Q4

מגישות - שני גולומב 325184653 , יובל בצר 212725048

הרצה רגילה:

ללא משקלים שליליים-

```
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ gcc -o dijkstra dijkstra.c
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ ./dijkstra
Enter the number of vertices: 3
Enter the weights of the graph:
Weight of edge between vertex 0 and vertex 0 (enter 0 if no edge): 0
Weight of edge between vertex 0 and vertex 1 (enter 0 if no edge): 1
Weight of edge between vertex 0 and vertex 2 (enter 0 if no edge): 2
Weight of edge between vertex 1 and vertex 0 (enter 0 if no edge): 4
Weight of edge between vertex 1 and vertex 1 (enter 0 if no edge): 5
Weight of edge between vertex 1 and vertex 2 (enter 0 if no edge): 7
Weight of edge between vertex 2 and vertex 0 (enter 0 if no edge): 1
Weight of edge between vertex 2 and vertex 1 (enter 0 if no edge): 0
Weight of edge between vertex 2 and vertex 2 (enter 0 if no edge): 8
Vertex           Distance from Source
0                        0
1                        1
2                        2
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$
```

עם משקלים שליליים-

```
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ ./dijkstra
Enter the number of vertices: 3
Enter the weights of the graph:
Weight of edge between vertex 0 and vertex 0 (enter 0 if no edge): 0
Weight of edge between vertex 0 and vertex 1 (enter 0 if no edge): 2
Weight of edge between vertex 0 and vertex 2 (enter 0 if no edge): -3
Cant enter a negative weight
Segmentation fault (core dumped)
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$
```

מקרה ראשון:

במקרה זה הרצנו gcov עבור גרף שבו הכנסנו צלע עם משקל שלילי- דבר הגורם לסיום ריצת התוכנית מכיוון שדייקסטרה לא תומך במשקלים שליליים.

ניתן לראות כי רק 32.61% כוסו, מכיוון שהכנסת המשקל השלילי גרמה לסיום התוכנית לפני הקריאה לפונקציות האלו, ולכן הם לא נקראו ולא כוסו.

```
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ gcov dijkstra.c
File 'dijkstra.c'
Lines executed:32.61% of 46
Creating 'dijkstra.c.gcov'

Lines executed:32.61% of 46
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ cat dijkstra.c.gcov
        -:    0:Source:dijkstra.c
        -:    0:Graph:dijkstra.gcno
        -:    0:Data:dijkstra.gcda
        -:    0:Runs:1
        -:    1:#include <limits.h>
        -:    2:#include <stdbool.h>
        -:    3:#include <stdio.h>
        -:    4:#include <stdlib.h>
        -:    5:
        -:    6:// Function to find the vertex with minimum distance value, from the set of vertices n
ot yet included in the shortest path tree
    #####:    7:int minDistance(int dist[], bool sptSet[], int V)
        -:    8:{
    #####:    9:    int min = INT_MAX, min_index;
    #####:   10:    for (int v = 0; v < V; v++)
    #####:   11:        if (sptSet[v] == false && dist[v] <= min)
    #####:   12:            min = dist[v], min_index = v;
    #####:   13:    return min_index;
        -:   14:}
        -:   15:
        -:   16:// Function to print the constructed distance array
    #####:   17:void printSolution(int dist[], int V)
        -:   18:{
    #####:   19:    printf("Vertex \t\t Distance from Source\n");
    #####:   20:    for (int i = 0; i < V; i++)
    #####:   21:        printf("%d \t\t\t %d\n", i, dist[i]);
    #####:   22:}
        -:   23:
        -:   24:// Function to implement Dijkstra's single source shortest path algorithm for a graph
represented using an adjacency matrix
    #####:   25:void dijkstra(int **graph, int V, int src)
    #####:   26:{
    #####:   27:    int dist[V];
    #####:   28:    bool sptSet[V];
        -:   29:
    #####:   30:    for (int i = 0; i < V; i++)
    #####:   31:        dist[i] = INT_MAX, sptSet[i] = false;
        -:   32:
    #####:   33:    dist[src] = 0;
        -:   34:
    #####:   35:    for (int count = 0; count < V - 1; count++)
        -:   36:    {
    #####:   37:        int u = minDistance(dist, sptSet, V);
    #####:   38:        sptSet[u] = true;
        -:   39:
    #####:   40:        for (int v = 0; v < V; v++)
    #####:   41:            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][
v] < dist[v])
    #####:   42:                dist[v] = dist[u] + graph[u][v];
        -:   43:    }
        -:   44:
    #####:   45:    printSolution(dist, V);
    #####:   46:}
        -:   47:
        -:   48:// Driver's code
        1:   49:int main()
        -:   50:{
        -:   51:    int V;
        1:   52:    printf("Enter the number of vertices: ");
        1:   53:    scanf("%d", &V);
        -:   54:
        -:   55:    // Dynamically allocate memory for the graph (adjacency matrix)
        1:   56:    int **graph = (int **)malloc(V * sizeof(int *));
        3:   57:    for (int i = 0; i < V; i++)
        2:   58:        graph[i] = (int *)malloc(V * sizeof(int));
        -:   59:
```

```
        -:   48:// Driver's code
        1:   49:int main()
        -:   50:{
        -:   51:    int V;
        1:   52:    printf("Enter the number of vertices: ");
        1:   53:    scanf("%d", &V);
        -:   54:
        -:   55:    // Dynamically allocate memory for the graph (adjacency matrix)
        1:   56:    int **graph = (int **)malloc(V * sizeof(int *));
        3:   57:    for (int i = 0; i < V; i++)
        2:   58:        graph[i] = (int *)malloc(V * sizeof(int));
        -:   59:
        1:   60:    printf("Enter the weights of the graph:\n");
       1*:   61:    for (int i = 0; i < V; i++) {
        2:   62:        for (int j = 0; j < V; j++) {
        2:   63:            printf("Weight of edge between vertex %d and vertex %d (enter 0 if no edge
): ", i, j);
        -:   64:            int temp;
        2:   65:            scanf("%d", &temp);
        2:   66:            if (temp < 0) {
        1:   67:                printf("Cant enter a negative weight\n");
        -:   68:                //graph[i][j] = 0;
        1:   69:                exit(1);
        -:   70:            }
        1:   71:            graph[i][j] = temp;
        -:   72:        }
        -:   73:    }
        -:   74:
    #####:   75:    dijkstra(graph, V, 0);
        -:   76:
    #####:   77:    for (int i = 0; i < V; i++)
    #####:   78:        free(graph[i]);
    #####:   79:    free(graph);
        -:   80:
    #####:   81:    return 0;
        -:   82:}
```

<u>מקרה שני:</u>

במקרה זה הרצנו gcov עבור גרף שבו הכנסנו צלעות תקינות- כלומר ללא משקלים שליליים.

ניתן לראות כי 89.80% כוסו, מכיוון שלא הכנסנו צלעות עם משקלים שליליים אז כל הקוד בוצע וכוסה חוץ מהתנאי שקורה רק כאשר הוכנסו משקלים שליליים (משחרר את הזיכרון ומסיים את ריצת התוכנית), ולכן עבור גרף 'תקין' שורות אלו לא יבוצעו.

```
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ gcov dijkstra.c
File 'dijkstra.c'
Lines executed:89.80% of 49
Creating 'dijkstra.c.gcov'

Lines executed:89.80% of 49
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$ cat dijkstra.c.gcov
        -:    0:Source:dijkstra.c
        -:    0:Graph:dijkstra.gcno
        -:    0:Data:dijkstra.gcda
        -:    0:Runs:1
        -:    1:#include <limits.h>
        -:    2:#include <stdbool.h>
        -:    3:#include <stdio.h>
        -:    4:#include <stdlib.h>
        -:    5:
        -:    6:// Function to find the vertex with minimum distance value, from the set of vertices n
ot yet included in the shortest path tree
        2:    7:int minDistance(int dist[], bool sptSet[], int V)
        -:    8:{
        2:    9:    int min = INT_MAX, min_index;
        8:   10:    for (int v = 0; v < V; v++)
        6:   11:        if (sptSet[v] == false && dist[v] <= min)
        2:   12:            min = dist[v], min_index = v;
        2:   13:    return min_index;
        -:   14:}
        -:   15:
        -:   16:// Function to print the constructed distance array
        1:   17:void printSolution(int dist[], int V)
        -:   18:{
        1:   19:    printf("Vertex \t\t Distance from Source\n");
        4:   20:    for (int i = 0; i < V; i++)
        3:   21:        printf("%d \t\t\t %d\n", i, dist[i]);
        1:   22:}
```

```
        1:   25:void dijkstra(int **graph, int V, int src)
        1:   26:{
        1:   27:    int dist[V];
        1:   28:    bool sptSet[V];
        -:   29:
        4:   30:    for (int i = 0; i < V; i++)
        3:   31:        dist[i] = INT_MAX, sptSet[i] = false;
        -:   32:
        1:   33:    dist[src] = 0;
        -:   34:
        3:   35:    for (int count = 0; count < V - 1; count++)
        -:   36:    {
        2:   37:        int u = minDistance(dist, sptSet, V);
        2:   38:        sptSet[u] = true;
        -:   39:
        8:   40:        for (int v = 0; v < V; v++)
        6:   41:            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][
 v] < dist[v])
        2:   42:                dist[v] = dist[u] + graph[u][v];
        -:   43:    }
        -:   44:
        1:   45:    printSolution(dist, V);
        1:   46:}
        -:   47:
        -:   48:// Driver's code
        1:   49:int main()
        -:   50:{
        -:   51:    int V;
        1:   52:    printf("Enter the number of vertices: ");
        1:   53:    scanf("%d", &V);
        -:   54:
        -:   55:    // Dynamically allocate memory for the graph (adjacency matrix)
        1:   56:    int **graph = (int **)malloc(V * sizeof(int *));
        4:   57:    for (int i = 0; i < V; i++)
        3:   58:        graph[i] = (int *)malloc(V * sizeof(int));
        -:   59:
        1:   60:    printf("Enter the weights of the graph:\n");
        4:   61:    for (int i = 0; i < V; i++) {
       12:   62:        for (int j = 0; j < V; j++) {
        9:   63:            printf("Weight of edge between vertex %d and vertex %d (enter 0 if no edge
): ", i, j);
        -:   64:            int temp;
        9:   65:            scanf("%d", &temp);
        9:   66:            if (temp < 0) {
    #####:   67:                printf("Cant enter a negative weight\n");
    #####:   68:                for (int k = 0; k < V; k++){
    #####:   69:                    free(graph[k]);
    #####:   70:                    free(graph);
        -:   71:                }
    #####:   72:                exit(1);
        -:   73:            }
        9:   74:            graph[i][j] = temp;
        -:   75:        }
        -:   76:    }
        -:   77:
        1:   78:    dijkstra(graph, V, 0);
        -:   79:
        4:   80:    for (int i = 0; i < V; i++)
        3:   81:        free(graph[i]);
        1:   82:    free(graph);
        -:   83:
        1:   84:    return 0;
        -:   85:}
shani@shani:~/Documents/OperatingSystem/Ex1/Q4$
```