CS-202 Exercises on Network Layer: IP & Routing (L15-L16)

Before we start

A few basic rules on how to allocate an IP prefix to a subnet:

- An IP prefix A/M is the range of IP addresses whose M most significant bits are the same as A's M most significant bits. E.g., 10.0.0.16 belongs to IP prefix 10.0.0.0/24 , because the 24 most significant bits of 10.0.0.16 are the same as the 24 most significant bits of 10.0.0.0.
- This implies that a /M IP prefix contains 2^{32-M} IP addresses. E.g., a /24 IP prefix contains $2^{32-24} = 2^8 = 256$ IP addresses. In other words, we don't have the freedom to create an IP prefix that contains an arbitrary number of IP addresses, it must always contain a number that is a power of 2.
- /M is called the "subnet mask", representing the bitmask consisting of M consecutive 1's followed by enough 0's to reach 32 bits. The subnet mask is applied to any IP address using bitwise AND to obtain the IP range the address belongs to. E.g. 100.52.12.18 belongs to the prefix 100.52.12.16/28 because: (1) the /28 mask represents the mask 1111 1111.1111 1111.1111 1111.1111 0000, (2) applying the mask to 100.52.12.18 (0110 0100.0011 0100.0000 1100.0001 0010 in binary) results in 100.52.12.16 (0110 0100.0011 0100.0000 1100.0001 0000 in binary).
- Each IP subnet must have its own IP prefix. Hence, IP prefixes allocated to different IP subnets must not overlap.

In addition to the network interfaces, each subnet has two IP addresses that are sometimes reserved for special use:

- The first IP address in the subnet's IP prefix (called "network address"). E.g., the first IP address in 10.0.0.0/24 is 10.0.0.0. This address is sometimes reserved for special uses, e.g., a discovery service provided by the subnet.
- The last IP address in the subnet's prefix (called "broadcast address"). E.g., the last IP address in 10.0.0.0/24 is 10.0.0.255. This address is sometimes reserved to be used as the subnet's broadcast address, i.e., as the destination IP address for packets that should be received by all end-systems in a subnet.

In practice, operators do not assign these addresses to any network interface. However, in this course you are free to assign them to a network interface or not except when it is clearly stated in the problem description.

Exercise 1: IP prefix allocation

IP subnets A, B and C contain 10, 5, and 3 network interfaces, respectively. Allocate an IP prefix to each subnet, and assign an IP address to each network interface, from IP prefix 1.2.3.0/27.

Consider two cases for allocating prefixes to subnets. In each case, follow the given order:

(a) A, B, C

(b) B, A, C

In the context of this exercise, when we say that allocation "follows a given order," we mean that, if Subnet X comes before Subnet Y in that order, the IP addresses for Subnet X should be arithmetically smaller than the IP addresses for Subnet Y (in the sense that IP address 1.2.3.4 is arithmetically smaller than IP address 1.2.3.5).

Note: Allocating addresses might be infeasible in some cases.

A. *Network prefix 1.2.3.0/27 contains 2^{32-27} = 32 addresses. We need to pick three subnets from the address space 1.2.3.0 - 1.2.3.31, while satisfying the requirements from above.*

   Subnet A must have at least 11 IP addresses (10 for end-systems and 1 for broadcast address). However, since we need to round to a power of 2 we will allocate 16 addresses: from 1.2.3.0 to 1.2.3.15. Let's represent some of them in binary format:

   0000 0001.0000 0010.0000 0011.0000 0000 = 1.2.3.0

   0000 0001.0000 0010.0000 0011.0000 0001 = 1.2.3.1

   0000 0001.0000 0010.0000 0011.0000 0010 = 1.2.3.2

   0000 0001.0000 0010.0000 0011.0000 0011 = 1.2.3.3

   ...

   The prefix is 0000 0001.0000 0010.0000 0011.0000 = 1.2.3.0 with length 28, thus Subnet A is 1.2.3.0/28, or 1.2.3.0 - 1.2.3.15.

   Note: to obtain the CIDR notation (that is 1.2.3.0/28), we:

   1. *Take the binary prefix of the subnet: 0000 0001.0000 0010.0000 0011.0000 0;*
   2. *Append zeros until we obtain 32 bits: 0000 0001.0000 0010.0000 0011.0000 0000;*

3. *Transform the value into dotted IP notation: 1.2.3.0;*
4. *Append "/" and the length of the binary prefix: 1.2.3.0/28.*

Subnet B needs 6 addresses, so we need to allocate 8: from 1.2.3.16 to 1.2.3.23. We expect that the mask has length 29. Let's represent some of the addresses in binary:

0000 0001.0000 0010.0000 0011.0001 0000 = 1.2.3.16

0000 0001.0000 0010.0000 0011.0001 0001 = 1.2.3.17

0000 0001.0000 0010.0000 0011.0001 0010 = 1.2.3.18

...

The interval is 1.2.3.16 - 1.2.3.23, and the prefix is 0000 0001.0000 0010.0000 0011.0001 0 with length 29, which corresponds to the block 1.2.3.16/29.

For Subnet C, we need to allocate 4 addresses. Thus we need a block with 4 addresses. We try to allocate starting from 1.2.3.24:

0000 0001.0000 0010.0000 0011.0001 10 00 = 1.2.3.24

0000 0001.0000 0010.0000 0011.0001 10 01 = 1.2.3.25

0000 0001.0000 0010.0000 0011.0001 10 10 = 1.2.3.26

0000 0001.0000 0010.0000 0011.0001 10 11 = 1.2.3.27

The interval is 1.2.3.24 - 1.2.3.27, the prefix is 0000 0001.0000 0010.00000 0011.0001 10 with length 30. Thus the CIDR notation is 1.2.3.24/30.

(b) It is not possible to allocate the addresses in this order.

The reason is that you have only two options in how to allocate addresses for Subnet A (either prefix 1.2.3.0/28 or prefix 1.2.3.16/28).

- *If you allocate prefix 1.2.3.0/28 to Subnet A, then there will be no room to allocate smaller addresses for Subnet B.*
- *Instead, if you allocate prefix 1.2.3.16/28, there will be no room to allocate bigger addresses to Subnet C.*

Exercise 2: network configuration

Consider the topology shown in Figure 1. There are three IP subnets (A, B and C) that contain some end-systems, and two IP subnets (D and E) that contain no end-systems. The green boxes (a, b, c, . . . g) denote network interfaces for routers R1, R2 and R3.

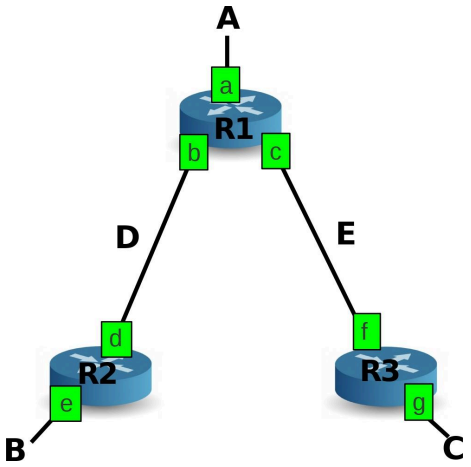

Figure 1: Network Topology.

1. Allocate an IP prefix to each subnet. Your allocation must respect the following constraints:
   - All prefixes must be allocated from 214.97.254.0/23.
   - Subnet A should have enough addresses to support 250 interfaces.
   - Subnet B should have enough addresses to support 120 interfaces.
   - Subnet C should have enough addresses to support 60 interfaces.
   - Each of subnets D and E should have enough addresses to support 2 interfaces.

First, you need to round the number of required addresses (number of interfaces + 1 for broadcast address) up to the nearest power of two.

Then, a good strategy for allocating subnets on the available address space is the following:

- *Sort the subnets in descending order according to the (rounded) number of addresses they require*
- *Start allocating addresses from one end of the address space (e.g. the beginning)*
- *For every remaining subnet (following the order in which you*

sorted them), allocate the address space after the previously allocated prefix. (i.e., do not leave gaps between consecutive subnets)

If you follow this strategy, you can guarantee that every subnet is well-aligned, and that there will be no gaps between allocated address blocks.

Nevertheless, there are multiple possible solutions for this exercise. Here is a possible allocation:

Subnet A: 214.97.254/24 (256 addresses)
Subnet B: 214.97.255.0/25 (128 addresses)
Subnet C: 214.97.255.128/26 (64 addresses)

Subnet D: 214.97.255.192/30 (4 addresses)
Subnet E: 214.97.255.196/30 (4 addresses)

2. Using your previous answer, provide the forwarding tables for each of the three routers (R1, R2, R3). Each table should contain two columns which show (i) the destination IP prefix, and (ii) the corresponding output link.

The simplest way to create a forwarding table is to create a separate rule for every subnet that we need to access.

If we take into account that the address ranges for different subnets do not overlap, we don't have to worry about longest-prefix-length matching.

Router 1:

Longest Prefix Match Outgoing:	*Interface:*
11010110 01100001 11111111 110000	*Port b*
11010110 01100001 11111111 110001	*Port c*
11010110 01100001 11111111	*10 Port c*
11010110 01100001 11111111 0	*Port b*
11010110 01100001 11111110	*Port a*

Router 2:

Longest Prefix Match Outgoing :	*Interface:*
11010110 01100001 11111111 110000	*Port d*
11010110 01100001 11111111 110001	*Port d*
11010110 01100001 11111111 10	*Port d*
11010110 01100001 11111111 0	*Port e*
11010110 01100001 11111110	*Port d*

Router 3:

Longest Prefix Match Outgoing:	*Interface:*
11010110 01100001 11111111 110000	*Port f*
11010110 01100001 11111111 110001	*Port f*
11010110 01100001 11111111 10	*Port g*
11010110 01100001 11111111 0	*Port f*
11010110 01100001 11111110	*Port f*

3. Can you reduce the number of entries of each forwarding table, i.e., for each table create an equivalent one, which has the same outcome but consists of fewer entries?

We can reduce the number of entries by grouping the Longest prefix matches for the same outgoing interface, starting from the longest matches (top-bottom). This will generate new entries with first column being the longest prefix between all (or some) of those entries and second column being the output port.

However, we need to be careful not to create conflict with an other outgoing interface. *For example, in the forwarding table of R1, we would need to summarize the entries for Port c with "11010110 01100001 11111111 1" (longest prefix match between "11010110 01100001 11111111 110001" and "11010110 01100001 11111111 10"). Then, if we wanted to summarize the entries for Port b we would need to add an entry with prefix "11010110 01100001 11111111" (longest prefix match between "11010110 01100001 11111111 110000" and "11010110 01100001 11111111 0"). But if we use this prefix and place it in the forwarding table, it would look like this:*

Longest Prefix Match Outgoing:	*Interface:*
11010110 01100001 11111111 1	*Port c*
11010110 01100001 11111111	*Port b*
11010110 01100001 11111110	*Port a*

Recall that the forwarding table must be ordered by the length of the prefix in the Longest Prefix Match column (as shown in the previous table). Therefore, when a router receives a packet, it will scan the forwarding table in order (from longest prefix to shortest prefix) until it matches the first compatible longest prefix match. In that case, a packet matching "11010110 01100001 11111111 110000 " will be forwarded to Port c instead of Port b (which is wrong, see the first line of the original forwarding table). In this situation, the prefixes of Port b can not be grouped.

Therefore, the tables with summarized entries are shown below.

Router 1:

Longest Prefix Match Outgoing	*Interface*
11010110 01100001 11111111 110000	*Port b*
11010110 01100001 11111111 1	*Port c*
11010110 01100001 11111111 0	*Port b*
11010110 01100001 11111110	*Port a*

Router 2:

Longest Prefix Match Outgoing	*Interface*
11010110 01100001 11111111 0	*Port e*
11010110 01100001 1111111	*Port d*

Router 3:

Longest Prefix Match Outgoing	*Interface*
11010110 01100001 11111111 10	*Port g*
11010110 01100001 1111111	*Port f*

Exercise 3: practice prefix allocation

Consider the graph shown in Figure 2, which consists of four network subnets, *A*, *B*, *C* and *D*, all connected to the Internet through Router 3.

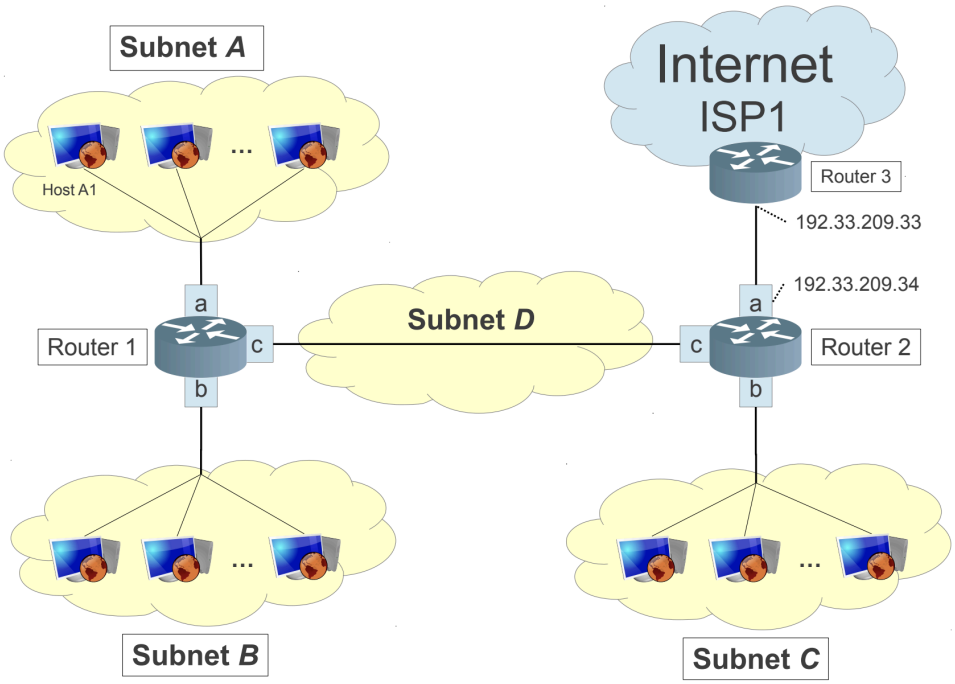

Figure 2: Network topology.

- Subnet *A* is attached to port *a* of Router 1.
- Subnet *B* is attached to port *b* of Router 1.
- Subnet *C* is attached to port *b* of Router 2.
- Subnet *D* has no hosts.
- All hosts have access to the Internet, through an Internet Service Provider "ISP1".

Assign network addresses to each of these four subnets, with the following constraints:

- All addresses must be allocated from 1.0.2.0/23 (in other words, they should have binary format 00000001.00000000.0000001x.xxxxxxxx);
- subnets *A*, *B* and *C* should have enough addresses to support 200, 100 and 50 interfaces, respectively;
- you should allocate the smallest possible range of IP addresses to each subnet;
- only in this exercise, assume subnets do not have a broadcast address (i.e., you should not allocate it);
- for each subnet, the assignment should take the form a.b.c.d/x.

1. *Subnet A: 200 interfaces. $2^7 < 200 < 2^8$, so we need to reserve $2^8 = 256$ IP addresses.*
2. *Subnet B: 100 interfaces. $2^6 < 100 < 2^7$, so we need to reserve $2^7 = 128$ IP addresses.*
3. *Subnet C: 50 interfaces. $2^5 < 50 < 2^6$, so we need to reserve $2^6 = 64$ IP addresses.*
4. *Subnet D: 2 interfaces. $2 = 2^1$, so we need to reserve 2 IP addresses.*

From 1.0.2.0/23, a possible assignment would be:

1. *Subnet A: 1.0.2.0/24 (256 addresses)*
2. *Subnet B: 1.0.3.0/25 (128 addresses)*
3. *Subnet C: 1.0.3.128/26 (64 addresses)*
4. *Subnet D: 1.0.3.192/31 (2 addresses)*

Exercise 4: identify subnets

Consider the network in Figure 3, consisting of hosts A, B, C, D, G and H, DNS server E, web server F, router R1 and switches S1, S2, S3 and S4.

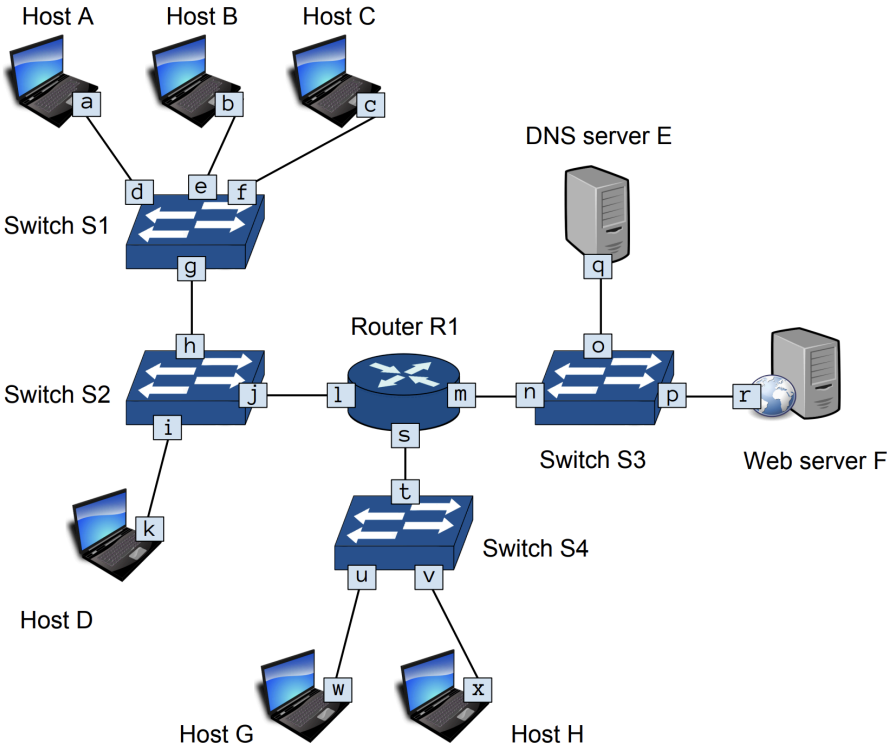

Figure 3: Network topology.

1. Allocate an IP prefix to each IP subnet that contains end-systems, following these rules:
   - All addresses must be allocated from 10.0.0.0/24 (they should have the binary format 00001010.00000000.00000000.xxxxxxxx), following the basic rules for allocating IP addresses.
   - You should allocate the smallest possible range of IP addresses to each subnet (*Note: be careful to consider* ***only*** *the network interfaces that should be assigned an IP address*)

The network consists of the following subnets:

Subnet 1: Hosts A, B, C, D, switches S1 and S2, and interface l of router R1 (5 interfaces + 1 broadcast address = 6 addresses)

Subnet 2: DNS server E, web server F, switch S3 and interface m of router R1 (3 interfaces + 1 broadcast address = 4 addresses)

Subnet 3: Hosts G and H, switch 4 and interface s of router R1 (3 interfaces + 1 broadcast address = 4 addresses)

The other interfaces belong to switches that as said in class, to keep things simple, you don't need to assign IP addresses to link-layer switches, even though, in reality, they also have IP addresses. A possible allocation of IP address spaces for each subnet is:

Subnet 1: 10.0.0.0/29 or 10.0.0.0 – 10.0.0.7
Subnet 2: 10.0.0.8/30 or 10.0.0.8 – 10.0.0.11
Subnet 3: 10.0.0.12/30 or 10.0.0.12 – 10.0.0.15
We can use the binary representation to check that the allocation is correct:

00001010.00000000.00000000.00000000 or 10.0.0.0/29 (interface a)
00001010.00000000.00000000.00000001 or 10.0.0.1/29 (interface b)
00001010.00000000.00000000.00000010 or 10.0.0.2/29 (interface c)
00001010.00000000.00000000.00000011 or 10.0.0.3/29 (interface k)
00001010.00000000.00000000.00000100 or 10.0.0.4/29 (interface l)
00001010.00000000.00000000.00000101 or 10.0.0.5/29 (not used)
00001010.00000000.00000000.00000110 or 10.0.0.6/29 (not used)
00001010.00000000.00000000.00000111 or 10.0.0.7/29 (broadcast addr. for Subnet 1)

00001010.00000000.00000000.00001000 or 10.0.0.8/30 (interface m)
00001010.00000000.00000000.00001001 or 10.0.0.9/30 (interface q)
00001010.00000000.00000000.00001010 or 10.0.0.10/30 (interface r)
00001010.00000000.00000000.00001011 or 10.0.0.11/30 (broadcast addr. for Subnet 2)

00001010.00000000.00000000.00001100 or 10.0.0.12/30 (interface s)
00001010.00000000.00000000.00001101 or 10.0.0.13/30 (interface w)
00001010.00000000.00000000.00001110 or 10.0.0.14/30 (interface x)
00001010.00000000.00000000.00001111 or 10.0.0.15/30 (broadcast addr. for Subnet 3)

2. Fill Table 1 with the IP address for the network interfaces that should be assigned an IP address. write "−" for interfaces for which an IP address is not needed.

Network interface	IP address
Example: y	1.2.3.4
Example: z	–
a	10.0.0.0
b	10.0.0.1
c	10.0.0.2
d	-
e	-
f	-
g	-
h	-
i	-
j	-
k	10.0.0.3
l	10.0.0.4
m	10.0.0.8
n	–
o	–
p	–
q	10.0.0.9
r	10.0.0.10
s	10.0.0.12
t	–
u	–
v	–
w	10.0.0.13
x	10.0.0.14

Table 1: IP address allocations for the interfaces from Figure 3

Exercise 5: link-state routing

Consider the network in Figure 4. Execute the link-state (Dijkstra's) algorithm we saw in class to compute the least-cost path from each of *x*, *v*, and *t* to all the other routers.

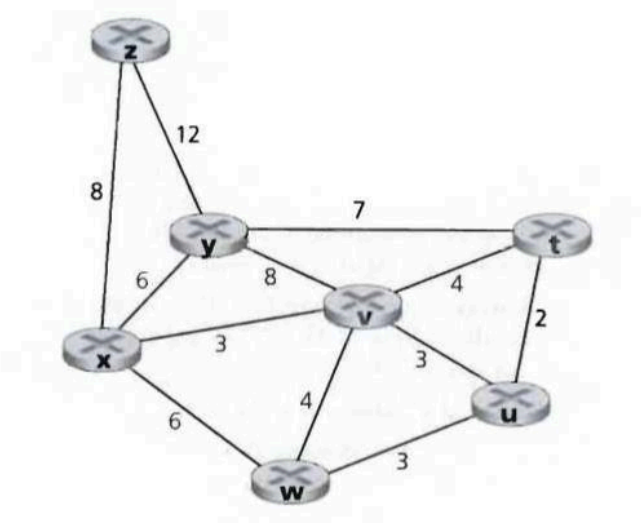

Figure 4: Network topology.

The least cost path from routers x, v, and t to all the other routers is displayed in the next table along with the execution steps of the link-state algorithm.

For each router i, C(i) stands for cost to i, and p(i) stands for predecessor to i.

1. Least-cost path from x to all network nodes.

step	*nodes visited*	*C(t),p(t)*	*C(u),p(u)*	*C(v),p(v)*	*C(w),p(w)*	*C(y),p(y)*	*C(z),p(z)*
0	*x*	*∞*	*∞*	*3,x*	*6,x*	*6,x*	*8,x*
1	*x,v*	*7,v*	*6,v*	*3,x*	*6,x*	*6,x*	*8,x*
2	*x,v,u*	*7,v*	*6,v*	*3,x*	*6,x*	*6,x*	*8,x*
3	*x,v,u,w*	*7,v*	*6,v*	*3,x*	*6,x*	*6,x*	*8,x*
4	*x,v,u,w,y*	*7,v*	*6,v*	*3,x*	*6,x*	*6,x*	*8,x*
5	*x,v,u,w,y,t*	*7,v*	*6,v*	*3,x*	*6,x*	*6,x*	*8,x*
6	*x,v,u,w,y,t,z*	*7,v*	*6,v*	*3,x*	*6,x*	*6,x*	*8,x*

2. Least-cost path from v to all network nodes.

step	nodes visited	C(t),p(t)	C(u),p(u)	C(w),p(w)	C(x),p(x)	C(y),p(y)	C(z),p(z)
0	v	4,v	3,v	4,v	3,v	8,v	∞
1	v,x	4,v	3,v	4,v	3,v	8,v	11,x
2	v,x,u	4,v	3,v	4,v	3,v	8,v	11,x
3	v,x,u,t	4,v	3,v	4,v	3,v	8,v	11,x
4	v,x,u,t,w	4,v	3,v	4,v	3,v	8,v	11,x
5	v,x,u,t,w,y	4,v	3,v	4,v	3,v	8,v	11,x
6	v,x,u,t,w,y,z	4,v	3,v	4,v	3,v	8,v	11,x

3. Least-cost path from t to all network nodes.

step	nodes visited	C(u),p(u)	C(v),p(v)	C(w),p(w)	C(x),p(x)	C(y),p(y)	C(z),p(z)
0	t	2,t	4,t	∞	∞	7,t	∞
1	t,u	2,t	4,t	5,u	∞	7,t	∞
2	t,u,v	2,t	4,t	5,u	7,v	7,t	∞
3	t,u,v,w	2,t	4,t	5,u	7,v	7,t	∞
4	t,u,v,w,x	2,t	4,t	5,u	7,v	7,t	15,x
5	t,u,v,w,x,y	2,t	4,t	5,u	7,v	7,t	15,x
6	t,u,v,w,x,y,z	2,t	4,t	5,u	7,v	7,t	15,x

Exercise 6: distance-vector routing

Consider the network in Figure 5. Execute the distance-vector (Bellman-Ford) algorithm we saw in class and show the information that router *z* knows after each iteration.

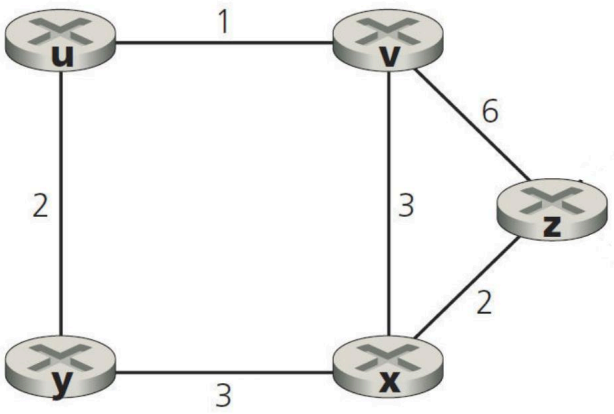

Figure 5: Network topology.

Each node in the topology has its own view of the network, which is updated independently from other nodes at the end of each step. Therefore, for every step of the algorithm, you also need to update each of the other cost tables. Otherwise, your solution may be incorrect.

In our solution we only show the cost table for node z, as required by the question. The cost table at node z consists of 5 columns (all possible destinations) and 3 rows (all possible sources—one row for node z and one row for each neighbor). Each entry of the table denotes the cost between the associated source-destination nodes.

Initially (at step 0), node z has the following view of the network:

		To				
		u	*v*	*x*	*y*	*z*
From	*v*	∞	∞	∞	∞	∞
	x	∞	∞	∞	∞	∞
	z	∞	6	2	∞	0

At step 1:

		To				
		u	*v*	*x*	*y*	*z*
From	*v*	1	0	3	∞	6
	x	∞	3	0	3	2
	z	7	5	2	5	0

At step 2:

		To				
		u	*v*	*x*	*y*	*z*
From	*v*	1	0	3	3	5
	x	4	3	0	3	2
	z	6	5	2	5	0

At step 3:

		To				
		u	*v*	*x*	*y*	*z*
From	*v*	*1*	*0*	*3*	*3*	*5*
	x	*4*	*3*	*0*	*3*	*2*
	z	*6*	*5*	*2*	*5*	*0*

We see that from step 2 to step 3 the cost tables did not change, indicating that the algorithm has converged.

Exercise 7: convergence

What is the maximum number of iterations required for the distance-vector (Bellman Ford) algorithm that we saw in class to converge (i.e., to finish, assuming no change occurs in the network graph and link costs)? Justify your answer.

At each iteration, a node exchanges cost tables with its neighbors. Thus, if you are node A, and your neighbor is B, all of B's neighbors (which are all one or two hops from you) will know the least-cost path of one or two hops to you after one iteration (i.e., after B tells them its cost to you).

Let d be the "diameter" of the network, which is computed as follows:

- *Find the least-cost path between each pair of nodes.*
- *The diameter is equal to the greatest length (in number of links) of any of those least-cost paths.*

Using the reasoning above, after $d - 1$ iterations, all nodes will know the least-cost path of length $\leq d$ hops to all other nodes. Since any path of length $> d$ hops is costlier than any of the least-cost paths, the cost tables of the nodes will not change after that point. Hence, the algorithm converges in at most $d - 1$ iterations.