ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

A polynomial time algorithm to find number of paths in directed acyclic graph between given two vertices s and t.

Dr. Ishwar Baidari[1], Nagaraj Honnikoll[2]

Associate Professor, Computer Science Department, Karnataka University, Dharwad, India [1]

Computer Science Department, Karnataka University, Dharwad, India [2]

Abstract: The algorithm takes an input a directed acyclic graph G = (V, E) and two vertices s and t and returns the number of paths from s to t in G.

Keywords: directed acyclic graph, paths, queues.

I. INTRODUCTION

Graphs are a pervasive data structure in computer science, and algorithms for working with them are fundamental to the field. There are hundreds of interesting computational problems defined in terms of graphs.

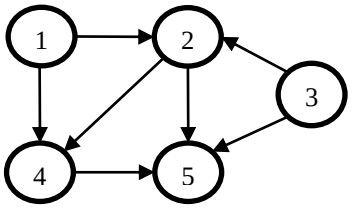
There are two standard ways to represent a graph G = (V, E) as a collection of adjacency list or as an adjacency matrix. Either way is applicable to both directed and undirected graphs.

A. Adjacency Matrix

For the adjacency matrix representation of a graph G = (V, E), we assume that the vertices are numbered 1, 2, . . . ,|V| in some arbitrary manner. Then the adjacency matrix representation of a graph G consists of |V| × |V| matrix A= $(a_{(i,j)})$ such that

$$a_{(i,j)} = \begin{cases} 1, & if (i,j) \in E, \\ x, & otherwise. \end{cases}$$

The adjacency matrix of a graph requires $O(V^2)$ memory, independent of the number of edges in the graph.



(a)

	1	**2**	**3**	**4**	**5**
1	0	1	0	1	0
2	0	0	0	1	1
3	0	1	0	0	1
4	0	0	0	0	1
5	0	0	0	0	0

(b)

Fig. 1. Representation of an directed graph. (a) An directed graph G having five vertices and seven edges. (b) The adjacency-matrix representation of G.

Path
A path is a walk in which all the edges and all the vertices are different.

B. Shortest Path

In a shortest path problem we are given a weighted directed graph G = (V, E), with weight function W: E→R mapping edges to real valued weights. The weight of path p = {V_0, V_1, . . . ,V_k} is the sum of the weights of its constituent edges.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

We define the shortest path weight from u to v by

$$\partial(u,v) = \begin{cases} \min\{w(p): u \rightsquigarrow v\}, if\ there\ is\ a\ path\ from\ u\ to\ v, \\ \infty, otherwise \end{cases}$$

A shortest path from vertex u to vertex v is then defined as any path p with weight w(p)= δ(u,v).

C. Variants of shortest Path Problem

Single Destination Shortest Path Problem: Finding a shortest path to a given destination vertex t from each vertex v.

Single Pair Shortest Path Problem: Finding a shortest path u to v for given vertices u and v. If we solve the single source problem with source vertex u, we solve this problem also.

All Pairs Shortest Path Problem: Finding a shortest path from u to v for every pair of vertices u and v. Although this problem can be solved by running a single source algorithm once for each vertex.

D. Multiple Queues using Single Array

We can implement multiple queues using single dimensional arrays. In a one dimensional array, multiple queues can be placed. Insertion from its rear end and deletion from its front end can be possible for desired queue. We can visualize this idea with the help of figure 2.

DOI 10.17148/IJARCCE.2015.41021

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

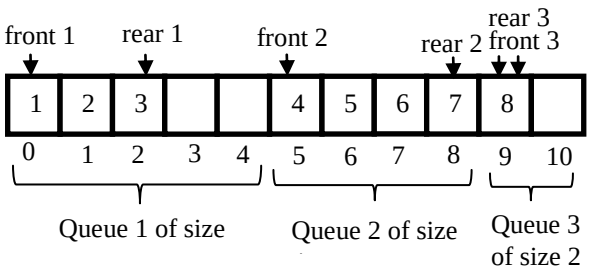


Fig. 2. Multiple queues using single array.

As shown in the Fig. 2, there are three queues having their own front and rear positioned at appropriate points in a single dimensional array.

In this paper we designed a polynomial time algorithm which takes an input a directed acyclic graph G = (V, E) and two vertices s and t and returns the number of paths from s to t in G.

II. ALGORITHM

Given a graph G = (V, E) and a distinguished source vertex "s" and distinguished destination vertex "d", this algorithm search systematically explores the edges of G to "discover" is there a path from "s" to "d" and count the number of paths from "s" to "d".

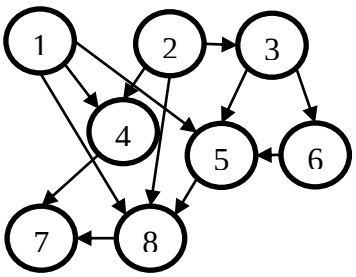
This algorithm works for the directed acyclic graphs. We have taken Graph G = (V, E) in the form of the adjacency matrix A= ($a_{(i,j)}$). Let Q be the multiple queues in a single array. F and R be two array to store front position and rear position of each queues in Q.

```
path( Matrix A)
1.  front ← 0, rear ← 0
2.  for i ←1 to u do
3.        F[i] ← front
4.        front ← rear
5.        for j ← 1 to v do
6.              if A[i][j] = 1
7.                    then Q[front] ← j
8.                          front ← front + 1
9.                          rear ← rear + 1
10.       repeat
11.       R[i] ← rear
12. repeat
13. s ← Source vertex in the graph G = (V, E).
14. d ← Destination vertex in the graph G = (V, E).
15. find_path(s, F[s], R[s], d)

find_path(s, f, r, d)
1.  f ←F[s]
2.  r ← R[s]
3.  if(s=d)
4.        then count ← count + 1
5.  while (f<r)
6.        find_path(Q[f], f, r, d)
7.        f ← f + 1
8.  repeat
```

III. WORKING OF ALGORITHM

Let us consider the following directed acyclic graph G= (V, E) = (8, 12).



(a)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	0	0	1	1	0	0	1
2	0	0	1	1	0	0	0	1
3	0	0	0	0	1	1	0	0
4	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	1
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0

(b)

Fig. 3. Representation of an directed graph. (a) An directed graph G having eight vertices and fourteen edges. (b) The adjacency-matrix representation of G.

This adjacency matrix is stored in the matrix A=($a_{(i,j)}$). We have taken Graph G = (V, E) in the form of the adjacency matrix A= ($a_{(i,j)}$). Let Q be the multiple queues in a single array. F[] and R[] be two array to store front position and rear position of each queues in Q. The path() method is going to add the each vertex out degree into the queue Q[], at the same time it is going to add front and rear of each queues in Q[] to the F[] and R[] respectively. This assignment is shown in the below figure 4.

	0	1	2	3	4	5	6	7	8	9	10	11	12
Q	4	5	8	3	4	8	5	6	7	8	5	7	

(a)

	0	1	2	3	4	5	6	7	8
F	0	0	3	6	8	9	10	11	11

(b)

	0	1	2	3	4	5	6	7	8
R	0	3	6	8	9	10	11	11	12

(c)

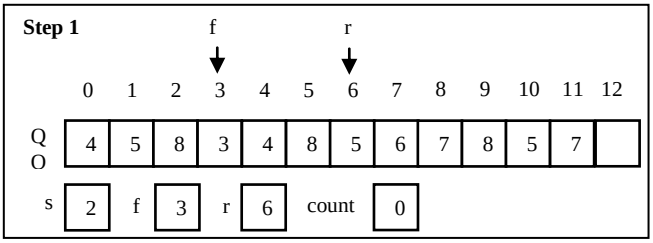
Fig. 4. (a) Multiple queues using single array Q. (b) F[] array to hold fronts of each queues in Q. (c) R[] array to hold rears of each queues in Q.

Let us consider the given source vertex 's' be '2' in the graph G, and destination vertex 'd' be '7', 'count= 0'. Now we call the method find_path() with '2' as source vertex 's', F[2] ← 6 as 'f' , R[2] ← 8 as 'r' and '7' as destination vertex 'd'. The below steps show how the algorithm works.

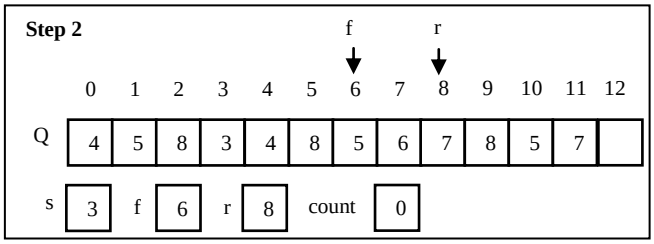
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

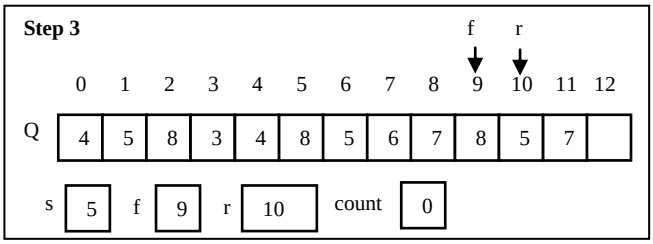
Step 1: find_path(2,6,8,7)
f ← F[2] ← 3 , r ← R[2] ← 6.
if(2==7) False. No increment in count.
while(3<6) True, Enter the loop
Call the method find_path(Q[3],3,6,7).



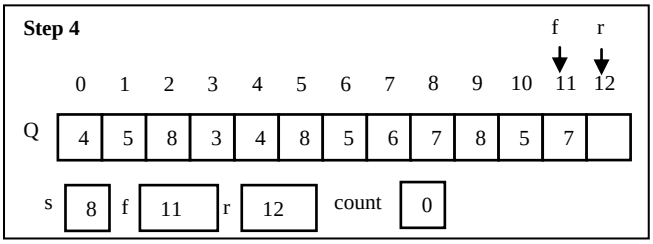
Step 2: find_path(3,3,6,7)
f ← F[3] ← 6 , r ← R[3] ← 8.
if(3==7) False. No increment in count.
while(6<8) True, Enter the loop
Call the method find_path(Q[6],6,8,7).



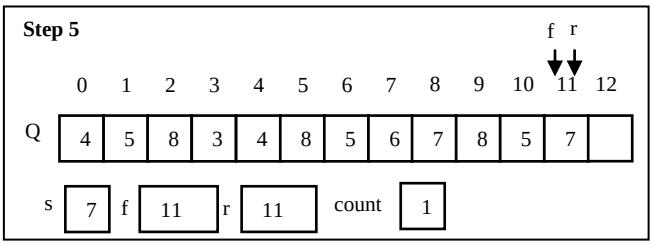
Step 3: find_path(5,6,8,7)
f ← F[5] ← 9 , r ← R[5] ← 10.
if(5==7) False. No increment in count.
while(9<10) True, Enter the loop
Call the method find_path(Q[9],9,10,7).



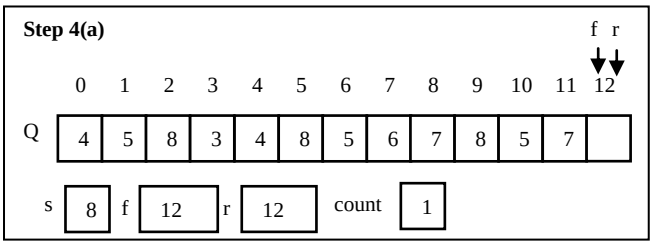
Step 4: find_path(8,9,10,7)
f ← F[8] ← 11 , r ← R[8] ← 12.
if(8==7) False. No increment in count.
while(11<12) True, Enter the loop
Call the method find_path(Q[11],11,12,7).



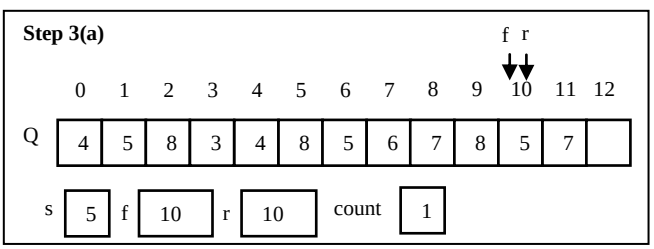
Step 5: find_path(7,11,12,7)
f ← F[7] ← 11 , r ← R[7] ← 11.
if(7==7) True. Increment count by one.
while(11<11) False, Exit the loop



Step 4(a): f ← 11+1 ← 12, r ← 12.
while(12<12) False, Exit the loop



Step 3(a): f ← 9+1 ← 10, r ← 10.
while(10<10) False, Exit the loop

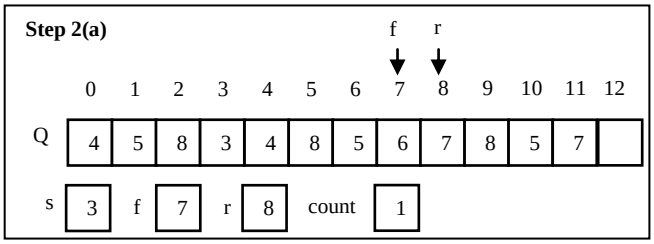


Copyright to IJARCCE
DOI 10.17148/IJARCCE.2015.41021

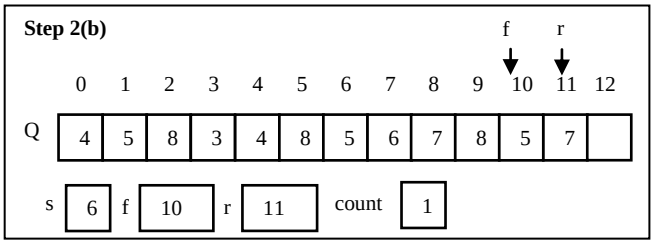
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

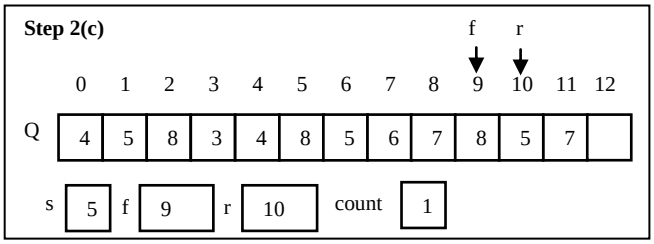
Step 2(a): f ← 6+1 ← 7, r ← 8.
while(7<8) True, Enter the loop
Call the method find_path(Q[7],7,8,7).



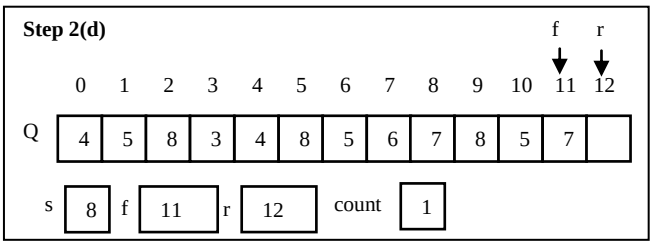
Step 2(b): find_path(6,7,8,7)
f ← F[6] ← 10 , r ← R[6] ← 11.
if(6==7) False. No increment in count.
while(10<11) True, Enter the loop
Call the method find_path(Q[10],10,11,7).



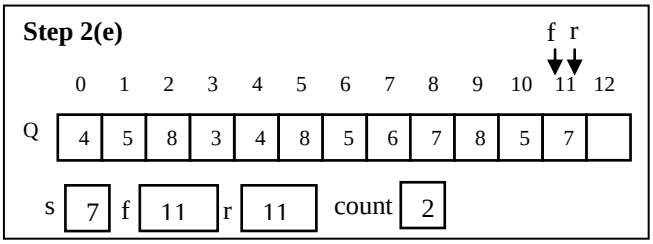
Step 2(c): find_path(5,10,11,7)
f ← F[5] ← 9 , r ← R[5] ← 10.
if(5==7) False. No increment in count.
while(9<10) True, Enter the loop
Call the method find_path(Q[9],9,10,7).



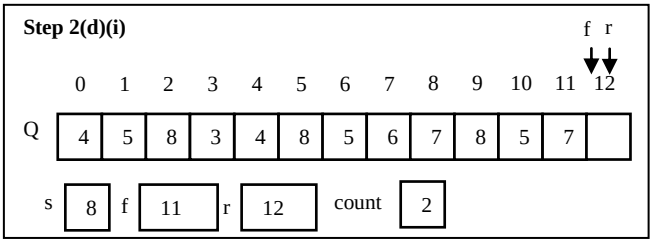
Step 2(d): find_path(8,9,10,7)
f ← F[8] ← 11 , r ← R[8] ← 12.
if(8==7) False. No increment in count.
while(11<12) True, Enter the loop
Call the method find_path(Q[11],11,12,7).



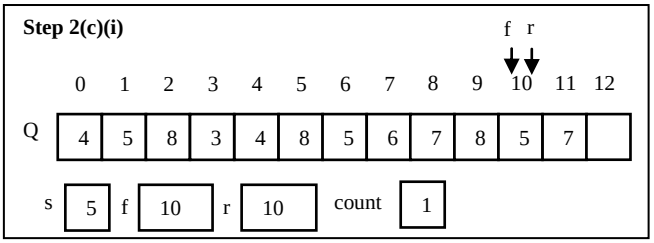
Step 2(e): find_path(7,11,12,7)
f ← F[7] ← 11 , r ← R[7] ← 11.
if(7==7) True. Increment count by one.
while(11<11) False, Exit the loop



Step 2(d)(i): f ← 11+1 ← 12, r ← 12.
while(12<12) False, Exit the loop



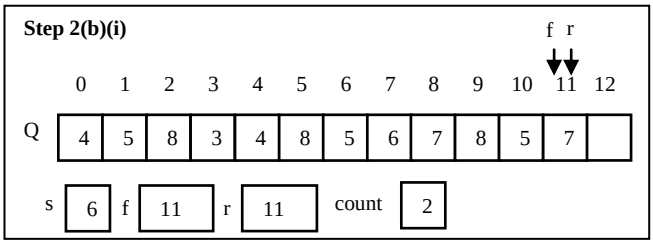
Step 2(c)(i): f ← 9+1 ← 10, r ← 10.
while(10<10) False, Exit the loop



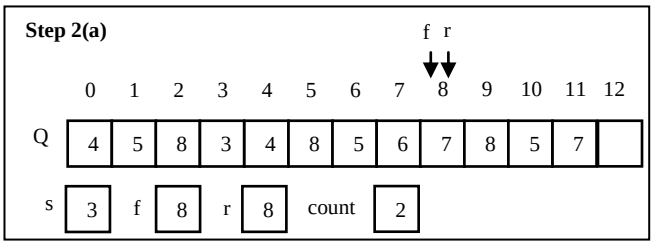
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

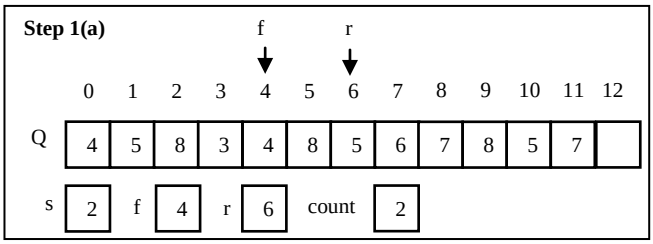
Step 2(b)(i): f ← 10+1 ← 11, r ← 11.
while(11<11) False, Exit the loop



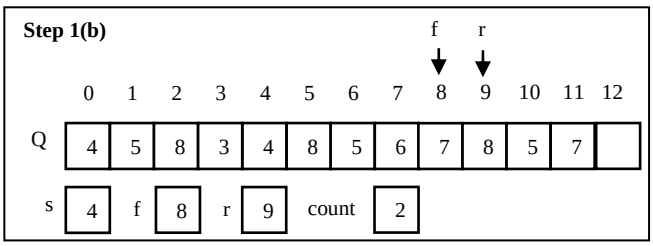
Step 2(a)(i): f ← 7+1 ← 8, r ← 8.
while(8<8) False, Exit the loop



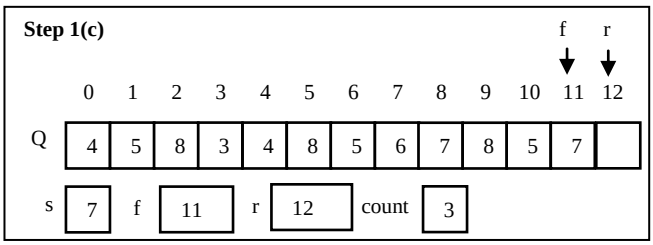
Step 1(a): f ← 3+1 ← 4, r ←6 .
while(4<6) True, Enter the loop
Call the method find_path(Q[4],4,6,7)



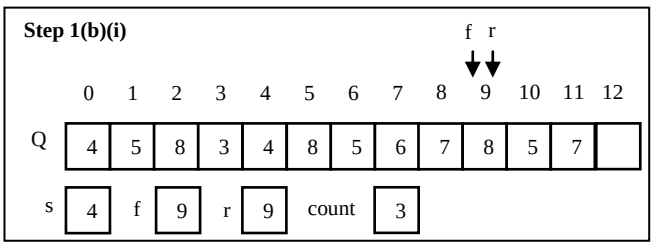
Step 1(b): find_path(4,4,6,7)
f ← F[4] ← 8 , r ← R[4] ← 9.
if(4==7) False. No increment in count.
while(8<9) True, Enter the loop
Call the method find_path(Q[8],8,9,7).



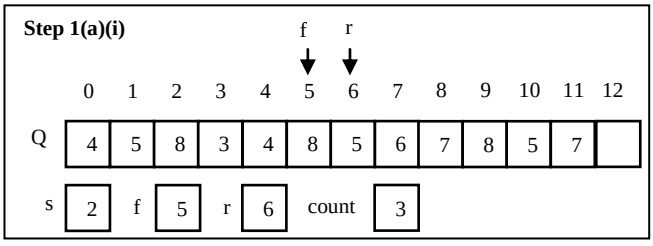
Step 1(c): find_path(7,8,9,7)
f ← F[7] ← 11 , r ← R[7] ← 11.
if(7==7) True. Increment count by one.
while(11<11) False, Exit the loop



Step 1(b)(i): f ← 8+1 ← 9, r ← 9.
while(9<9) False, Exit the loop



Step 1(a)(i): f ← 4+1 ← 5, r ←6 .
while(5<6) True, Enter the loop
Call the method find_path(Q[5],5,6,7)

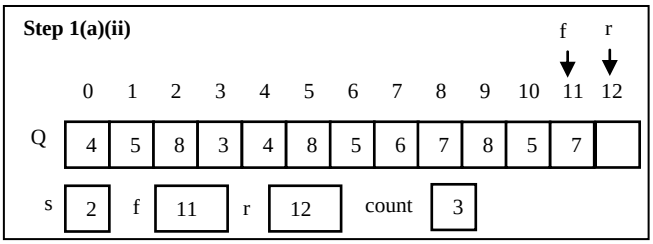


DOI 10.17148/IJARCCE.2015.41021

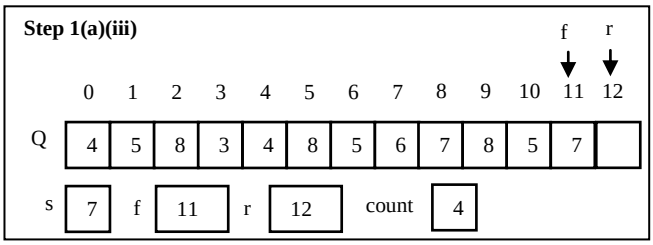
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 10, October 2015

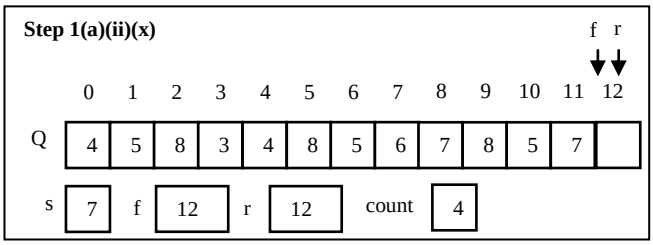
Step 1(a)(ii): find_path(8,5,6,7)
f ← F[8] ← 11 , r ← R[8] ← 12.
if(8==7) False. No increment in count.
while(11<12) True, Enter the loop
Call the method find_path(Q[11],11,12,7).



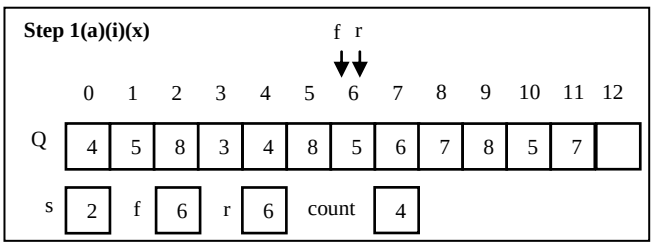
Step 1(a)(iii): find_path(7,11,12,7)
f ← F[7] ← 11 , r ← R[7] ← 11.
if(7==7) True. Increment count by one.
while(11<11) False, Exit the loop



Step 1(a)(ii)(x): f ← 11+1 ← 12, r ← 12.
while(12<12) False, Exit the loop



Step 1(a)(i)(x): f ← 5+1 ← 6, r ← 6.
while(6<6) False, Exit the loop



After exiting all loops the number paths from '2' to '7' is stored in count which will be '4'.

IV. TIME COMPLEXITY

For an n-state diagraph with outdegree d, the time complexity of the algorithm is O (n^3d).

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction To Algorithms, 2nd ed., MIT Press, Cambridge, MA, U.S.A.
[2] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Data Structures and Algorithms, 4th Impression, Dorling Kindersley, India.
[3] A. A. Puntambekar, Data Structures Using 'c', 1st ed., Technical Publications, Pune, India.