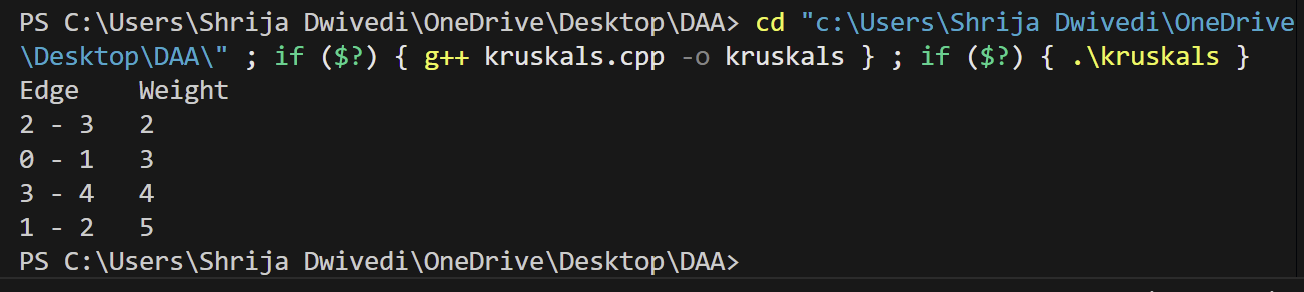
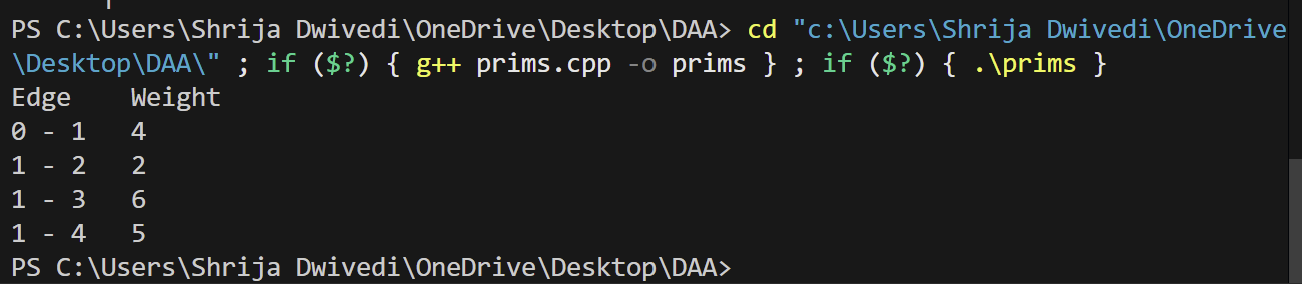
1. MST implementation using Kruskal’s Algorithm
2. #include <stdio.h>
4. #define V 5
5. #define E 7
7. typedef struct {
8. int start, end, weight;
9. } Edge;
11. typedef struct {
12. int parent;
13. int rank;
14. } Subset;
16. int findSet(Subset subsets[], int i) {
17. if (subsets[i].parent != i)
18. subsets[i].parent = findSet(subsets, subsets[i].parent);
19. return subsets[i].parent;
20. }
22. void unionSets(Subset subsets[], int x, int y) {
23. int xroot = findSet(subsets, x);
24. int yroot = findSet(subsets, y);
26. if (subsets[xroot].rank < subsets[yroot].rank)
27. subsets[xroot].parent = yroot;
28. else if (subsets[xroot].rank > subsets[yroot].rank)
29. subsets[yroot].parent = xroot;
30. else {
31. subsets[yroot].parent = xroot;
32. subsets[xroot].rank++;
33. }
34. }
36. void kruskalAlgorithm(Edge edges[]) {
37. Edge mst[V];
38. int e = 0, i = 0;
40. // bubble sort edges by weight
41. for (int a = 0; a < E - 1; a++)
42. for (int b = 0; b < E - a - 1; b++)
43. if (edges[b].weight > edges[b + 1].weight) {
44. Edge temp = edges[b];
45. edges[b] = edges[b + 1];
46. edges[b + 1] = temp;
47. }
49. Subset subsets[V];
50. for (int v = 0; v < V; v++) {
51. subsets[v].parent = v;
52. subsets[v].rank = 0;
53. }
55. while (e < V - 1 && i < E) {
56. Edge nextEdge = edges[i++];
58. int x = findSet(subsets, nextEdge.start);
59. int y = findSet(subsets, nextEdge.end);
61. if (x != y) {
62. mst[e++] = nextEdge;
63. unionSets(subsets, x, y);
64. }
65. }
66. printf("Edge \tWeight\n");
67. for (i = 0; i < e; i++)
68. printf("%d - %d \t%d\n", mst[i].start, mst[i].end, mst[i].weight);
69. }
71. int main() {
72. Edge edges[E] = {
73. {0, 1, 3},
74. {0, 2, 9},
75. {1, 2, 5},
76. {1, 3, 6},
77. {2, 3, 2},
78. {2, 4, 8},
79. {3, 4, 4}
80. };
82. kruskalAlgorithm(edges);
83. return 0;
84. }

**Output:  
**

1. MST implementation using Prim’s Algorithm
2. #include <stdio.h>
3. #include <limits.h>
4. #define V 5
5. int findMinKey(int cost[], int visited[]) {
6. int min = INT\_MAX, minIndex;
7. for (int i = 0; i < V; i++)
8. if (visited[i] == 0 && cost[i] < min)
9. min = cost[i], minIndex = i;
10. return minIndex;
11. }
12. void primAlgorithm(int graph[V][V]) {
13. int parent[V];
14. int cost[V];
15. int visited[V];
17. for (int i = 0; i < V; i++)
18. cost[i] = INT\_MAX, visited[i] = 0;
20. cost[0] = 0;
21. parent[0] = -1;
23. for (int i = 0; i < V - 1; i++) {
24. int u = findMinKey(cost, visited);
25. visited[u] = 1;
27. for (int v = 0; v < V; v++)
28. if (graph[u][v] && visited[v] == 0 && graph[u][v] < cost[v])
29. parent[v] = u, cost[v] = graph[u][v];
30. }
31. printf("Edge \tWeight\n");
32. for (int i = 1; i < V; i++)
33. printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
34. }
36. int main() {
37. int graph[V][V] = {
38. {0, 4, 0, 8, 0},
39. {4, 0, 2, 6, 5},
40. {0, 2, 0, 0, 7},
41. {8, 6, 0, 0, 9},
42. {0, 5, 7, 9, 0}
43. };
44. primAlgorithm(graph);
45. return 0;
46. }

**output:  
**