(DAA:- Assignment)

Roll No:- 43 Name:- Parth Prasun Reg. No:- 22096228

Q2) Discuss the applicability of DFS in the following graph-related problems:-

a) Check for the cycles in the graph
b) Check if the given graph is bipartite or not
c) To compute transitive closure for a given sparse gr

⇒ DFS is a fundamental graph traversal algorithm that explores as far as possible before backtracking. This method is highly versatile and finds applications in numerous graph related problems.

a) Check for cycles in the given Graph:-
The process involves traversing the graph while marking the visited vertices, if during traversal if you encounter a vertex that is already visited and is not the direct parent of current vertex, a cycle is detected.
It works efficiently for both directed and undirected graph

Pseudo code:-

Undirected Graph	Directed Graph
DFS(vertex, parent)	DFS(vertex)
1) Mark vertex as visited.	1) Mark vertex as visited and mark in recursion stack.
2) For every adjacent vertex adj of vertex	2) For every adjacent vertex adj of vertex
a) If adj is not visited i) Recursive call DFS(adj, vertex) ii) If DFS(adj, vertex) → true cycle detected	a) If adj is not visited and DFS(adj) finds a cycle return true b) else if in recursion stack, return true
b) Else if adj is not the parent a cycle is found	3) Remove vertex from from rec stack
3) If no cycle is found, return false	4) Return false if no cycle

Parth Prasun

b) Check if the given graph is Bipartite

A bipartite graph is a graph whose vertices can be divided into 2 disjoint sets U & V such that every edge connects a vertex in U to one in V. DFS can be utilized by attempting to color the graph using 2 colo and no two adjacent vertices share the same color.

It starts at any random vertex and mark 1 color then it attempts to color all neighbouring vertices with opposite color and recursively applying this strategy.

Algorithm

1) use a color[] array which stores (0/1) for every denoting opposite colors.

ii) call the DFS function for every node

iii) If the node u hasn't been visited previously then assign !color[v] to color[u] and call DFS again to visit nodes connected to u.

iv) If at any point, color[u] is equal to color[v], then node is not bipartite

c) Transitive closure for a given sparse graph

The transitive closure of a graph is a measure of which vertices are reachable from other vertices through a path of directed edges. In other words for every pair of vertices (u,v) it determines if there is a path from u to v.

Parth Prasun

Algorithm:-

1) create a matrix tc[V][V] that would finally have the transitive closure of the given graph. Initialize all entries as 0.

11) Call DFS for every node of the graph to mark reachable vertices in tc[][]. In recursive calls to DFS, we don't call DFS for an adjacent vertex if it is already marked as reachable in tc[][].

111) It uses adjacency list to visit the neighbours.

Q1) Explain the concept of B-trees? State the properties of B-trees.

=) B tree is special type of self-balancing search tree in which each node can contain more than one key and can have more than 2 children. It is a generalized form of the BST. It is also known as a m-way tree.

The need for B-tree arose with the rise in the need for lesser time in accessing physical storage media like a hard disk. There was a need for such types of data structures that minimize the disk access.

Properties:-

1) For each node x the keys are stored in increasing order.

11) In each node, there is a boolean value x.leaf which is true if x is leaf.

Classmate
Date :
Page :

Parth Prasun

3) If n is the order of the tree, each internal node can contain at most $(n-1)$ keys along with a pointer to each child.

4) Each node except the root can have at most n children and at least $n/2$ children

5) All leaves are at the same level.

6.) The root has at least 2 children and contains min^n of 1 ke

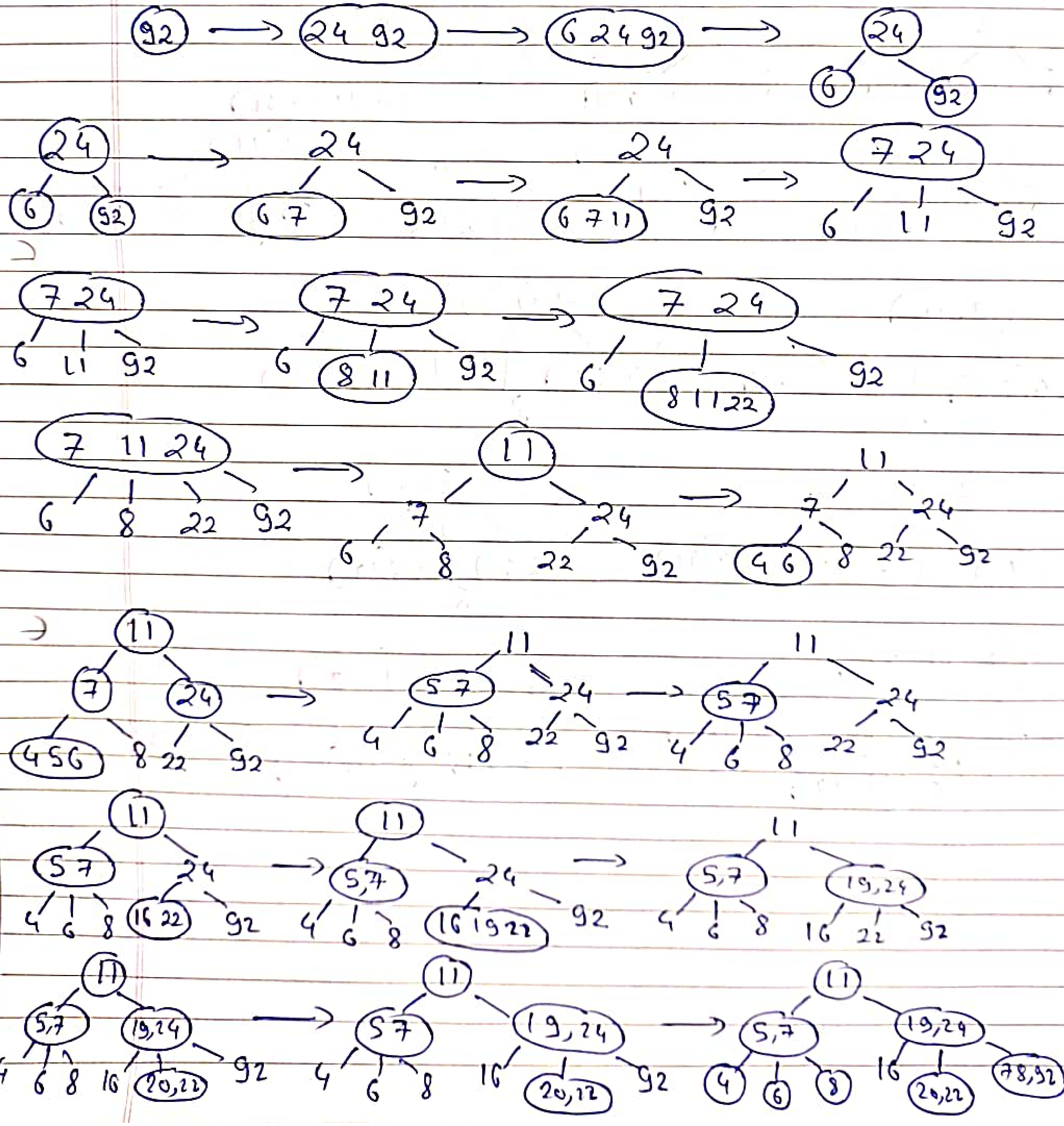
7) If $n \geq 1$ then for any n-key B-tree of height h and min^n degree $t \geq 2$ $h \geq \log_t (n+1)/2$

For order 3	For order 4
• Max no of keys $= m-1 = 3-1 = 2$	• Max no of keys $= m-1 = 4-1 = 3$
• Min no of keys $= 1$	• Min no of keys $= 1$
• Max no of children = order $= 3$	• Max no of children = order $= 4$
• Min no of children $= 2$	• Min no of children $= 2$

Parth Prasun

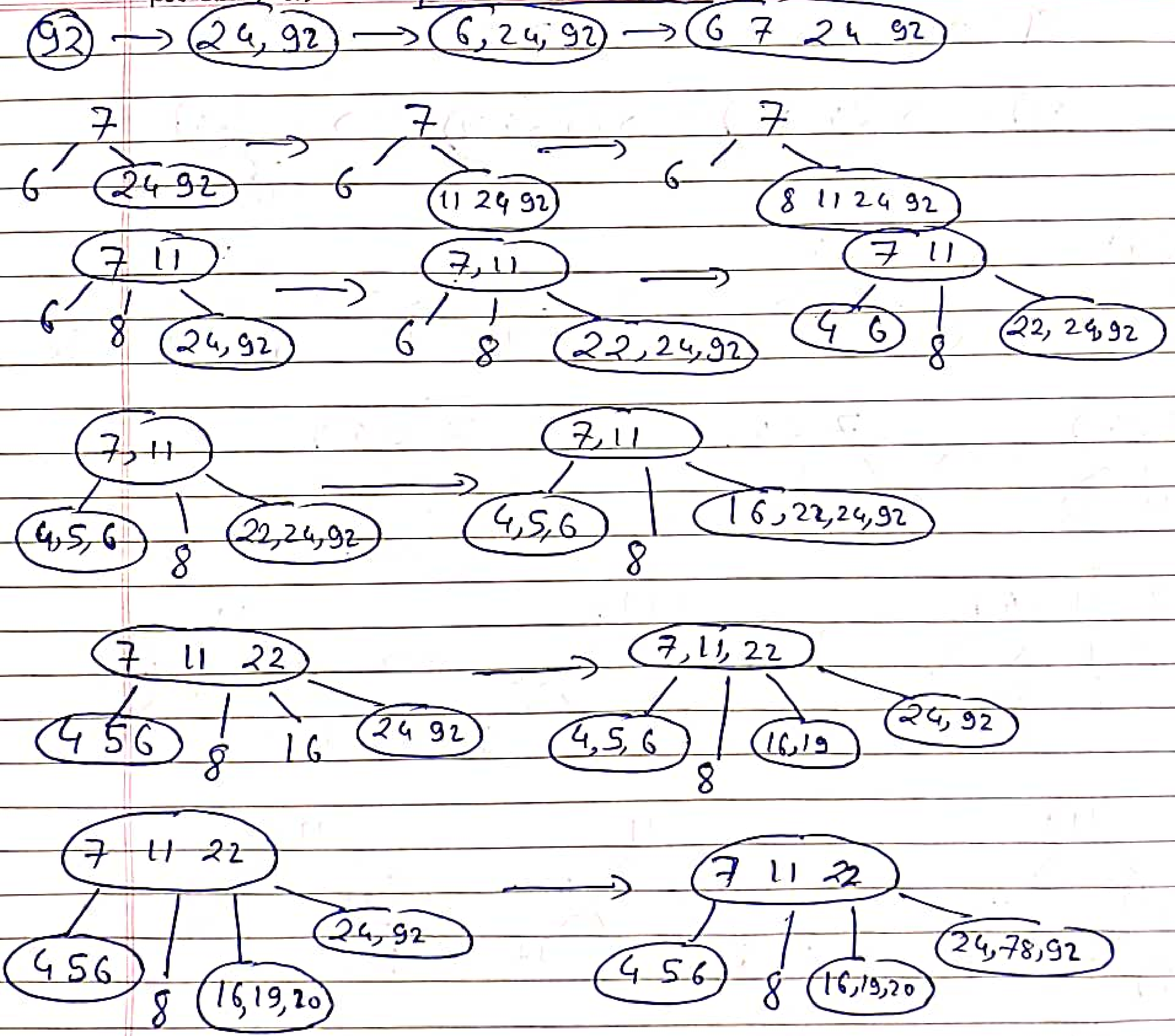
$\boxed{\text{Order-3 tree}}$

92, 24, 6, 7, 11, 8, 22, 4, 5, 16, 19, 20, 78



Parth Prasun

Order 4-tree



classmate
Date:
Page:

Q3) Write a divide and conquer algorithm that computes the leftmost minimum element in each row of an m×n Monge array A. Illustrate with an example.

⇒ A Monge array is 2-D array of numbers where for any elements A[i][j] and A[k][l] such that i<k and j<l, the following condition holds:-

$$A[i][j] + A[k][l] \le A[i][l] + A[k][j]$$

This property ensures that if we pick any four elements forming a submatrix such that two of them are diagonally opposite, the sum of the diagonal elements is less than or equal to the sum of other 2 elements. One interesting property of a Monge array is the leftmost minimum element in any row is no righter than the leftmost minᵐ element in row below it.

Algorithm

Initialization Step:-

1) Start with entire Array A as your search space. Define 4 variables to track the bounds of the current search space : top (init 0), bottom (init m-1), left (init 0), right (init n-1).

Recursive Divide and Conquer:-

1) Base Case : If the current search space is reduced to a single column (i.e ⇒ left == right) then the leftmost minᵐ element for each row in this space is simply the minᵐ element in this col for the rows b/w top and bottom. Find these and return as the result.

11) Dividing Step :- i) Find the middle column of the current search space :- mid_col = (left + right)/2

ii) Perform a linear search in this middle col to find the minm element b/w rows top and bottom. Let's denote the pos of minm element as (min_row, min_col).

Conquering and Combining :-

1) Left Subarray :- Recursively apply the algo the subarray the lies to left of 'mid_col' (b/w L & mid_col-1) for rows top through 'min_row This finds the leftmost minm elements for these rows in subarray.

2) Right Subarray :- Similar application to this array that lies to right of mid_col (b/w mid_col+1 and right) for rows 'min_row' through bottom. In the Monge property, the leftmost minm element in any row below 'min_row' can't be left ~~of~~ the col 'mid_col'.

3) Combining :- combine the results from the left and right subarrays with the minm element found in the middle column.

Example:-

$$B = \begin{matrix} 37 & 23 & 22 & 32 \\ 21 & 6 & 7 & 10 \\ 53 & 34 & 30 & 31 \\ 32 & 13 & 9 & 6 \\ 43 & 21 & 15 & 8 \end{matrix}$$

Condition for monge array:-

$$B[i,j] + B[i+1,j+1] \le B[i,j+1] + B[i+1,j] \quad \forall i = 1,2,3,4$$
$$\forall j = 1,2,3$$

→

$i=1, j=1$: $B[1,1] + B[2,2] \le B[1,2] + B[2,1] = 37+6 \le 23+21 = 43 \le 44$

$i=1, j=2$: $B[1,2] + B[2,3] \le B[1,3] + B[2,2] = 23+7 \le 22+6 = 30 \not\le 28$

$i=1, j=3$: $B[1,3] + B[2,4] \le B[1,4] + B[2,3] = 22+10 \le 32+7 = 32 \le 39$

$i=2, j=1$: $B[2,1] + B[3,2] \le B[2,2] + B[3,1] = 21+34 \le 6+53 = 55 \le 59$

$i=2, j=2$: $B[2,2] + B[3,3] \le B[2,3] + B[3,2] = 6+30 \le 7+34 = 36 \le 41$

All other cases validate the condtn, only ($i=1, j=2$) doesn't satisfy the constraint. We need to change one of these terms so that this holds true.

→

Reducing $B[1,2]$ by 2 will invalidate the case $i=1, j=1$ so we can't use this value. By reducing $B[2,3]$ by 2 will not invalidate any case. Thus by setting $B[2,3] = 5$ this array becomes Monge.

Q4) Explain the Knuth-Morris-Pratt (KMP) String Matching Algorithm with an example.

=> KMP algorithm revolutionized string matching by achieving linear time complexity O(n). A matching time of O(n) is achieved by avoiding comparison with an element of 'S' (string) that have previously been involved in comparison with some element of the pattern 'p' to be matched. By comparing characters sequentially and leveraging the preprocessed table, KMP minimizes backtracking, making it a cornerstone algorithm for efficient string-pattern recognition.

Components of KMP Algorithm:-

1) The Prefix Function (Π) :- The prefix function for a pattern encapsulates knowledge about how the pattern matches against the shift of itself. This information can be used to avoid a useless shift of pattern p. It enables avoiding backtracking of the string 'S'.

2) KMP Matcher:- With the String 'S' and pattern 'p' and prefix function as inputs, find the occurrence of 'p' in 'S' and returns the no of shifts of 'p' after which occurrences are found.

Pseudo code for Prefix Function Computation

1) $m \leftarrow length[p]$
2) $\pi[1] \leftarrow 0$
3) $K \leftarrow 0$

```
for q ← 2 to m
  do while k > 0 and P[k+1] ≠ P[q]
    do k ← π[k]
  If P[k+1] = P[q]
    then k ← k+1
  π[q] ← k
  return π
```

Pseudo code for KMP Matcher

```
n ← length [T]
m ← length [P]
π ← compute-prefix-function
q ← 0
for i ← 1 to n
  do while q > 0 and P[q+1] ≠ T[i]
    do q ← π[q]
   If P[q+1] = T[i]
     then q ← q+1
    If q = m
  then Print "Pattern occurs with shift" i-m
    q ← π[q]
```

Analysis of Time Complexity:-

LPS compute function is computed in linear T.C so we can say T.C is O(m).

Total time complexity of our KMP algorithm is O(n+m)

Example of KMP Algorithm

T:- b a c b a b a b a b a c a a b
P:- a b a b a c a

Prefix function for p

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

Soln:- Initially n = Size of T:- 15 , m = size of P = 7

Step 1:- i=1, q=0 comparing P[1], T[1]
T:- b a c b a b a b a b a c a a b
P:- a b a b a c a
P[1] ≠ T[1]
p shifted to one pos right.

Step 2:- i=2, q=0 comparing P[1], T[2]
T:- b a c b a b a b a b a c a a b
P:- a b a b a c a
P[1] = T[2], p not shifted

Step 3:- i=3, q=1 comparing P[2], T[3]
T:- b a c b a b a b a b a c a a b
P:- a b a b a c a
P[2] ≠ T[3]
Backtracking on P comparing P[1], T[3]

Step 4:- i=4, q=0 comparing P[1], T[4]
T:- b a c b a b a b a b a c a a b
P:- a b a b a c a
P[1] ≠ T[4]

Step 5:- i=5, q=0 P[1], T[5] comparison
T:- b a c b a b a b a b a c a a b
P:- a b a b a c a
P[1] = T[5]

Step 6:- $i = 6, q = 1$ $P[2]$ & $T[6]$

T:- b a c b a b a b a b a c a a b

P:- a b a b a c a $P[2] = T[6]$

Step 7:- $i = 7, q = 2$ $P[3]$ & $T[7]$, $P[3]$ matches with $T[7]$

Step 8:- $i = 8, q = 3$, $P[4]$ & $T[8]$, $P[4] = T[8]$

Step 9:- $i = 9, q = 4$ $P[5]$ & $T[9]$, $P[5] = T[9]$

Step 10:- $i = 10, q = 5$, $P[6]$ & $T[10]$, $P[6] \neq T[10]$

Backtracking on p, comparing $P[4]$ with $T[10]$ because after mismatch $q = \pi[5] = 3$

Step 11:- $i = 11, q = 4$. Comparing $P[5]$ & $T[11]$

T:- b a c b a b a b a b a c a a b

P:- ~~a b~~ a b a b a c a

$P[5] = T[11]$

Step 12:- $i = 12, q = 5$ Comparing $P[6]$ & $T[12]$, $P[6] = T[12]$

Step 13:- $i = 13, q = 6$ Comparing $P[7]$ & $T[13]$, $P[7] = T[13]$

Pattern P found in string T and total no of shifts $i - m = 13 - 7 = 6$ shifts