DATE: / /
PAGE NO.:

Unit 3

Network flow problem → Maximum flow
→ Ford Fulkerson Method

Pattern Matching → Naive Method
→ Finite Automata
→ Boyer Moore
→ KMP Matcher
→ Rabin Karp.

- **Pattern Matching**

The pattern matching problem is related to locate all or some occurrences of given pattern string consisting of sequence of characters within a given text string.

Given a text string 'T' of length 'n' over an alphate Σ^* and a pattern string 'p' of length 'm'. The problem is to locate all or some occurence of 'p' in 'T'.

1. **Brute Force / Naive Method.**

The most simple approach for the string matching problem is Brute Force algorithm which is also known as Naive method.

The algorithm doesn't require any preprocessing on text or pattern.

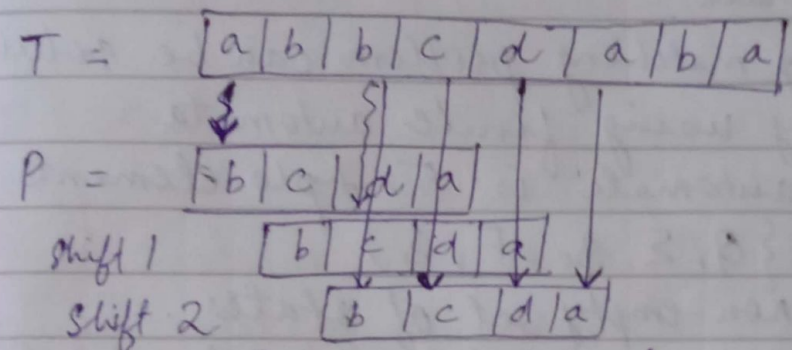
The basic idea behind it is that the pattern and text are compared character by character.

In case if a pattern is not matched, then pattern is shifted one position to right and process is repeated until a match is found or end of text is reached.

Example:

$T = \langle a\ b\ b\ c\ d\ a\ b\ a \rangle$

$P = \langle b\ c\ d\ a \rangle$



Pattern match found after 2 shifts.

Step 1 → Initialization loop:

```
set n → length [T]
set m → length [P]
for i ← 0 to n - m
set i ← 0
while (j < m and P[j] = T[i+j])
    set j ← j + 1
    if (j == m) then
        return i
```

Step 2 → return value at the point of all

```
return -1
```

The algorithm Brute force string matching scan the whole text string from left to right in order to search the exact pattern within the text string. In worst case, running time is

proportional to $\frac{n}{m}$.

where n and m are length of 2 strings.
Worst case time complexity is

$$\Theta[(n-m+1)m] \quad n \times m$$
$$\Theta(n^2)$$

(2) Finite Automata

The string matching problem can be solved more efficiently using finite automata.
A finite automata is 5-tuple elements

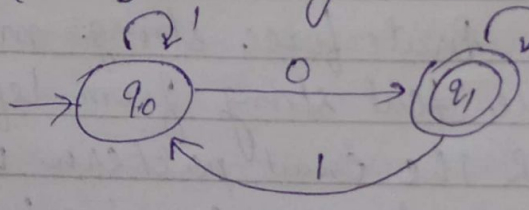
$$\{Q, \Sigma, q_0, F, \delta)$$

Q = non-empty set of states.
Σ = non-empty state of input alphabets
q_0 = initial state
F = final state
δ = transition function that maps

$$Q \times \Sigma \to Q.$$

DFA is Deterministic Finite Automata in which for a particular input character the machine goes to one state only.
Null move is not allowed in DFA.
DFA can't change state without any input character.

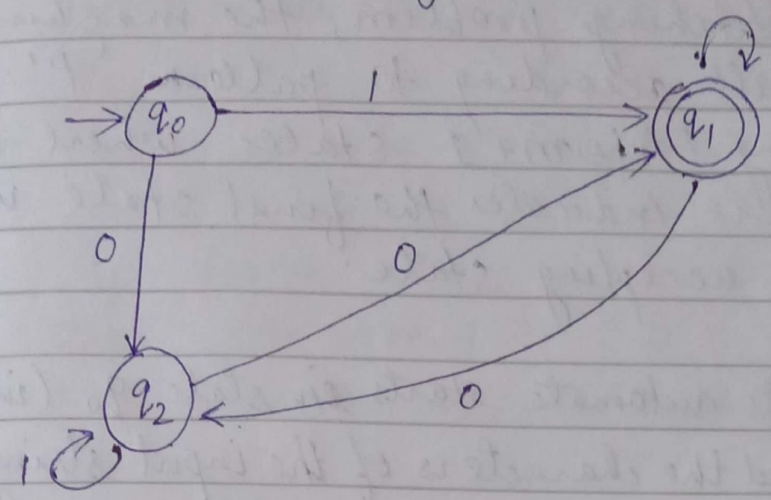
$\Sigma = \{0, 1\}$ string ends with 0's.



no. of state = 2.

DATE : / /
PAGE NO. :

- DFA to accept string with even number of 0's.



For using Finite automata as a solution to the string matching problem, the machine has to be built according to pattern 'P'. The resulting FA has m+1 states where the last state indicates the final state which shows accepting state.

The finite automata starts in state q_0 (initial state) and read the characters of the input string one at a time. If automata is in state q and reads input character 'a' it moves from state q to state $\delta(q, a) = q_1$. Whenever its current state q is a member of 'A', the machine M has accepted the string read so far. An input that is not allowed is rejected. A Finite automata M includes a function ϕ called the final state function from Σ^* to Q such that:

$\phi(w)$ is the state in M ends up in after scanning string W.

Thus M accepts a string W if and only if $\phi(w) \in A$.

The function ϕ is defined as

$$\phi(\epsilon) = q_0$$

$$\phi(wa) = \delta(\phi(w), a) \text{ for } w \in \Sigma^*, a \in \Sigma$$

DATE : / /
PAGE NO. :

FINITE AUTOMATA MATCHER (T, δ, m)

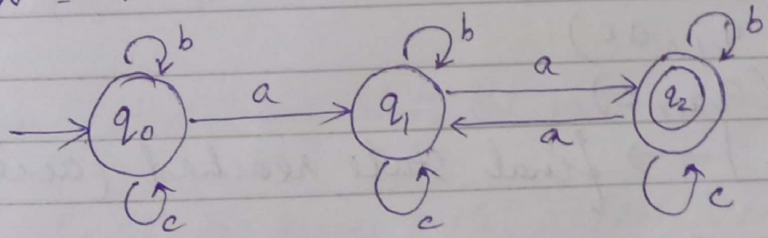
1. $n \leftarrow$ length $[T]$
2. $q \leftarrow 0$
3. for $i \leftarrow 1$ to n
4. do $q \leftarrow \delta(-q, T[i])$
5. if $q = m$
6. then $s \leftarrow i - m$
7. print " pattern occurs with shift s"
   return s

$O(n)$ for finite automata matcher

COMPUTE TRANSITION - FUNCTION (P, Σ)

1. $m \leftarrow$ length $[P]$
2. for $q \leftarrow 0$ to m
3. do for each character $a \in \Sigma^*$
4. do $k \leftarrow \min(m+1, q+2)$
5. repeat $k \leftarrow k+1$
6. until
7. $\delta(q, a) \leftarrow k$
8. return δ.

- $\Sigma = \{a, b, c\}$
  $W = bc\,aa\,bc\,aaa\,ba\,c$



initial state = q_0
final state = q_2 (or acceptance state)

$\delta(q_0, a) = q_1$
$\delta(q_0, b) = q_0$
$\delta(q_0, c) = q_0$
$\delta(q_1, a) = q_2$
$\delta(q_1, b) = q_1$
$\delta(q_1, c) = q_1$
$\delta(q_2, a) = q_1$
$\delta(q_2, b) = q_2$
$\delta(q_2, c) = q_2$

$w = bcaabcaaabac$

$\Rightarrow \delta(q_0, \underline{b}caabcaaabac)$
$\Rightarrow \delta(q_0, caabcaaabac)$
$\Rightarrow \delta(q_0, aabcaaabac)$
$\Rightarrow \delta(q_1, abcaaabac)$
$\Rightarrow \delta(q_2, bcaaabac)$
$\Rightarrow \delta(q_2, caaabac)$
$\Rightarrow \delta(q_2, aaabac)$
$\Rightarrow \delta(q_1, aabac)$
$\Rightarrow \delta(q_2, abac)$
$\Rightarrow \delta(q_1, bac)$
$\Rightarrow \delta(q_1, ac)$
$\Rightarrow \delta(q_2, c)$
$\Rightarrow \boxed{q_2}$ → final state reached (accepted)

Q. $\Sigma = \{a, c, g\}$
Pattern = acacaga

DATE : / /
PAGE NO. :

8. Rabin Karp

The Rabin Karp algorithm involves both the step of pattern matching : preprocessing and Pattern matching :

For Rabin Karp algorithm we assume string character contains $\Sigma = \{0, 1, \ldots\}$

Thus, each character is a decimal digit for given pattern $P[1 \ldots m]$.

p denotes the decimal value for given text $T[1 \ldots n]$

t_s denotes the decimal value for length m substring $T[s+1 \ldots s+m]$ where $0 \leq s \leq n-m$

For given pattern, S is valid shift if and only if $p = t_s$ which means $p[1 \ldots m] = T[s+1 \ldots s+m]$

p is calculated using Horner's rule as

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \ldots + 10(P[1])))$$

t_{s+1} is calculated using t_s as :

$$t_{s+1} = 10(t_s - 10^{m-1} T[s+1] + T[s+m+1])$$

- t_{s+1} calculations shift the pattern by 1 digit.

Substracting the term $10^{m-1} T[s+1]$ removes the higher order digit from $T[s]$ and multiplying it by 10 shift it one position towards left.

- Adding the term $T[s+m+1]$ bring lower order digit to the number. But the difficulty of this method is the

value of p and t_s may be too large to compute. So the improvement in this method generate p and t_{s+1} calculation as →

$d = 10$ (decimal digit)

$p = P \bmod q$

$h = d^{m-1} \pmod q$

higher order digit $= d^{m-1}$

$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

If $t_s = p$, it doesn't assure that it is valid shift s but it will definitely rule out invalid shift s.

Problem: $T = \langle 2,3,5,9,0,2,3,1,4,1,5,2,6,7,3,9,9,2,1 \rangle$

$P = \langle 3,1,4,1,5 \rangle$

$q = 13$

$m = 5$

$h = d^{m-1} = 10^4 = 10000$

$p = P \bmod q$

$= 31415 \bmod 13$

$\boxed{p = 7}$

(i) $T = \boxed{2|3|5|9|0|2}\ldots$

$P = \boxed{3|1|4|1|5}$

$s = 0$

$t_s = 23590 \bmod 13$

$\boxed{t_s = 8}$

$\boxed{t_s \neq p}$

(ii) T = [3|5|9|0|2|3 ...
P = [3|1|4|1|5]

s = 1

$t_{s+1} = [10[23590 - 2\times10000] + 2] \bmod 13$

$t_2 = 35902 \bmod 13$.

$t_2 = 9 \neq 7$ [shift R]

(iii) T = [5|9|0|2|3|1 ...
P = [3|1|4|1|5]

s = 2

$t_3 = [10\times[35902 - 3\times10000] + 3] \bmod 13$.

$t_3 = 59023 \bmod 13$

$t_3 = 3 \neq 7$ (shift)

(iv) T = [9|0|2|3|1|4 ...
P = [3|1|4|1|5]

s = 3

$t_4 = [10 \times [59023 - 5\times10000) + 1] \bmod 13$

$= 90231 \bmod 13 = 11 \neq 7$ (shift)

(v) T = [0|2|3|1|4|1 ...
P = [3|1|4|1|5]

s = 4

$t_5 = [10 \times [90231 - 9\times10000] + 4] \bmod 13$

$= 2314 \bmod 13$

$= 0 \neq 7$ (shift

(vi) T = [2|3|1|4|1|5 ...
P = [3|1|4|1|5]

s = 5

$t_6 = [10 \times [02314 - 0\times10000] + 1] \bmod 13$

$= 23141 \bmod 13 = 1 \neq 7$ (shift)

(vii)

T =	3	1	4	1	5	2
P =	3	1	4	1	5	

$s = 6$

$t_7 = [10(23141 - 2 \times 10000) + 5] \bmod 13$

$= 31415 \bmod 13$

$= 7$

$\boxed{t_7 = p}$

→ Pattern match found after 6 shift.

Q. $T = \langle 3\,1\,4\,1\,5\,9\,2\,6\,5\,3\,5\,8\,9\,7\,9\,3 \rangle$

$P = \langle 2\,6 \rangle$

$\bmod q = 11$

Q. $T = \langle 2\,3\,9\,5\,6\,2\,10\,9\,2\,6\,5\,1\,7\,9 \rangle$

$P = \langle 6\,2\,10 \rangle$

$\bmod q = 7$

DATE : / /
PAGE NO. :

4° KMP Matcher

[Knuth - Morris - Pratt]

In Naive algorithm, when there is mismatch between text and pattern then shifting by a single character in this situation is useless by most of the time.

KMP algorithm avoids testing of such useless shift. For this purpose, an auxillary function called prefix function for a pattern denoted by 'Π' is computed.

Compute Prefix function.

① Compute the length of P
   m ← Length [P]

② First value of prefix function would be 0
   So Π[1] ← 0
      k ← 0

③ for q ← 2 to m
   do while k > 0 & P[k+1] ≠ P[q]
      do k ← Π[k] // mismatch so k is decremented
   if P[k+1] = P[q]
      then k = k+1
      Π[q] ← k

④ Return Π function.

KMP - Matcher.

① n ← Length [T], m ← Length [P]
② Π ← Compute Prefix function for P.
③ set q ← 0
④ Compare text with Pattern P (char by char. Left to right)
   for i ← 1 to n

Do while $q > 0$ & $P[q+1] \neq T[i]$
do $q \leftarrow \pi[q]$
if $P[q+1] = T[i]$; then $q \leftarrow q+1$
if $q = m$ then
Print "Pattern occurs with shift"; $i-m$
$q \leftarrow \pi[q]$
⑤ Exit.

Problem

① $P = \langle ababababca \rangle$
$m = 10$
$\pi[1] = 0$ & $k = 0$

② for $q = 2$ to 10.
$q=2$ $P[k+1] = P[1] = a$
$P[q] = b$
$a \neq b$
$\pi[2] = 0$

③ $q=3$ $P[k+1] = P[1] = a$
$P[q] = a$
$a = a$ $k=1$
$\pi[3] = 1$

④ $q=4$ $P[k+1] = P[2] = b$
$P[q] = P[4] = b$
$b = b$ $k = 2$
$\pi[4] = 2$

⑤ $q=5$ $P[k+1] = P[3] = a$
$P[q] = P[5] = a$
$a = a$ $k = 3$
$\pi[5] = 3$

DATE : / /
PAGE NO. :

⑥ $q=6$ $P[k+1] = P[4] = b$.
$P[q] = P[6] = b$.
$b = b$ $\boxed{k=4}$
$\pi[6] = 4$

⑦ $q=7$ $P[k+1] = P[5] = a$
$P[q] = P[7] = a$
$a = a$ $\boxed{k=5}$
$\pi[7] = 5$

⑧ $q=8$ $P[k+1] = P[6] = b$.
$P[q] = P[8] = b$.
$b = b$ $\boxed{k=6}$
$\pi[8] = 6$

⑨ $q=9$ $P[k+1] = P[7] = a$
$P[q] = P[9] = c$

$k>0$ & $a \neq c$
$k = \pi[k]$
$k = \pi[6] = 4$!

$k>0$ $P[k+1] \neq P[q]$
$P[5]$ $P[9]$
$a \neq c$

$k = \pi[4] = 2$

$k>0$ $P[k+1] \neq P[q]$
$a \neq c$

$k = \pi[2] = 0$

$k=0$ $P[k+1] \neq P[q]$
$a \neq c$

$\pi[q] = \pi[9] = k = 0$.

⑩ $q=10$ $P[k+1] = P[1] = a$
$P[q] = P[10] = a$
$a = a$ $\boxed{k=1}$ $\pi[10] = 1$

q	1	2	3	4	5	6	7	8	9	10
P[q]	a	b	a	b	a	b	a	b	c	a
π[q]	0	0	1	2	3	4	5	6	0	1

Q. $T = \langle b\ a\ c\ b\ a\ b\ a\ b\ a\ a\ b\ c\ b\ a\ b \rangle$

$P = \langle a\ b\ a\ b\ a\ a \rangle$

Prefix function (π):

① $m = 6$.

$k = 0$; $\pi[1] = 0$

② for $q = 2$ to $q = m$

$q = 2$; $P[k+1] = P[1] = a$

$P[q] = P[2] = b$

$a \neq b$;

$\pi[2] = k = 0$

③ $q = 3$; $P[k+1] = P[1] = a$

$P[q] = P[3] = a$

$a = a$; $k = k+1 = 1$

$\pi[3] = k = 1$

④ $q = 4$; $P[k+1] = P[2] = b$

$P[q] = P[4] = b$

$b = b$; $k = k+1 = 2$

$\pi[4] = k = 2$

⑤ $q = 5$; $P[k+1] = P[3] = a$

$P[q] = P[5] = a$; $k = k+1 = 3$

$\pi[5] = k = 3$

⑥ $q = 6$; $P[k+1] = P[4] = b$

$P[q] = a$;

$b \neq a$ and $k = 3 > 0$

$k = \pi[k] = \pi[3] = 1$

$P[k+1] = P[2] = b$

$P[q] = a$; $b \neq a$ and $k = 1 > 0$

DATE : / /
PAGE NO. :

$k = \pi[1] = 0$

$P[k+1] = P[1] = a$

$P[q] = P[6] = a$

$P[k+1] = P[q]$

$\Rightarrow k = k+1 = 1$

$\underline{\pi[6] = k = 1}$

q	1	2	3	4	5	6
P[q]	a	b	a	b	a	a
π[q]	0	0	1	2	3	1

Pattern Matcher.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T =	b	a	c	b	a	b	a	b	a	a	b	c	b	a	b

j	0	1	2	3	4	5	6
P =	a	b	a	b	a	a	

n = length [T] = 15

m = length [P] = 6

1. $\underline{i=1}, j=0$ $T[i] = T[1] = b$
   $P[j+1] = P[1] = a$
   $b \neq a$

2. $\underline{i=2}, j=0$ $T[i] = T[2] = a$, $P[j+1] = P[1] = a$ $\Big\}$ $\boxed{a = a}$ $j \to j{+}{+}$

3. $j=1, i=3$ $T[i] = c$, $P[j+1] = b$ $\Big\}$ $\boxed{c \neq b}$ $j = \pi[j] = \pi[1] = 0$

4. $i=3, j=0$ $T[i] = c$, $P[j+1] = a$ $\Big\}$ $\boxed{c \neq a}$

5. $i=4, j=0$ $T[i] = b$, $P[j+1] = a$ $\boxed{b \neq a}$

6. $i=5, j=0$ $T[i] = a$, $P[j+1] = a$ $\boxed{a = a}$ $j \to j{+}{+}$

7. $i=6, j=1$ $T[i] = b$, $P[j+1] = b$ $\boxed{b = b}$ $j \to j{+}{+}$

8. $i=7, j=2$ $T[7] = a$ $P[2+1] = a$
$a = a$ $j \to j+1$

9. $i=8, j=3$ $T[8] = b$ $P[3+1] = b$
$b = b$ $j \to j+1$

10. $i=9, j=4$ $T[9] = a$ $P[4+1] = a$
$a = a$ $j \to j+1$

11. $i=10, j=5$ $T[10] = a$ $P[5+1] = a$
$a = a$ $j \to j+1$

12. $j = 6 = m$

Pattern occur with shift = $i - m$
$= 10 - 6$
$= 4$ shifts

5. Boyer Moore Algorithm

It is very efficient pattern matching algorithm. This algorithm is often implemented in text editor and word processor for search and find operation.

A special feature of this algorithm is that it process the pattern to be searched from text before starting the search. The processing of pattern is independent of text.

Preprocessing of pattern includes calculations of a function based on same heuristic.

→ Bad character Heuristic
→ Good Suffix Heuristic.

DATE : / /
PAGE NO. :

- In Naive algorithm when there is a mismatch we shift the pattern by 1 position in right direction and start the matching after the new alignment.
- But in Boyer Moore algo, the number of shift is calculated using bad character shift and good suffix shift. The another feature is that after aligning the pattern under the text, matching is performed in reverse order (right to left)

- Bad Character Heuristic →

① T : _ _ _ _ _ _ _ _ [X]
P : A N P A N M A [N]

While performing matching, we found that very first character in text is mismatched with corresponding character in pattern. We can easily see that the character X is not available in pattern.
A character in text which causes mismatch while performing the reverse matching is called 'bad character'

In such case, it is not wise to shift the pattern by one position. So pattern can be shifted by its length (in this case, $m = 8$)

DATE : / /
PAGE NO. :

(2) T : _ _ _ _ _ _ [X] A N _ _
P : A N P A N M A N

In reverse matching, 2 characters have been matched and there is a bad character X. In this situation, number of shift = length of pattern − no. of matched characters. Here, it is 8 − 2 = 6 shifts.

(3) T : _ _ _ _ _ _ _ _ [A]
P : A N P A N M A N

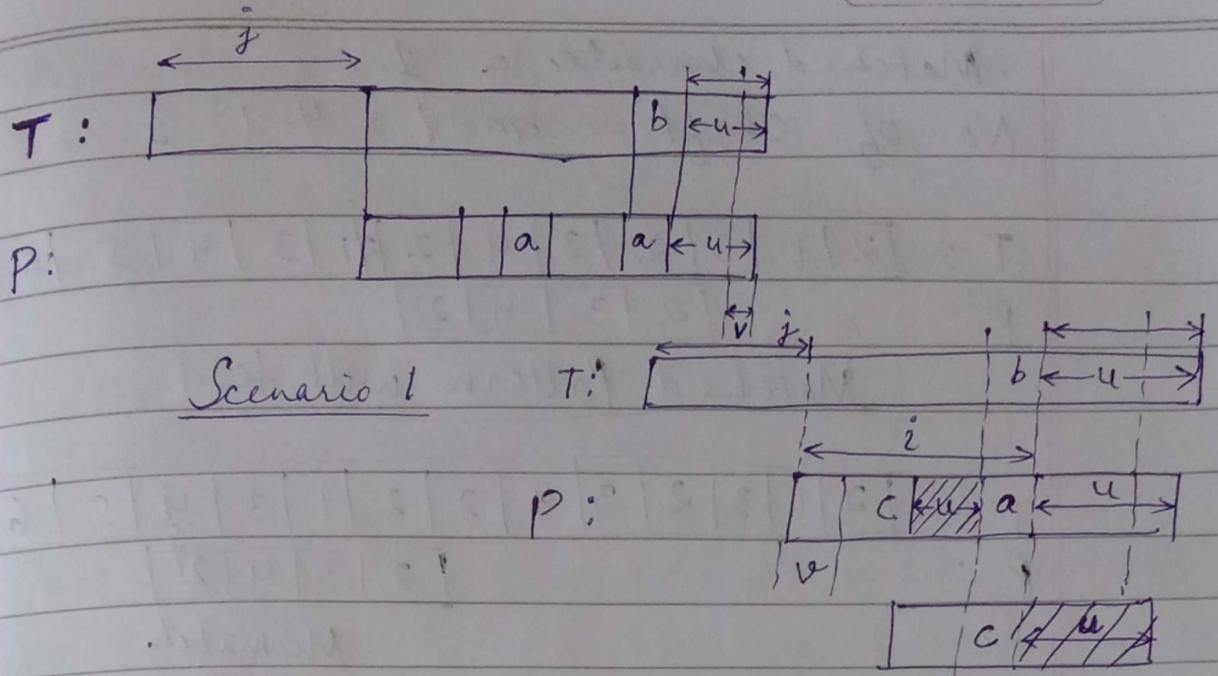
Here, A is bad character. In such case, we should shift the pattern in such a manner the rightmost occurrence of A in the pattern should be aligned below the bad character.

There will be a shift of 1 character in the given case :

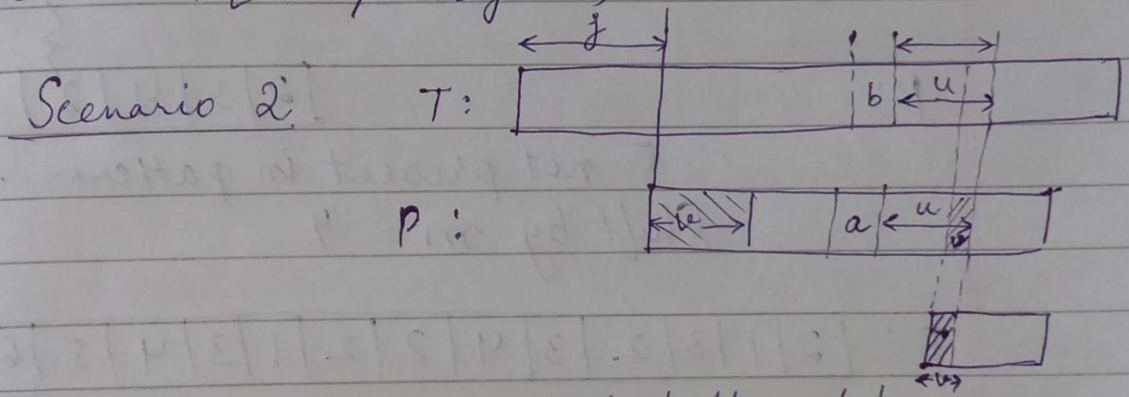
T : _ _ _ _ _ _ _ _ [A]
P : A N P A N M A N

- Good Suffix Heuristic

Good Suffix heuristic state that if the pattern P contain occurence of u not followed by a, then rightmost occurrence of u not followed by a can be aligned under u in the text as shown in the diagram.



In this scenario, pattern is aligned the text after j shifts. While performing reverse matching, substring 'u' has been matched and the character $P[i] \neq T[j+i]$



Consider a case when u not followed by a as in scenario 1, is not available in pattern. In that case, the alignment shift can be done in the following manner.

Problem

T : <2 1 3 2 3 4 2 2 1 3 4 5 6 1>

P : <2 3 4 2>

T :	2	1	3	2	3	4	2	2	1	3	4	5	6	1
P :	2	3	4	2										

3 mismatch 4

(Bad character Heuristic shift)

Matched character = 1

No. of shift = $m - 1 = 4 - 1 = 3$

T:	2	1	3	2	3	4	2	2	1	3	4	5	6	1
P:			2	3	4	2								

Matched Pattern with text

T:	2	1	3	2	3	4	2	2	1	3	4	5	6	1
P:							2	3	4	2				

Mismatch.

Shift by 1 place and place 4 below 4.

T:	2	1	3	2	3	4	2	2	1	3	4	5	6	1
P:								2	3	4	2			

5 not present in pattern.

Shift by $m = 4$

T:	2	1	3	2	3	4	2	2	1	3	4	5	6	1
										2	3	4	2	

No more match found.

DATE: / /
PAGE NO.:

Network Flow Problem:

→ Connected directed graph
→ Maximum flow
↔ Ford Fulkerson Method.

Source → flow originate
Sink → consume

Flow Network can be represented as a directed graph. In the maximum flow problem, we have to compute the greatest rate at which material can be shifted from the source to the sink without violating any capacity constraint.

A flow network $G = (V, E)$ is a directed graph in which edge $(u, v) \in E$ has a non negative capacity $c(u, v) \geq 0$

Assume that there are 2 vertices in the flow network : a source vertex 'S' and a sink vertex 'T'. Every other vertex in the path, S between and T

A flow in the flow network G is a real valued function $f : V \times V \rightarrow R$ and it satisfies 3 properties :

① Capacity constraint
For all $(u, v) \in E$ we require $f(u, v) \leq c(u, v)$
Thus, the flow from 1 vertex to another must not exceed the given capacity.

② Skew Symmetry
For all $(u, v) \in E$, we require $f(u, v) = -f(v, u)$

It simply means that flow from a vertex u to v is the negative of flow in the reverse direction.

(c) Flow conservation

For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(u, v) = 0$$

total

It says that the flow out of vertex other than source or sink is zero.

Ford Fulkerson Method.

It computes the maximum flow in the given flow network. This method works behind 3 main ideas:

1. Residual Network.
2. Augumenting Path.
3. Cut

- Residual Network is a network consisting of edges that can augument more flow through the network. For a given flow network G with capacity C, source vertex S and sink T ~~and flow f~~, the residual capacity defined by a flow f is any additional capacity that f has not fully taken advantage of pushing its flow from S to T.

It is the amount of additional flow we can push from u to v before exceeding the capacity $C(u, v)$.

For edge (u, v) in graph G, residual capacity from u to v with respect to flow f denoted by

$$C_f(u, v) = c(u, v) - f(u, v)$$

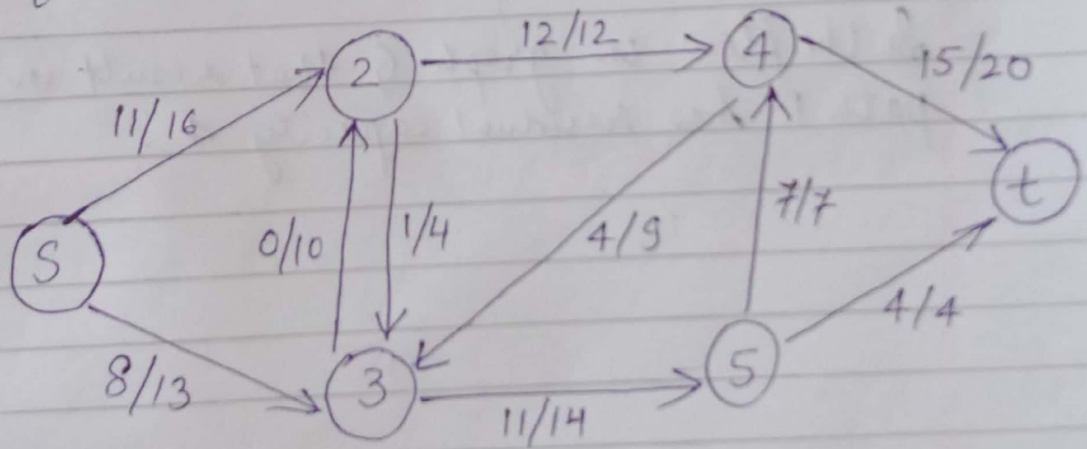
The residual capacity from v to u.

$$C_f(v, u) = f(u, v)$$

- **Augumenting Path**

For the given flow network $G(V, E)$ and a flow f, augumenting path P is the simple path S to T in the residual network G_f. Each edge (u, v) in the augumenting path puts some additional positive flow from u to v, without violating capacity constraint on the edge. The maximum amount by which we can increase the flow on each edge in an augumenting path P, the residual capacity of P is given by

$$C_f(P) = \min \{C_f(u, v) : (u, v) \text{ is on } p\}$$



DATE : / /
PAGE NO. :

Augumenting Path

$\Rightarrow \min \{ c_f(u, v) : (u, v) \text{ in } p \}$
$\Rightarrow \min \{ c_f(5,3), c_f(3,4), c_f(4,$
$\Rightarrow \min \{ 5, 4, 5 \}$
$= 4$

So the flow in graph G that result in augumenting path P has residual capacity 4.

DATE : / /
PAGE NO. :

Cut of flow network

Given flow network $G(V, E)$ where S is the source and T is the sink. cut (S, T) is the partition of V into S and T where $T = V - S$, $s \in S$, $t \in T$. The net flow $f(S,T)$ and capacity of cut $c(S, T)$ The net flow consists of both positive and negative flow from S to T are added, while flow from T to S are substracted. The net capacity consists only positive value for S to T.

$\{(s, 2, 3), (4, 5, t)\}$

Net flow $= f(2,4) + f(3,5) + f(3,4)$
$= 12 + 11 + (-4)$
$= 19$.

Net capacity $= c(2,4) + c(3,5)$
$= 12 + 14 = 26$.

Q.

(S)