34) Comparison count sort & distribution count sort

Input enhancement → we need extra space

→ The idea is to preprocess the problem's input, in whole or a part & store the additional information obtained to accelerate solving the problem afterwards

1) Sorting - counting ⟨ Comparison counting sort
                      ⟨ Distribution counting sort

2) String matching → Horspool's
                     Boyer Moore's

Comparison Count Sort

Input Array A[0 ... 5] → 62 31 84 96 19 47

	62	31	84	96	19	47
	0	1	2	3	4	5
Count	0	0	0	0	0	0
Count 1	3	0	1	1	0	0
Count 2	3	1	2	2	0	1
Count 3	3	1	4	3	0	1
Count 4	3	1	4	5	0	1
Count 5	3	1	4	5	0	2
Count 6						

Count	3	1	4	5	0	2

62 31 84 96 19 47

↓

S

	0	1	2	3	4	5
Final Sorted array	19	31	47	62	84	96

Algorithm ComparisonCountSort(A[0...n-1])
// Sorts an array by comparison counting
// Input: Array A Output: Array S in increasing order

for i ← 0 to n-1 do count[i] ← 0

for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[i] < A[j]
            count[j] ← 1
        else
            count[j] ← 1

for i ← 0 to n-1 do S[count] ← A[i]

Return S

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} n-1-i$$

$$= (n-1)^2 - \frac{(n-1)(n)}{2}$$

$$= n^2 - 2n + 1 - n + 1$$

$$= n^2 - 3n + 2$$

$$= O(n^2)$$

$$= (n-1) + (n-2) + \cdots 1$$

$$= \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = n^2$$

Distribution Count Sort

	0	1	2	3	4	5
Input Array A[0..5] =	13	11	12	13	12	12

l ← lower (11)
u ← upper (13)

	11	12	13
Frequency	1	3	2
Distribution value	1	4	6

S =

0	1	2	3	4	5
11	12	12	12	13	13

→ is the sorted array

	11	12	13
12	1	4	6
12 →	1	3	6
13	1	2	6
12	1	2	5
11	1	1	5
13	0	1	5

Algorithm Distribution Count Sort (A[0..n-1], l, u)

// Sorts an array of integers from a limited range

// Input: Array A & range l & u

// Output: An array S

for j ← 0 to u-l do D[j] = 0

for i ← 0 to n-1 do D[A[i] - l] ← D[A[i] - l] + 1

for j ← 1 to u-l do D[j] ← D[j-1] + D[j]

for i ← n-1 down to 0 do

    j ← A[i] - l

    S[D[j] - 1] ← A[i]

    D[j] ← D[j] - 1

return S

Time Complexity :- O(n)

35 Horspool's String matching

→ Goal is to find the pattern (P) in the text (T)

→ Size of text = n characters T[0 ... n-1]
Size of pattern = m characters P[0 ... m-1]

→ In Brute Force (Naive) string matching limitation is that the shift size is ~~fixed~~ limited to 1.

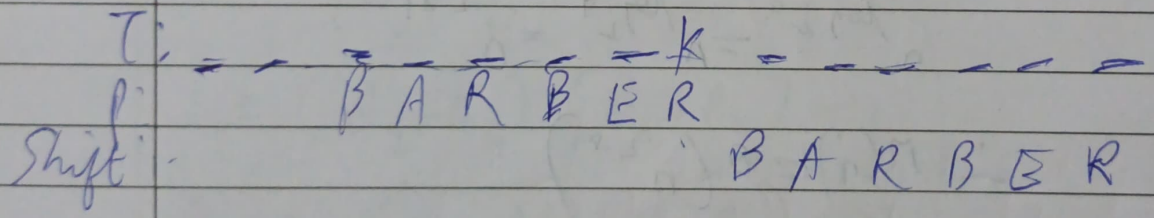
→ In Horspool algorithm, we do some sort of preprocessing to the pattern, before searching it in the text

→ In naive (Brute Force) string Matching → left to right comparison

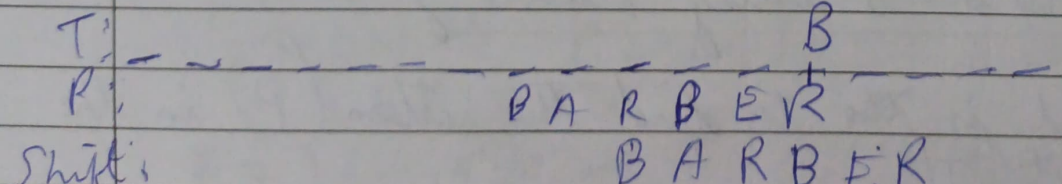
→ In Horspool string matching → Right to left comparison

Text size = n char
Pattern size = m char

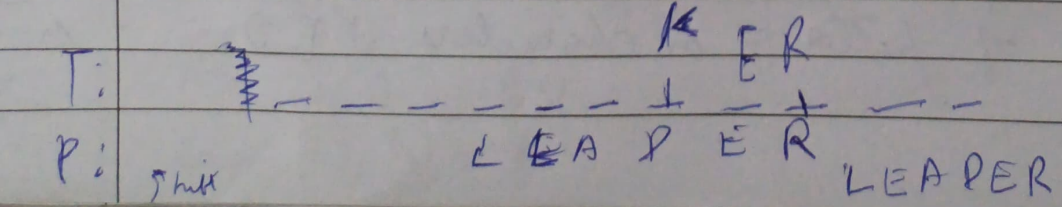
Case 1



Case 2



Case 3



Case 4

T: _ _ _ _ _ _ _ _ _ X E R

B A R B E R
B A R B E R

Shift table is calculated in Horspool for all the characters based on Pattern

Shift Value

$$t(c) = \begin{cases} \text{The pattern length } m \text{ if } c \text{ is not among the } (n-1) \text{ characters of the pattern} \\ \text{the distance from the rightmost } c \text{ among the } m-1 \text{ characters of the pattern to its last character, otherwise} \end{cases}$$

Pattern as BARBER

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
4	2	6	6	1	6	6	6	6	6	6	6	6	6	6	6	6	3

. . . . . X Y Z
6 6 1

Pattern as LEADER

L	E	A	D	E	R	others
5	1	3	2	1	0	6

Eg:-

Shift table:- B A R E
2 4 3 1

Text:- J I M _ S A W _ M E _ I N _ A _ B A R B E R _ S H O P

Pattern:- B A R B E R
B A R B E R
B A R B E R
B A R B E R
B A R B E R
B A R B E R

Eg 2)

Text:- F A I L _ M E A N S _ F I R S T _ A T T E M P T _ I N _ L E A R N

Pattern E A R N

Shift table:- E A R N Others
3 2 1 4

~~E A R N~~
~~E A~~

F A I L _ M E A N S _ F I R S T _ A T T E M P T _ I N _ L E A R N

E A R N
E A R N
E A R N
E A R N
E A R N
E A R N
E A R N
E A R N
E A R N

Algorithm ShiftTable(P[0 ... m-1])

// Fills the shift table used by Horspool Algorithm
// Input : Pattern P[0 ... m-1]
// Output : Table [0 ... size-1] indexed by the alphabet's characters and filled with shift size

Initialise all the elements of Table with m
for j ← 0 to m-2 do Table[P[j]] ← m-1-j
return Table

Algorithm Horspool(P[0 ... m-1], T[0 ... n-1])

// Implements Horspool's algorithm for string matching
// Output: The index of the left end of the first matching substring of -1 if there are no matches

ShiftTable(P)
i ← m-1
while i <= n-1 do
    k ← 0 // No. of matched characters
    while k <= m-1 and P[m-1-k] == T[i-k] do
        k ← k+1
    if k == m
        return i-m+1
    else
        i ← i + Table[T[i]]
return -1

Horspool's time complexity

Worst case = O(nm)
Avg case = O(n)

36) C Program to implement Horspool String matching

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 256
int Table[256]

void Shifttable(char P[25])
{
    int i, m;
    m = strlen(P);
    for(i=0; i<SIZE; i++)
        Table[i] = m;
    for(i=0; i<m-2; i++)
        Table[P[i]] = m-i-1;
}

int horspool(char T[100], char P[25])
{
    int m, n, k, i;
    m = strlen(P);
    n = strlen(T);
    i = m-1;
    while(i<=n-1)
```

```
    {
        k=0;
        while((k<m) && (P[m-1-k] == T[i-k]))
            k++;
        if k==m
            return i-m+1;
        else
            i = i + Table[T[i]]
    }
    return -1;
}

int main()
{
    char text[100], pattern[25];
    int res;
    printf("\n Read text \n");
    scanf("%[^\n]s", text);
    printf("\n Read pattern \n");
    scanf("%s", pattern);
    res = horspool(text, pattern);
    if (res == -1)
        printf("\n Pattern not found");
    else
        printf("\n Pattern found at position
        %d", res);
    return 0;
}
```

37) Boyer Moore's String Matching

Bad - Symbol Shift (d_1)

Text: _ _ _ _ _ _ _ _ _ _ S E R _ _ _

Pattern: B A R B E R k = 2

$d_1 = \max\{t(c) - k, 1\}$

$d_1 = t(c) - k$
$= 6 - 2 = 4$

Text: _ _ _ _ _ _ _ _ _ _ A E R _ _ _

Pattern: B A R B E R

$d_1 = t(A) - k$
$= 4 - 2 = 2$

Good Suffix Shift (d_2)

↓

is created based on the pattern

Pattern is ABCBAB

i	Pattern	d_2
1	ABCBAB	2
2	ABCBAB	4
3	ABCBAB	4
4	ABCBAB	4
5		

Text: _ _ _ _ X B A B _ _ _ _

Pattern: _ _ D B C B A B

D B C B A B

Text: _ _ _ X B A B _ _ _ _

Pat: A B C B A B

A B C B A B

BAO BAB

K	Pattern	d_2
1	BAOBAB	2
2	BAO BAB	5
3	BAO BAB	5

Algorithm

1) For the given pattern and the alphabet used in both pattern and text, construct a bad symbol shift d as
$d_1 = t(i) - k$, where t is the shift Table & k is the no. of matching characters
If d_1 is negative, then shift size should be
$d_1 = \max\{t(i) - k, 1\}$

2) Using the patterns, construct the good Suffix shift d_2

3) Align the pattern against beginning of the text

4) Repeat these steps until a matching substring is found of the pattern reaches beyond the last character of text
Starting with the last character of the pattern, compare the corresponding characters of pattern & Text
If characters do not match after some comparison, the shift size d is calculated as
$$d = \begin{cases} d & \text{if } k = 0 \\ \max\{d_1, d_2\} & \text{if } k > 0 \end{cases}$$

Text: BESS_KNEW_ABOUT_BAOBAB

Pattern: BAOBAB

A	B	O	other } Shift table t
1	2	3	6

k	Patterns	d_2
1	BAOBAB	2
2	BAOBAB	5
3	BAOBDB	5

BESS_KNEW_ABOUT_BAOBAB

BAOBAB

$d_1 = t(k) - 0 = 6 - 0 = 6$

BAOBAB

$d_1 = 6 - 2 = 4$, $d_2 = 5$

BAOBAB

$d_1 = 6 - 1 = 5$, $d_2 = 5$

BAOBAB