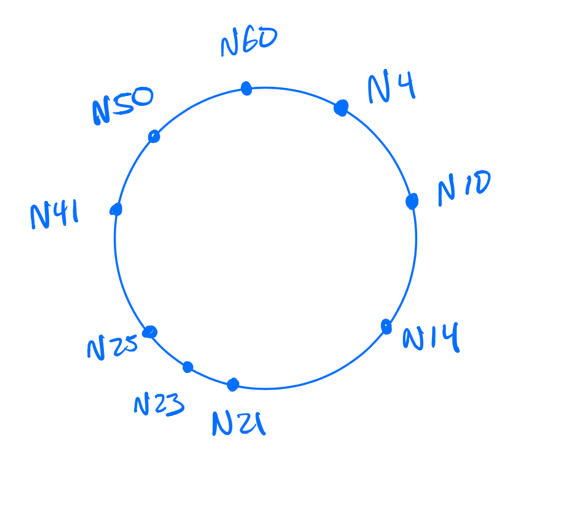
**Week 4 Chord Exercises**



The identifier ring of a Chord DHT overlay network.

1. This ring uses 6-bit identifiers. How many unique identifiers exist on this identifier ring?

26 = 64 identifiers

1. Key *k* is assigned to the node that follows *k*. This is the *successor(k)*. What is the *successor(12)?*

*successor(12)* = N14

1. A node is responsible for all keys between itself and its predecessor node. Which node is responsible for address 32?

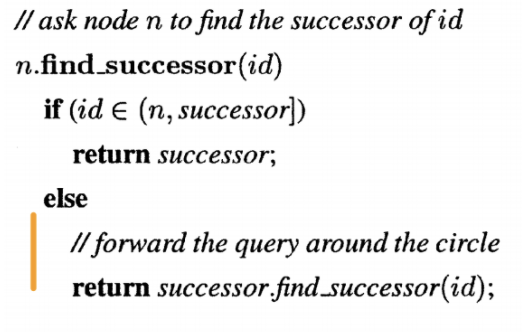
N41 is responsible for address 32

1. What is the set of keys that N21 is responsible for? Your answer should use interval notation e.g. [start, end) using brackets to denote that number is included in the set and parens to denote the number is not included in the set

N21 is responsible for the address range (13, 20]

1. The *successor(50) = 60.* If we assume that node 50 is aware that node 60 is its current successor, is this enough information for node 50 to know the set of keys that node 60 is responsible for? Explain your answer in a sentence or two

Yes; since node 50 is aware that node 60 is its current successor, node 50 knows that there isn’t a single node in between itself and node 60. Therefore node 50 can conclude that any address greater than its own address and less than or equal to node 60’s address is node 60’s responsibility; i.e. the range (50, 60].

1. If every node in the ring has a pointer to its current successor (i.e. we are ignoring nodes joining and leaving), a simple routing algorithm is possible. To locate the node responsible for a given key *Ksearch*, each node determines if its successor is responsible for *Ksearch*. If yes, then return the successor’s node ID. Else, forward the query onwards to your successor who repeats the process. The pseudocode is shown below. Using this algorithm, how many network hops would it take for node 50 to locate the node responsible for key 12?  
     
   

3 hops. 50 knows 60 is its successor, so it checks (50, 60] for 12, which fails and it jumps to the successor (60). 60’s successor is 4, so it checks the range (60, 4] (modulo 64) for 12, fails, and jumps to the successor 4 (that’s 2 hops so far). Node 4’s successor is 10; it checks the range between it and its successor (4, 10] for the address 12, which fails, so the algorithm makes a 3rd hop to node 10. Node 10’s successor is 14, so the algorithm checks the range (10, 14] for 12, which succeeds, so node 14 is returned. 3 hops were made.

1. If every node stored the current (again, we are ignoring nodes joining or leaving), how many network hops would it take for node 4 to determine the node that is responsible for key 22?

3 hops. 4 has successor 10, so checks (4, 10] for 22, which fails; first hop to node 10. 10 has successor 14, so checks (10, 14] for 22, which fails; second hop to node 14. 14 has successor 21, so checks (14, 21] for 22, which fails; third hop to node 21. 21 has successor 23, so checks (21, 23] for 22, which succeeds, so node 23 is returned after 3 hops.

1. Each node *Nsource* maintains a small table of “finger entries”. The table entry for *finger[k]* on a specific node lists the address of another node *Ntarget*. This ‘target’ is the node that is responsible for key *Nsource + 2k-1*. In precise terms, this is *successor(Nsource + 2k-1)*  
     
   If we wanted to create a finger table entry for the “next” node on the identifier ring, (e.g. the node immediately following *Nsource* which we would expect to be the node responsible for the key at *Nsource + 1*), what would be the value of *k*?   
     
   Hint: Remember that node identifiers and key identifiers share the same address space.   
   Hint 2: The node we are after is the *successor(Nsource + 1).* This question is directly asking “what value of *k* would give us our direct successor”?

The finger table entry for the next node on the identifier ring would be for k = 1.

1. In the example ring, what is the value for *finger[1]* on node 14?

N14’s *finger[1] = N21*.

1. This example ring uses 6-bit identifiers. If there was a Node 0 on this ring, what key would be identified by value of *finger[6]*? What would be the successor of that key (in other words, what would be the value of the finger table for *finger[6]*)?   
     
   Hint: Refer to question 1.   
   Hint: Do not miss that finger is calculated with *k-1,* not with *k* directly

*Successor(0 + 26-1) = Successor(32) = N41*.

1. If a ring uses *m* bits for addresses, then each node stores from *finger[1]* to *finger[m]* entries in the finger table. Our ring uses 6-bit addresses, so we store up to *finger[6]*. A node can only route to the addresses stored in its finger table. What is the maximum number of address values a node can skip past when routing? Express your answer both as a concrete number for our 6-bit example  
     
   Bonus: Express your answer as a formula

For 6-bit addresses, the maximum number of skipped addresses is:  
*26-1-1 = 31* (if you don’t count the address landed on as a “skipped past” address)  
  
In general, for an *m*-bit address-space, the maximum number of skipped addresses would be (again, not counting the address landed on as a “skipped past” address):  
*2m-1-1*.

1. For node N10, what is its *finger[4]* entry?

N10’s *finger[4] = successor(10 + 24-1 mod 64)  
 = successor(28)  
 = N41*.

1. Is node N21 responsible for key K16?

Yes, there is no node in the range [16, 21), so N21 is responsible for K16.

1. If node 10 wants to find the node responsible for K34 e.g. *successor(34),* it will search it’s finger table. Would it be possible for node 21 to be the *successor(34)*?  
     
   Hint: Look at the definition of successor. Is it possible for a lower-valued node to be the successor for a higher-valued key?   
     
   Clarification: We are ignoring modular arithmetic here….please don’t worry about the edge case ☺

No, node 10 inherently knows itself exists and (in checking node 21 from its finger table) knows node 21 exists. Even with modular arithmetic, if K34’s successor-search circled back around to K0, it would hit node 10 first before node 21. N10 would be the successor of K34 before hitting N21. It’s not possible given what we know for N21 to be K34’s successor.

1. If node 10 wants to find the node responsible for K34 e.g. *successor(34),* it will search its finger table. Would it be possible for node 50 to be the *successor(34)*?

Yes, since node 10 only knows that itself and 50 exist, it’s possible that nothing exists in the range [34, 50), so 50 could indeed be the successor of 34.

1. In general, does a node’s identifier need to be lower or higher than a key’s identifier for the node to be a possible candidate for *successor(key)*?

The node’s identifier needs to be higher than the key’s identifier (modular arithmetic-adjusted) for it to be a possible successor of that key.

1. If a node *N* knows that it is not responsible for a key K, and we assume routing can only go in the positive direction, then is the key identifier for K bigger or smaller than the node identifier for *N*?

*K* would have to be bigger than *N*.

1. Build the finger table for node 4  
   *Finger[i] = Successor(4 + 2i-1 mod 64)*

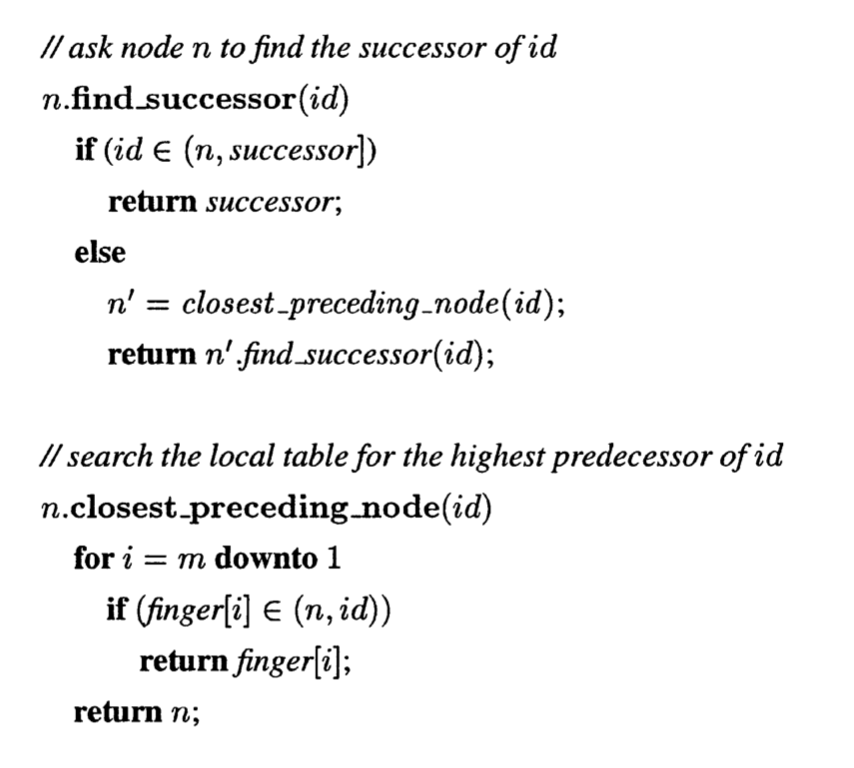
|  |  |
| --- | --- |
| *i* | *Finger[i]* |
| 1 | N10 |
| 2 | N10 |
| 3 | N10 |
| 4 | N14 |
| 5 | N21 |
| 6 | N41 |

1. The following is the chord routing algorithm. Attempt to route from node 4, looking for key 12  
   1. Check if *id ∈ (n, successor] : 12 ∈ (4, 10]* *= False*
   2. *n' = 4.closest\_preceding\_node(12)*
      1. *4.closest\_preceding\_node(12):*
         1. loop for *i = 6..1* and check if *finger[i] ∈ (4, 12):*

|  |  |  |  |
| --- | --- | --- | --- |
| *i* | *X = n + 2i-1 mod N* | *finger[i] (i.e. successor(X)* | *finger[i] ∈ (4, 12)* |
| 6 | 36 | 41 | False |
| 5 | 20 | 21 | False |
| 4 | 12 | 14 | False |
| 3 | 8 | 10 | True |

* + 1. Returns *N10* to *n’*
  1. *n'.find\_successor(id) 🡪 10.find\_successor(12)*
     1. Check if *id ∈ (n, successor] 🡪 12 ∈ (10, 14]* *🡪 True*
     2. Return *N14*
  2. Return *N14* (final routing solution)

The route from N4 looking for K12 is N4 🡪 N10 🡪 N14.

**