MATHEMATICS SECOND PERIODICAL

VAISHNAVI V RAO, AM.EN.P2CSN19009 05/05/2020

GRAPH THEORY

A graph is a data structure which represents the objects. These objects are connected with a pair of links. Every node connect to every other node with a pair of links called edges. This section mainly focuses on finding different shortest path algorithms and researching about how one graph is different from other.

The different shortest path algorithms are listed below:

- Bellman-Ford algorithm
- Floyd Warshall algorithm
- Dijkstras algorithm

Bellman-Ford algorithm:
The algorithm was first proposed by Alfonso Shimbel (1955), but is instead named after Richard Bellman and Lester Ford Jr., who published it in 1958 and 1956, respectively. Edward F. Moore also published the same algorithm in 1957, and for this reason it is also sometimes called the Bellman–Ford–Moore algorithm.
Bellman-Ford algorithm computes shortest path from single source vertex to all the vertices in a weighted graph. This is more versatile graph computation as it can handle the negative weights. This algorithm uses relaxation technique where in correct distances are replaced by better ones until they reach the solution.

Listing 1: Bellman-Ford pseudocode.

```
1  function BellmanFord(list vertices, list edges, vertex source) is ::↩
       distance[], predecessor[]
2
3  // Step 1: initialize graph
4      for each vertex v in vertices do
5          distance[v] := inf             // Initialize the distance to all ↩
               vertices to infinity
6          predecessor[v] := null         // And having a null predecessor
7          distance[source] := 0              // The distance from the source↩
               to itself is, of course, zero
8  // Step 2: relax edges repeatedly
9      for i from 1 to size(vertices) minus 1 do
```

```
        for each edge (u, v) with weight w in edges do
            if distance[u] + w < distance[v] then
                distance[v] := distance[u] + w
                predecessor[v] := u

// Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges do
        if distance[u] + w < distance[v] then
            error "Graph contains a negative-weight cycle"

    return distance[], predecessor[]
```

Floyd-Warshall Algorithm:

Floyd-Warhshall algorithm is also called as Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm or WFI algorithm.

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles.This algorithm follows the dynamic programming approach to find the shortest paths.

Time Complexity:

There are three loops. Each loop has constant complexities. So, the time complexity of the Floyd-Warshall algorithm is O(n^3)

Space Complexity:

The space complexity of the Floyd-Warshall algorithm is O(n^2)

Listing 2: Floyd warshall pseudocode.

```
let dist be a |V| x |V| array of minimum distances initialized to (↩
   infinity)
for each edge (u, v) do
    dist[u][v] <- w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] <- 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] <-dist[i][k] + dist[k][j]
            end if
```

Dijkstras Algorithm:

Dijkstra's algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph.For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.
Dijkstas algorithm does not work for negative weights.

Listing 3: Dijksta's algorithm C program

```
#include<stdio.h>
#define HIGHVALUE 999
#define MAX 10

void dij(int Adj_Matrix[MAX][MAX],int v,int starting_n)
{

    int visited[MAX],count,min_distance,nextnode,i,j;
    int cos_Matrix[MAX][MAX],distance[MAX],previous[MAX];


    for(i=0;i<v;i++)
        for(j=0;j<v;j++)
            if(Adj_Matrix[i][j]==0) //Check if the value at that point is 0
                cos_Matrix[i][j] = HIGHVALUE; //So that no other value can replace this
            else
                cos_Matrix[i][j] = Adj_Matrix[i][j];


    for(i=0;i<v;i++)
    {
        distance[i] = cos_Matrix[starting_n][i];
        previous[i] = starting_n;
        visited[i]=0;
    }

    distance[starting_n]=0;
    visited[starting_n]=1;
    count=1;

    while(count<v-1)
    {
        min_distance=HIGHVALUE;

```

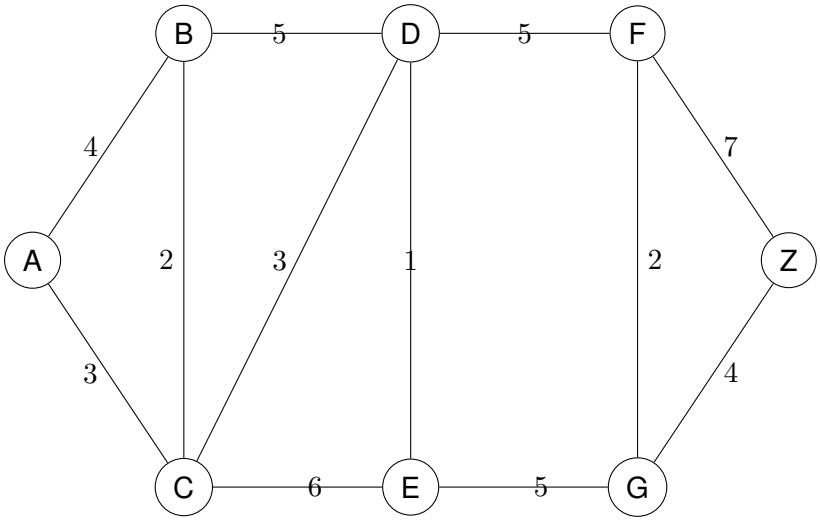
```
35          //nextnode is the node at minimum distance
36          for(i=0;i<v;i++)
37              if(distance[i] < min_distance && !visited[i])
38              {
39                  min_distance=distance[i];
40                  nextnode=i;
41              }
42
43              //Check for a better path
44              visited[nextnode] = 1;
45              for(i=0;i<v;i++)
46                  if(!visited[i])
47                      if(min_distance + cos_Matrix[nextnode][i] < distance[i↩
                           ])
48                      {
49                          distance[i] = min_distance + cos_Matrix[nextnode][↩
                               i];
50                          previous[i] = nextnode;
51                      }
52          count++;
53      }
54
55      //print the path and distance of each node
56      for(i=0;i<v;i++)
57          if(i!=starting_n)
58          {
59              printf("\nDistance of node %d = %d ",i,distance[i]);
60              printf("\nPath = %d",i);
61
62              j=i;
63              do
64              {
65                  j = previous[j];
66                  printf(" <-% d ",j);
67              }while(j != starting_n);
68      }
69
70  printf("\n");
71
72  }
73
74  int main()
75  {
76      int Adj_Matrix[MAX][MAX],i,j,v,u;
77      printf("Enter the Total Number of Vertices: ");
78      scanf("%d",&v);
79      printf("\nEnter the Adjacency matrix by each Row:\n");
```

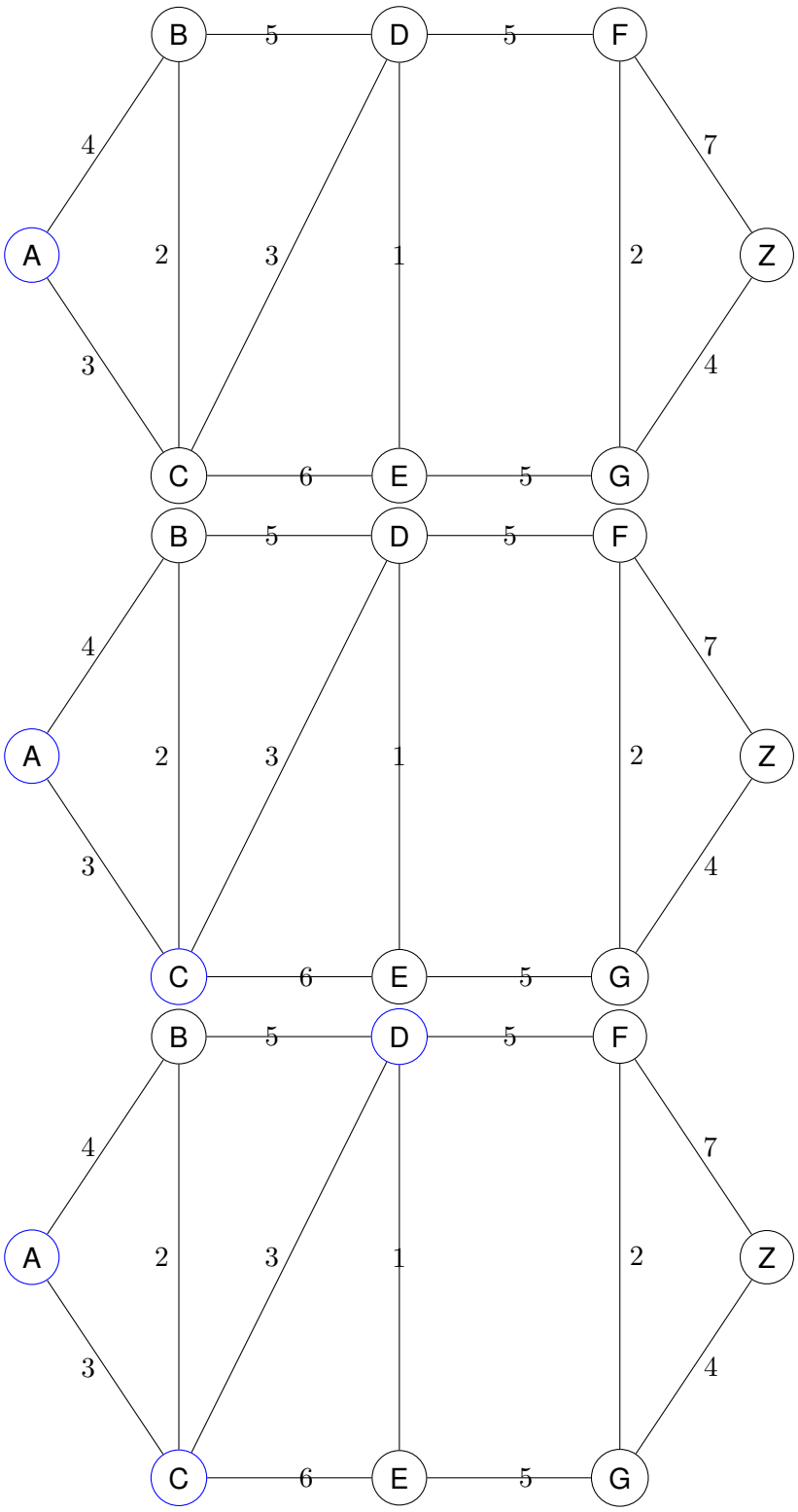
```
80
81      for(i=0;i<v;i++) //Loop to get the values from user and store
82          for(j=0;j<v;j++)
83              scanf("%d",&Adj_Matrix[i][j]);
84
85      printf("\nEnter the starting node:");
86      scanf("%d",&u);
87      dij(Adj_Matrix,v,u);
88
89      return 0;
90  }
```

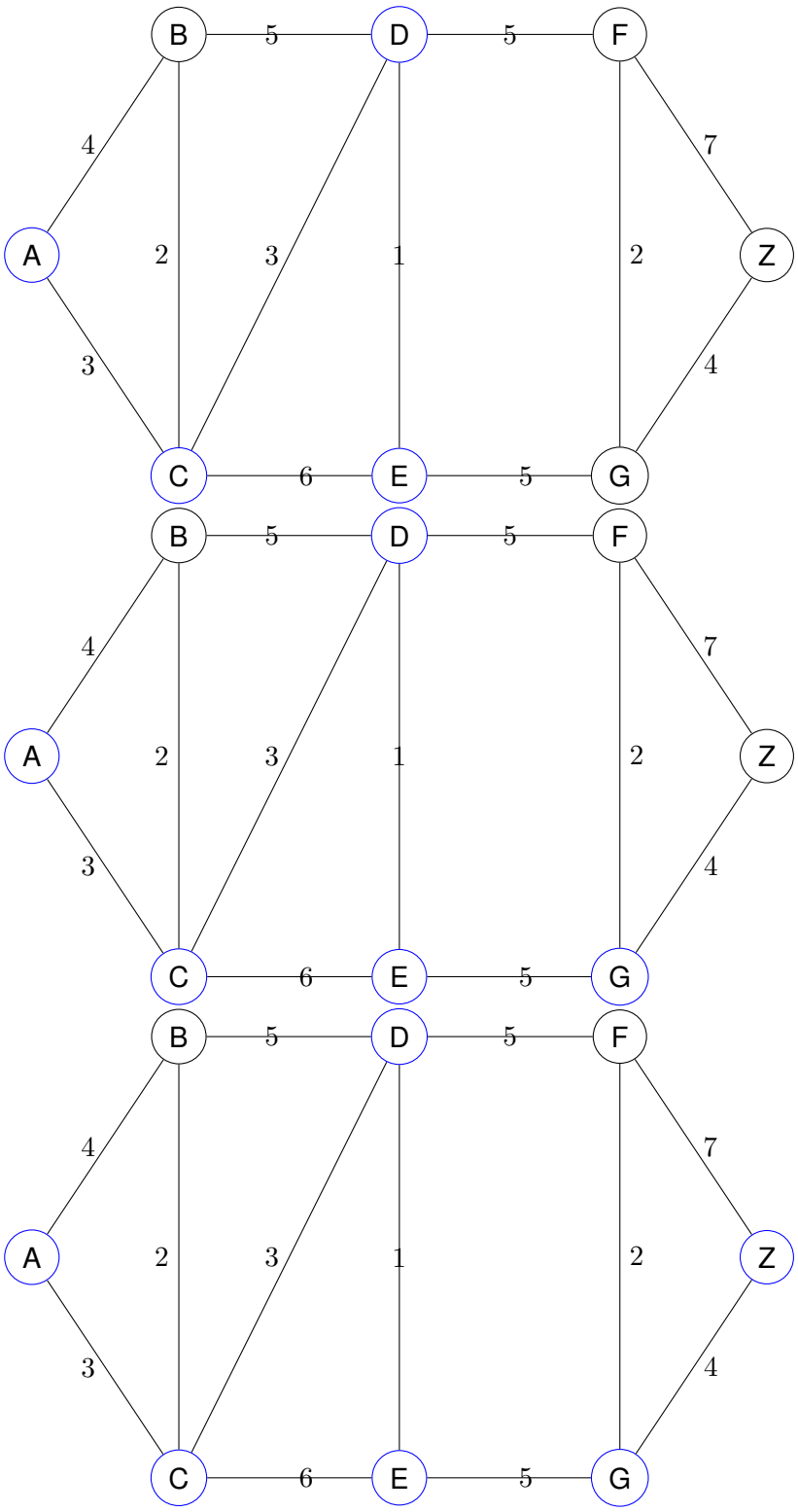
Comparison Dijkstas algorithm with other shortest path algorithms

- Bellman-Ford algorithm works fine on a graph having negative edge weights but Dijkstra algorithm may or may not give the correct result in this case.
- Bellman-Ford algorithm does not work if a graph contains negative weight cycles although it can detect a negative weight cycle and it is slower than Dijkstra's Algorithm.
- complexity of bellman ford algorithm is O(V.E) in worst case and O(E) in best case.
- The Floyd–Warshall algorithm is a good choice for computing paths between all pairs of vertices in dense graphs, in which most or all pairs of vertices are connected by edges.
- For sparse graphs with non-negative edge weights, a better choice is to use Dijkstra's algorithm from each possible starting vertex, since the running time of repeated Dijkstra is better than the $O(|V|^3)$ running time of the Floyd–Warshall algorithm, when |E| is significantly smaller than $|V|^2$

Dijkstra's algorithm for this graph

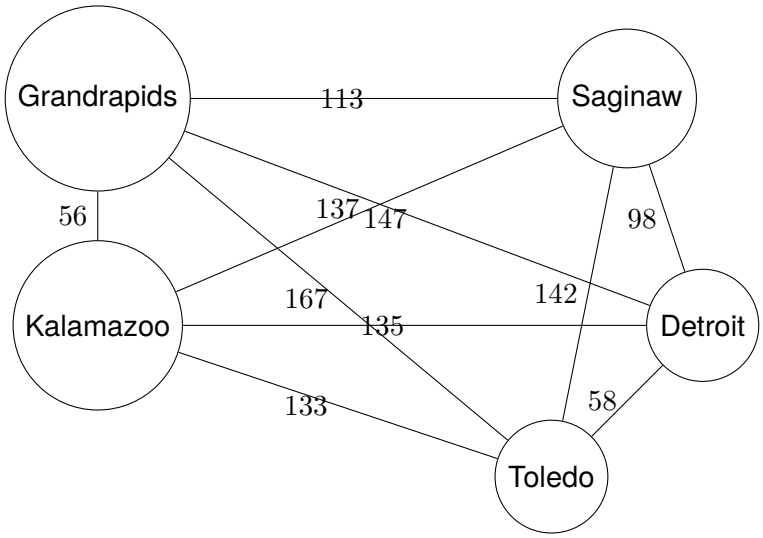



B
5
D
5
F
4
7
A
2
3
1
2
Z
3
4
C
6
E
5
G
B
5
D
5
F
4
7
A
2
3
1
2
Z
3
4
C
6
E
5
G
B
5
D
5
F
4
7
A
2
3
1
2
Z
3
4
C
6
E
5
G


B
5
D
5
F
4
7
A
2
3
1
2
Z
3
4
C
6
E
5
G
B
5
D
5
F
4
7
A
2
3
1
2
Z
3
4
C
6
E
5
G
B
5
D
5
F
4
7
A
2
3
1
2
Z
3
4
C
6
E
5
G

Travelling salesman problem

The traveling salesman problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited. In the problem statement, the points are the cities a salesperson might visit. The salesman‘s goal is to keep both the travel costs and the distance traveled as low as possible.



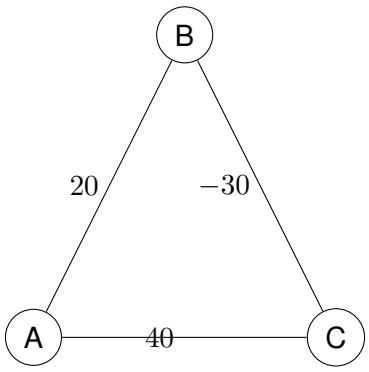
to find the path between all the cities we have 12 paths let us start from city Detroit. hence

PATH	TOTAL COST
Detroit-Toledo-Grandrapids-Saginaw-Kalamazoo-Detroit	610
Detroit- Toledo -Grandrapids-Kalamazoo-Saginaw-Detroit	516
Detroit- Toledo -Kalamazoo-Saginaw-Grandrapids-Detroit	588
Detroit- Toledo -Kalamazoo-Grandrapids-Saginaw-Detroit	458
Detroit- Toledo -Saginaw-Kalamazoo-Grandrapids-Detroit	540
Detroit- Toledo -Saginaw-Grandrapids-Kalamazoo-Detroit	504
Detroit-Saginaw- Toledo -Grandrapids-Kalamazoo-Detroit	598
Detroit-Saginaw- Toledo -Kalamazoo-Grandrapids-Detroit	576
Detroit-Saginaw-Kalamazoo-Toledo-Grandrapids-Detroit	682
Detroit-Saginaw-Grandrapids- Toledo -Kalamazoo-Detroit	646
Detroit-Grandrapids-Saginaw- Toledo -Kalamazoo-Detroit	670
Detroit-Grandrapids- Toledo -Saginaw-Kalamazoo-Detroit	728

here we can note that the shortest path is 458 with lowest cost.

NEGATIVE EDGES IN DIJKSTRA'S ALGORITHM

Dijkstra's algorithm does not work on graphs with negative weights. The algorithm works in a greedy manner. Once it selects the minimum edge weight, it cannot replace it for better choice. At each iteration, it extracts the minimum edged weight by assuming that in the next iterations there won't come any other lesser edge weight for the shortest path.



In the above figure, we are trying to get shortest paths from source node A to all other nodes (node B and node c). Since Dijkstra's algorithm works by employing a greedy process, it outputs 20 as the shortest path cost to node B.

As we can see, from node A, we can go to two nodes – node B and node C, at a cost of 20 and 40 respectively. Hence, going to node B is cheaper. And that is why, it outputs 20 to be the cheapest cost to reach node B.

However, we know that the cheapest cost to reach node B is through node C. And the associated cost is: 40 + (-30) = 10. So Dijkstra's algorithm gets it wrong. It gets it wrong because it cannot foresee that later, a negative edge can bring down the total cost to below 20.

If we carefully observe, we see that the wrong calculation by Dijkstra's algorithm happens due to the negative edge. Had cost from node C to node B not been negative, it could never bring down the total cost to lower than 20, after getting added to 40.

BELLMAN-FORD ALGORITHM FOR EXPLAINING NEGATIVE EDGES

Listing 4: C program on bellman ford.

```
1  #include<stdio.h>
2
3  #include<math.h>
4
5  Bellman(int k, int* a, int* b, int n){
6      int i,j;
7      for(i=0;i<n;i++){
8          *(b+(k+1)*n+i)=*(b+k*n+i);
9          for(j=0;j<n;j++){
10             if(*(b+(k+1)*n+i)>*(b+k*n+j)+*(a+*(b+j)*n+*(b+i)))
11                 *(b+(k+1)*n+i)=*(b+k*n+j)+*(a+*(b+j)*n+*(b+i));
```

```
        }
    }
    if(k<n)
        Bellman(k+1,a,b,n);
}

int main(){
    int n,i,j,k,z;
    printf("\nTHIS PROGRAM ONLT CHECKS FOR NEGATIVE CYCLES IN BELLMAN FORD\n");
    printf("\nENTER THE NUMBER OF VERTICES\n");
    scanf("%d",&n);
    int a[n][n],b[n+2][n];
    printf("\nENTER THE MATRIX PLEASE\n");
    for(i=0;i<n;i++){
        printf("\n");
        for(j=0;j<n;j++){


            scanf("%d",&a[i][j]);
        }
    }

    printf("\nENTER THE SOURCE VERTEX");
    scanf("%d",&z);

    /*Calculation for the first vertex*/
    b[0][0]=z-1;
    b[1][0]=0;
    for(j=1;j<n;j++){
        if(j>z-1)
            b[0][j]=j;
        else
            b[0][j]=j-1;
    }
    for(j=1;j<n;j++)
        b[1][j]=99;

    Bellman(1,a,b,n);


    for(i=0;i<n;i++){
        if(b[n+1][i]!=b[n][i]){
            printf("\n THERE IS A NEGATIVE EDGE \n");
            return 0;
        }
    }
```

```
58 printf("\nThe answer is:\n");
59     for(i=1;i<n;i++)
60         printf("\nWeight of vertex no.%d is %d",b[0][i]+1,b[n+1][i]);
61 }
```

MATRICES

GAUSSIAN ELIMINATION

Gaussian elimination, also known as row reduction, is an algorithm in linear algebra for solving a system of linear equations. It is usually understood as a sequence of operations performed on the corresponding matrix of coefficients.

Listing 5: Python program on guass elimination.

```
ef Gauss_Eliminate(Matrix, b, n):

    l = [0 for x in range(n)]

    s = [0.0 for x in range(n)]

    for i in range(n):

        l[i] = i

        smax = 0.0

        for j in range(n):

            if abs(Matrix[i][j]) > smax:

                smax = abs(Matrix[i][j])

        s[i] = smax



    for i in range(n - 1):

        rmax = 0.0

        for j in range(i, n):

            ratio = abs(Matrix[l[j]][i]) / s[l[j]]

            if ratio > rmax:

                rmax = ratio

                rindex = j

        temp = l[i]
```

```
38
39          l[i] = l[rindex]
40
41          l[rindex] = temp
42
43          for j in range(i + 1, n):
44
45              multiplier = Matrix[l[j]][i] / Matrix[l[i]][i]
46
47              for k in range(i, n):
48
49                  Matrix[l[j]][k] = Matrix[l[j]][k] - multiplier * Matrix[l[↩
                        i]][k]
50
51              b[l[j]] = b[l[j]] - multiplier * b[l[i]]
52
53
54
55      x = [0.0 for y in range(n)]
56
57      x[n - 1] = b[l[n - 1]] / Matrix[l[n - 1]][n - 1]
58
59      for j in range(n - 2, -1, -1):
60
61          summ = 0.0
62
63          for k in range(j + 1, n):
64
65              summ = summ + Matrix[l[j]][k] * x[k]
66
67          x[j] = (b[l[j]] - summ) / Matrix[l[j]][j]
68
69
70
71      print ("The values for solution are [", end="")
72
73      for i in range(n):
74
75          if i != (n - 1):
76
77              print(x[i], ",", end="")
78
79          else:
80
81              print(x[i], "].")
82
83
```

```
print("Enter the Number of Rows: ")
rows = int(input())

print("Enter the Row Values: ")
arr1 = list(map(int, input().split()))

arr2 = list(map(int, input().split()))

arr3 = list(map(int, input().split()))

arr4 = list(map(int, input().split()))




print("Enter the RHS: ")
rhs = list(map(int, input().split()))



matrix0 = [arr1,arr2,arr3,arr4]

vector0 = rhs


Gauss_Eliminate(matrix0, vector0, rows)
```

MANUAL SOLUTION.

$$B = \begin{bmatrix} 0 & 1 & 1 & -2 & -3 \\ 1 & 2 & -1 & 0 & 2 \\ 2 & 4 & 1 & -3 & -2 \\ 1 & 4 & -7 & -1 & -19 \end{bmatrix}$$

Interchange R2 by R1

$$B = \begin{bmatrix} 1 & 2 & -1 & 0 & 2 \\ 0 & 1 & 1 & -2 & -3 \\ 2 & 4 & 1 & -3 & -2 \\ 1 & 4 & -7 & -1 & -19 \end{bmatrix}$$

R3=-2R1+R3

R4=-R1+R4

$$B = \begin{bmatrix} 1 & 2 & -1 & 0 & 2 \\ 0 & 1 & 1 & -2 & -3 \\ 0 & 0 & 3 & -3 & -6 \\ 0 & -6 & -6 & -1 & -21 \end{bmatrix}$$

R4=6R2+R4

$$B = \begin{bmatrix} 1 & 2 & -1 & 0 & 2 \\ 0 & 1 & 1 & -2 & -3 \\ 0 & 0 & 3 & -3 & -6 \\ 0 & 0 & 0 & -13 & -39 \end{bmatrix}$$

Divide R3 by 3 and R4 by 13

$$B = \begin{bmatrix} 1 & 2 & -1 & 0 & 2 \\ 0 & 1 & 1 & -2 & -3 \\ 0 & 0 & 1 & -1 & -2 \\ 0 & 0 & 0 & -1 & -3 \end{bmatrix}$$

(command gcc <program name>, time ./a.out)

time taken by the program:25.38s

memory used by the program:785bytes

INVERSE MATRIX SOLVER

If the product between two matrices is the identity matrix, then we say that the matrices are "inverse"; because by multiplying them we obtain the neutral element for the product.

Listing 6: C program on inverse matrix multiplication

```
#include<stdio.h>
int main()
{
    float mat[10][10], sum,inverse;
    int row,column,k,n;
    printf("\n======THIS IS A C PROGRAM TO CALCULATE INVERSE OF A MATRIX==========\n");
    printf("\nENTER THE ORDER OF MATRIX 5 OR BELOW 5\n\n");
    scanf("%d",&n);
    printf("\nENTER THE MATRIX PLEASE\n\n");
    for(row=0;row<n;row++){ /*code to enter row*/
        for(column=0;column<n;column++) { /*code to enter column*/
            scanf("%f",&mat[row][column]);
                  }
            }
/* Since solving huge matrix requires some way to solve them with upper triangular matrix, Small Matrices like 2x2 or 3x3 could be solved directly using the formula*/
```

```
16
17 for(row=0;row<n;row++) {
18     for(column=n;column<2*n;column++) {
19     if(row==(column-n))
20         mat[row][column]=1.0;
21     else
22         mat[row][column]=0.0;
23 }
24 }
25 for(row=0;row<n;row++) {
26     for(column=0;column<n;column++) {
27         if(row!=column){
28
29             sum=mat[column][row]/mat[row][row];
30             for(k=0;k<2*n;k++)
31             {
32             mat[column][k]-=sum* mat[row][k];
33 }
34 }
35 }
36 }
37
38 for(row=0; row<n;row++) {
39     inverse=mat[row][row];
40     for(column=0;column<2*n;column++) {
41         mat[row][column]/=inverse;
42 }
43 }
44
45 printf("\n====================INVERSE MATRIX====================\n");
46 for(row=0;row<n;row++)
47 {
48 for(column=n;column<2*n;column++){
49
50 printf("%.2f\t",mat[row][column]);
51 printf("\t");
52 }
53 printf("\n");
54 }
55 return 0;
56 }
```

---

manual method to calculate inverse of a matrix

$$\left[\begin{array}{ccccc|ccccc} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 0 & 1 & 0 & 0 & 0 \\ 1 & 3 & 6 & 10 & 15 & 0 & 0 & 1 & 0 & 0 \\ 1 & 4 & 10 & 20 & 35 & 0 & 0 & 0 & 1 & 0 \\ 1 & 5 & 15 & 35 & 69 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccccc|ccccc} 1 & 2 & 3 & 4 & 5 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 3 & 6 & 10 & 15 & 0 & 0 & 1 & 0 & 0 \\ 1 & 4 & 10 & 20 & 35 & 0 & 0 & 0 & 1 & 0 \\ 1 & 5 & 15 & 35 & 69 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

elimination of first coumn

$$\left[\begin{array}{ccccc|ccccc} 1 & 2 & 3 & 4 & 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & -3 & -5 & -7 & -9 & 1 & -2 & 0 & 0 & 0 \\ 0 & 1 & 3 & 6 & 10 & 0 & -1 & 1 & 0 & 0 \\ 0 & 2 & 7 & 16 & 30 & 0 & -1 & 0 & 1 & 0 \\ 0 & 3 & 12 & 31 & 64 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

finding pivot element and swapping third and second row

$$\left[\begin{array}{ccccc|ccccc} 1 & 2 & 3 & 4 & 5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 6 & 10 & 0 & -1 & 1 & 0 & 0 \\ 0 & -3 & -5 & -7 & -9 & 1 & -2 & 0 & 0 & 0 \\ 0 & 2 & 7 & 16 & 30 & 0 & -1 & 0 & 1 & 0 \\ 0 & 3 & 12 & 31 & 64 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

eliminate second column

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & -3 & -8 & -15 & 0 & 3 & -2 & 0 & 0 \\ 0 & 1 & 3 & 6 & 10 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 4 & 11 & 21 & 1 & -5 & 3 & 0 & 0 \\ 0 & 0 & 1 & 4 & 10 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 3 & 13 & 34 & 0 & 2 & -3 & 0 & 1 \end{array}\right]$$

swap fourth and third rows

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & -3 & -8 & -15 & 0 & 3 & -2 & 0 & 0 \\ 0 & 1 & 3 & 6 & 10 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 10 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 4 & 11 & 21 & 1 & -5 & 3 & 0 & 0 \\ 0 & 0 & 3 & 13 & 34 & 0 & 2 & -3 & 0 & 1 \end{array}\right]$$

delete third row

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 4 & 15 & 0 & 6 & -8 & 3 & 0 \\ 0 & 1 & 0 & -6 & -20 & 0 & -4 & 7 & -3 & 0 \\ 0 & 0 & 1 & 4 & 10 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & -5 & -19 & 1 & -9 & 11 & -4 & 0 \\ 0 & 0 & 0 & 1 & 4 & 0 & -1 & 3 & -3 & 1 \end{array}\right]$$

swap fifth and fourth rows

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 4 & 15 & 0 & 6 & -8 & 3 & 0 \\ 0 & 1 & 0 & -6 & -20 & 0 & -4 & 7 & -3 & 0 \\ 0 & 0 & 1 & 4 & 10 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 0 & -1 & 3 & -3 & 1 \\ 0 & 0 & 0 & -5 & -19 & 1 & -9 & 11 & -4 & 0 \end{array}\right]$$

delete fourth column

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & -1 & 0 & 10 & -20 & 15 & -4 \\ 0 & 1 & 0 & 0 & 4 & 0 & -10 & 25 & -21 & 6 \\ 0 & 0 & 1 & 0 & -6 & 0 & 5 & -14 & 13 & -4 \\ 0 & 0 & 0 & 1 & 4 & 0 & -1 & 3 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -14 & 26 & -19 & 5 \end{array}\right]$$

delete column 5

$$\left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & 1 & 0 & 0 & 0 & -4 & 46 & -79 & 55 & -14 \\ 0 & 0 & 1 & 0 & 0 & 6 & -79 & 142 & -101 & 26 \\ 0 & 0 & 0 & 1 & 0 & -4 & 55 & -101 & 73 & -19 \\ 0 & 0 & 0 & 0 & 1 & 1 & -14 & 26 & -19 & 5 \end{array}\right]$$

(command gcc <program name>, time ./a.out)

time taken by the program:23.961s

memory used by the program:1.7kb

DETERMINANT OF THE MATRIX

In algebra, a determinant is a function depending on n that associates a scalar, det(A), to every n×n square matrix A. The fundamental geometric meaning of a determinant acts as the scale factor for volume when A is regarded as a linear transformation. Determinants are important both in Calculus, where they enter the substitution rule for several variables, and in Multilinear Algebra.

Determinants basically help to describe the nature of solutions of linear equations. The determinant of a real matrix is just some real number, telling you about the invertibility of the matrix and hence telling you things about linear equations wrapped up in the matrix.

The determinant being non-zero is equivalent to the matrix being invertible, which is equivalent to the corresponding sets of linear equations having EXACTLY one solution.

The determinant being zero means the matrix is not invertible. In this case the corresponding sets of linear equations can either have infinitely many solutions or none at all.

APPLICATIONS OF DETERMINANT

- Graphic software such as Adobe Photoshop.
- In physics related applications, they are used in the study of electrical circuits, quantum mechanics and optics.

PROPERTIES

- If one row of a matrix consists entirely of zeros, then the determinant is zero.
- If two rows of a matrix are interchanged, the determinant changes sign.
- If two rows of a matrix are identical, the determinant is zero.
- If the matrix B is obtained from the matrix A by multiplying every element in one row of A by the scalar λ , then | B|=λ |A|.
- For an n × n matrix A and any scalar λ, det(λA)= λn det(A).
- If a matrix B is obtained from a matrix A by adding to one row of A, a scalar times another row of A, then |A|=|B|.
- det(A) = det(AT).
- The determinant of a triangular matrix, either upper or lower, is the product of the elements on the main diagonal. Property 9: If A and B are of the same order, then det(AB)=det(A) det(B).

what happens to the determinant if we change rows

If any two rows of a determinant are interchanged, then the sign of determinant changes.

multiple a row by a constant and so forth

If each element of a row of a determinant is multiplied by a constant k, then its value gets multiplied by k.

C PROGRAM TO CALCULATE DETERMINANT

Listing 7: C program on determinant of matrix

```
#include<stdio.h>
#include<math.h>


float determinant(float matrix[30][30],float size)
{
    int row,column,m,n,temp;
    float start=1,det=0,minor[30][30];

    if (size==1)
    {
        return (matrix[0][0]);
    }
    else
    {
        det=0;
        for (temp=0;temp<size;temp++)
        {
            m=0;
            n=0;
            for (row=0;row<size;row++)
            {
                for (column=0;column<size;column++)
                {
                    minor[row][column]=0;
                    if (row!= 0 && column != temp)
                    {
                       minor[m][n]=matrix[row][column];
                       if (n<(size-2))
                          n++;
                       else
                       {
                           n=0;
                           m++;
                       }
                    }
                }
            }
            det=det + start * (matrix[0][temp] * determinant(minor,size-1)↩
               );
            start=-1 * start;
        }
    }
```

```
43
44      return (det);
45  }
46
47
48  int main()
49  {
50      int i,j;
51      float matrix[30][30],size,d;
52      printf("\nEnter the Order of the Matrix\n");
53      scanf("%f",&size);
54      printf("\nEnter the Elements in the Matrix\n");
55      for (i=0;i<size;i++)
56      {
57          for (j=0;j<size;j++)
58          {
59               scanf("%f",&matrix[i][j]);
60          }
61          printf("\n");
62      }
63      d=determinant(matrix,size);
64      printf("\nDeterminant of the Matrix is = %f\n",d);
65
66
67  }
```

manual chio's method to solve determinant.

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 69 \end{bmatrix}$$

ANSWER:

$$1/1 = \begin{bmatrix} \begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 4 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 5 \end{vmatrix} \\ \begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 6 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 10 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 15 \end{vmatrix} \\ \begin{vmatrix} 1 & 1 \\ 1 & 4 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 10 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 20 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 35 \end{vmatrix} \\ \begin{vmatrix} 1 & 1 \\ 1 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 15 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 35 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 69 \end{vmatrix} \end{bmatrix},$$

$$1/1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 9 & 14 \\ 3 & 9 & 19 & 34 \\ 4 & 14 & 34 & 68 \end{bmatrix}$$

$$1/1 = \begin{bmatrix} \begin{vmatrix} 1 & 2 \\ 2 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 3 & 9 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 4 & 14 \end{vmatrix} \\ \begin{vmatrix} 1 & 3 \\ 2 & 9 \end{vmatrix} & \begin{vmatrix} 1 & 3 \\ 3 & 19 \end{vmatrix} & \begin{vmatrix} 1 & 3 \\ 4 & 34 \end{vmatrix} \\ \begin{vmatrix} 1 & 4 \\ 2 & 14 \end{vmatrix} & \begin{vmatrix} 1 & 4 \\ 3 & 34 \end{vmatrix} & \begin{vmatrix} 1 & 4 \\ 4 & 68 \end{vmatrix} \end{bmatrix},$$

$$1/1 = \begin{bmatrix} 1 & 3 & 6 \\ 3 & 10 & 22 \\ 6 & 22 & 52 \end{bmatrix}$$

divide R3 and C3 2

$$1/1 = \begin{bmatrix} 1 & 3 & 3 \\ 3 & 10 & 11 \\ 3 & 11 & 13 \end{bmatrix}$$

$$4 = \begin{bmatrix} \begin{vmatrix} 1 & 3 \\ 3 & 10 \end{vmatrix} & \begin{vmatrix} 1 & 3 \\ 3 & 11 \end{vmatrix} \\ \begin{vmatrix} 1 & 3 \\ 3 & 11 \end{vmatrix} & \begin{vmatrix} 1 & 3 \\ 3 & 11 \end{vmatrix} \end{bmatrix},$$

$$4 = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

$$4 = \begin{bmatrix} 4 - 4 = 0 \end{bmatrix}$$