|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **07 / 02 / 2023** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Dr. Aarti Phadke** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 4**

**Title: Single Source Shortest Path**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To analyze performance of sorting method. |

|  |
| --- |
| **COs to be achieved:** |
| **CO2:** Describe various algorithms design strategies to solve different problems. |

|  |
| --- |
| **Apparatus / Software Tools Used:** |
| 1. VS Code 2. Microsoft Excel |

|  |
| --- |
| **Theory:** |
| **Historical Profile:**  Sometimes the problems have more than one solution. With the size of the problem, every time it’s not feasible to solve all the alternative solutions and choose a better one. The greedy algorithms aim at choosing a greedy strategy as a solutioning method and proves how the greedy solution is better one.  Though greedy algorithms do not guarantee optimal solutions, they generally give a better and feasible solution. The path finding algorithms work on graphs as input and represent various problems in the real world.  The greedy method suggests that one can devise an algorithm that work in stages, considering one input at a time. At each stage, a decision is made regarding whether a particular input is in an optimal solution. This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of the next input into the partially constructed optimal solution will result in an infeasible solution, then this input is not added to the partial solution. Otherwise, it is added. The selection procedure itself is based on some optimization measures that may be plausible for a given problem. Most of these, however, will result in algorithms that generate suboptimal solutions.  This version of the greedy technique is called the **subset paradigm**.  **Code Abstraction:**  SolType Greedy(Type s[], int n)  {  SolType solution = EMPTY;  // Initialize the solution.  for (int i = 1; i <= n; i++) {  Type x = Select(a);  if (Feasible(solution, x))  solution = Union(solution, x);  }  return solution;  } |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| **Link for Shortest Path Algorithm in Virtual Labs -**  <https://ds1-iiith.vlabs.ac.in/List%20of%20experiments.html> |

|  |
| --- |
| **Codes and Graph:** |
| **Code:**  import time  def shortest\_paths(v, n, cost):      L = 10      S = [False] \* n      dist = [L] \* n      dist[v] = 0      for num in range(1, n):          u = -1          for i in range(n):              if not S[i] and (u == -1 or dist[i] < dist[u]):                  u = i          S[u] = True          for w in range(n):              if dist[w] > dist[u] + cost[u][w]:                  dist[w] = dist[u] + cost[u][w]      return dist  n = int(input("\nenter the number of vertices: "))  cost = []  print("\nenter the cost matrix (enter inf for infinity):")  for i in range(n):      row = input().split()      row = [float('inf') if x == 'inf' else float(x) for x in row]      print("Row", i, ":", row)      cost.append(row)    print("\ncost matrix:")  for row in cost:      print(row)  for i in range(n):      cost[i][i] = 0  # print("\ncost matrix: ", cost)  v = int(input("\nenter source index: "))  start\_time = time.perf\_counter()  shortest\_distances = shortest\_paths(v, n, cost)  end\_time = time.perf\_counter()  timeTaken=end\_time-start\_time  print("\nshortest distances from vertex", v, "are -")  for i in range(n):      print("vertex", i + 1, ":", shortest\_distances[i])  print(f"\ntime taken: {timeTaken:.10f} seconds")  **Output:**    **Calculation of Input Taken:**      **Observation Table:**      **Graph:** |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Find the shortest path from node 1 to node 8 of the distance network shown in figure below using Dijkstra’s Algorithm.** |

|  |
| --- |
| **Conclusion:** |
| The experiment explores sorting performance with greedy algorithms, using a simple iterative approach to construct optimal solutions for problems with multiple options. |

**Signature of faculty in-charge with Date:**