Banker’s Algorithm in Operating System

Last Updated: 03-04-2020

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

Why Banker’s algorithm is named so?

Banker’s algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

Following **Data structures** are used to implement the Banker’s Algorithm:

Let **‘n’** be the number of processes in the system and **‘m’** be the number of resources types.

Available :

- It is a 1-d array of size **‘m’** indicating the number of available resources of each type.
- Available[j] = k means there are **‘k’** ins es of resource type **R_j**

system.

- Max[i, j] = k means process P_i may request at most **'k'** instances of resource type R_j.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process P_i is currently allocated **'k'** instances of resource type R_j

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process P_i currently need **'k'** instances of resource type R_j

for its execution.

- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Banker's algorithm consists of Safety algorithm and Resource request algorithm

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

> *1) Let Work and Finish be vectors of length 'm' and 'n' respectively.*
> *Initialize: Work = Available*
> *Finish[i] = false; for i=1, 2, 3, 4….n*
>
> *2) Find an i such that both*
> *a) Finish[i] = false*
> *b) $Need_i$ <= Work*
> *if no such i exists goto step (4)*
>
> *3) Work = Work + Allocation[i]*
> *Finish[i] = true*
> *goto step (2)*
>
> *4) if Finish [i] = true for all i*
> *then the system is in a safe state*

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i$ [j] = k means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

> *1) If $Request_i$ <= $Need_i$*
> *Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.*
>
> *2) If $Request_i$ <= Available*
> *Goto step (3); otherwise, P_i must wait, since the resources are not available.*

follows:

Available = Available – Requesti

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i – Request_i$

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Question1. What will be the content of the Need matrix?

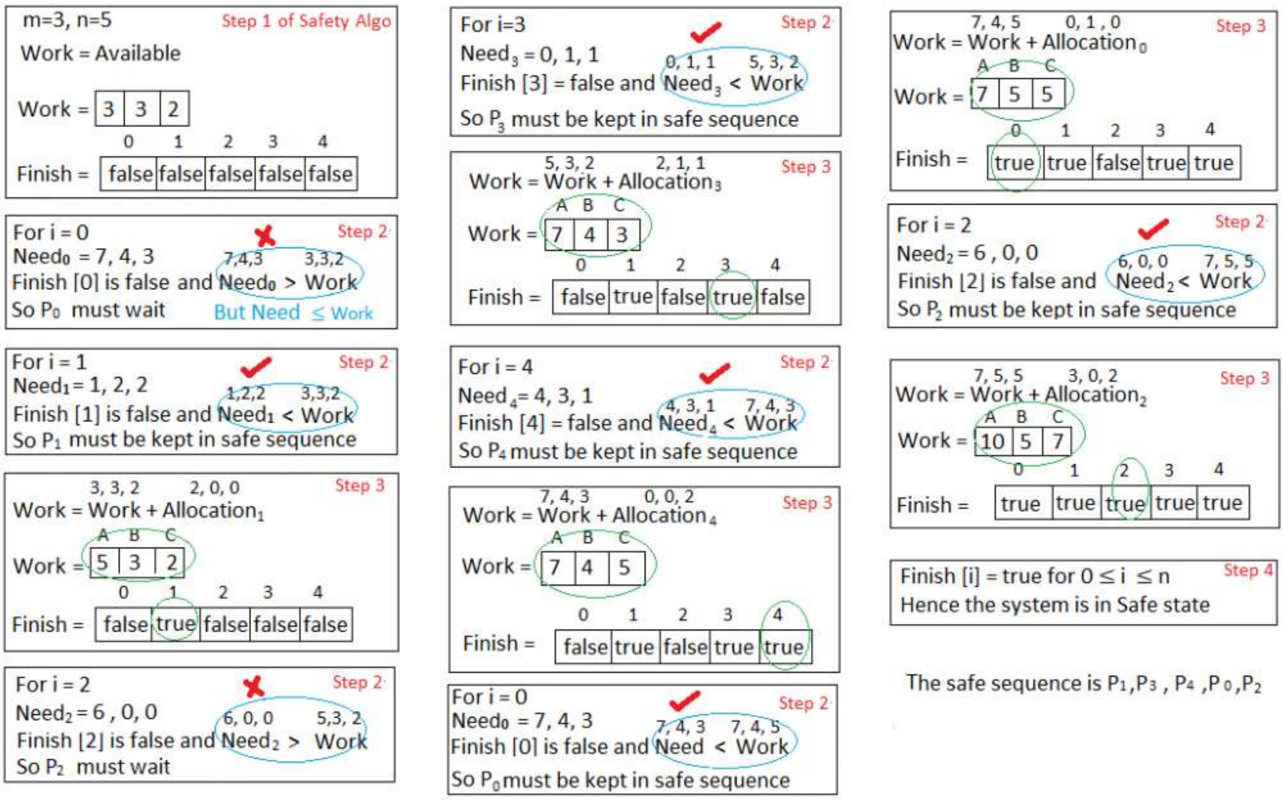
Need [i, j] = Max [i, j] – Allocation [i, j]

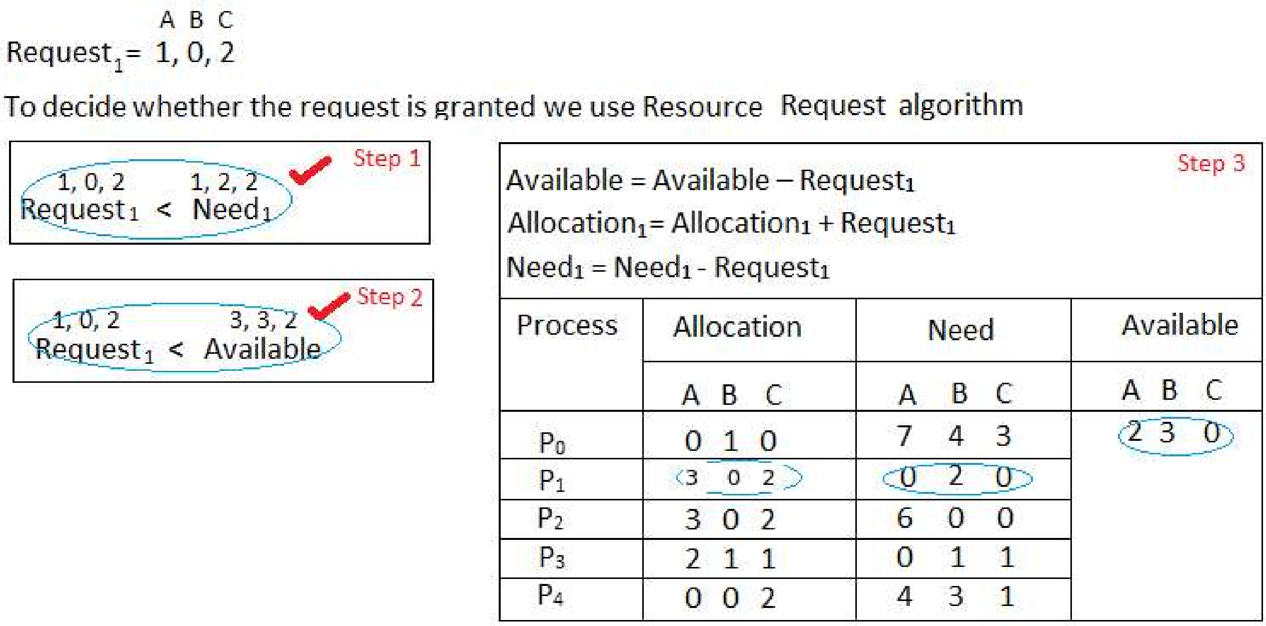
So, the content of Need Matrix is:

P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

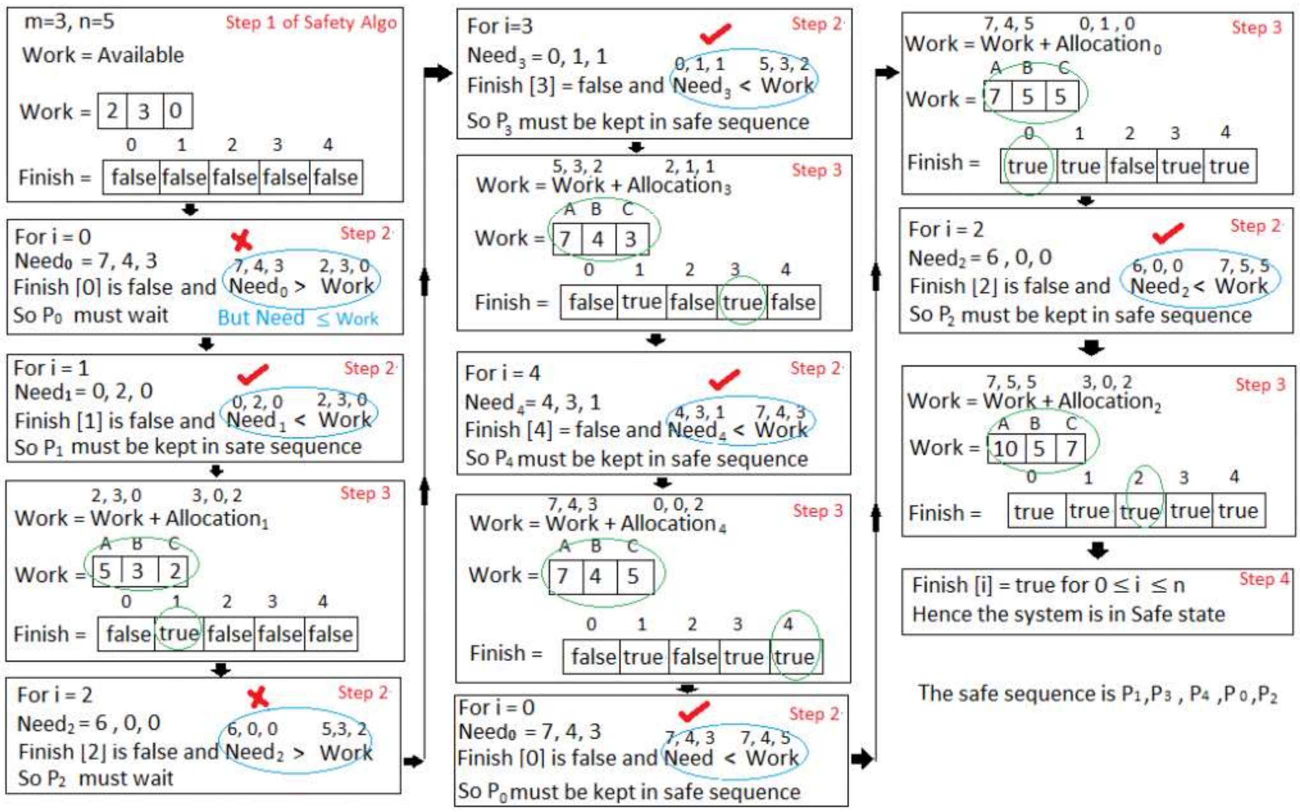
Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,





We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



Hence the new system state is safe, so we can immediately grant the request for process $\mathbf{P_1}$.

Code for Banker's Algorithm

```
// Banker's Algorithm
#include <iostream>
using namespace std;

int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources
    int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4

    int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4

    int avail[3] = { 3, 3, 2 }; // Available Resources

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {

                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                        break;
                    }
                }

                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
```

```
    for (i = 0; i < n - 1; i++)
        cout << " P" << ans[i] << " ->";
    cout << " P" << ans[n - 1] <<endl;

    return (0);
}

// This code is contributed by SHUBHAMSINGH10
```

C

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources
    int alloc[5][3] = { { 0, 1, 0 }, // P0    // Allocation Matrix
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4

    int max[5][3] = { { 7, 5, 3 }, // P0    // MAX Matrix
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4

    int avail[3] = { 3, 3, 2 }; // Available Resources

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {

                int flag = 0;
                for (j = 0; j < m; j+
```

```
                }

                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);

    return (0);

    // This code is contributed by Deep Baldha (CandyZack)
}
```

Java

```
//Java Program for Bankers Algorithm
public class GfGBankers
{
    int n = 5; // Number of processes
    int m = 3; // Number of resources
    int need[][] = new int[n][m];
    int [][]max;
    int [][]alloc;
    int []avail;
    int safeSequence[] = new int[n];

    void initializeValues()
    {
    // P0, P1, P2, P3, P4 are the Process names here
    // Allocation Matrix
    alloc = new int[][] { { 0, 1, 0 }, //P0
                          { 2, 0, 0 }, //P1
                          { 3, 0, 2 }, //P2
                          { 2, 1, 1 }, //P3
                          { 0, 0, 2 } }; //P4

    // MAX Matrix
    max = new int[][] { { 7, 5, 3 }, //P0
                  { 3, 2, 2 }, //P1
                  { 9, 0, 2 }, //P2
                  { 2, 2, 2 }, //P3
```

```
    }

    void isSafe()
    {
    int count=0;

    //visited array to find the already allocated process
    boolean visited[] = new boolean[n];
    for (int i = 0;i < n; i++)
    {
        visited[i] = false;
    }

    //work array to store the copy of available resources
    int work[] = new int[m];
    for (int i = 0;i < m; i++)
    {
        work[i] = avail[i];
    }

    while (count<n)
    {
        boolean flag = false;
        for (int i = 0;i < n; i++)
        {
            if (visited[i] == false)
             {
            int j;
            for (j = 0;j < m; j++)
            {
                if (need[i][j] > work[j])
                break;
            }
            if (j == m)
            {
               safeSequence[count++]=i;
               visited[i]=true;
               flag=true;

                for (j = 0;j < m; j++)
                {
                work[j] = work[j]+alloc[i][j];
                }
            }
             }
        }
        if (flag == false)
        {
            break;
        }
    }
    if (count < n)
    {
        System.out.println("The Syste     UnSafe!");
```

```
        System.out.println("Following is the SAFE Sequence");
                for (int i = 0;i < n; i++)
        {
            System.out.print("P" + safeSequence[i]);
            if (i != n-1)
            System.out.print(" -> ");
        }
    }
    }

    void calculateNeed()
    {
    for (int i = 0;i < n; i++)
    {
        for (int j = 0;j < m; j++)
         {
        need[i][j] = max[i][j]-alloc[i][j];
         }
    }
    }

    public static void main(String[] args)
    {
      int i, j, k;
      GfGBankers gfg = new GfGBankers();

      gfg.initializeValues();
      //Calculate the Need Matrix
      gfg.calculateNeed();

       // Check whether system is in safe state or not
      gfg.isSafe();
    }
}
```

Python3

```
# Banker's Algorithm

# Driver code:
if __name__=="__main__":

    # P0, P1, P2, P3, P4 are the Process names here
    n = 5 # Number of processes
    m = 3 # Number of resources

    # Allocation Matrix
    alloc = [[0, 1, 0 ],[ 2, 0, 0 ],
            [3, 0, 2 ],[2, 1, 1] ,[ 0, 0, 2]]
```

```
    avail = [3, 3, 2] # Available Resources

    f = [0]*n
    ans = [0]*n
    ind = 0
    for k in range(n):
        f[k] = 0

    need = [[ 0 for i in range(m)]for i in range(n)]
    for i in range(n):
        for j in range(m):
            need[i][j] = max[i][j] - alloc[i][j]
    y = 0
    for k in range(5):
        for i in range(n):
            if (f[i] == 0):
                flag = 0
                for j in range(m):
                    if (need[i][j] > avail[j]):
                        flag = 1
                        break

                if (flag == 0):
                    ans[ind] = i
                    ind += 1
                    for y in range(m):
                        avail[y] += alloc[i][y]
                    f[i] = 1

    print("Following is the SAFE Sequence")

    for i in range(n - 1):
        print(" P", ans[i], " ->", sep="", end="")
    print(" P", ans[n - 1], sep="")

# This code is contributed by SHUBHAMSINGH10
```

C#

```
// C# Program for Bankers Algorithm
using System;
using System.Collections.Generic;

class GFG
{
static int n = 5; // Number of processes
static int m = 3; // Number of resources
int [,]need = new int[n, m];
int [,]max;
int [,]alloc;
```

```
    {
        // P0, P1, P2, P3, P4 are the Process
        // names here Allocation Matrix
        alloc = new int[,] {{ 0, 1, 0 }, //P0
                            { 2, 0, 0 }, //P1
                            { 3, 0, 2 }, //P2
                            { 2, 1, 1 }, //P3
                            { 0, 0, 2 }};//P4

        // MAX Matrix
        max = new int[,] {{ 7, 5, 3 }, //P0
                            { 3, 2, 2 }, //P1
                          { 9, 0, 2 }, //P2
                          { 2, 2, 2 }, //P3
                          { 4, 3, 3 }};//P4

        // Available Resources
        avail = new int[] { 3, 3, 2 };
    }

    void isSafe()
    {
        int count = 0;

        // visited array to find the
        // already allocated process
        Boolean []visited = new Boolean[n];
        for (int i = 0; i < n; i++)
        {
            visited[i] = false;
        }

        // work array to store the copy of
        // available resources
        int []work = new int[m];
        for (int i = 0; i < m; i++)
        {
            work[i] = avail[i];
        }

        while (count<n)
        {
            Boolean flag = false;
            for (int i = 0; i < n; i++)
            {
                if (visited[i] == false)
                {
                    int j;
                    for (j = 0; j < m; j++)
                    {
                        if (need[i, j] > work[j])
                        break;
                    }
                    if (j == m)
```

```
                        for (j = 0; j < m; j++)
                        {
                            work[j] = work[j] + alloc[i, j];
                        }
                    }
                }
            }
            if (flag == false)
            {
                break;
            }
        }
        if (count < n)
        {
            Console.WriteLine("The System is UnSafe!");
        }
        else
        {
            //System.out.println("The given System is Safe");
            Console.WriteLine("Following is the SAFE Sequence");
            for (int i = 0; i < n; i++)
            {
                Console.Write("P" + safeSequence[i]);
                if (i != n - 1)
                Console.Write(" -> ");
            }
        }
    }

    void calculateNeed()
    {
        for (int i = 0;i < n; i++)
        {
            for (int j = 0;j < m; j++)
            {
                need[i, j] = max[i, j] - alloc[i, j];
            }
        }
    }

    // Driver Code
    public static void Main(String[] args)
    {
        GFG gfg = new GFG();

        gfg.initializeValues();

        // Calculate the Need Matrix
        gfg.calculateNeed();

        // Check whether system is in
        // safe state or not
        gfg.isSafe();
    }
```

Output:

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

GATE question:
http://quiz.geeksforgeeks.org/gate-gate-cs-2014-set-1-question-41/

Reference:
Operating System Concepts 8th Edition by Abraham Silberschatz, Peter B. Galvin, Greg Gagne

This article has been contributed by Vikash Kumar. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

Sequence Step Algorithm in Operating System
Banker's Algorithm in Operating System
Deadlock Detection Algorithm in Operating System
System Protection in Operating System
User View Vs Hardware View Vs System View of Operating System
System Programs in Operating System
File System Implementation in Operating Sy

Introduction of Deadlock in Operating System
Thread in Operating System
Paging in Operating System
Segmentation in Operating System
File Systems in Operating System
Virtual Memory in Operating System
Real Time Operating System (RTOS)
Remote Procedure Call (RPC) in Operating System
Lottery Process Scheduling in Operating System
Resource Allocation Graph (RAG) in Operating System
Page Fault Handling in Operating System

Improved By : Deepanshu8391, CandyZack, ShJos, tarlisonbrito, Rajput-Ji, more

Article Tags : Operating Systems Process Synchronization

Practice Tags : Operating Systems

25

2.9

To-do Done

Based on **42** vote(s)

Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us
Careers
Privacy Policy
Contact Us

Learn

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

Practice

Courses
Company-wise
Topic-wise
How to begin?

Contribute

Write an Article
Write Interview Experience
Internships
Videos