**Single Source Shortest Path DAG SSSP**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

Single Source Shortest Path (SSSP) is a fundamental problem in graph theory and network analysis, aiming to find the shortest path from a given source node to all other nodes in a weighted graph. The goal is to determine the minimum distance and the corresponding path between the source node and every other node in the graph.

The algorithm in this document works by first doing a topological ordering, assuming that all the paths are infinitely distant and initializing the first node distance to 0 then fixing the distance if we find a shorter one for every element. This algorithm works at a time complexity of .

**Use cases:**

The SSSP DAG problem finds applications in various fields, such as route planning, network routing, and scheduling, where finding the shortest paths from a given starting point to all other reachable points is a fundamental optimization task.

**Algorithm:**

1. # Variables

2. graph = adjacency list

3.

4. # SSSP DAG Algorithm

5. def SSSP(graph):

6. n = len(graph)

7. order = topsort(graph)

8. distances = [float('inf')] \* n

9. distances[ order[0] ] = 0

10.

11. for arg in order:

12. for el in graph[arg]:

13. distances[el[0]] = min(distances[arg] + el[1], distances[el[0]])

14.

15. return order, distances

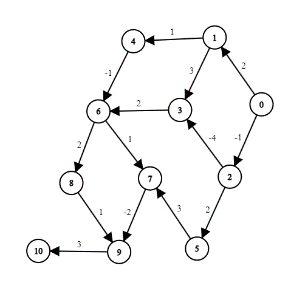
16.

17. order, distances = SSSP(graph)

18. print(distances)

**Example:**

Here’s a small example illustrating an example of input outputs for the SSSP DAG:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. # Variables

2. graph = [

3. [(1, 2), (2, -1)],

4. [(3, 3), (4, 1)],

5. [(3, 4), (5, 2)],

6. [(6, 2)],

7. [(6, 1)],

8. [(7, 3)],

9. [(7, 1), (8, 2)],

10. [(9, -2)],

11. [(9, 1)],

12. [(10, 3)],

13. []

14. ]

15.

16. # The Depth-First Search Algorithm

17. def dfs(arg, visited, graph):

18. if arg in visited:

19. return []

20. else:

21. visited.add(arg)

22. l = []

23. for el in graph[arg]:

24. l = dfs(el[0], visited, graph) + l

25. return [arg] + l

26.

27. # The Topological Ordering Algorithm

28. def topsort(graph):

29. n = len(graph)

30. visited = set()

31. l = []

32. for i in range(n):

33. if not i in visited:

34. l = dfs(i, visited, graph) + l

35. return l

36.

37. # SSSP DAG Algorithm

38. def SSSP(graph):

39. n = len(graph)

40. order = topsort(graph)

41. distances = [float('inf')] \* n

42. distances[ order[0] ] = 0

43.

44. for arg in order:

45. for el in graph[arg]:

46. distances[el[0]] = min(distances[arg] + el[1], distances[el[0]])

47.

48. return order, distances

49.

50. order, distances = SSSP(graph)

51. print(order, distances, sep="\n")

The corresponding output is:

1. Python>> [0, 2, 5, 1, 4, 3, 6, 8, 7, 9, 10]

2. >> [0, 2, -1, 3, 3, 1, 4, 4, 6, 2, 5]



**Longest Path Algorithm:**

Here you should just multiply all the edges by -1, find the shortest path using any SSSP algorithm and then remultiply them by -1 again and there you go, the longest path to each node

