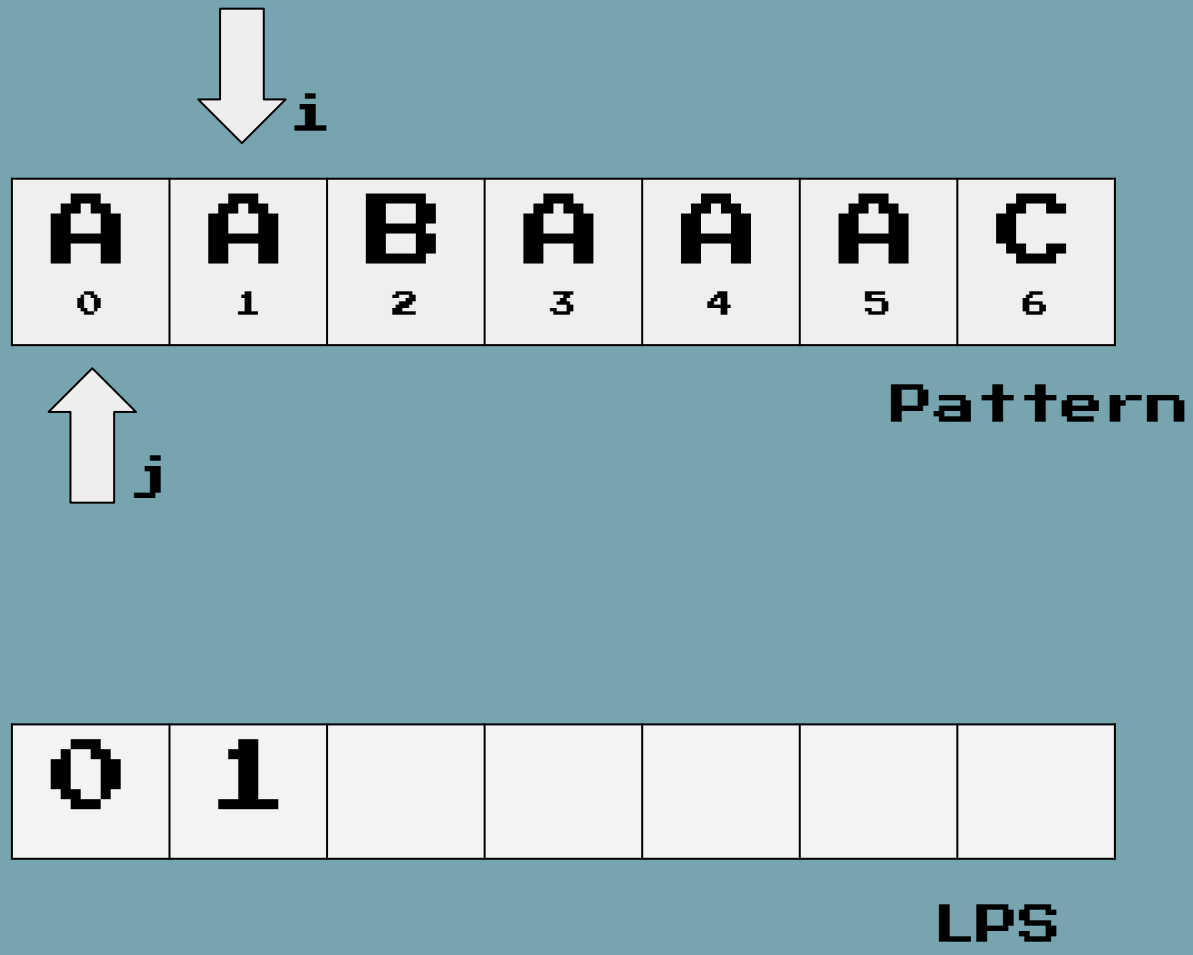
Two pointers pointing at the same array



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

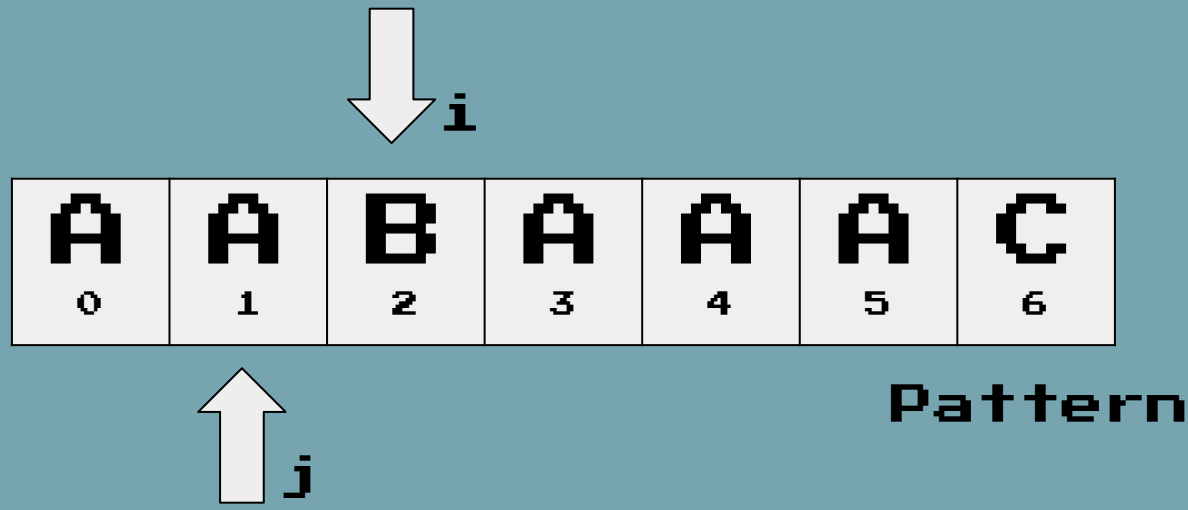
ANALYSIS



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

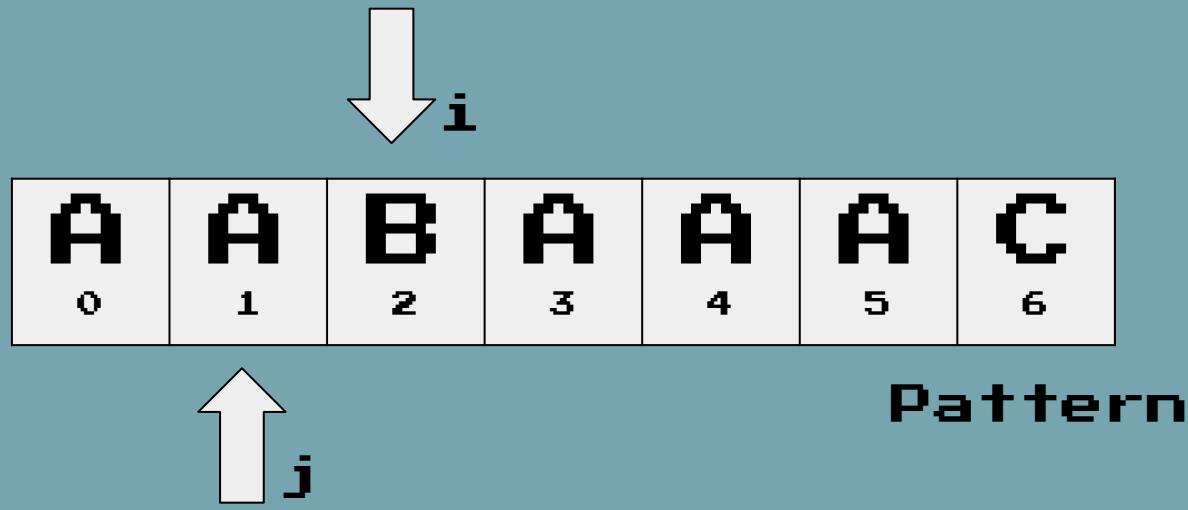
ANALYSIS



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

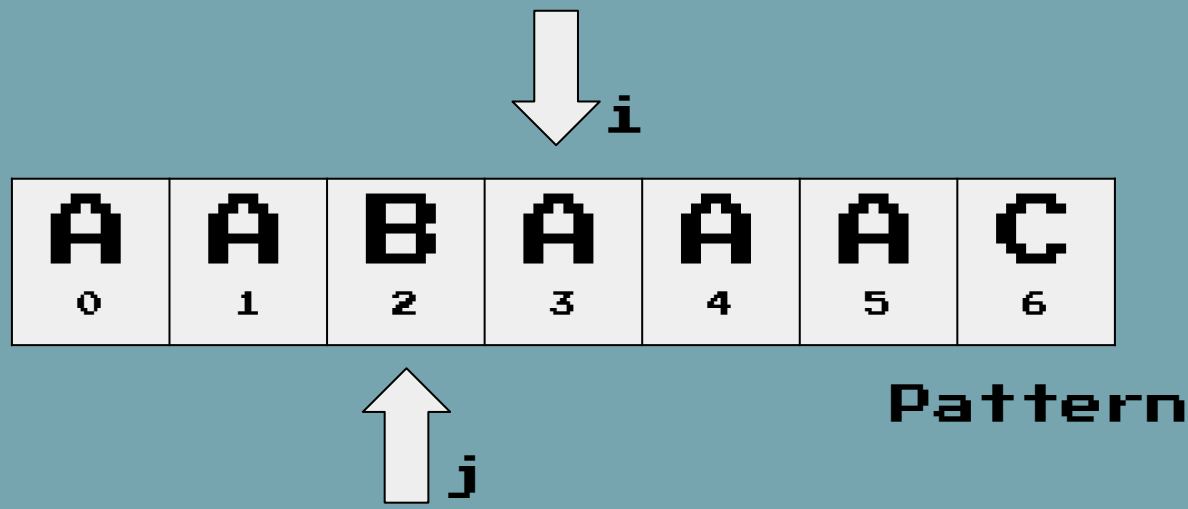
ANALYSIS

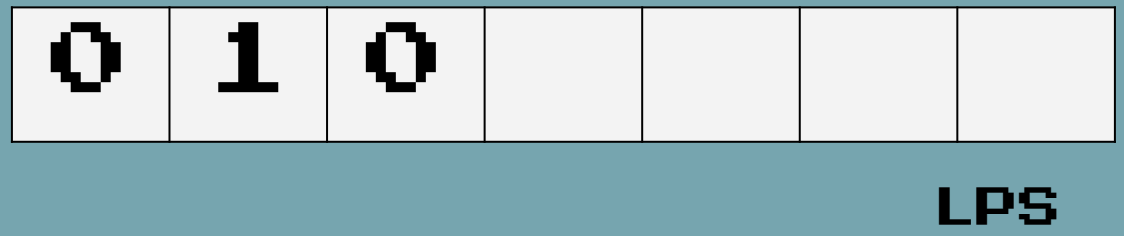


CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

ANALYSIS

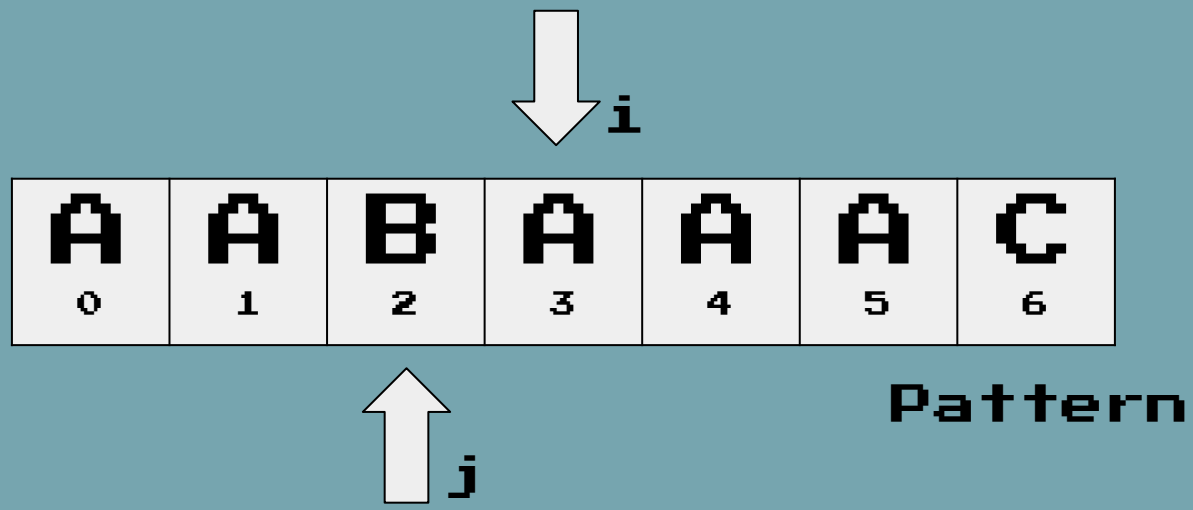




CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

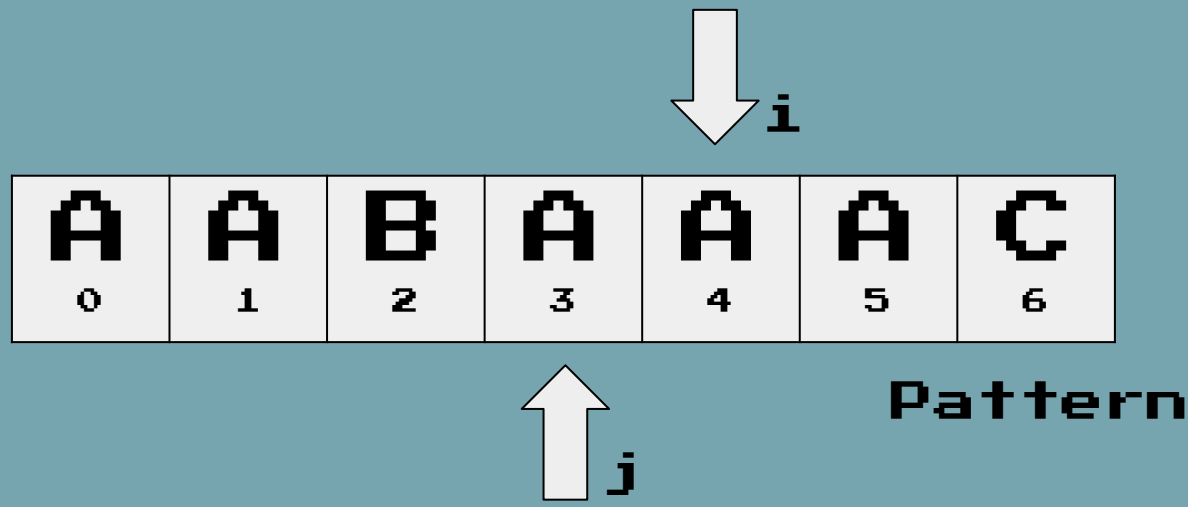
ANALYSIS



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

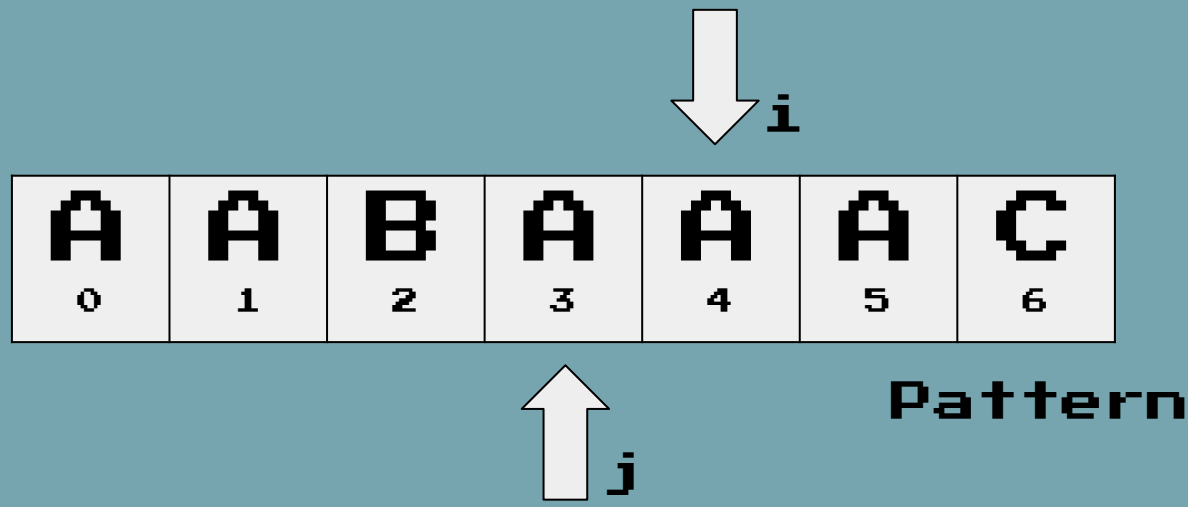
ANALYSIS



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

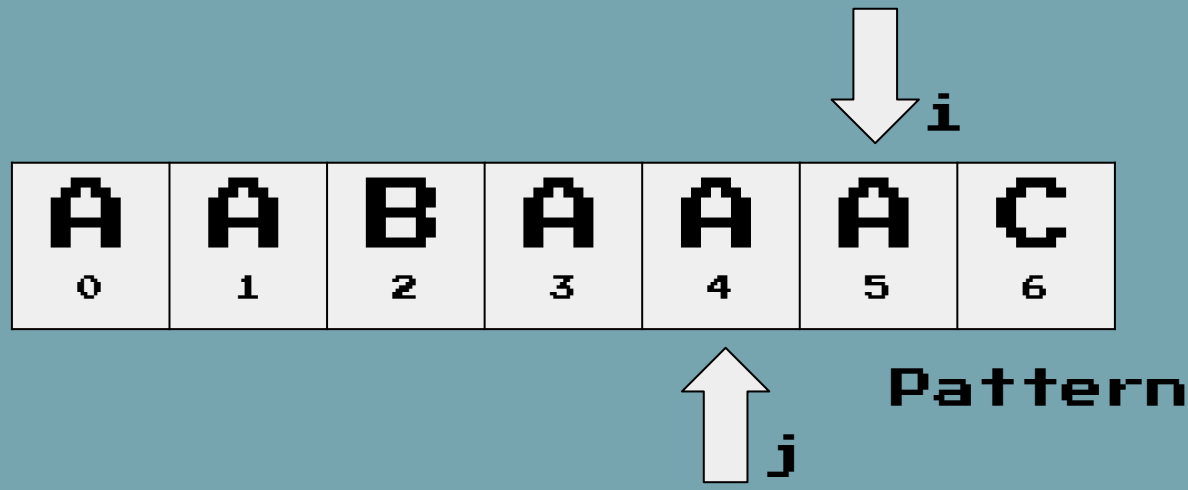
ANALYSIS

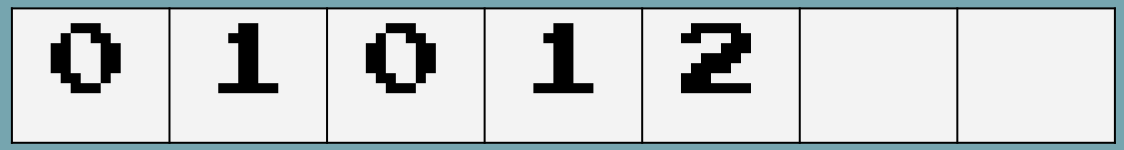


CODES

```python
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

ANALYSIS

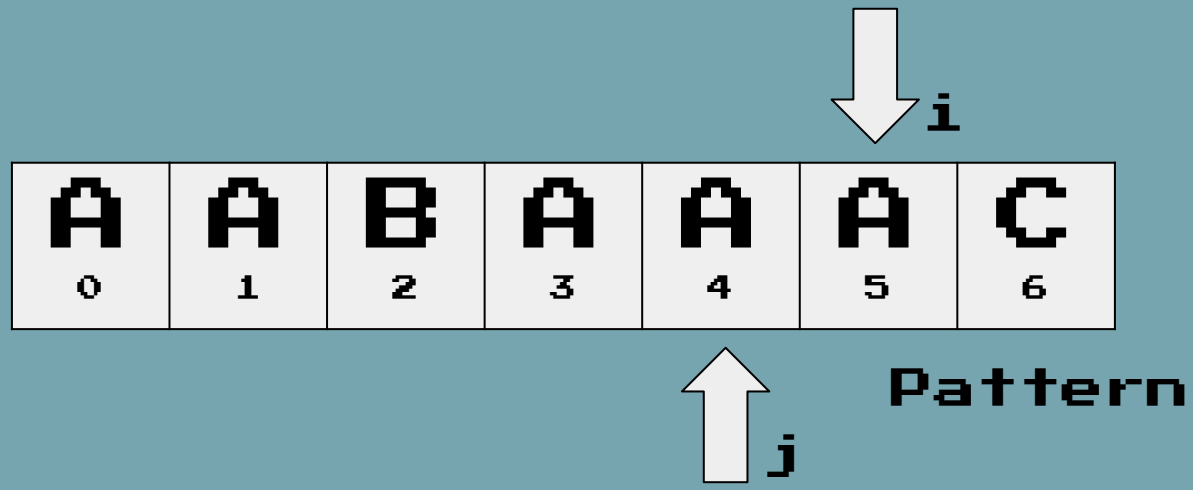




CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

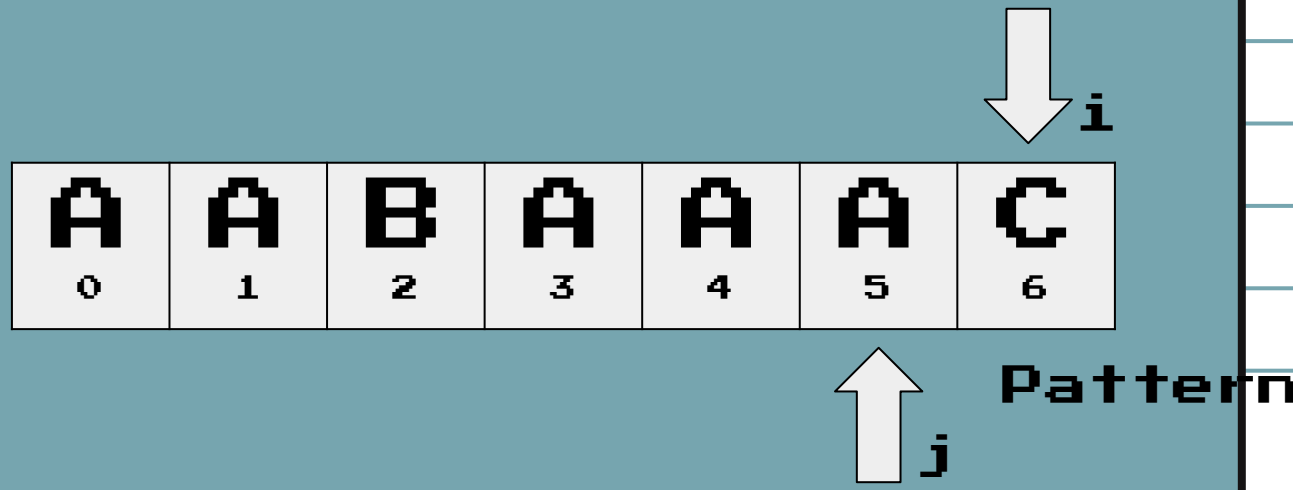
ANALYSIS



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

ANALYSIS



CODES

```
def π_table(self, pattern):

    n = len(pattern)
    lps = [0]

    for i in range(1, n):
        j = lps[i-1]

        while j>0 and pattern[j] != pattern[i]:
            j = lps[j-1]

        if pattern[j] == pattern[i]:
            lps.append(j+1)
        else:
            lps.append(j)

    return lps
```

ANALYSIS

A	A	B	A	A	A	C
0	1	2	3	4	5	6

Pattern

0	1	0	1	2	2	0

LPS

ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

i

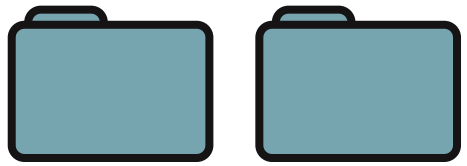
Two pointers pointing at two different arrays

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

i

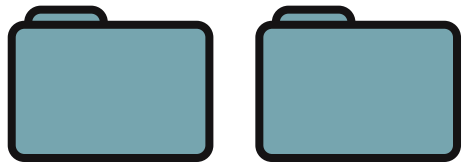
To check if:
text[i] == pattern[j+1]

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

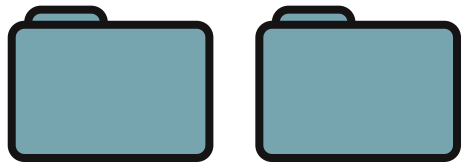
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

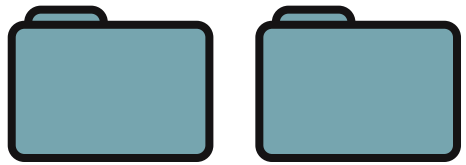
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

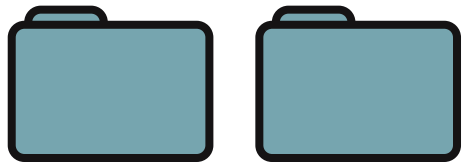
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

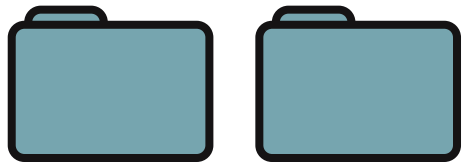
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

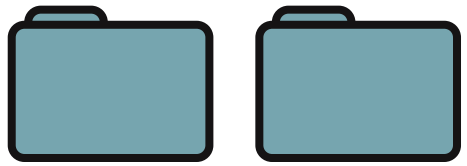
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

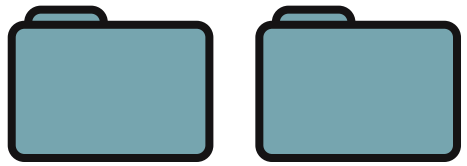
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

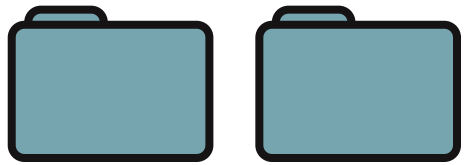
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

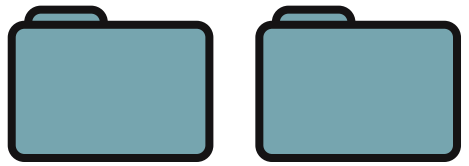
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

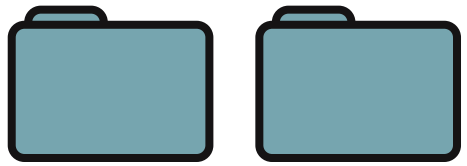
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

i

i moves 2 steps forwards from where it initially was

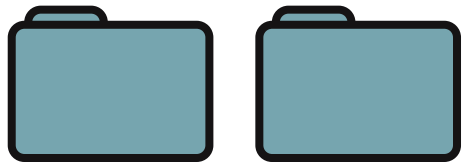
j moves 2 steps backwards

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

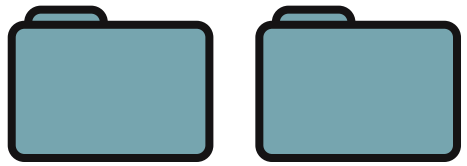
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

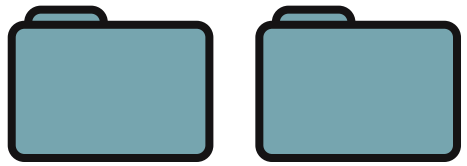
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

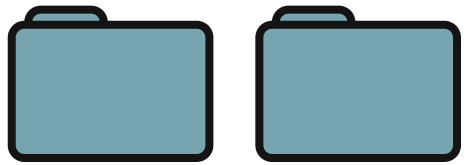
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

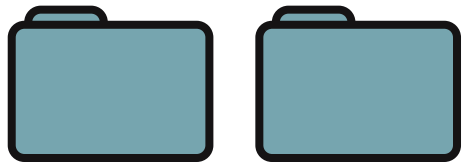
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

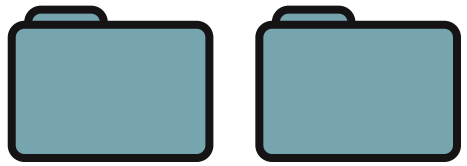
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

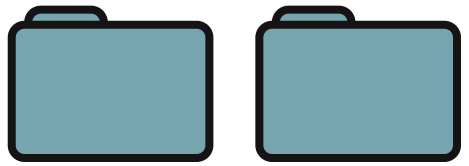
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

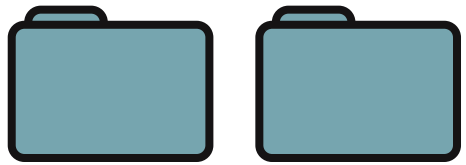
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

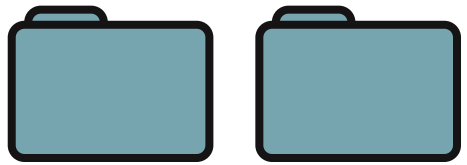
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

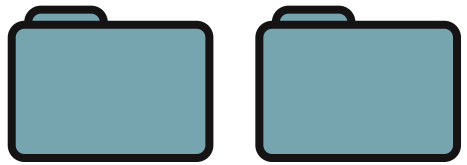
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



ANALYSIS

0	1	2	3	4	5	6	7	8	9	10
A	B	A	B	A	B	A	B	B	A	C

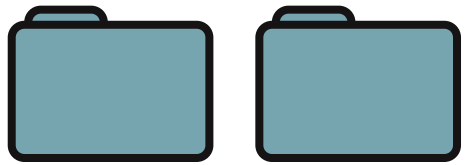
i

j

Pat.		A	B	A	B	B	A	C
Pos.	0	1	2	3	4	5	6	7
LPS		0	0	1	2	0	1	0

CODES

```python
def knuth_morris_pratt(self, text, pattern):

    lps = self.π_table(pattern)
    result, j = [], 0
    m, n = len(text), len(pattern)

    for i in range(m):
        while j>0 and text[i] != pattern[j]:
            j = lps[j-1]

        if text[i] == pattern[j]:
            j += 1

        if j == n:
            result.append((i-j)+1)
            j = 0
```



TIME COMPLEXITY

Building LPS array:

time complexity = O(m)

Searching:

comparison is a multiple of n, time complexity = O(n)

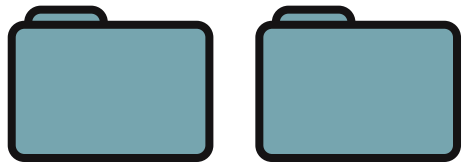
Total time complexity: O(n+m)

- lps array is **NOT actually** linear
- the pattern is usually short

Overall (searching)=> O(n+m)

=> O(n)

for n>>m



TIME COMPLEXITY

Best case: each time it shifts a suffix could be found

Virus genome: A C T G G A C T G G A C T G G A C T ...

Target genome: A C T G G A C G

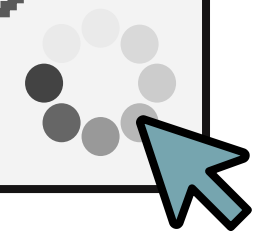
Best case time complexity = O(n)

TIME COMPLEXITY

Worst case: All the characters in the pattern are distinct. BUT...

pat	T	A	C	G
pos	0	1	2	3
lps	0	0	0	0

Worst case time complexity = O(n)

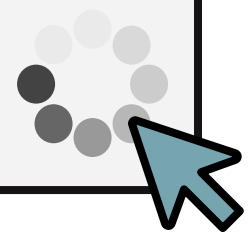


TIME COMPLEXITY

Best case: O(n)

Worst case: O(n)

Average case: O(n+m)



Boyer-Moore-Horspool (BMH)

Boyer -Moore -Horspool

- Search genome from back to front

```
current_index=len(self.genome)-len(target)

while(current_index>0):

    flag=1

    for i in range(len(target)):

        genome_character=self.genome[current_index+i]
        target_character=target[i]

        if genome_character!=target_character:
            flag=0
            break

    if flag==1:
        results.append(current_index)

    skip=skipper.get(self.genome[current_index],len(target))
    current_index-=skip
```

Boyer -Moore -Horspool

- Search genome from back to front
- Check pattern is from front to back

```
current_index=len(self.genome)-len(target)

while(current_index>0):

    flag=1

    for i in range(len(target)):

        genome_character=self.genome[current_index+i]
        target_character=target[i]

        if genome_character!=target_character:
            flag=0
            break

    if flag==1:
        results.append(current_index)

    skip=skipper.get(self.genome[current_index],len(target))
    current_index-=skip
```

Boyer -Moore -Horspool

- Search genome from back to front
- Check pattern is from front to back
- Use 'bad match table'

  E.g. {'A':3,'T':2}

```
results=[]
skipper={c:len(target)-i-1 for i,c in enumerate(target[:0:-1])}
current_index=len(self.genome)-len(target)

while(current_index>0):

    flag=1

    for i in range(len(target)):

        genome_character=self.genome[current_index+i]
        target_character=target[i]

        if genome_character!=target_character:
            flag=0
            break

    if flag==1:
        results.append(current_index)

    skip=skipper.get(self.genome[current_index],len(target))
    current_index-=skip
```

Boyer -Moore -Horspool

- Search genome from back to front
- Check pattern is from front to back
- Use 'bad match table'

  E.g. {'A':3,'T':2}

- Skips by len(pattern) otherwise

```
results=[]
skipper={c:len(target)-i-1 for i,c in enumerate(target[:0:-1])}
current_index=len(self.genome)-len(target)

while(current_index>0):

    flag=1

    for i in range(len(target)):

        genome_character=self.genome[current_index+i]
        target_character=target[i]

        if genome_character!=target_character:
            flag=0
            break

    if flag==1:
        results.append(current_index)

    skip=skipper.get(self.genome[current_index],len(target))
    current_index-=skip
```

How the Bad Match Table is made?

- Ignore first character
- Based on first occurrence of a character

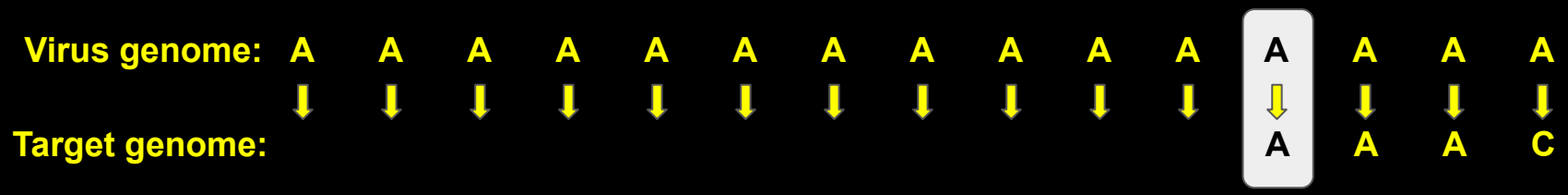
E.g. T A A A G

0 1 2 3 4

{'A':1,'G':4}

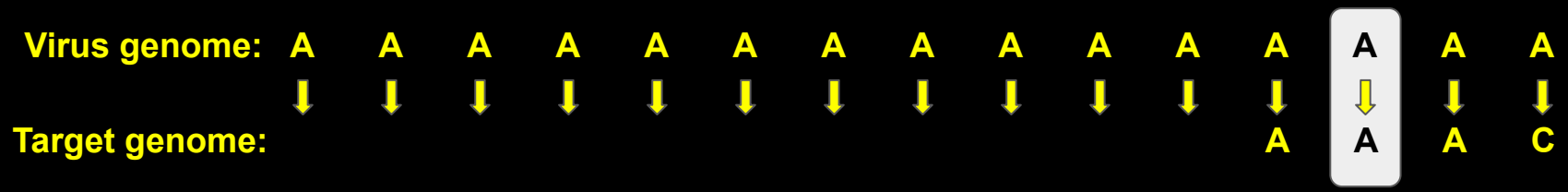
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



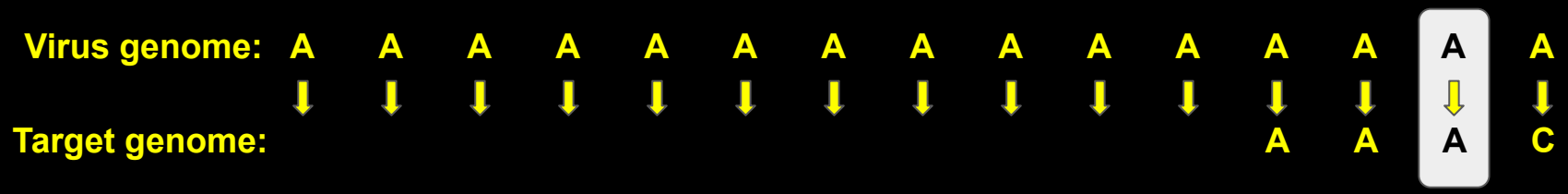
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



Time complexity: Worst case

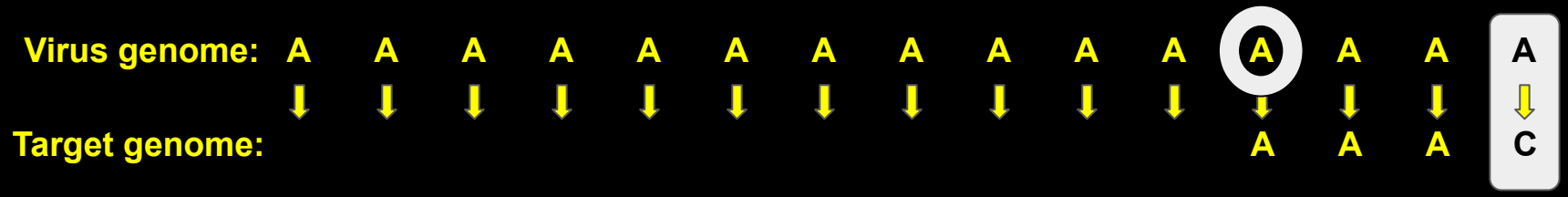
```
Dictionary = { A: 1,
               C: 3}
```



Time complexity: Worst case

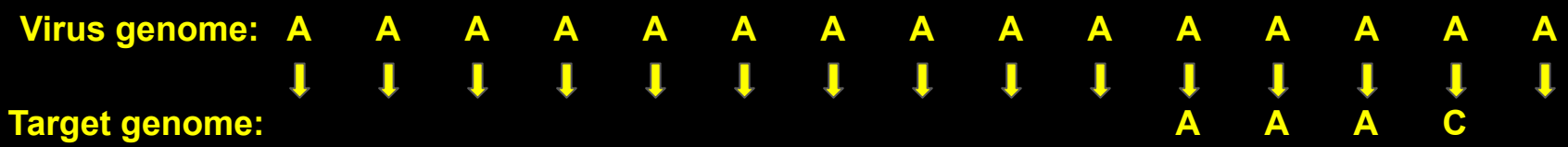
```
Dictionary = { A: 1,
               C: 3}
```

MISMATCH!!



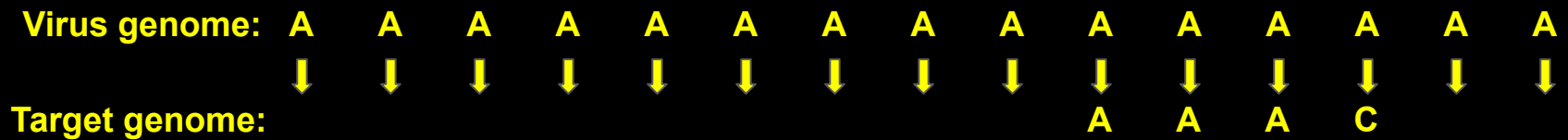
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



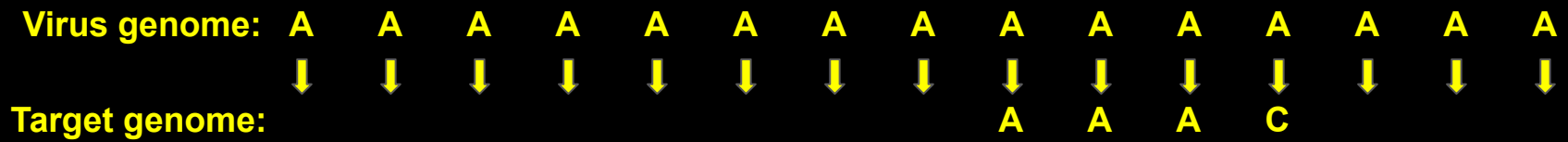
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



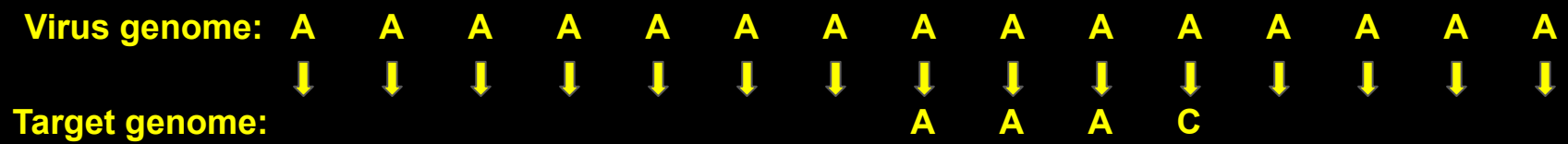
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



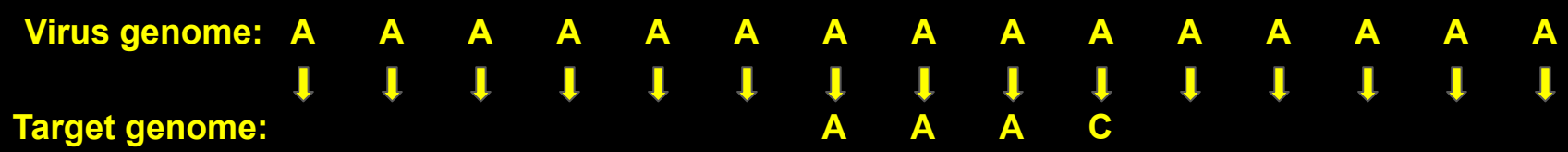
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



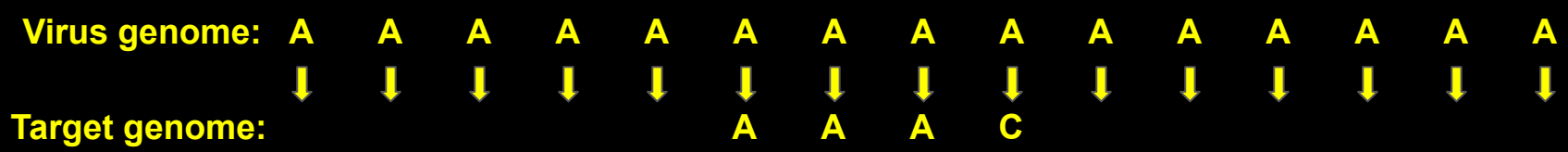
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



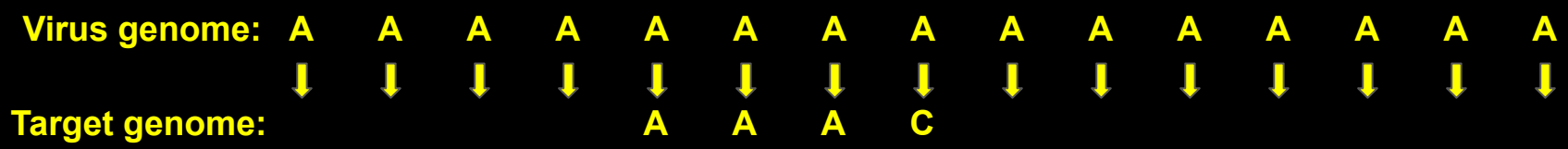
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



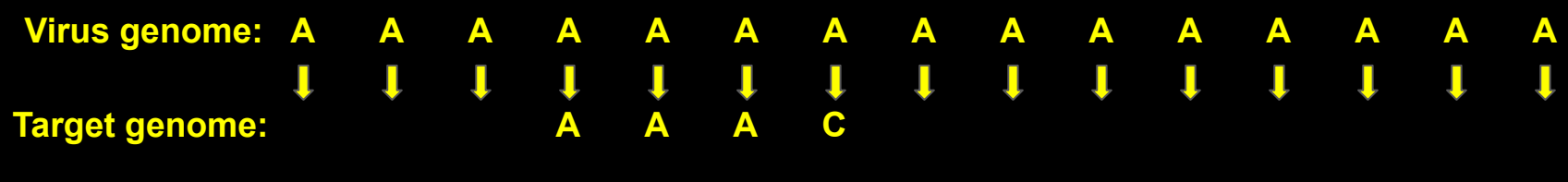
Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```



Time complexity: Worst case

```
Dictionary = { A: 1,
               C: 3}
```

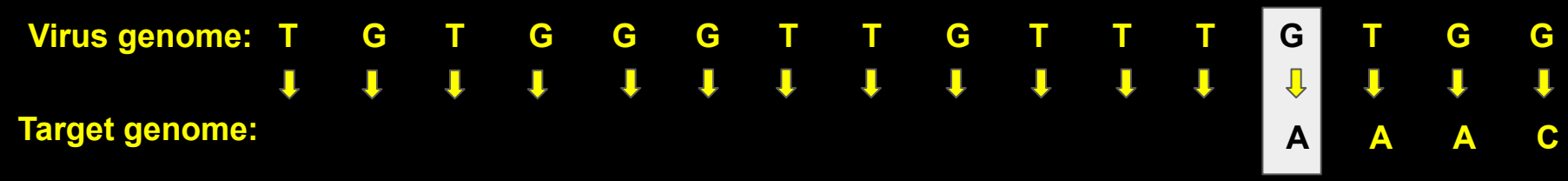


Worst Case: O(nm)

Time complexity: Best case

```
Dictionary = { A: 1,
               C: 3}
```

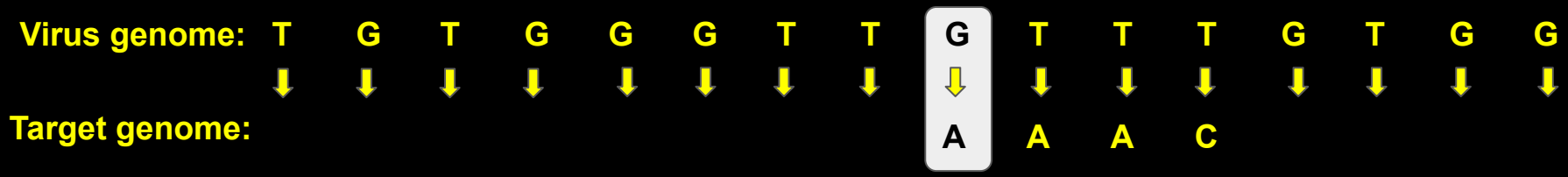
NOT FOUND!



Time complexity: Best case

```
Dictionary = { A: 1,
               C: 3}
```

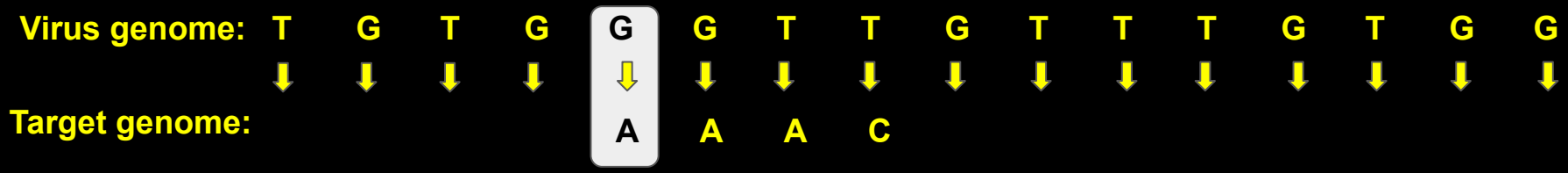
NOT FOUND!



Time complexity: Best case

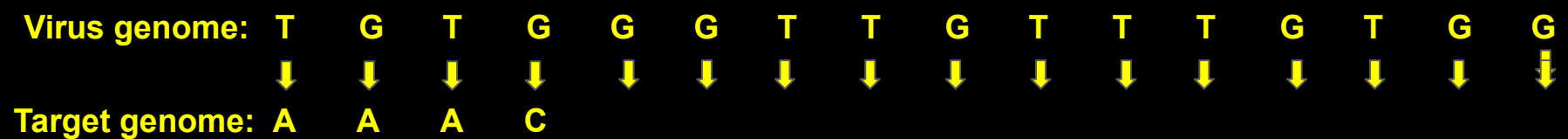
```
Dictionary = { A: 1,
               C: 3}
```

NOT FOUND!



Time complexity: Best case

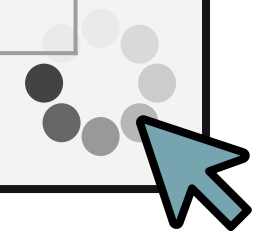
Dictionary = { A: 1,
C: 3}



Best Case: O(n)

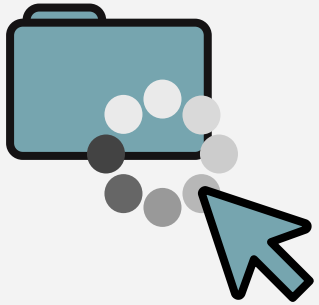
Conclusion

Algorithm	Preprocessing Time	Matching Time
Naive	0	O(nm)
KMP	O(m)	O(n)
BMH	O(m)	O(nm)



BEST ALGORITHM?

Ideas from **Knuth_Morris_Pratt** Algorithm



However, Comparing the three

Naive Algorithm

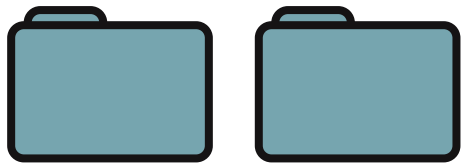
- Easy to think of
- Easy for beginners to implement
- Inefficient

KMP Algorithm

- LPS is not applicable for every case
- Code and concept is not easy to think of

BMH Algorithm

- Simple to code, concept based on index of character
- Basically brute force in worst case



Thank you!