CHAPTER 29

PRACTICE SET

Questions

Q29-1.

a. In a centralized network, the directory system uses the client-server paradigm, but storing and downloading of the files are done using peer-to-peer paradigm.

b. A decentralized network does not depend on a central directory system.

Q29-2. In an unstructured network, the nodes are linked together randomly. A structured network uses a predefined set of rules to link nodes together.

Q29-3. The number of points is 2^{10} or 1024.

Q29-4. There are not only IP addresses that we need to store; we need to store other identities such a file names. A hash function creates a unified set of identities.

Q29-5. For this network, $m = \log_2 1024 = 10$. This means each identifier has 10 bits. The height of the tree is also 10. The number of leaves is 1024. Each node has 10 subtrees. Each routing table also has 10 rows.

Q29-6. There are only $2^m = 16$ points on the circle (0 to 15). We need to use modular arithmetic to find the location of the node. Since 18 mod 16 = 2, the node is located at the point marked 2 on the address space circle.

Q29-7.

a. In the direct method, the file is stored in node 20.

b. In the indirect method, the file is stored in node 4, but a reference is given in node 20.

Q29-8. We need to find the node with the shortest distance from k3. We use the exclusive-or operation: $3 \oplus 4 = 7$, $3 \oplus 7 = 4$, and $3 \oplus 12 = 15$, which means node N7 is the closest to k3. The key k3 is stored in node N7.

Q29-9. A decentralized network generates less traffic and is less vulnerable to attacks, but is more difficult to build.

Q29-10. A central network makes the maintenance of the directory simple, but has such drawbacks such as creating huge traffic and vulnerability to attacks.

Q29-11. The two strategies are called direct and indirect.

a. In the direct strategy, the object is stored in a peer whose ID is somehow *closest* to the object ID.

b. In the indirect strategy, the object is stored in a peer that owns the object, but a reference is stored in a peer whose ID is somehow *closest* to the object ID.

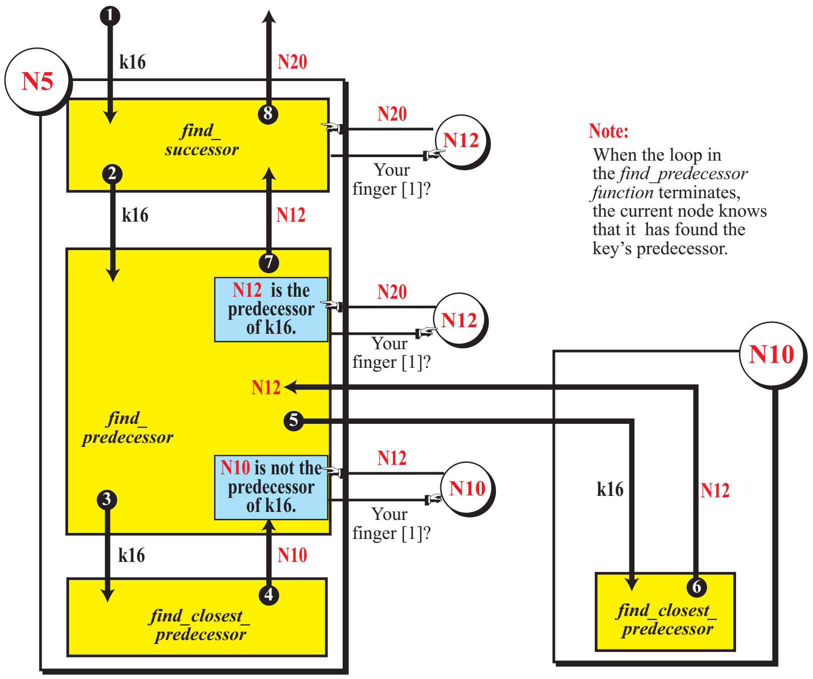
Q29-12. If the identifier of a key and a node is the same, the node is responsible for the key. In other words, the node is the successor of the key, but not the predecessor of it. In this case, N5 is the successor of k5, but the predecessor of k5 is the node before N5 in the ring.

Problem

P29-1. The number of rows in the finger table is $\log_2 16 = 4$. The rows in the table for node N6 show the successors of target keys $(6 + 2^{m-1})$ for $m = 1$ to 4 as shown below:

m	Target key	Successor
1	6 + 1	N8
2	6 + 2	N8
3	6 + 4	N12
4	6 + 8	N3

P29-2. Let us go through the event as we did in Example 2.16 in the text. The following shows how N5 with some help from N10 and N12 goes through the sequence of events to find the successor of k16:


N5
k16
N20
find_ successor
N12
k16
find_ predecessor
N12 is the predecessor of k16.
N10 is not the predecessor of k16.
k16
N10
find_closest_ predecessor
N20
N12
Your finger [1]?
N20
N12
Your finger [1]?
N12
N10
Your finger [1]?
N12
N10
k16
N12
find_closest_ predecessor
Note:
When the loop in the *find_predecessor* *function* terminates, the current node knows that it has found the key's predecessor.

a. Event 1: Since N5 is not the responsible node for k16, it calls its *find_successor* function.

b. Event 2: N5 calls the *find_predecessor* function.

c. Event 3: Since N5 is not the predecessor node of k16, N5 calls the *find_closest_predecessor* function. This function checks the finger table of N5 from bottom to top. It finds that N10 is the closest predecessor because 10 ∈ (5, 16).

d. Event 4: The *find_closest_predecessor* function returns N10 to the *find_predecessor* function as the closest predecessor of k16. N5 finds that N10 is not the predecessor of k16 (by remotely calling the N10.finger[1]).

e. Event 5: N5 remotely asks N10 to find the closest predecessor of k16.

f. Event 6: Node 10 finds that N12 is the closest predecessor of k16 according to its knowledge. It returns N12 to N5. N5 finds that N12 is actually the predecessor of k16 (by asking N12.finger[1].

g. Event 7: The *find_predecessor* function returns N12, as the predecessor of k16, to the *find_successor* function.

h. Event 8: Node N5 finds that the successor node of k16 is N20 by asking N12 to send its finger [1]).

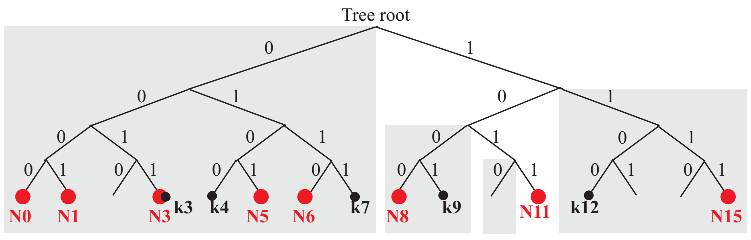
P29-3. We have $m = \log_2 16 = 4$. The number of digits in each identifier is $n = m/b$ or 2. The identifiers are in base $2^2 = 4$ as follows: 00 … 03, 10 … 13, 20 … 23, 30 … 33.

P29-4. We have $n = m/b = 32/4 = 8$. The routing table is of size $n \times 2^b$ or 8×16 (8 rows and 16 columns). The leaf set is a table of one row and 16 columns.

P29-5. We have $m = \log_2 16 = 4$ and $n = m/b = 4/2 = 2$. The routing table has n rows (here 2) and 2^b columns (here 4). The identifiers are 00 to 33 in base 4. The following shows a possible routing table for node N21. In row 0, there should be no common digits with the node identifier; the first digit defines the column number. In row 1, the first digit should be 2 (one common digit with node N21); the second digit defines the column. Note that some cells can have no entry.

	0	1	2	3
0	00	10	▨	31
1	20	▨	22	23

P29-6. The following shows the four subtrees.



P29-7. The following takes place:

a. The length of the common prefix between N0 (N0000) and k12 (k1100) is 0. Node N0 sends the query to the node specified in row 0 of its routing table, node N8.

b. The length of the common prefix between N8 (N1000) and k12 (k1100) is 1. Node N8 sends the query to the node specified in row 1 of its routing table, node N15.

c. Node 15 is responsible for k12. It responds to the query.

P29-8. A node can quickly find if it is the predecessor of a key when key ∈ (node, successor]. If the key is not the predecessor, the node needs to find the closest predecessor. For this purpose, it needs to check the entries in the finger table (from bottom to top) to find a node, between itself and the key, that is closest to the key as far as the information in the table can tell. The table needs to be searched from bottom to top because many entries can be between the node and the key, but the closest to the key is definitely the one which is found first if we start from the bottom of the table. When the closest predecessor is found, the node can ask it to continue with the search.

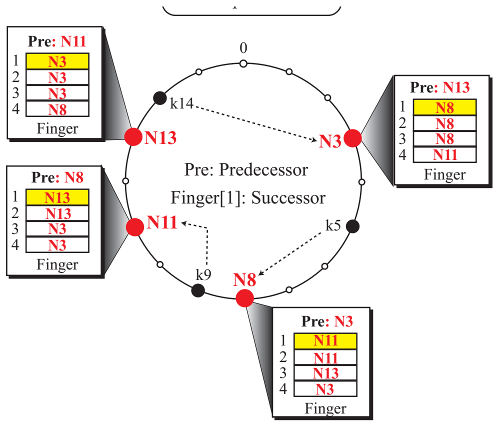
a. Since k1 ∉ (N2, N4], N2 is not the predecessor of k1. Node N2 needs to find the closest predecessor of k1. The table is searched from the bottom. N12 ∈ (N2, k1) because 1 here means 17. So, as far as N2 knows, N12 is the closest predecessor for k1. N2 will ask N12 to search further.

b. Since k6 ∉ (N2, N4], N2 is not the predecessor of k6. Node N2 needs to find the closest predecessor of k6. The table is searched from the bottom. N12 ∉ (N2, k6), N10 ∉ (N2, k6), N7 ∉ (N2, k6), but N4 ∈ (N2, k6). So, as far as N2 knows, N4 is the closest predecessor for k6. N2 will ask N4 to search further.

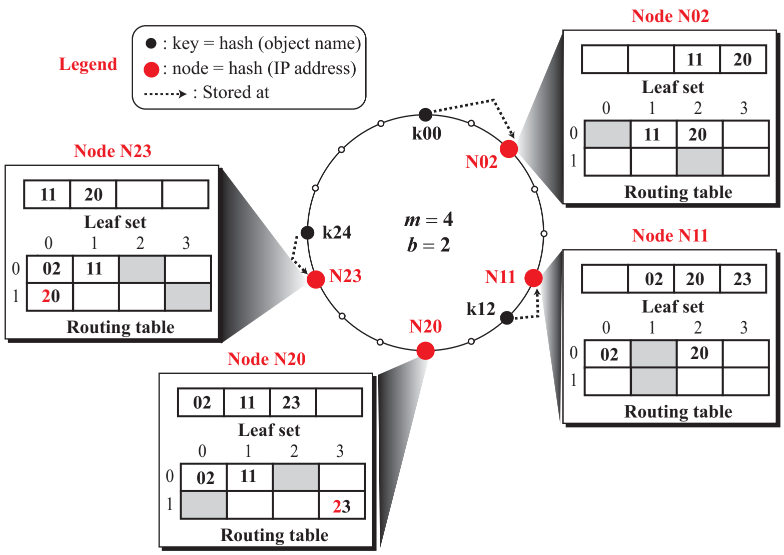
c. Since k9 ∉ (N2, N4], N2 is not the predecessor of k9. Node N2 needs to find the closest predecessor of k9. The table is searched from the bottom. N12 ∉ (N2, k9), N10 ∉ (N2, k9), but N7 ∈ (N2, k9). So, as far as N2 knows, N7 is the closest predecessor for k9. N2 will ask N7 to search further.

d. Since k13 ∉ (N2, N4], N2 is not the predecessor of k13. Node N2 needs to find the closest predecessor of k13. The table is searched from the bottom. N12 ∈ (N2, k13), so, as far as N2 knows, N12 is the closest predecessor for k13. N2 will ask N12 to search further.

P29-9. The following shows the ring.



P29-10. The following shows the ring, nodes, leaf sets, and routing tables.



P29-11. We use the algorithm in the text:

a. Key k24 is not in the range of the leaf set of N02. The routing table should be used. In this case $p = 0$ and $v = 2$, which means we need to look at the routing table entry [0, 2] = N20. The query is passed to node N20. N20 is not responsible for k24; the key k24 is not in the range of leaf set of N20;

the routing table is used. In this case, $p = 1$ and $v = 4$. Since table entry [1, 4] does not exist, the query is passed to a node sharing a prefix as long as the current node, but numerically closer to the key, which is N23. The node N23 is responsible for k24.

b. Key k12 is in the range of the leaf set of N20. The closest node in the leaf set to k12 is the node N11. The query is passed to N11, which is responsible for k12.

P29-12. The node identities in binary are N2(0010), N3(0011), N7(0111), N10(1010), and N12(1100). The following shows the routing tables. The explanation follows:

	N2	**N3**	**N7**	**N10**	**N12**
0	N10	N10	N12	N2	N7
1	N3	N2	N3	N12	N10
2	N3	N2			
3	N3	N2			

a. Node N2 has no common prefix with node N10 and node N12, but it is closest to N10 (using XOR criteria). It has one common prefix with N3 and N7, but it is closest to N3. It has two common prefixes with node N3. Finally, it has three common prefixes with N3.

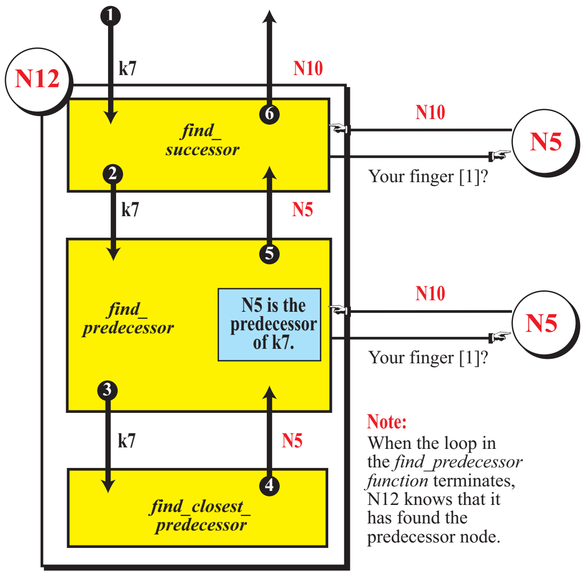
b. Node N3 has no common prefix with node N10 and node N12, but it is closest to N10. It has one common prefix with N2 and N7, but it is closest to N2. It has two common prefixes with node N2. Finally, it has three common prefixes with N2.

c. Node N7 has no common prefix with node N10 and node N12, but it is closest to N12. It has one common prefix with N2 and N3, but it is closest to N3. It has two common prefixes with no nodes. It has three common prefixes with no nodes.

d. Node N10 has no common prefix with nodes N2, N3, and N7, but it is closest to N2. It has one common prefix with N12. It has two common prefixes with no nodes. It has three common prefixes with no nodes.

e. Node N12 has no common prefix with nodes N2, N3, and N7, but it is closest to N7. It has one common prefix with N10. It has two common prefixes with no nodes. It has three common prefixes with no nodes.

P29-13. Let us go through the event as we did in Example 2.16 in the text. The following shows how N12 with some help from N5 goes through the sequence of events to find the successor of k7:



N12
1
k7
N10
6
find_ successor
2
k7
N5
5
find_ predecessor
N5 is the predecessor of k7.
3
k7
N5
4
find_closest_ predecessor
N10
N5
Your finger [1]?
N10
N5
Your finger [1]?
Note:
When the loop in the *find_predecessor function* terminates, N12 knows that it has found the predecessor node.

a. Event 1: Since N12 is not the responsible node for k7, it calls its *find_successor* function.

b. Event 2: N12 calls the *find_predecessor* function.

c. Event 3: Since N12 is not the predecessor node of k7, N12 calls the *find_closest_predecessor* function. This function checks the finger table of N12 from bottom to top. It finds that N5 is the closest predecessor because $5 \in (12, 7)$, N5 is between N12 and k7 when going around the ring in the clockwise direction. This can also be seen if we think of 5 as 37 (in modulo 32 arithmetic) and 7 as 39. Definitely $37 \in (12, 39)$. Note that we need to add 32 to 5 and 7 because we move from 12 to these two points as we pass point 0.

d. Event 4: The *find_closest_predecessor* function returns N5 to the *find_predecessor* function as the closest predecessor of k7. N12 finds that N5 is actually the predecessor of k7 (by remotely calling the N5.finger[1]) because $7 \in (5, 10]$.

e. Event 5: The *find_predecessor* function returns N5 to the *find_successor* function as the predecessor of k7.

f. Event 6: Node N12 finds that the successor node of k7 is N10 by remotely asking N5 to send its finger [1].

P29-14. For a node to be the predecessor of a key, the key needs to be a member of the half-open interval made of the node and its successor node. In other words, *key* $\in$ (12, 17]. Note that the interval is 13, 14, 15, 16, and 17; it does not include 12, but it includes 17.

a. k12 $\notin$ (12, 17], which means that N12 is not the predecessor of k12.

b. k15 $\in$ (12, 17], which means that N12 is the predecessor of k15.

c. k17 $\in$ (12, 17], which means that N12 is the predecessor of k17.

d. k22 $\notin$ (12, 17], which means that N12 is not the predecessor of k22.