Barret Jackson
66122573
November 20, 2020

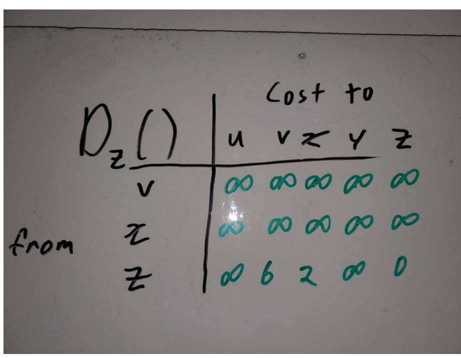
COSC 328 Lab 6

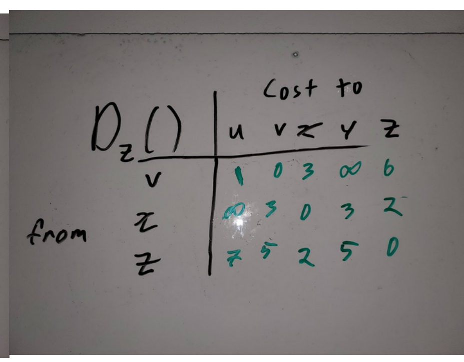
1) According to the distance-vector algorithm using the Bellman-Ford equation, the least-cost path from any node A to any node B will be $D_A(B) = \min_v \{ c_{A,v} + D_v(b) \}$ for all neighbouring nodes v. Therefore, to build the distance table from z, we use the BF equation to calculate the least-cost path to each node, starting with the neighbours of z.
$D_z(x) = \min \{ 2 + 0, 6 + 3 \} = 2$
$D_z(v) = \min \{ 6 + 0, 2 + 3 \} = 5$
$D_z(y) = \min \{ 2 + 3, 2 + 3 + 1 + 2\} = 5$
$D_z(u) = \min \{ 2 + 3 + 2, 2 + 3 + 1 \} = 6$
The resulting distance table is:

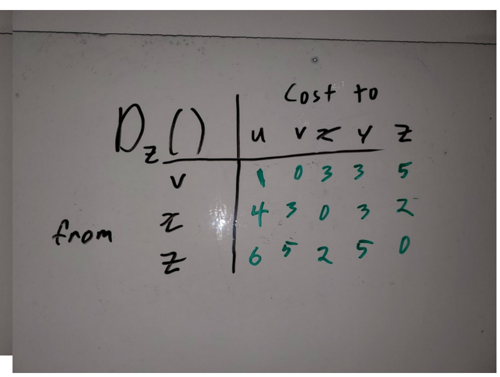
Distance table from node z

Node	***Path***	***Cost***
x	{z,x}	2
v	{z,x,v}	5
y	{z,x,y}	5
u	{z,x,v,u}	6

Alternatively, here is a sequence of tables showing how z and its neighbours x and v obtain their least cost distances:







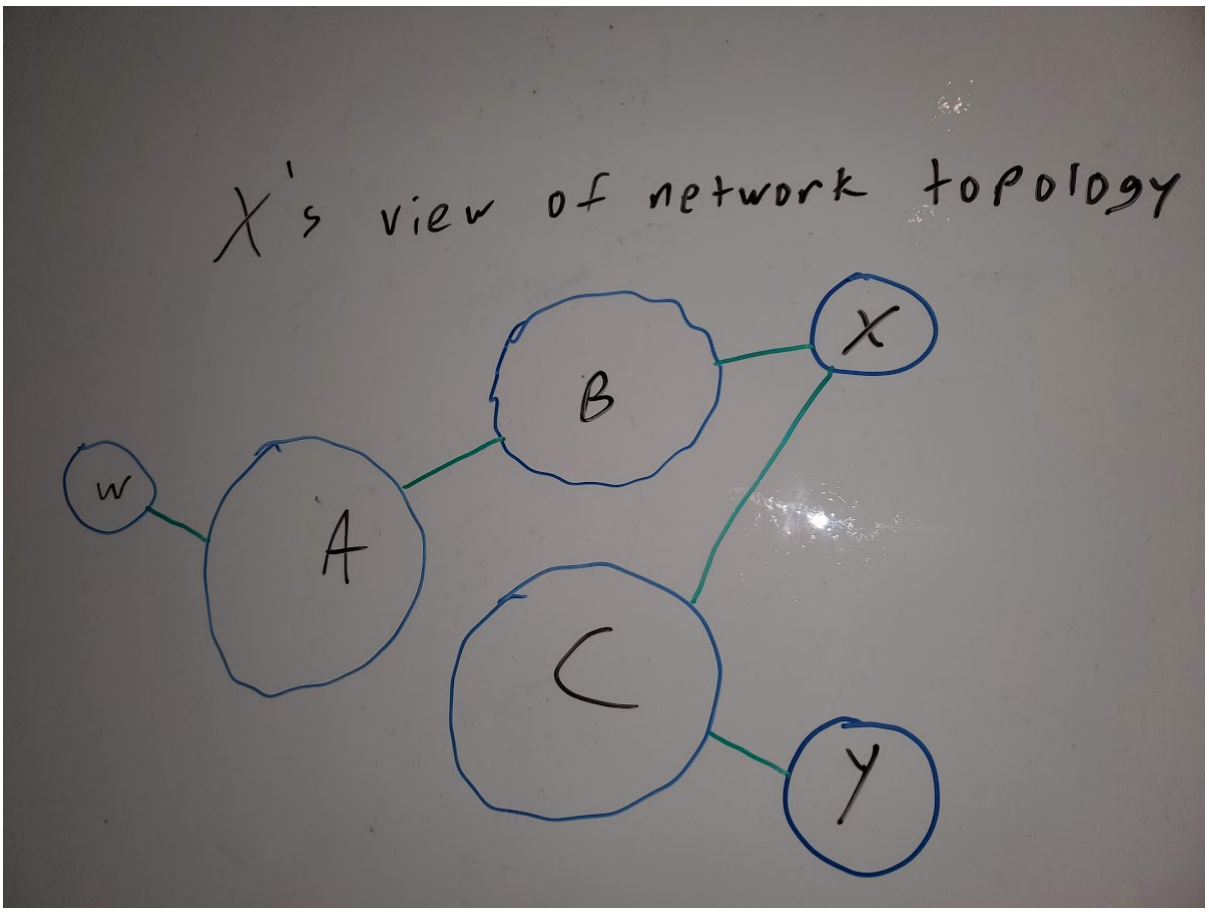
2) a) Using poisoned reverse, z informs w that $D_z(x)=\infty$, and informs y that $D_z(x)=6$. w informs y that $D_w(x)=\infty$, and z that $D_w(x)=5$. Finally, y informs both w and z that $D_y(x)=4$.
b) At time t0, z, w, and y have the information from part a), and the link cost c(x,y) changes from 4 to 60. At t1, y informs z that $D_y(x)=\infty$ and informs w that $D_y(x)=9$. At t2, w informs y that $D_w(x)=\infty$ and z that $D_w(x)=10$. At t3, z must update its least costs, so it informs w that $D_z(x)=\infty$ and y that $D_z(x)=11$. At t4, y must update its table, and informs w that $D_y(x)=14$ and z that $D_y(x)=\infty$. At t4, w must update its table… at this point we can tell we have a count to infinity problem.

Continuing with this logic, at t27, z calculates that the least cost to x is 50 from its direct path to x, so it advertises to both y and w that $D_z(x)=50$. At t28, w advertises to y that $D_w(x)=\infty$ and to z that $D_w(x)=\infty$, and y advertises to w that $D_y(x)=53$ and to z that $D_y(x)=\infty$. At t29, w calculates that the least cost to x is 51, so it advertises to y that $D_w(x)=51$ and to z that $D_w(x)=\infty$. At t30, y must update its least costs, so it advertises to w that $D_y(x)=\infty$ and to z that $D_y(x)=52$. Finally, the system is stabilized at t31, where for z, the path to x through w has cose ∞, through y it has cost 55, and through its direct link to z, cost 50. For w, the cost to access x either directly or through y is ∞, and through z is 51. Finally, for y, to access x through w is 52, through y is 60, and through z is 53.

c) Since poisoned reverse does not work when a router is connected to more than two other routers, the only solution to the count to infinity problem in this graph is to eliminate the connection between y and z.

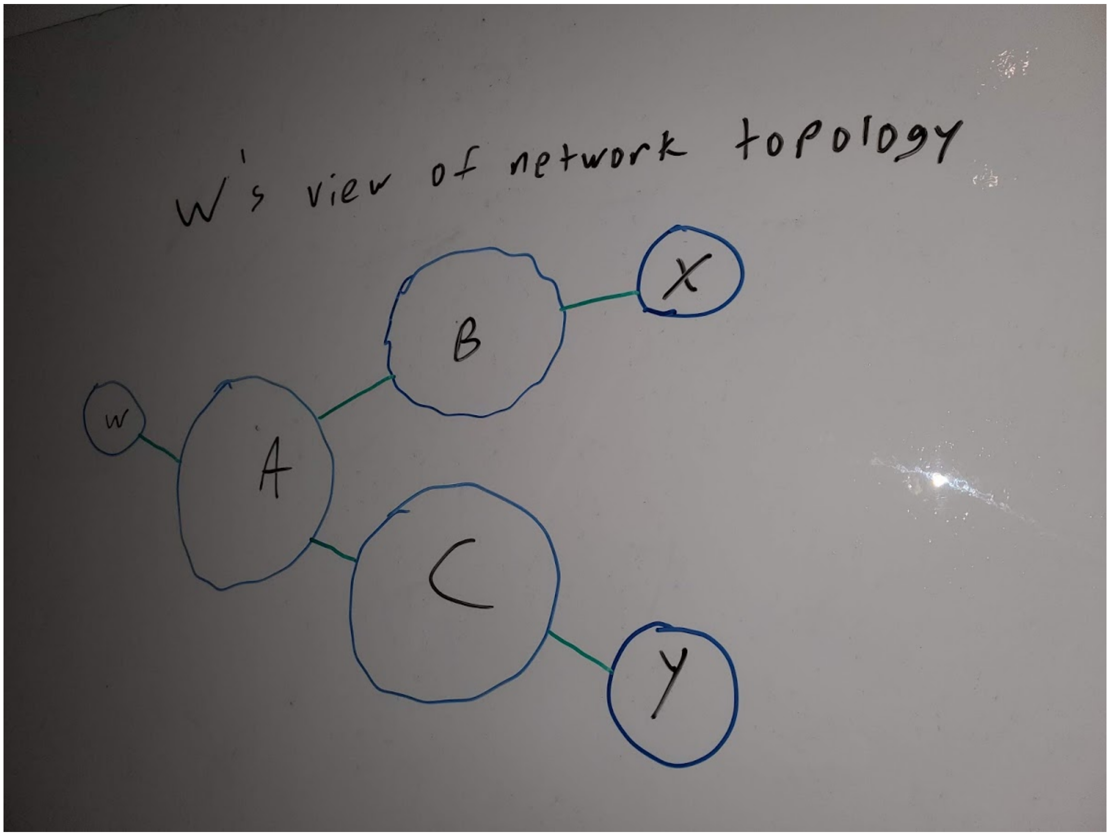
3) a) eBGP – 3c learns of x from 4c in a different autonomous system, using eBGP.
b) iBGP – 3a learns of x from 3b in the same autonomous system, so iBGP.
c) eBGP – 1c learns of x from 3a: different autonomous system, so eBGP.
d) iBGP – 1d learns of x from either 1a (most likely) or 1b: same autonomous system, so iBGP.

4) X's view of the network topology is as follows:



Since X is a multi-homed access ISP, it can reach both B and C directly. By policy, any traffic passing through X must have X as either its source or destination. Therefore, to reach Y, it must pass through C with a path XCY. To reach W, it appears as though there are two possible paths, XBAW and XCAW. W has advertised a path to A, X has advertised a path to B and C, and Y has advertised a path to C. If B learns of a path to W through A which is then advertised to X, X will let C know that it has a path to W. Since X already has a path to W, C will not advertise a path to W through A since this could potentially increase the load on C, so the path AC is not advertised. Similarly, B would not advertise a route to C to reduce the traffic from B to C.

W’s view of the network topology is as follows:



Similar to X’s topology, W advertises a path to A, Y advertises a path to C, and X advertises a path to B. Since X will not accept any traffic without itself as the source or destination, it does not advertise its link to C. To reduce traffic through itself, B also neglects to advertise a path from B to C.

<u>"Fun" with coding</u>

a) The output for the given graph is as follows (note I modified the code to use the python functions for undirected graphs)

Legend:

A = 0   B=1   C=2   D=3   E=4

```
testing dijkstraDumb_shortestPaths on a graph with 5 nodes from lecture notes
Graph with:
         Vertices:
        0,1,2,3,4,
         Edges:
        (0,1; wt:2) (0,2; wt:4) (0,3; wt:1) (1,0; wt:2) (1,2; wt:1) (1,4; wt:9) (1,3; wt:6) (2,0; wt:4) (2,1;
wt:1) (2,3; wt:4) (3,0; wt:1) (3,1; wt:6) (3,2; wt:4) (3,4; wt:2) (4,1; wt:9) (4,3; wt:2)

['0']
['0', '1']
['0', '1', '2']
['0', '3']
['0', '3', '4']
testing dijkstra_shortestPaths on a graph with 5 nodes from class
Graph with:
         Vertices:
        0,1,2,3,4,
         Edges:
        (0,1; wt:2) (0,2; wt:4) (0,3; wt:1) (1,0; wt:2) (1,2; wt:1) (1,4; wt:9) (1,3; wt:6) (2,0; wt:4) (2,1;
wt:1) (2,3; wt:4) (3,0; wt:1) (3,1; wt:6) (3,2; wt:4) (3,4; wt:2) (4,1; wt:9) (4,3; wt:2)

['0']
['0', '1']
['0', '1', '2']
['0', '3']
['0', '3', '4']
```

This output is as expected

b) The output for the graph in question 1 is as follows (again using undirected graphs)

Legend:

u=0   v=1   x=2   y=3   z=4

```
testing dijkstraDumb_shortestPaths on a graph with 5 nodes from lecture notes
Graph with:
         Vertices:
        0,1,2,3,4,
         Edges:
        (0,1; wt:1) (0,3; wt:2) (1,0; wt:1) (1,2; wt:3) (1,4; wt:6) (2,1; wt:3) (2,3; wt:3) (2,4; wt:2) (3,0;
wt:2) (3,2; wt:3) (4,1; wt:6) (4,2; wt:2)

['4', '2', '1', '0']
['4', '2', '1']
['4', '2']
['4', '2', '3']
['4']
testing dijkstra_shortestPaths on a graph with 5 nodes from class
Graph with:
         Vertices:
        0,1,2,3,4,
         Edges:
        (0,1; wt:1) (0,3; wt:2) (1,0; wt:1) (1,2; wt:3) (1,4; wt:6) (2,1; wt:3) (2,3; wt:3) (2,4; wt:2) (3,0;
wt:2) (3,2; wt:3) (4,1; wt:6) (4,2; wt:2)

['4', '2', '1', '0']
['4', '2', '1']
['4', '2']
['4', '2', '3']
['4']
```

Once again, the output is as expected

c) From the generated plot, we see the array implementation increasing in time complexity as a function of n as a second order polynomial, which matches our expectations of the time complexity of Dijkstra's algorithm being $O(n^2)$. Since a heap is a tree-based data structure, the best case time complexity (if the tree is a complete binary tree) is $O(\log_a n)$ where a is the number of branches in the tree. This is reflected by the lower observed time complexity in the generated plot.