Lab -

DIP : Design and Analysis of Algorithm

AIM : Write a program to solve the following Algorithm

1. Dijkstra algorithm

2. All Pairs Shortest path - Floyd - Warshal Algorithm

3. Kruskal's algorithm

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

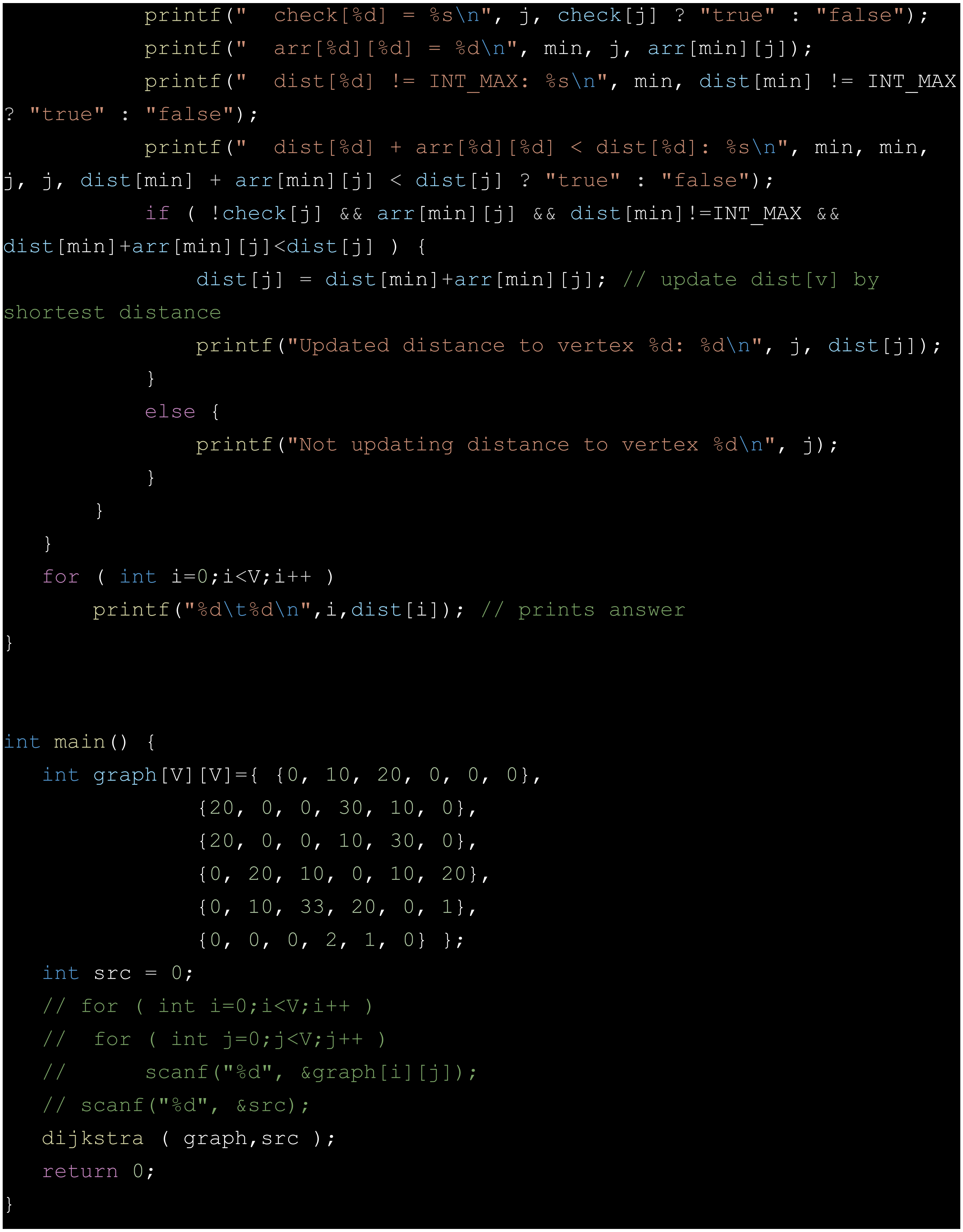
Roll no : MT001

Date : Jan 2024

Program 01 : Dijkstra algorithm

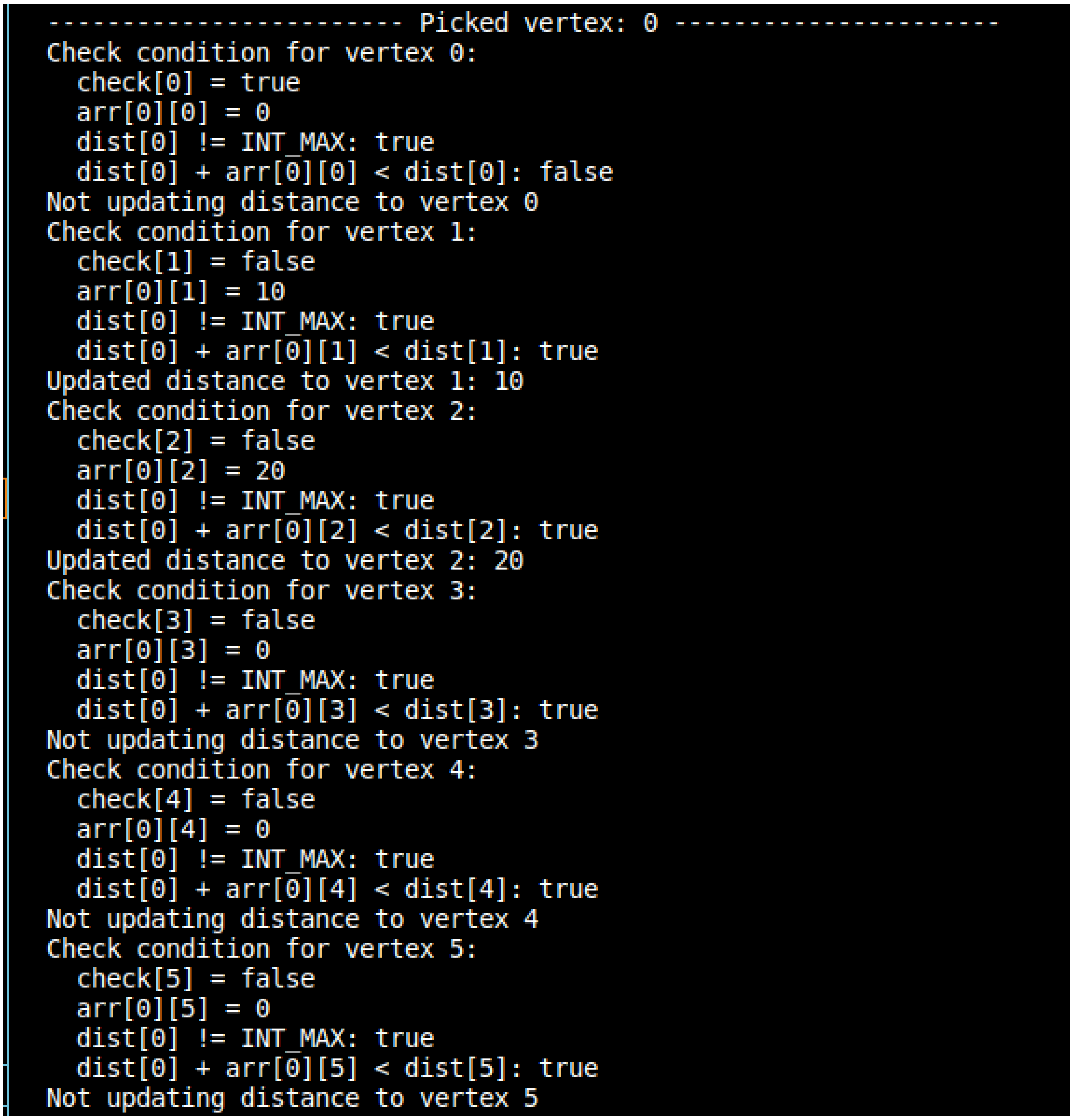
- Code :

```
// Dijkstra algorithm

#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define V 6

int mindist ( int dist[], bool check[] ) {
    int min = INT_MAX, minindex;
    for ( int i=0;i<V;i++ )
        if ( check[i]==false && dist[i]<=min ) {
            min = dist[i]; // update when min is greater than dist recorded
            minindex = i;
        }
    return minindex;
}

void dijkstra(int arr[V][V], int src) {
    int dist[V]; // shortest distance from src to i.
    bool check[V]; // check[i] will true if vertex i is included in
shortest path
    for ( int i=0;i<V;i++ ) {
        dist[i] = INT_MAX;
        check[i] = false;
    }
    dist[src] = 0; // cause starting node
    for ( int i=0;i<V-1;i++ ) {
        int min = mindist(dist, check); // pick the minimum distance
vertex.
        check[min] = true; // mark the picked vertex as processed
        printf("------------------------ Picked vertex: %d
----------------------\n", min);
        for ( int j=0;j<V;j++ ) {
            printf("Check condition for vertex %d:\n", j);
```

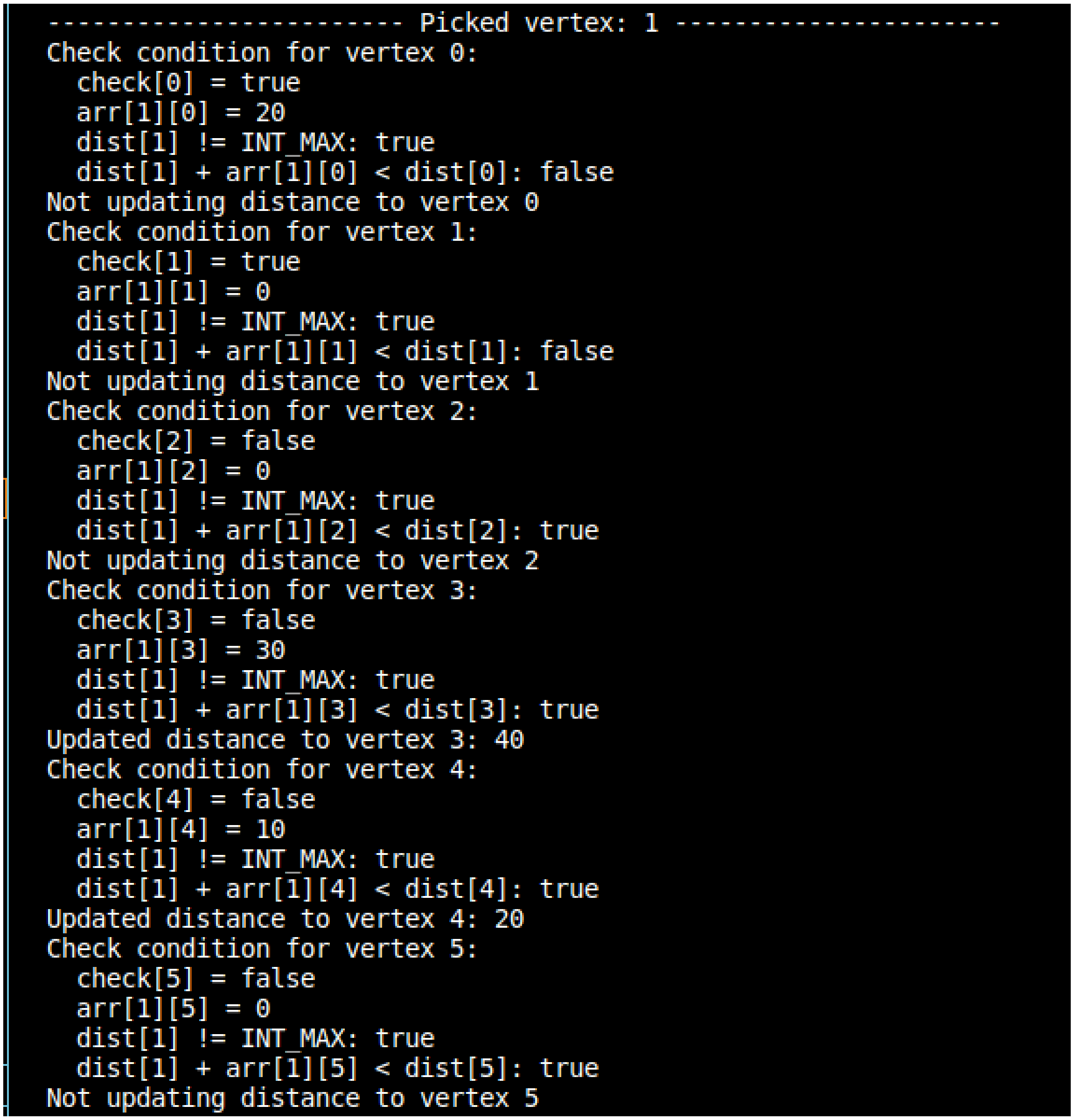


```
           printf("  check[%d] = %s\n", j, check[j] ? "true" : "false");
           printf("  arr[%d][%d] = %d\n", min, j, arr[min][j]);
           printf("  dist[%d] != INT_MAX: %s\n", min, dist[min] != INT_MAX
? "true" : "false");
           printf("  dist[%d] + arr[%d][%d] < dist[%d]: %s\n", min, min,
j, j, dist[min] + arr[min][j] < dist[j] ? "true" : "false");
           if ( !check[j] && arr[min][j] && dist[min]!=INT_MAX &&
dist[min]+arr[min][j]<dist[j] ) {
               dist[j] = dist[min]+arr[min][j]; // update dist[v] by
shortest distance
               printf("Updated distance to vertex %d: %d\n", j, dist[j]);
           }
           else {
               printf("Not updating distance to vertex %d\n", j);
           }
       }
   }
   for ( int i=0;i<V;i++ )
       printf("%d\t%d\n",i,dist[i]); // prints answer
}


int main() {
   int graph[V][V]={ {0, 10, 20, 0, 0, 0},
               {20, 0, 0, 30, 10, 0},
               {20, 0, 0, 10, 30, 0},
               {0, 20, 10, 0, 10, 20},
               {0, 10, 33, 20, 0, 1},
               {0, 0, 0, 2, 1, 0} };
   int src = 0;
   // for ( int i=0;i<V;i++ )
   //  for ( int j=0;j<V;j++ )
   //      scanf("%d", &graph[i][j]);
   // scanf("%d", &src);
   dijkstra ( graph,src );
   return 0;
}
```

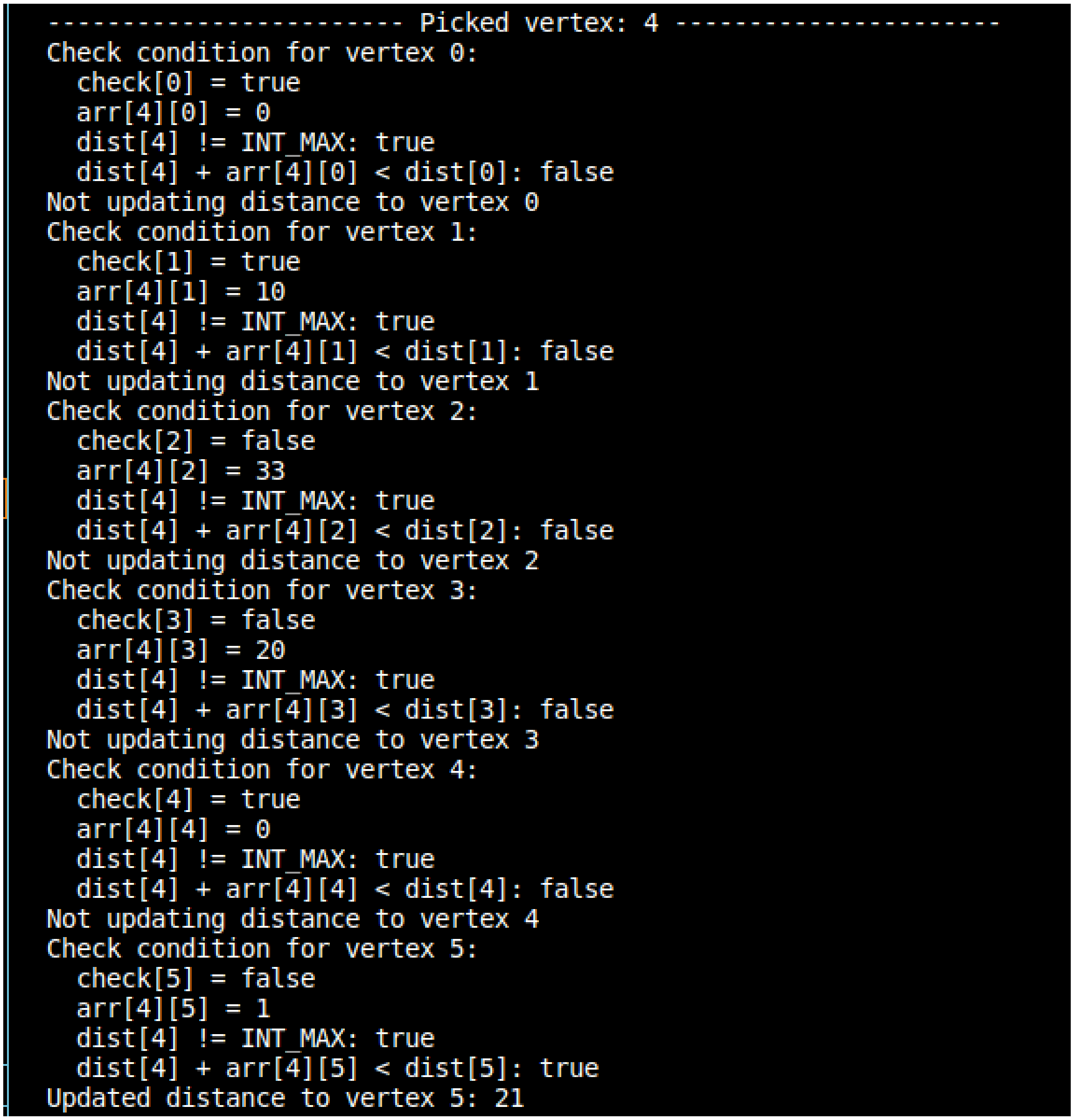
- Output Screen-shots / Tracing :



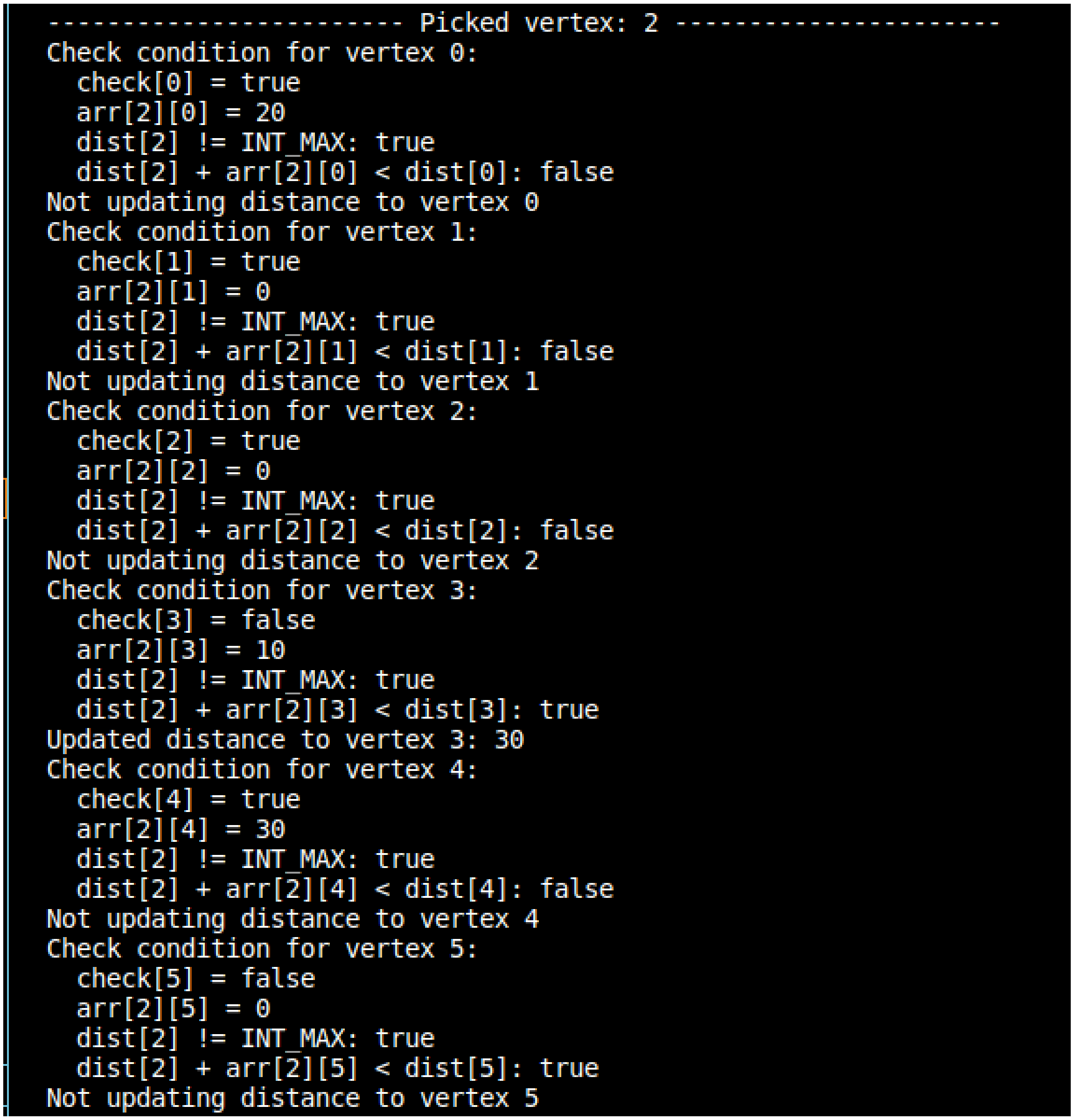
```
------------------------ Picked vertex: 0 ----------------------
Check condition for vertex 0:
  check[0] = true
  arr[0][0] = 0
  dist[0] != INT_MAX: true
  dist[0] + arr[0][0] < dist[0]: false
Not updating distance to vertex 0
Check condition for vertex 1:
  check[1] = false
  arr[0][1] = 10
  dist[0] != INT_MAX: true
  dist[0] + arr[0][1] < dist[1]: true
Updated distance to vertex 1: 10
Check condition for vertex 2:
  check[2] = false
  arr[0][2] = 20
  dist[0] != INT_MAX: true
  dist[0] + arr[0][2] < dist[2]: true
Updated distance to vertex 2: 20
Check condition for vertex 3:
  check[3] = false
  arr[0][3] = 0
  dist[0] != INT_MAX: true
  dist[0] + arr[0][3] < dist[3]: true
Not updating distance to vertex 3
Check condition for vertex 4:
  check[4] = false
  arr[0][4] = 0
  dist[0] != INT_MAX: true
  dist[0] + arr[0][4] < dist[4]: true
Not updating distance to vertex 4
Check condition for vertex 5:
  check[5] = false
  arr[0][5] = 0
  dist[0] != INT_MAX: true
  dist[0] + arr[0][5] < dist[5]: true
Not updating distance to vertex 5
```



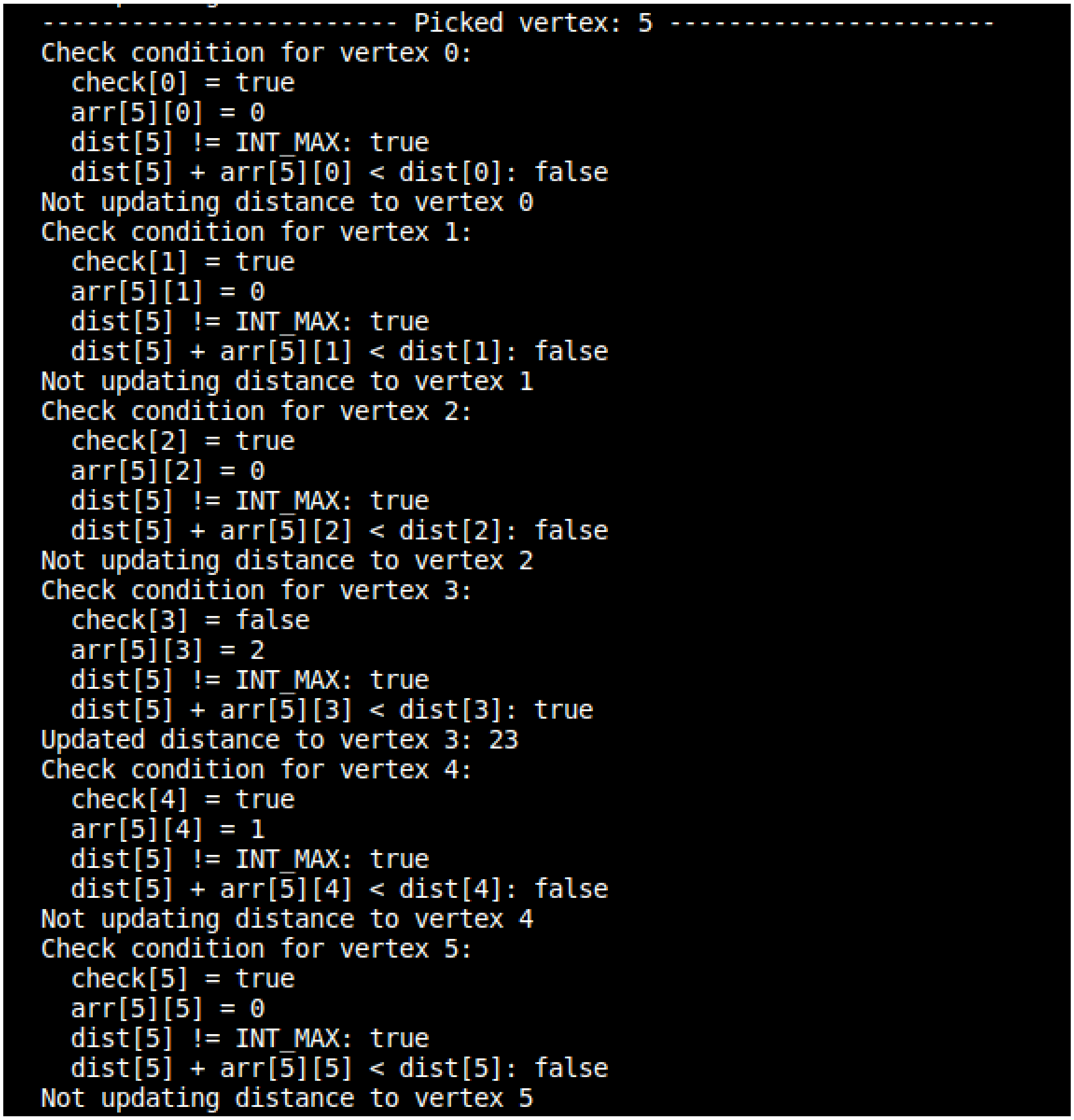
```
------------------------ Picked vertex: 1 ----------------------
Check condition for vertex 0:
  check[0] = true
  arr[1][0] = 20
  dist[1] != INT_MAX: true
  dist[1] + arr[1][0] < dist[0]: false
Not updating distance to vertex 0
Check condition for vertex 1:
  check[1] = true
  arr[1][1] = 0
  dist[1] != INT_MAX: true
  dist[1] + arr[1][1] < dist[1]: false
Not updating distance to vertex 1
Check condition for vertex 2:
  check[2] = false
  arr[1][2] = 0
  dist[1] != INT_MAX: true
  dist[1] + arr[1][2] < dist[2]: true
Not updating distance to vertex 2
Check condition for vertex 3:
  check[3] = false
  arr[1][3] = 30
  dist[1] != INT_MAX: true
  dist[1] + arr[1][3] < dist[3]: true
Updated distance to vertex 3: 40
Check condition for vertex 4:
  check[4] = false
  arr[1][4] = 10
  dist[1] != INT_MAX: true
  dist[1] + arr[1][4] < dist[4]: true
Updated distance to vertex 4: 20
Check condition for vertex 5:
  check[5] = false
  arr[1][5] = 0
  dist[1] != INT_MAX: true
  dist[1] + arr[1][5] < dist[5]: true
Not updating distance to vertex 5
```



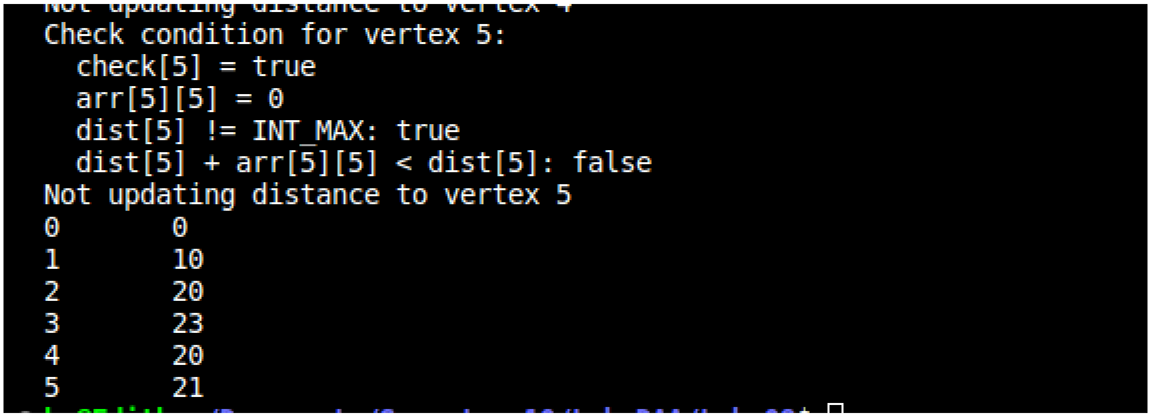
```
------------------------ Picked vertex: 4 ----------------------
Check condition for vertex 0:
  check[0] = true
  arr[4][0] = 0
  dist[4] != INT_MAX: true
  dist[4] + arr[4][0] < dist[0]: false
Not updating distance to vertex 0
Check condition for vertex 1:
  check[1] = true
  arr[4][1] = 10
  dist[4] != INT_MAX: true
  dist[4] + arr[4][1] < dist[1]: false
Not updating distance to vertex 1
Check condition for vertex 2:
  check[2] = false
  arr[4][2] = 33
  dist[4] != INT_MAX: true
  dist[4] + arr[4][2] < dist[2]: false
Not updating distance to vertex 2
Check condition for vertex 3:
  check[3] = false
  arr[4][3] = 20
  dist[4] != INT_MAX: true
  dist[4] + arr[4][3] < dist[3]: false
Not updating distance to vertex 3
Check condition for vertex 4:
  check[4] = true
  arr[4][4] = 0
  dist[4] != INT_MAX: true
  dist[4] + arr[4][4] < dist[4]: false
Not updating distance to vertex 4
Check condition for vertex 5:
  check[5] = false
  arr[4][5] = 1
  dist[4] != INT_MAX: true
  dist[4] + arr[4][5] < dist[5]: true
Updated distance to vertex 5: 21
```



```
------------------------ Picked vertex: 2 ----------------------
Check condition for vertex 0:
  check[0] = true
  arr[2][0] = 20
  dist[2] != INT_MAX: true
  dist[2] + arr[2][0] < dist[0]: false
Not updating distance to vertex 0
Check condition for vertex 1:
  check[1] = true
  arr[2][1] = 0
  dist[2] != INT_MAX: true
  dist[2] + arr[2][1] < dist[1]: false
Not updating distance to vertex 1
Check condition for vertex 2:
  check[2] = true
  arr[2][2] = 0
  dist[2] != INT_MAX: true
  dist[2] + arr[2][2] < dist[2]: false
Not updating distance to vertex 2
Check condition for vertex 3:
  check[3] = false
  arr[2][3] = 10
  dist[2] != INT_MAX: true
  dist[2] + arr[2][3] < dist[3]: true
Updated distance to vertex 3: 30
Check condition for vertex 4:
  check[4] = true
  arr[2][4] = 30
  dist[2] != INT_MAX: true
  dist[2] + arr[2][4] < dist[4]: false
Not updating distance to vertex 4
Check condition for vertex 5:
  check[5] = false
  arr[2][5] = 0
  dist[2] != INT_MAX: true
  dist[2] + arr[2][5] < dist[5]: true
Not updating distance to vertex 5
```



```
------------------------ Picked vertex: 5 ----------------------
Check condition for vertex 0:
  check[0] = true
  arr[5][0] = 0
  dist[5] != INT_MAX: true
  dist[5] + arr[5][0] < dist[0]: false
Not updating distance to vertex 0
Check condition for vertex 1:
  check[1] = true
  arr[5][1] = 0
  dist[5] != INT_MAX: true
  dist[5] + arr[5][1] < dist[1]: false
Not updating distance to vertex 1
Check condition for vertex 2:
  check[2] = true
  arr[5][2] = 0
  dist[5] != INT_MAX: true
  dist[5] + arr[5][2] < dist[2]: false
Not updating distance to vertex 2
Check condition for vertex 3:
  check[3] = false
  arr[5][3] = 2
  dist[5] != INT_MAX: true
  dist[5] + arr[5][3] < dist[3]: true
Updated distance to vertex 3: 23
Check condition for vertex 4:
  check[4] = true
  arr[5][4] = 1
  dist[5] != INT_MAX: true
  dist[5] + arr[5][4] < dist[4]: false
Not updating distance to vertex 4
Check condition for vertex 5:
  check[5] = true
  arr[5][5] = 0
  dist[5] != INT_MAX: true
  dist[5] + arr[5][5] < dist[5]: false
Not updating distance to vertex 5
```



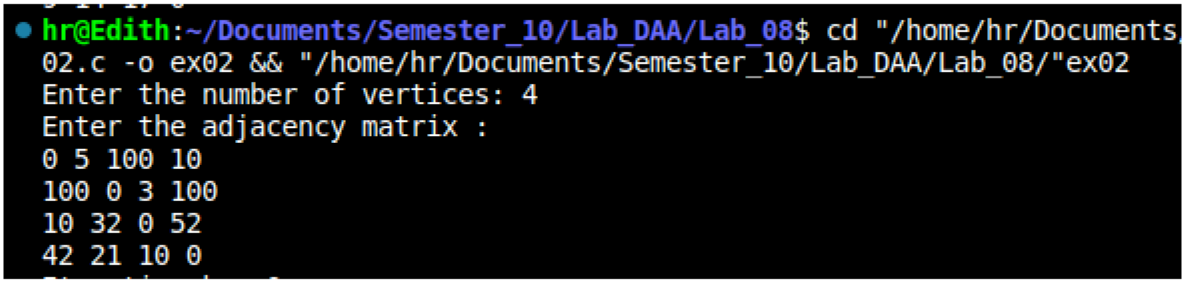
```
Not updating distance to vertex 4
Check condition for vertex 5:
  check[5] = true
  arr[5][5] = 0
  dist[5] != INT_MAX: true
  dist[5] + arr[5][5] < dist[5]: false
Not updating distance to vertex 5
0       0
1       10
2       20
3       23
4       20
5       21
```

Program 02 : All Pairs Shortest path - Floyd - Warshal Algorithm

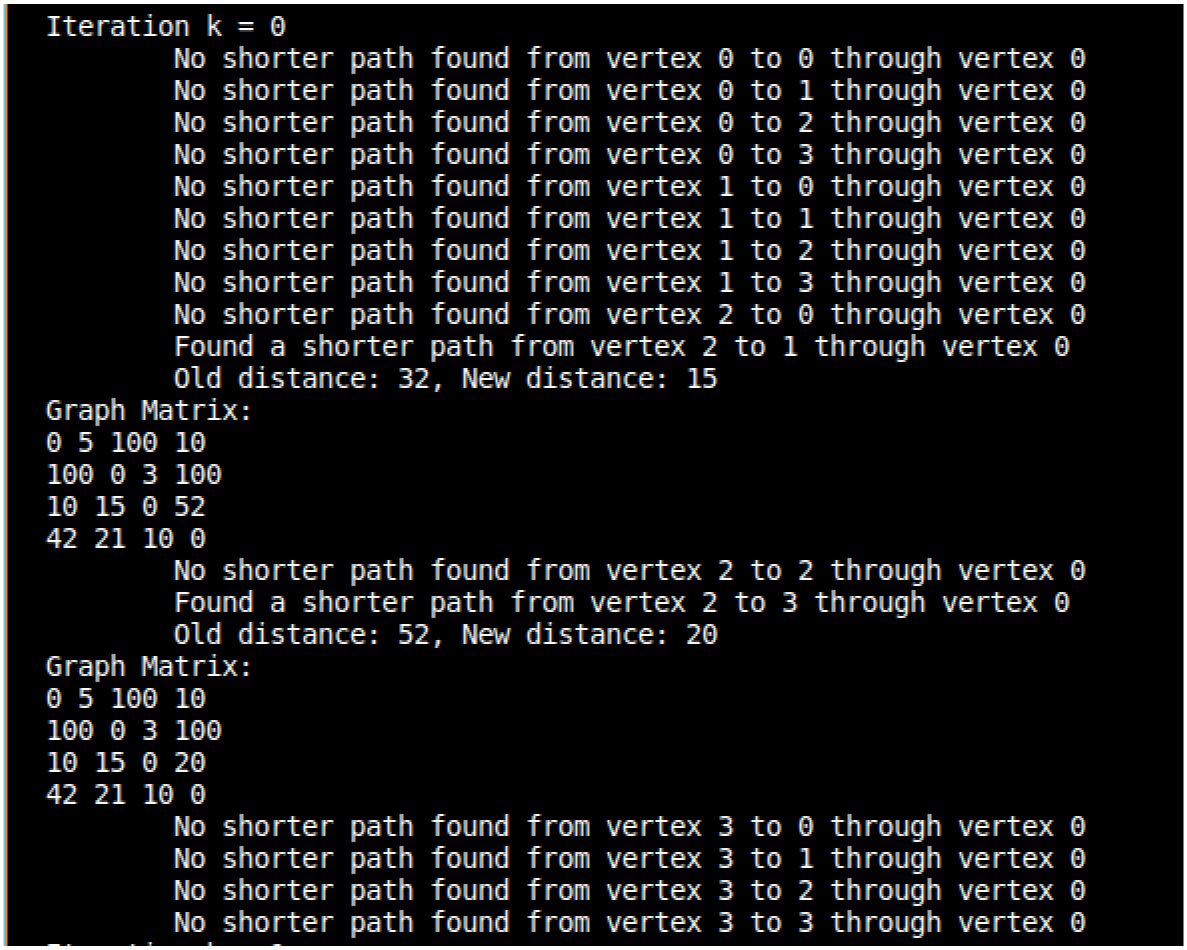
- Code :

```
// All Pairs Shortest path - Floyd - Warshal Algorithm

#include <stdio.h>

#define MAX_VERTICES 100

void printGraph(int graph[MAX_VERTICES][MAX_VERTICES], int n) {
    printf("Graph Matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
}

void floydWarshall(int graph[MAX_VERTICES][MAX_VERTICES], int n) {
    int i, j, k;
    for (k = 0; k < n; k++) {
        printf("Iteration k = %d\n", k);
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (graph[i][j] > graph[i][k] + graph[k][j]) {
                    printf("        Found a shorter path from vertex %d to
%d through vertex %d\n", i, j, k);
                    printf("        Old distance: %d, New distance: %d\n",
graph[i][j], graph[i][k] + graph[k][j]);
                    graph[i][j] = graph[i][k] + graph[k][j];
                    printGraph(graph, n);
                } else {
                    printf("        No shorter path found from vertex %d to
%d through vertex %d\n", i, j, k);
                }
            }
        }
    }
}

int main(void) {
    int n, i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix :\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    floydWarshall(graph, n);

    printf("Shortest path matrix is:\n");
    printGraph(graph, n);
    return 0;
}
```

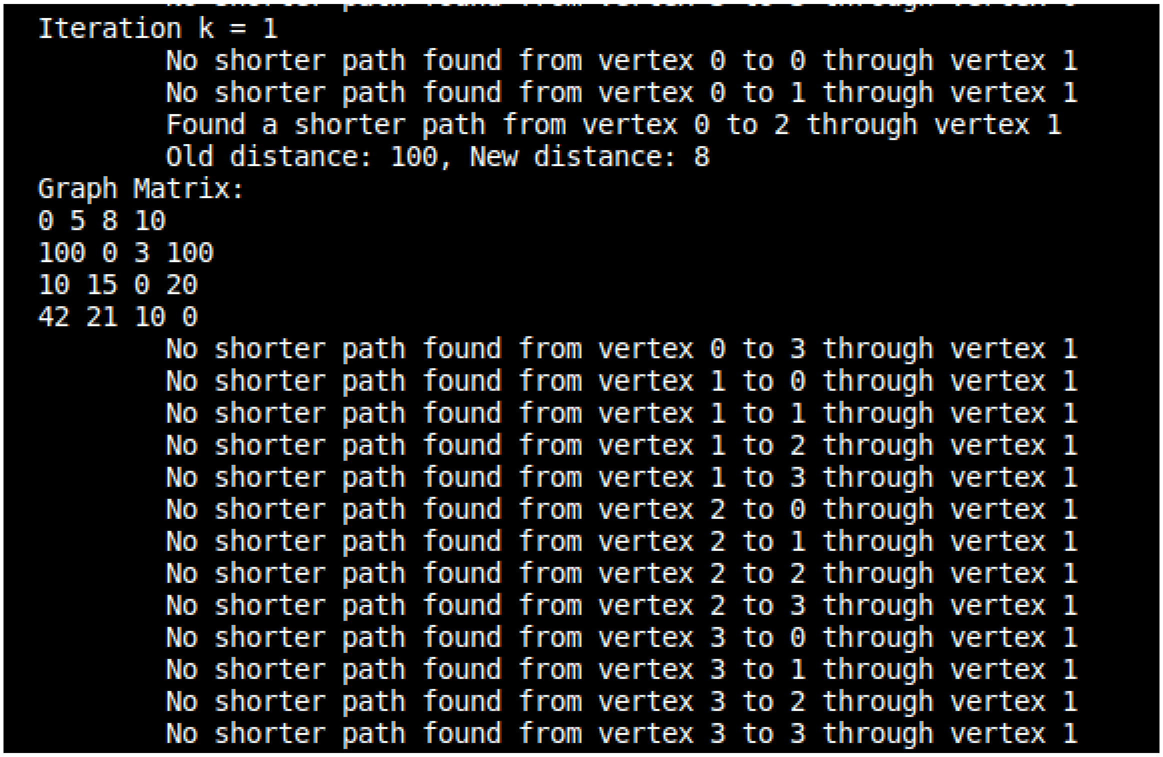
- Output Screen-shots / Tracing :



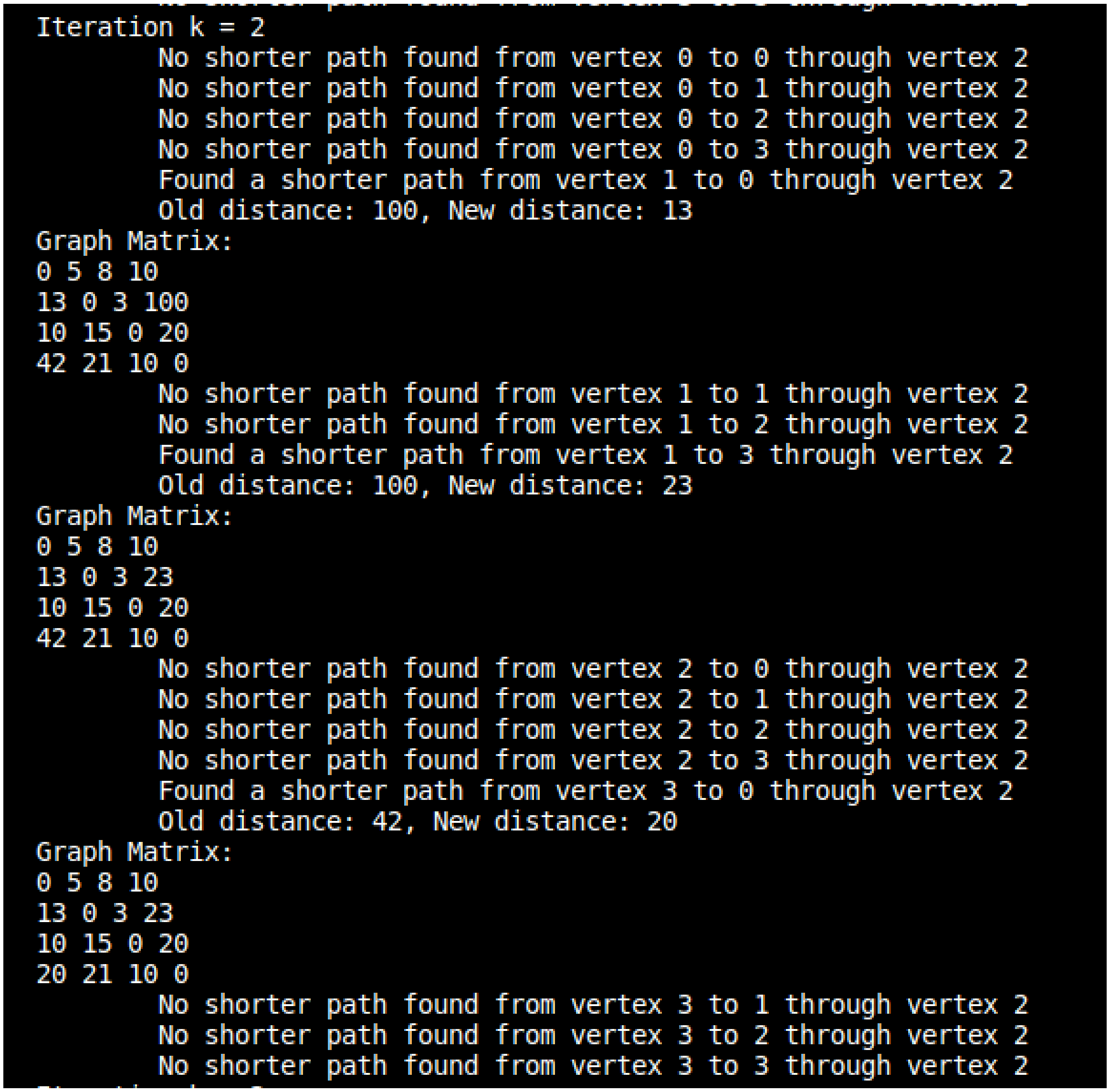
```
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_08$ cd "/home/hr/Documents
  02.c -o ex02 && "/home/hr/Documents/Semester_10/Lab_DAA/Lab_08/"ex02
  Enter the number of vertices: 4
  Enter the adjacency matrix :
  0 5 100 10
  100 0 3 100
  10 32 0 52
  42 21 10 0
```



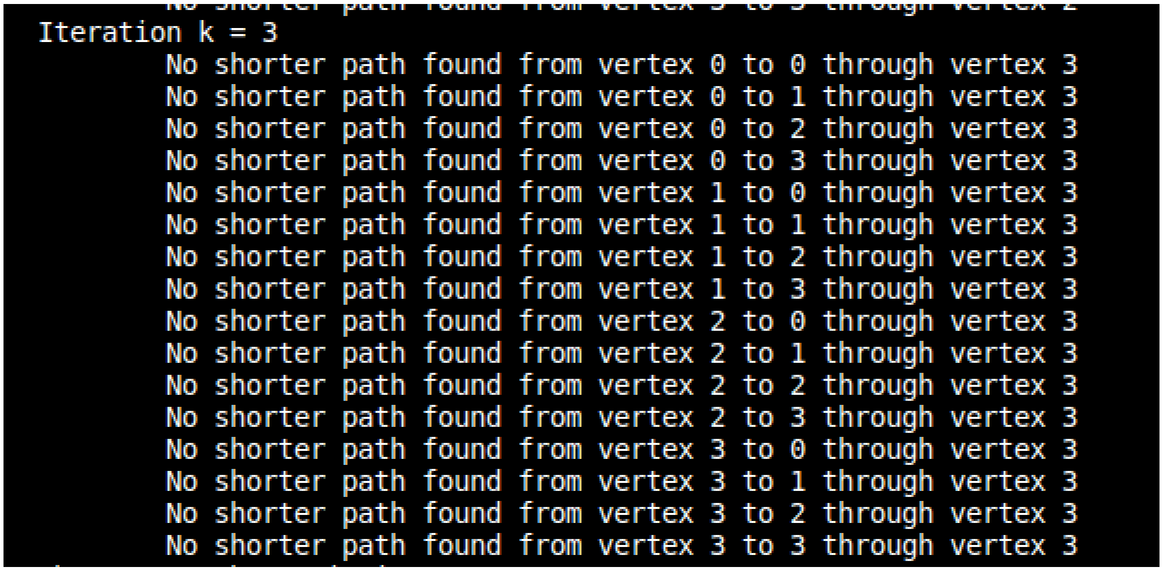
```
Iteration k = 0
        No shorter path found from vertex 0 to 0 through vertex 0
        No shorter path found from vertex 0 to 1 through vertex 0
        No shorter path found from vertex 0 to 2 through vertex 0
        No shorter path found from vertex 0 to 3 through vertex 0
        No shorter path found from vertex 1 to 0 through vertex 0
        No shorter path found from vertex 1 to 1 through vertex 0
        No shorter path found from vertex 1 to 2 through vertex 0
        No shorter path found from vertex 1 to 3 through vertex 0
        No shorter path found from vertex 2 to 0 through vertex 0
        Found a shorter path from vertex 2 to 1 through vertex 0
        Old distance: 32, New distance: 15
Graph Matrix:
0 5 100 10
100 0 3 100
10 15 0 52
42 21 10 0
        No shorter path found from vertex 2 to 2 through vertex 0
        Found a shorter path from vertex 2 to 3 through vertex 0
        Old distance: 52, New distance: 20
Graph Matrix:
0 5 100 10
100 0 3 100
10 15 0 20
42 21 10 0
        No shorter path found from vertex 3 to 0 through vertex 0
        No shorter path found from vertex 3 to 1 through vertex 0
        No shorter path found from vertex 3 to 2 through vertex 0
        No shorter path found from vertex 3 to 3 through vertex 0
```



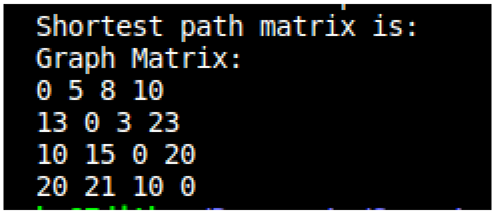
```
Iteration k = 1
        No shorter path found from vertex 0 to 0 through vertex 1
        No shorter path found from vertex 0 to 1 through vertex 1
        Found a shorter path from vertex 0 to 2 through vertex 1
        Old distance: 100, New distance: 8
Graph Matrix:
0 5 8 10
100 0 3 100
10 15 0 20
42 21 10 0
        No shorter path found from vertex 0 to 3 through vertex 1
        No shorter path found from vertex 1 to 0 through vertex 1
        No shorter path found from vertex 1 to 1 through vertex 1
        No shorter path found from vertex 1 to 2 through vertex 1
        No shorter path found from vertex 1 to 3 through vertex 1
        No shorter path found from vertex 2 to 0 through vertex 1
        No shorter path found from vertex 2 to 1 through vertex 1
        No shorter path found from vertex 2 to 2 through vertex 1
        No shorter path found from vertex 2 to 3 through vertex 1
        No shorter path found from vertex 3 to 0 through vertex 1
        No shorter path found from vertex 3 to 1 through vertex 1
        No shorter path found from vertex 3 to 2 through vertex 1
        No shorter path found from vertex 3 to 3 through vertex 1
```



```
Iteration k = 2
        No shorter path found from vertex 0 to 0 through vertex 2
        No shorter path found from vertex 0 to 1 through vertex 2
        No shorter path found from vertex 0 to 2 through vertex 2
        No shorter path found from vertex 0 to 3 through vertex 2
        Found a shorter path from vertex 1 to 0 through vertex 2
        Old distance: 100, New distance: 13
Graph Matrix:
0 5 8 10
13 0 3 100
10 15 0 20
42 21 10 0
        No shorter path found from vertex 1 to 1 through vertex 2
        No shorter path found from vertex 1 to 2 through vertex 2
        Found a shorter path from vertex 1 to 3 through vertex 2
        Old distance: 100, New distance: 23
Graph Matrix:
0 5 8 10
13 0 3 23
10 15 0 20
42 21 10 0
        No shorter path found from vertex 2 to 0 through vertex 2
        No shorter path found from vertex 2 to 1 through vertex 2
        No shorter path found from vertex 2 to 2 through vertex 2
        No shorter path found from vertex 2 to 3 through vertex 2
        Found a shorter path from vertex 3 to 0 through vertex 2
        Old distance: 42, New distance: 20
Graph Matrix:
0 5 8 10
13 0 3 23
10 15 0 20
20 21 10 0
        No shorter path found from vertex 3 to 1 through vertex 2
        No shorter path found from vertex 3 to 2 through vertex 2
        No shorter path found from vertex 3 to 3 through vertex 2
```



```
Iteration k = 3
        No shorter path found from vertex 0 to 0 through vertex 3
        No shorter path found from vertex 0 to 1 through vertex 3
        No shorter path found from vertex 0 to 2 through vertex 3
        No shorter path found from vertex 0 to 3 through vertex 3
        No shorter path found from vertex 1 to 0 through vertex 3
        No shorter path found from vertex 1 to 1 through vertex 3
        No shorter path found from vertex 1 to 2 through vertex 3
        No shorter path found from vertex 1 to 3 through vertex 3
        No shorter path found from vertex 2 to 0 through vertex 3
        No shorter path found from vertex 2 to 1 through vertex 3
        No shorter path found from vertex 2 to 2 through vertex 3
        No shorter path found from vertex 2 to 3 through vertex 3
        No shorter path found from vertex 3 to 0 through vertex 3
        No shorter path found from vertex 3 to 1 through vertex 3
        No shorter path found from vertex 3 to 2 through vertex 3
        No shorter path found from vertex 3 to 3 through vertex 3
```



```
Shortest path matrix is:
Graph Matrix:
0 5 8 10
13 0 3 23
10 15 0 20
20 21 10 0
```

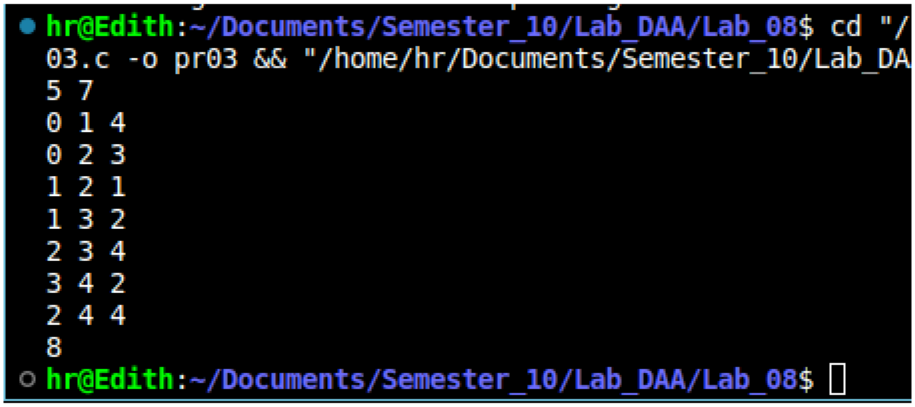
Program 03 : Kruskal's algorithm

- Code :

```
// Kruskal's algorithm

#include <stdio.h>
#include <stdlib.h>

struct Edge {
    int src;
    int dest;
    int wght;
};
struct Edge arr[3000];
int parent[100000];

void sort_by_weight(int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j].wght > arr[j + 1].wght) {
                struct Edge temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int find_par(int a) {
    if (parent[a] == -1)
        return a;
    return (parent[a] = find_par(parent[a]));
}

void kruskal(int edges) {
    int sum = 0;
    int a, b;
    sort_by_weight(edges);
    for (int i = 0; i < edges; i++) {
```

```c
        a = find_par(arr[i].src);
        b = find_par(arr[i].dest);
        if (a != b) {
            sum += arr[i].wght;
            parent[a] = b;
        }
    }
    printf("%d\n", sum);
}

int main() {
    int n, m;
    int source, destination, weight;
    scanf("%d", &n);
    scanf("%d", &m);
    for (int i = 0; i < n; i++)
        parent[i] = -1;
    for (int i = 0; i < m; i++) {
        scanf("%d %d %d", &source, &destination, &weight);
        arr[i].src = source;
        arr[i].dest = destination;
        arr[i].wght = weight;
    }
    kruskal(m);
    return 0;
}
```

- Output Screen-shots / Tracing :



```
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_08$ cd "/
  03.c -o pr03 && "/home/hr/Documents/Semester_10/Lab_DA
  5 7
  0 1 4
  0 2 3
  1 2 1
  1 3 2
  2 3 4
  3 4 2
  2 4 4
  8
○ hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_08$ ▯
```