**1. Use Dijksra's algorithm to compute the least cost routes from node A to all other nodes. Show your working in a table with columns "Step" (number of times in the loop), N, and "D(X), p(X)" for all X not equal to A.**



Key

N – Set of nodes who’s least cost is positively known

D (N) – Distance (to node from A)

P (N) – Previous node (in direction of receiving node)

**1, A** - Bold figures – The resolved least cost route

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Step** | **N** | **D(B), p(B)** | **D(C), p(C)** | **D(D), p(D)** | **D(E), p(E)** | **Comments** |
| **0** | {A} | 5, A | **1, A** | ∞ | ∞ | A to C is the least cost route to C |
| **1** | {A, C} | 3, C | ∞ | **2, C** | ∞ | ACD is the least cost route to D |
| **2** | {A, C, D} | **3, C** | ∞ | ∞ | 5, D | ACB is the least cost route to B |
| **3** | {A, C, D, B} | ∞ | ∞ | ∞ | **4, B** | ACBE is the least cost route to E |

At each step the least cost route is found to the next node. In step 0 only the neighbouring nodes of A are examined, with the route to B cost 5 and route to C cost 1. A to C is therefore saved as the least cost route and the routes from C examined next.

In step 1 the costs to get to node C’s neighbouring nodes are added to the original cost of getting from node A. Node C can see Nodes B with cost 3 and can see Node D with cost 2. The route from C to D is the least cost; therefore node D is saved and examined next. A faster route has now been found to Node B through C, so this is replaced in the table.

In step 2 from Node D the routes to nodes B and E can be viewed (cost 3 and 5). The table is updated with the route to B saved. Although a route to B has always been visible, it is only now that it can be determined no faster route to B is possible.

At step 3 from Node B the only unresolved route visible is the path to Node E. From B the cost is only 4, therefore the table is updated and this figure saved. The table is now resolved.  **2. Use the Distance Vector Algorithm to compute the least cost from every node to every other node. Show the distance vector, DX, for each node X. Do this for 3 iterations of the algorithm (i.e. show 3 tables for each node).**



T0 - For the distance vector algorithm only the routes to each nodes neighbour are recorded, so for example in the tables below the table for A only contains routes that go via B and C. In step 0 each cost to the neighbouring node is inputted and stored as red text. Where no path exists it is left blank.

Key - DA – Distance from node A.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DA | To B | To C | To D | To E |
| Via B | 5 | - | - | - |
| Via C | - | 1 | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DB | To A | To C | To D | To E |
| Via A | 5 | - | - | - |
| Via C | - | 2 | - | - |
| Via E | - | - | - | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DC | To A | To B | To D | To E |
| Via A | 1 | - | - | - |
| Via B | - | 2 | - | - |
| Via D | - | - | 1 | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DD | To A | To B | To C | To E |
| Via C | - | - | 1 | - |
| Via E | - | - | - | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DE | To A | To B | To C | To D |
| Via B | - | 1 | - | - |
| Via D | - | - | - | 3 |

T1 – At step 1 each table is updated to include the red figures from the neighbouring nodes. For example Node A can now look at the tables for B and C and calculate the distance from Nodes B to E, and from C to D. The distance from Node A to Node E (via B) is cost 5 from A to B, and then 1 from B to E, giving cost 6. The new figures are stored in blue.

At this stage some faster routes are found, it is now apparent that to get from Node A to Node B it is faster to go via C then directly to B.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DA | To B | To C | To D | To E |
| Via B | 5 | 7 | - | 6 |
| Via C | 3 | 1 | 2 | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DB | To A | To C | To D | To E |
| Via A | 5 | 6 | - | - |
| Via C | 3 | 2 | 3 | - |
| Via E | - | - | 4 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D**C** | To A | To B | To D | To E |
| Via A | 1 | 6 | - | - |
| Via B | 7 | 2 | - | 3 |
| Via D | - | - | 1 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DD | To A | To B | To C | To E |
| Via C | 2 | 3 | 1 | - |
| Via E | - | 4 | - | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DE | To A | To B | To C | To D |
| Via B | 6 | 1 | 3 | - |
| Via D | - | - | 4 | 3 |

T2 – At step 2 each table is again updated with the blue figures to determine new or faster routes. For example a route from Node A to D (via B) can now be calculated, as table B has 2 routes to D (via C and E). The cost from A to B (3) and B to C (3) is added are stored in green. In this example I have determined the least cost possible from Node x -> Node z.

Some faster routes are found which replace the costs recorded at step 1, and these are stored in **bold** with the previous cost in brackets ()**.** An example of a route being faster is Node C via A to Node B. In the previous set of tables the cost was 6. At this step the route from C to A is still 1, but in step 1 table A has a faster route to C, at cost 3. This involves the message coming back to its originating node (**C**) and the message forwarded to Node B. This is cost **4.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DA | To B | To C | To D | To E |
| Via B | 5 | **5 (7)** | 6 | **4 (6)** |
| Via C | 3 | 1 | 2 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DB | To A | To C | To D | To E |
| Via A | 5 | **4 (6)** | 5 | 9 |
| Via C | 3 | 2 | 3 | 5 |
| Via E | 7 | 4 | 4 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D**C** | To A | To B | To D | To E |
| Via A | 1 | **4 (6)** | 3 | 7 |
| Via B | **5 (7)** | 2 | 5 | 3 |
| Via D | 3 | 4 | 1 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DD | To A | To B | To C | To E |
| Via C | 2 | 3 | 1 | 4 |
| Via E | 9 | 4 | 6 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DE | To A | To B | To C | To D |
| Via B | **4 (6)** | 1 | 3 | 4 |
| Via D | 5 | 6 | 4 | 3 |

**3. Briefly comment on which algorithm was more efficient in this exercise and on what other factors might influence the choice of algorithm. How would you compute the least hop routes?**

Dijksra’s algorithm is easier to calculate whilst the network is small, and for this 4 rows were required to find the least cost path from A to all other nodes. The choice of the least cost was only ever between two nodes. The disadvantages of this type of network is that the cost of the entire network need to be known meaning in a larger network the look up tables increase vastly in size. Another disadvantage to Dijksra’s algorithm is that when changes occur to the costs the whole algorithm could need recalculation.

The distance vector algorithm required 36 rows, and whilst the process was much more time consuming to work out mathematically, it did return the results of all the other routes between each node. Distance vectors hold local information and share this with its neighbouring nodes, meaning that updates in costs on short paths are very quick. The distance vector algorithm is less efficient when the variations in cost are high, as more iteration are needed to determine which is the best path between nodes (on some networks it may be quicker to go through several nodes rather than take one direct high cost path).

For Question 1 and 2 the efficiency of both algorithms had advantages. On this small network it was far quicker to calculate by hand Dijksra’s to return the fastest routes from Node A as at each of the 4 steps a simple choice of lowest cost was required.

Question 2 asked a different question, to determine the cost between each node and every other node. Although this took longer to work out by hand as all tables required addition, all routes were found in only 3 steps, quicker than Dijksra’s and returning a better quality and quantity of results. Whilst Dijksra’s is acceptable for this size of network, the distance vector algorithm is also acceptable and has a lot more advantages in any larger network. Obviously the time cost of working out each table step is negated when a computer is processing the figures.

Another way to choose a path is to compute the least hop route. This means that regardless of the number of hops involved in a network, the route between two nodes with the least hops will be chosen.

The least hops route can be determined with the distance vector algorithm, where only the first cost found for each node is used. For example in table 0 from node A to node B a route with cost 5 was found. In table 1 a faster route with cost 3 was found (via node C) and thus that figure was then used. However in the least hop route table only the first route is used, as that will have the least amount of hops.

To compute the least hop route with Dijksra’s algorithm the costs between each node need to be replaced with 1 and then calculated (and the route with the smallest cost will therefore have the least cost).

An advantage to analysing the least hop route compared to the distance vector algorithm is it would identify a bottle neck in the system. For example if Node A-B was cost 10, but A-B via C was 3, then the administrator may choose to upgrade the A-B line to relieve the congestion.