Skip Lists

Saikrishna Arcot
M. Hudachek-Buswell

July 18, 2020

Idea of Skip Lists

- A linked list can be used to store items in ascending or descending order.

Idea of Skip Lists

- A linked list can be used to store items in ascending or descending order.
- However, searching for a particular item will still be $O(n)$.

Idea of Skip Lists

- A linked list can be used to store items in ascending or descending order.
- However, searching for a particular item will still be $O(n)$.
- This could be reduced by having some items (where these items are randomly chosen) be in another list, and have these items act like a "marker". That way, you can reduce the search time by searching fewer items to see where in the list it might be.

Skip Lists

- A skip list is similar to a linked list with the items in ascending order, but a skip list has multiple **levels**.

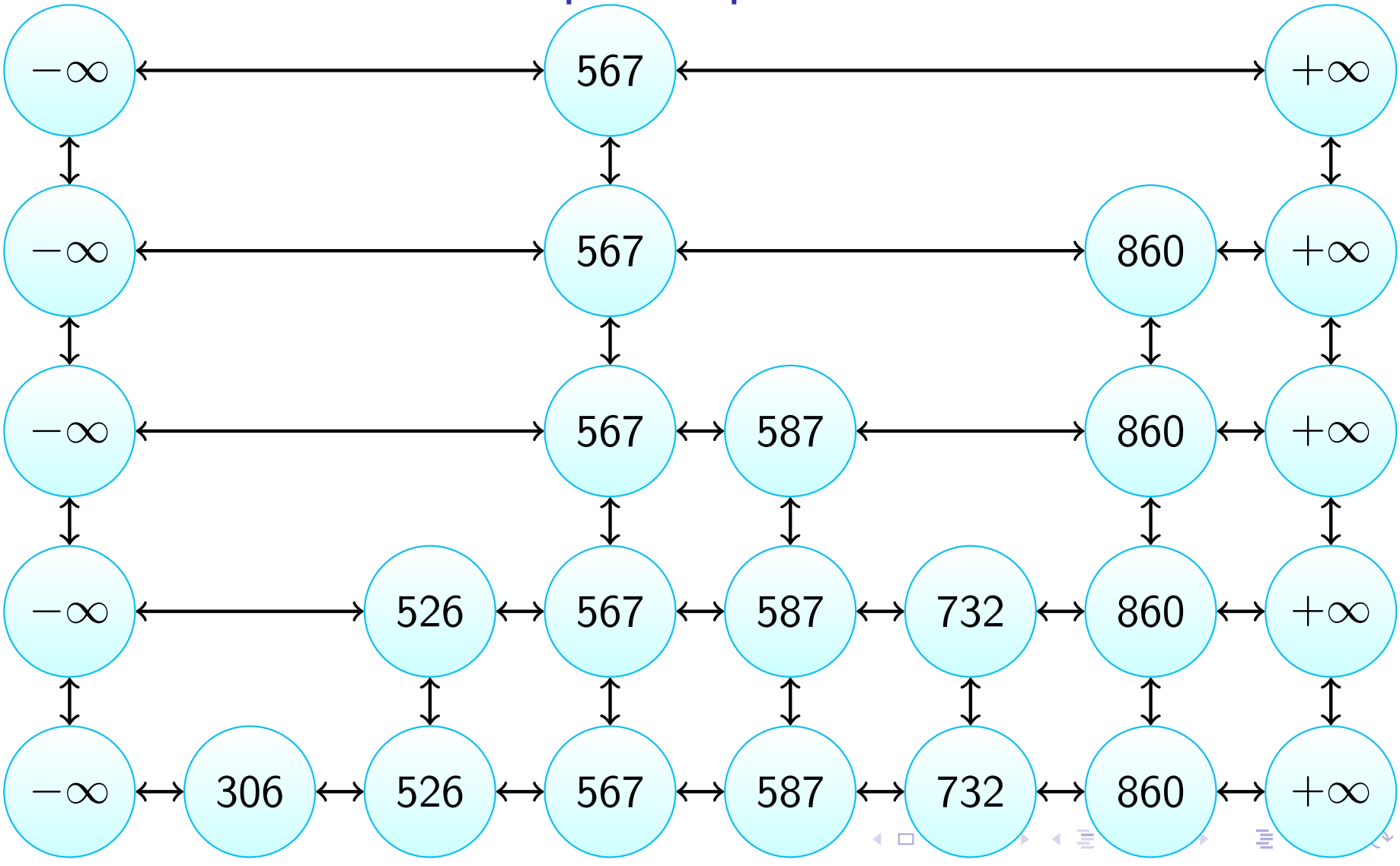
Skip Lists

- A skip list is similar to a linked list with the items in ascending order, but a skip list has multiple **levels**.
- Each level has (ideally) half of the items on the level below it. This means that the first level (usually called level 0) would have all of the items, the second level would have half of all of the items, the third level would have quarter of all of the items, etc.

Skip Lists

- A skip list is similar to a linked list with the items in ascending order, but a skip list has multiple **levels**.
- Each level has (ideally) half of the items on the level below it. This means that the first level (usually called level 0) would have all of the items, the second level would have half of all of the items, the third level would have quarter of all of the items, etc.
- When adding a new item, a **coin toss** is used to determine if the item gets promoted to the next level and, if so, how many times.

Example Skip List



Coin Toss

- When adding a new node into a skip list, a coin toss is used to promote the item.

Coin Toss

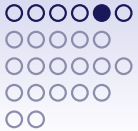
- When adding a new node into a skip list, a coin toss is used to promote the item.
- A coin toss can either be someone flipping an actual coin or some method that randomly gives heads or tails (or some representation of it). The idea is that this "coin" is random.

Coin Toss

- When adding a new node into a skip list, a coin toss is used to promote the item.
- A coin toss can either be someone flipping an actual coin or some method that randomly gives heads or tails (or some representation of it). The idea is that this "coin" is random.
- If the coin comes up heads, then the item is promoted to the next level, and another flip is done. If the coin comes up tails, then the item is not promoted, and no more flips are done.

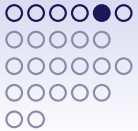
Coin Toss

- When adding a new node into a skip list, a coin toss is used to promote the item.
- A coin toss can either be someone flipping an actual coin or some method that randomly gives heads or tails (or some representation of it). The idea is that this "coin" is random.
- If the coin comes up heads, then the item is promoted to the next level, and another flip is done. If the coin comes up tails, then the item is not promoted, and no more flips are done.
- Because of this randomness, the resulting skip list might not be perfectly arranged (with only half of the items on a level being present on the level above it).



Skip List Nodes

- Each circle in the previous example represents a unique node in a skip list.



Skip List Nodes

- Each circle in the previous example represents a unique node in a skip list.
- Each node holds a reference to the item being stored in the node.

Skip List Nodes

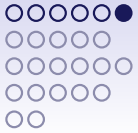
- Each circle in the previous example represents a unique node in a skip list.
- Each node holds a reference to the item being stored in the node.
- Each node also holds a reference to the previous and next nodes in that level, and the nodes directly above and below that node. The nodes directly above and below a node should hold a reference to the same item.

Skip List Nodes

- Each circle in the previous example represents a unique node in a skip list.
- Each node holds a reference to the item being stored in the node.
- Each node also holds a reference to the previous and next nodes in that level, and the nodes directly above and below that node. The nodes directly above and below a node should hold a reference to the same item.
- Finally, each node also contains the level the node is on; all nodes on the bottom level typically have a level of 0.

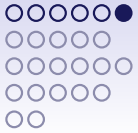
Skip List Nodes

- Each circle in the previous example represents a unique node in a skip list.
- Each node holds a reference to the item being stored in the node.
- Each node also holds a reference to the previous and next nodes in that level, and the nodes directly above and below that node. The nodes directly above and below a node should hold a reference to the same item.
- Finally, each node also contains the level the node is on; all nodes on the bottom level typically have a level of 0.
- Because each node has four references (up, down, left, right), this type of node might also be known as a **quad-node**.



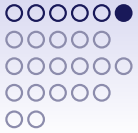
Phantom Nodes

- Unlike most other data structures, skip lists need to use **phantom nodes**.



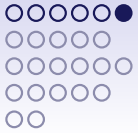
Phantom Nodes

- Unlike most other data structures, skip lists need to use **phantom nodes**.
- Phantom nodes are typically special nodes that are used as a start and/or end marker.



Phantom Nodes

- Unlike most other data structures, skip lists need to use **phantom nodes**.
- Phantom nodes are typically special nodes that are used as a start and/or end marker.
- For skip lists, at the start of each level, there needs to be a phantom node that represents negative infinity. At the end of each level, there can *optionally* be a phantom node that represents positive infinity.

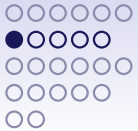


Phantom Nodes

- Unlike most other data structures, skip lists need to use **phantom nodes**.
- Phantom nodes are typically special nodes that are used as a start and/or end marker.
- For skip lists, at the start of each level, there needs to be a phantom node that represents negative infinity. At the end of each level, there can *optionally* be a phantom node that represents positive infinity.
- The negative infinity phantom nodes are required; the positive infinity phantom nodes are not.

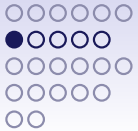
Phantom Nodes

- Unlike most other data structures, skip lists need to use **phantom nodes**.
- Phantom nodes are typically special nodes that are used as a start and/or end marker.
- For skip lists, at the start of each level, there needs to be a phantom node that represents negative infinity. At the end of each level, there can *optionally* be a phantom node that represents positive infinity.
- The negative infinity phantom nodes are required; the positive infinity phantom nodes are not.
- Positive and negative infinity may not be represented in all programming languages, another special value can be used.



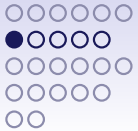
Searching

- Start at the top-left phantom node.



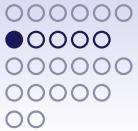
Searching

- Start at the top-left phantom node.
- Check the data in the node to the right.



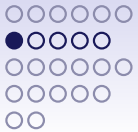
Searching

- Start at the top-left phantom node.
- Check the data in the node to the right.
  - If the data you're looking for is less than the data in the node, or there is no node there, go down one level.



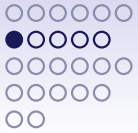
Searching

- Start at the top-left phantom node.
- Check the data in the node to the right.
  - If the data you're looking for is less than the data in the node, or there is no node there, go down one level.
  - If the data you're looking for is greater than the data in the node, go to that node.



Searching

- Start at the top-left phantom node.
- Check the data in the node to the right.
  - If the data you're looking for is less than the data in the node, or there is no node there, go down one level.
  - If the data you're looking for is greater than the data in the node, go to that node.
  - If the data you're looking for is equal to the data in the node, then you've found the data.

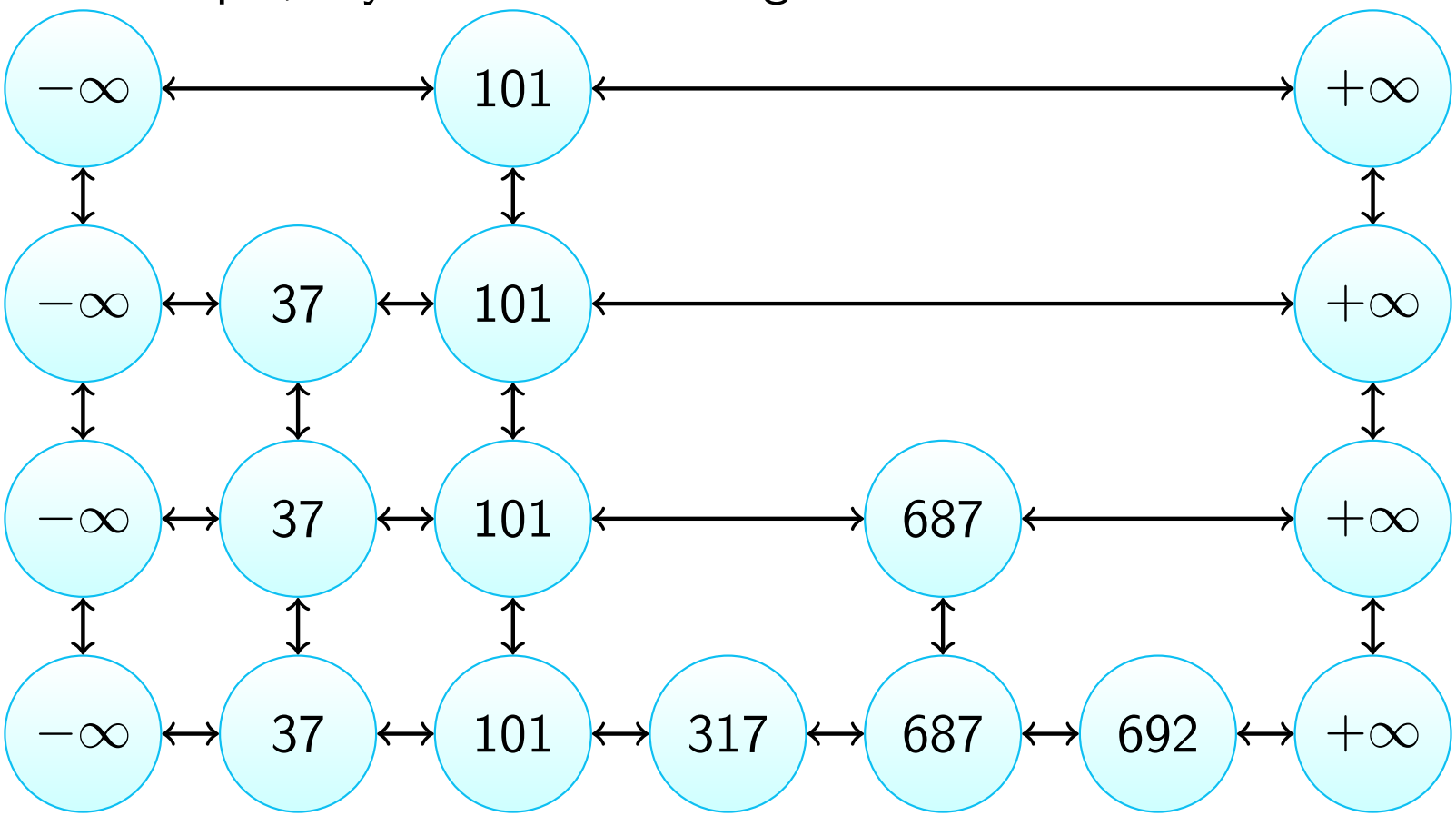


Searching

- Start at the top-left phantom node.
- Check the data in the node to the right.
  - If the data you're looking for is less than the data in the node, or there is no node there, go down one level.
  - If the data you're looking for is greater than the data in the node, go to that node.
  - If the data you're looking for is equal to the data in the node, then you've found the data.
- Repeat the previous step, until you find the node or go off of the list, in which case the data you're looking for isn't in the skip list.

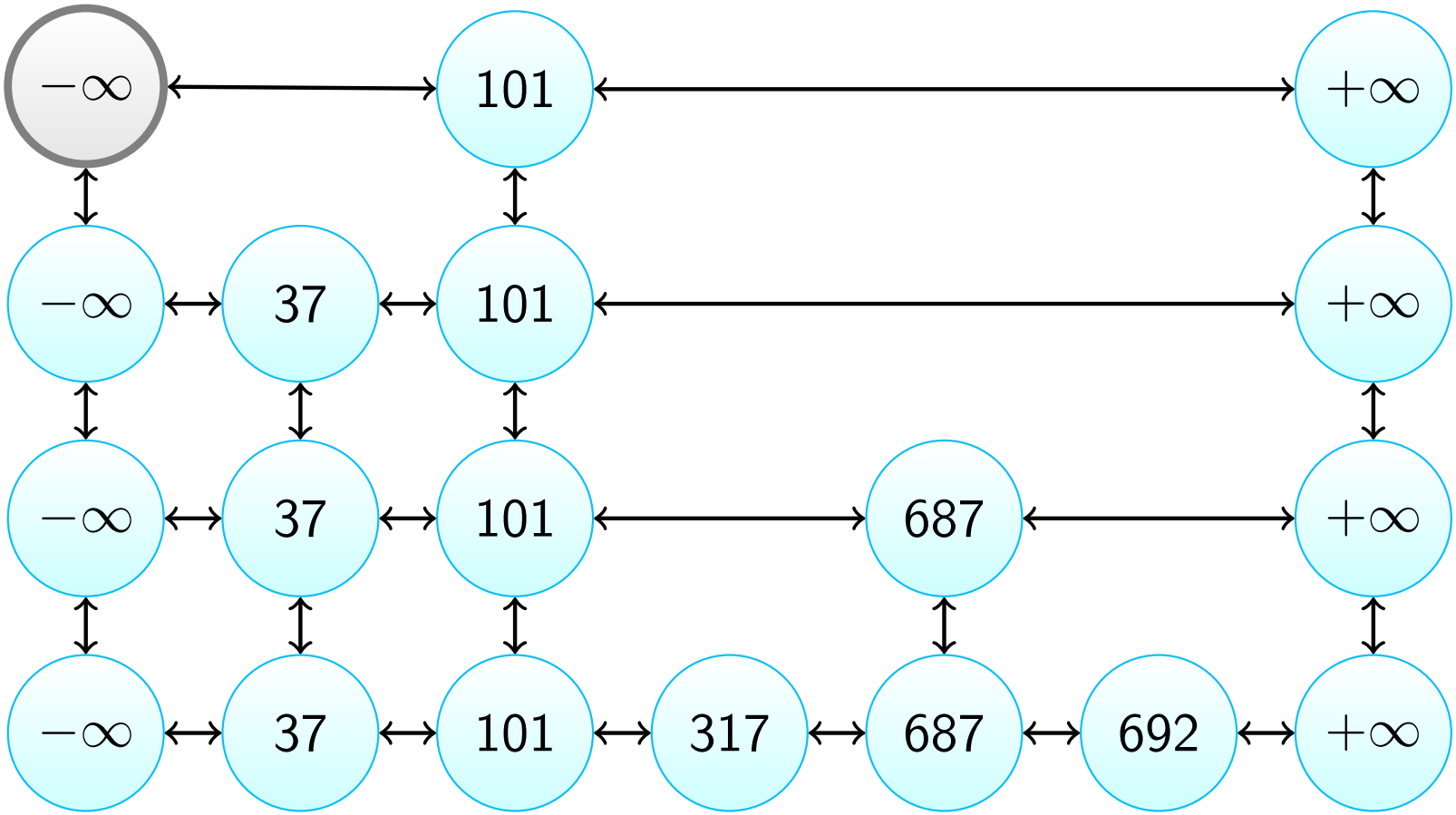
Searching

For example, if you were searching for 687:



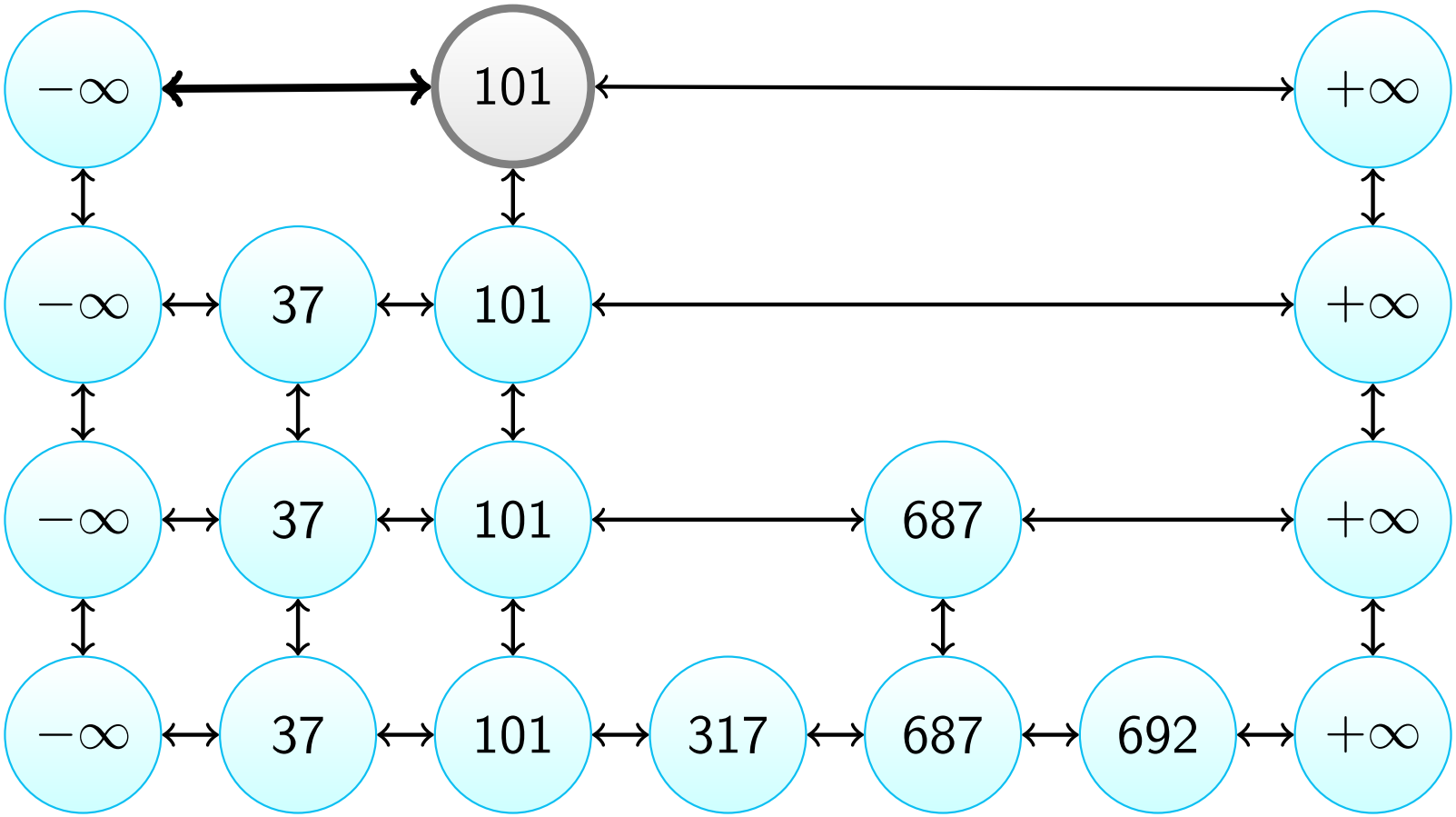
Searching

For example, if you were searching for 687:



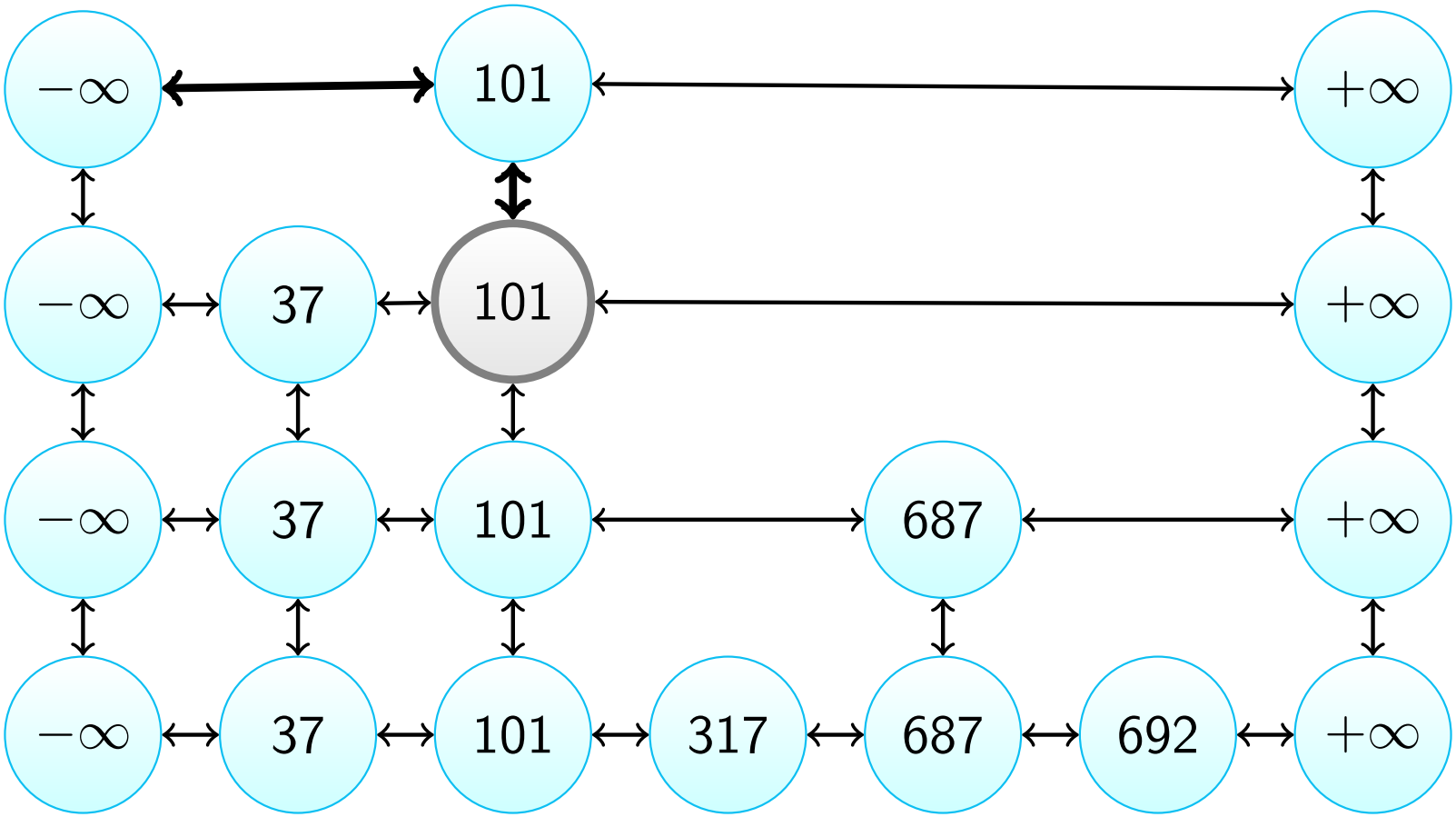
Searching

For example, if you were searching for 687:



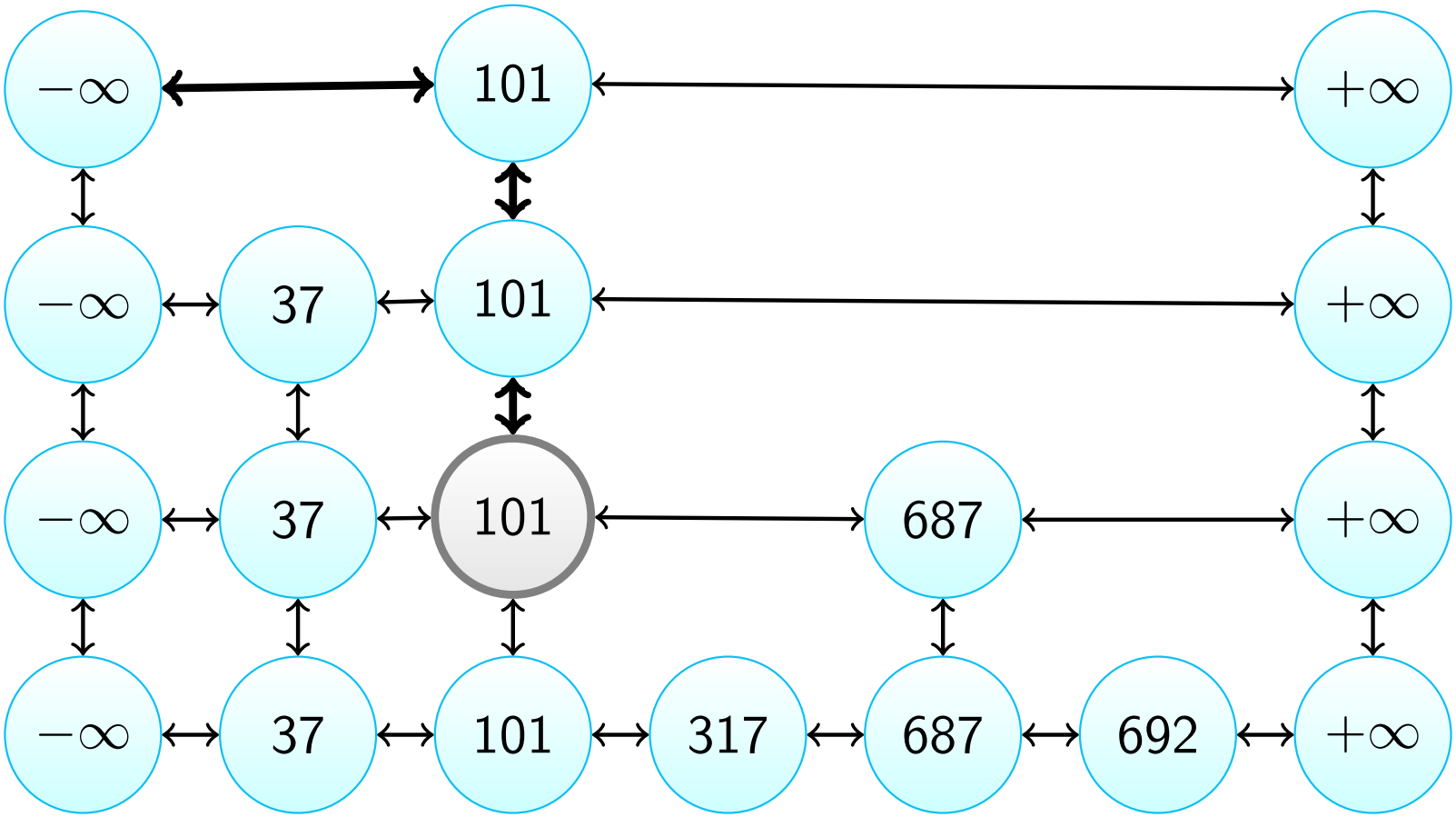
Searching

For example, if you were searching for 687:



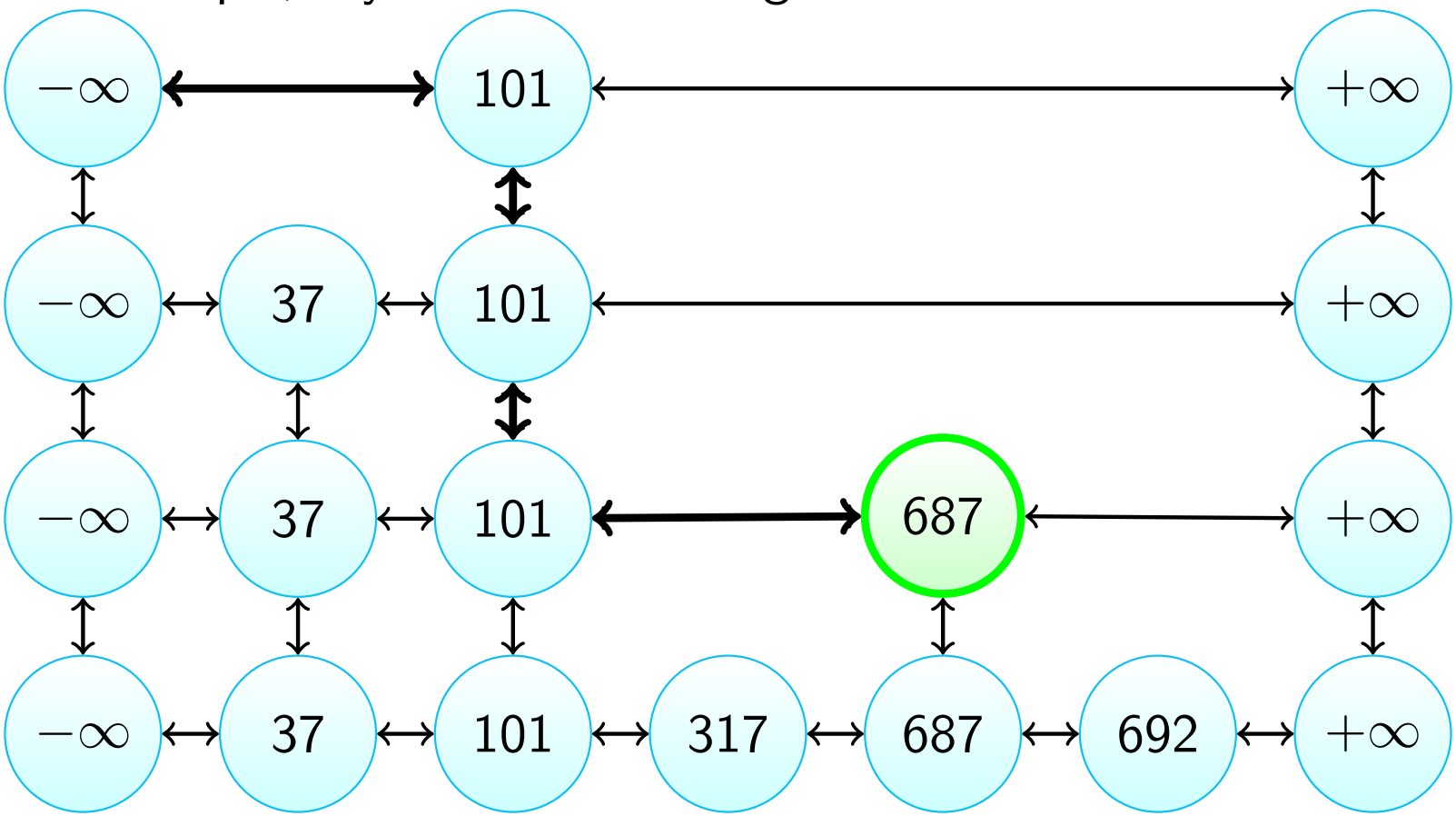
Searching

For example, if you were searching for 687:



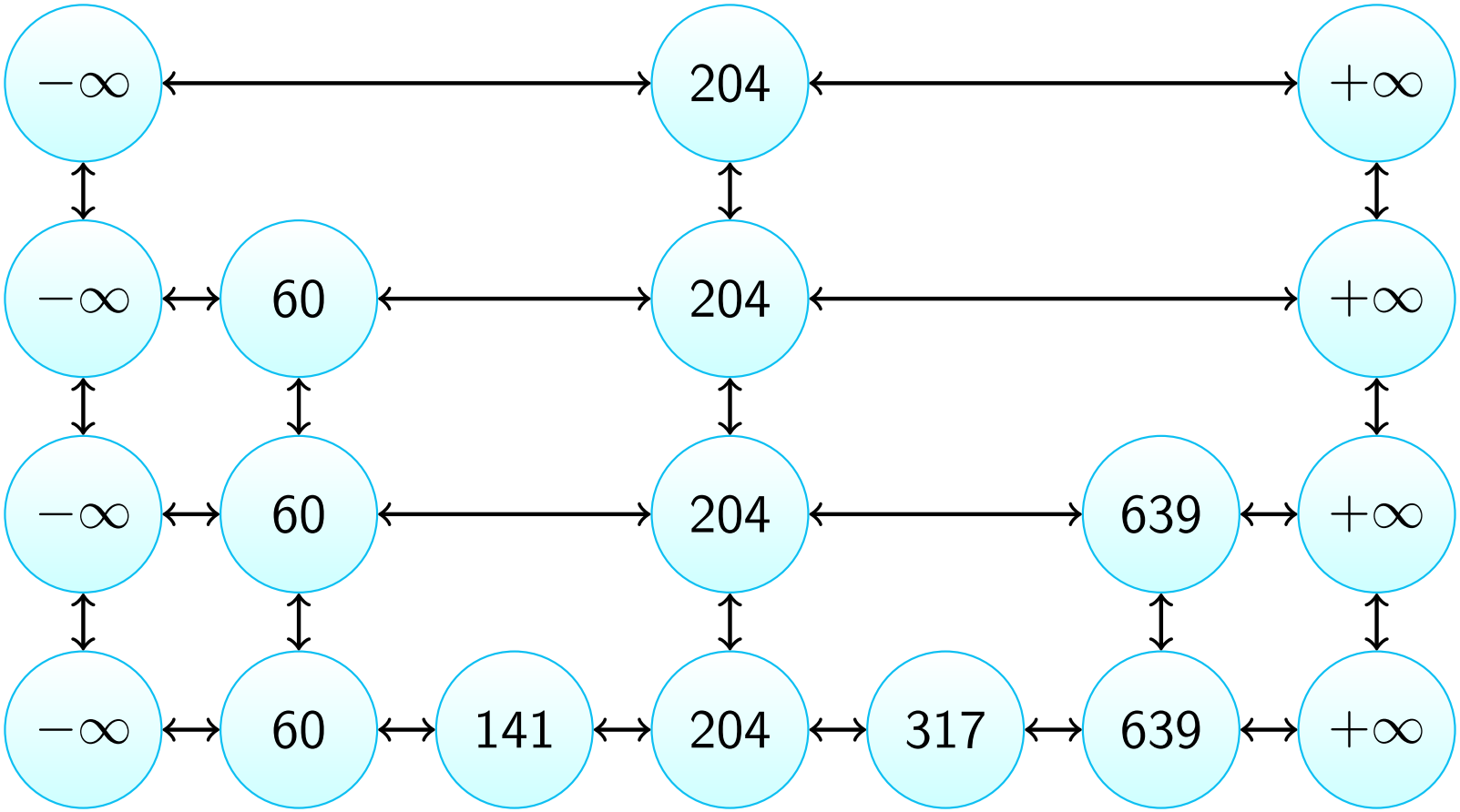
Searching

For example, if you were searching for 687:



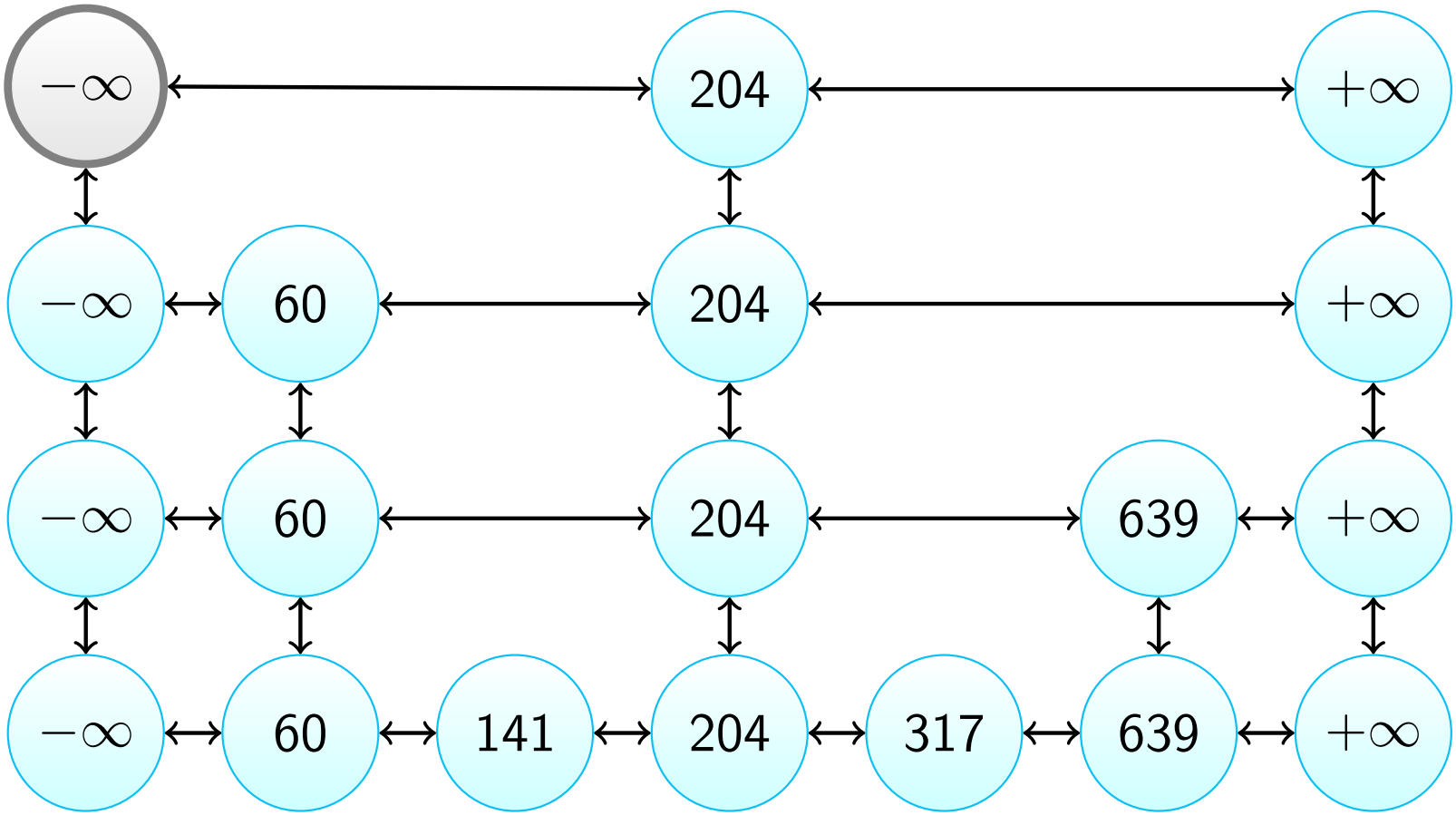
Searching

For example, if you were searching for 141:



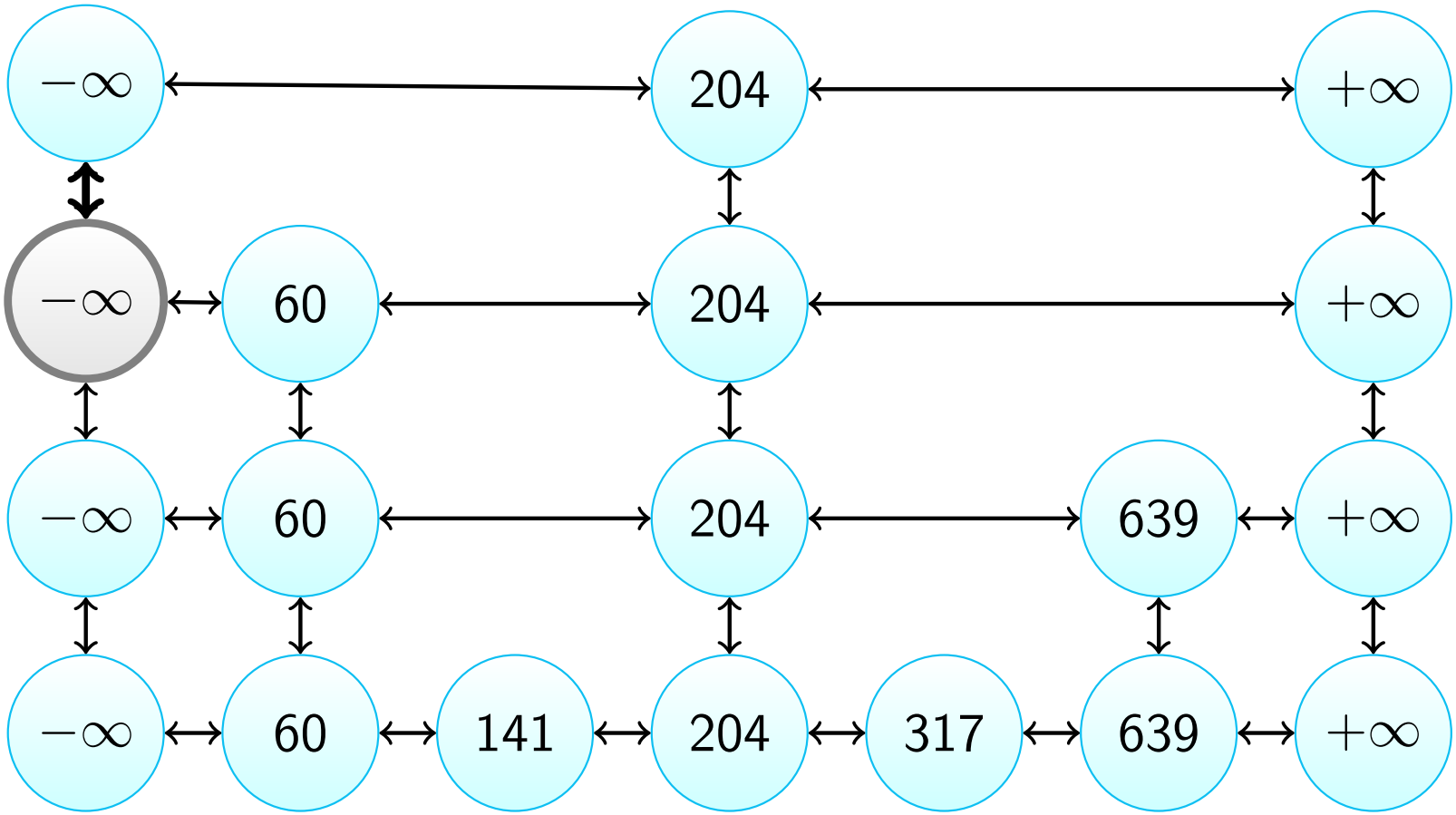
Searching

For example, if you were searching for 141:



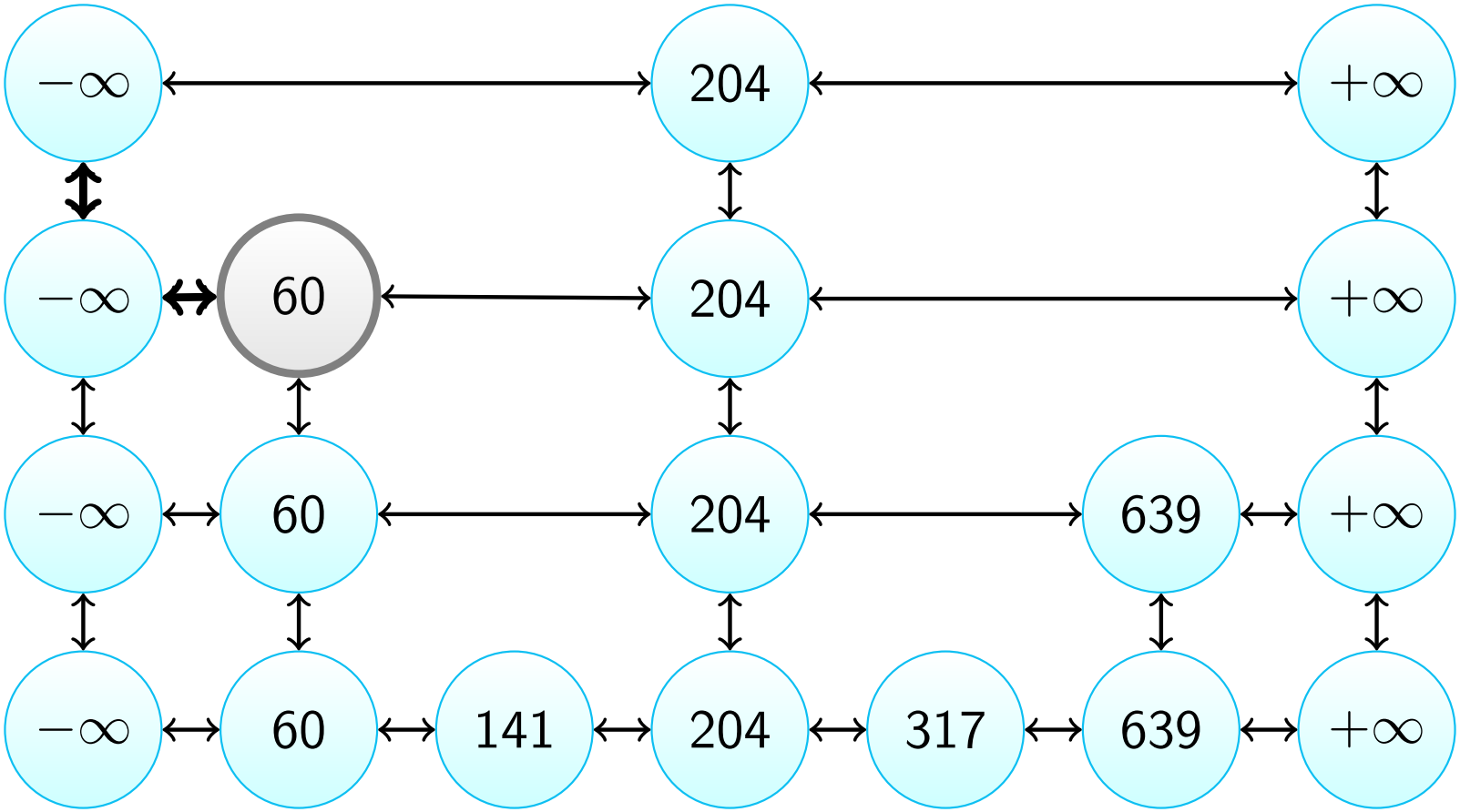
Searching

For example, if you were searching for 141:



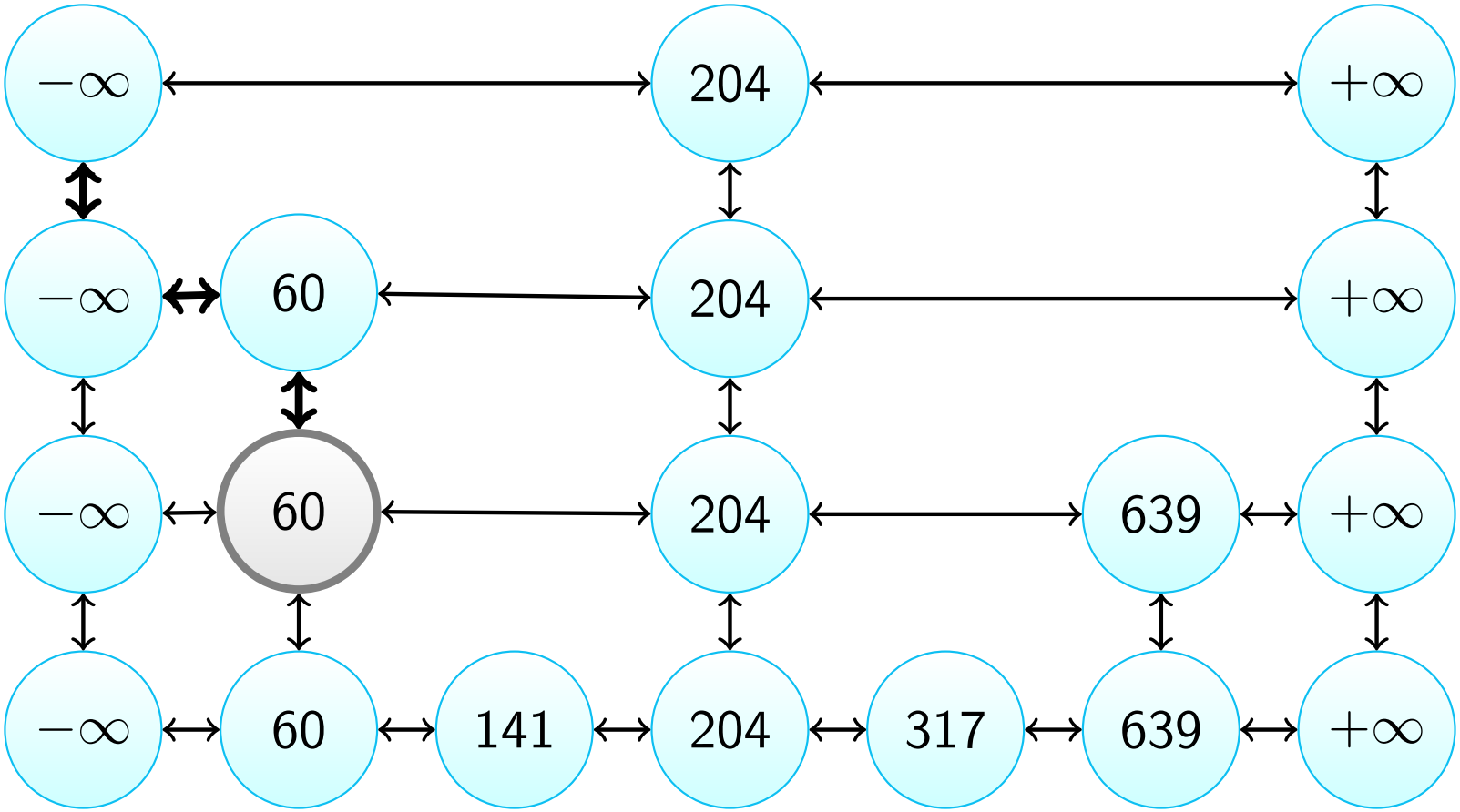
Searching

For example, if you were searching for 141:



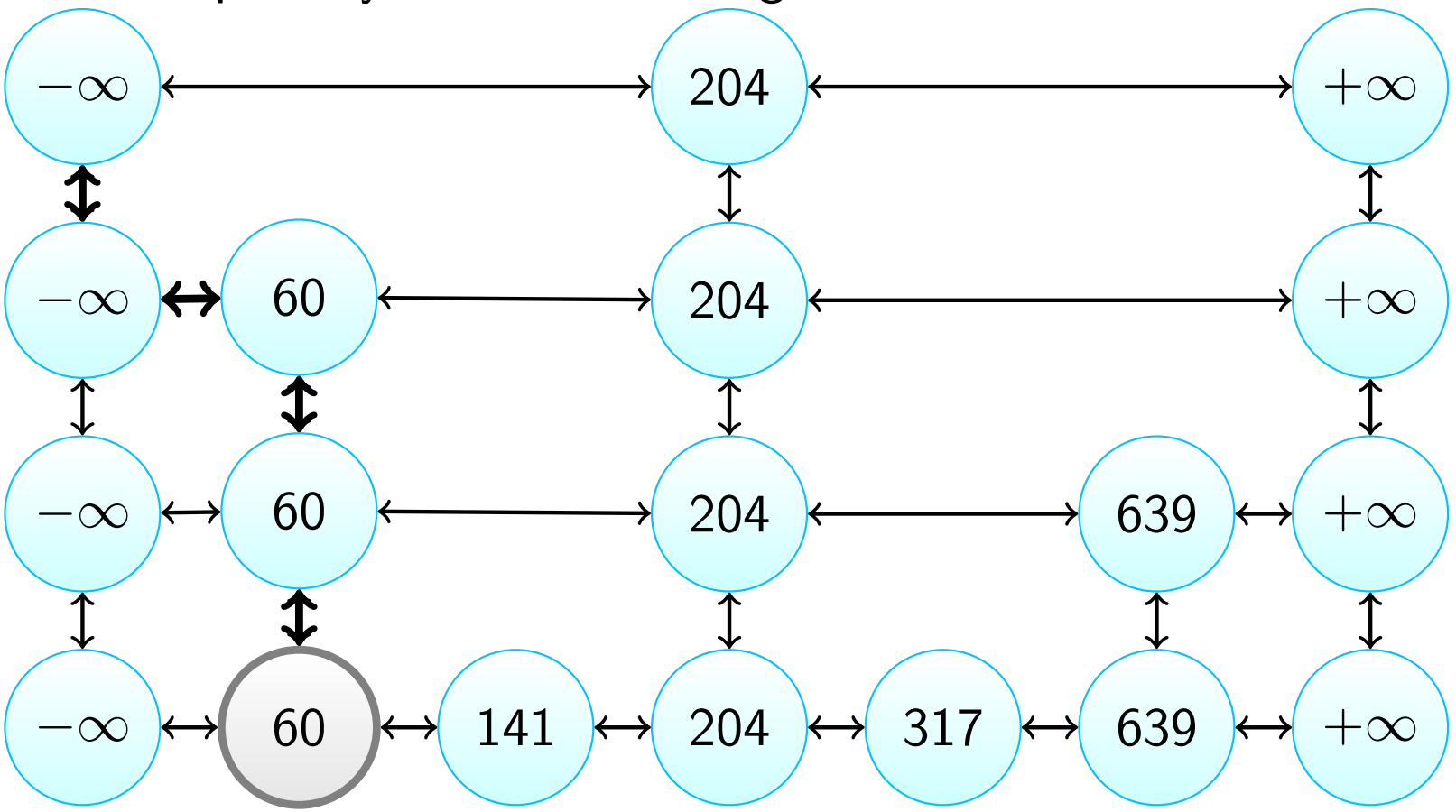
Searching

For example, if you were searching for 141:



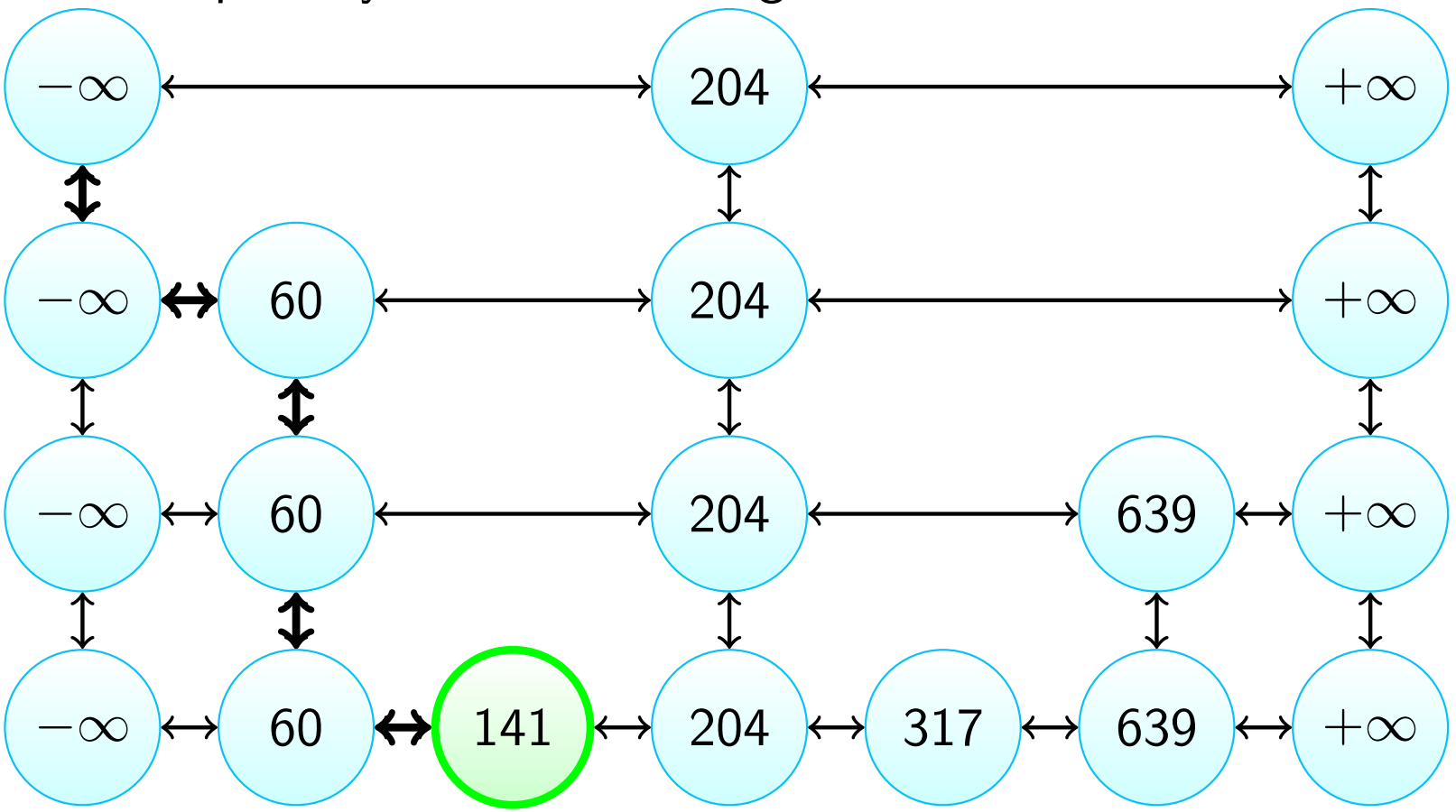
Searching

For example, if you were searching for 141:



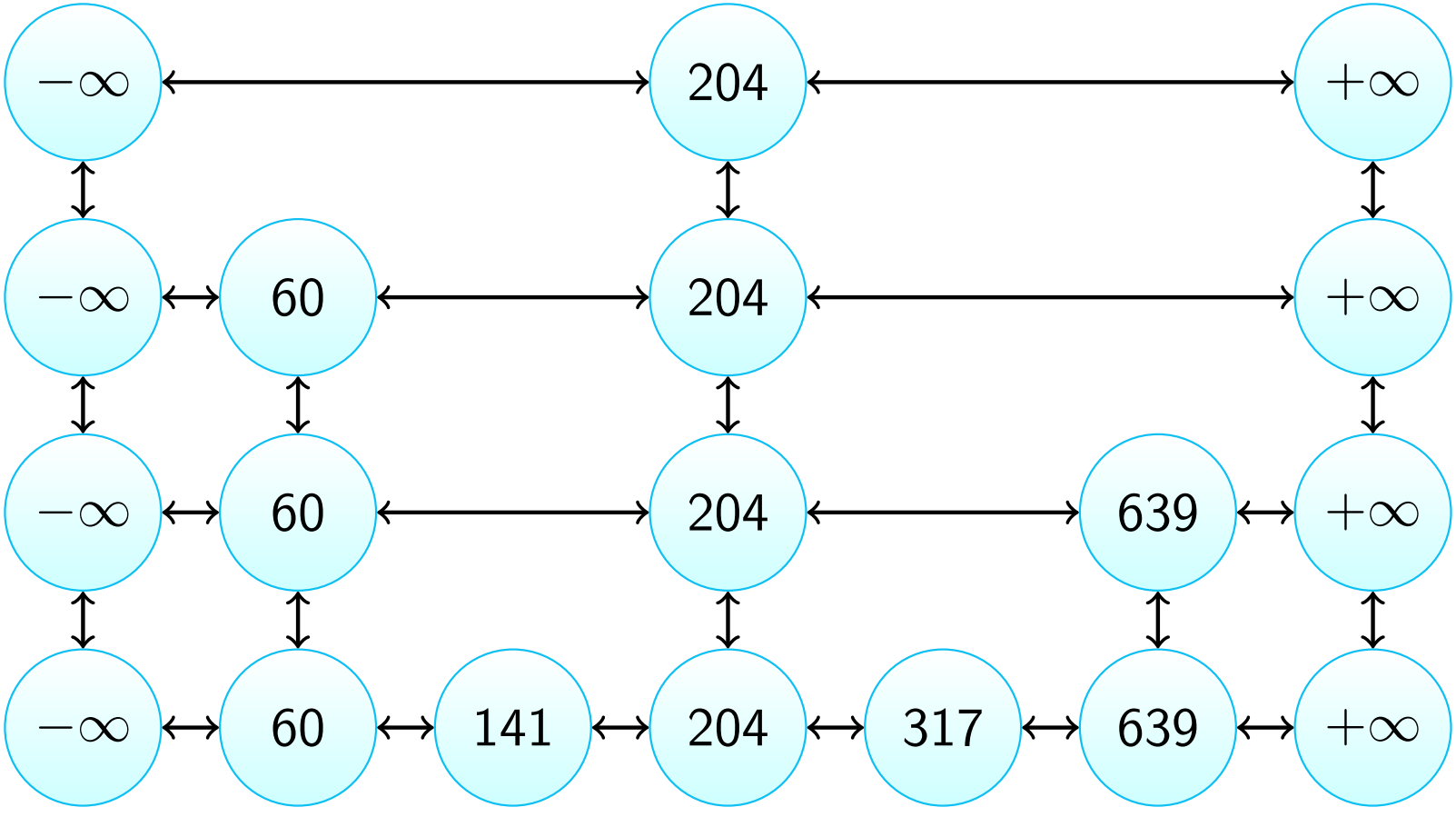
Searching

For example, if you were searching for 141:



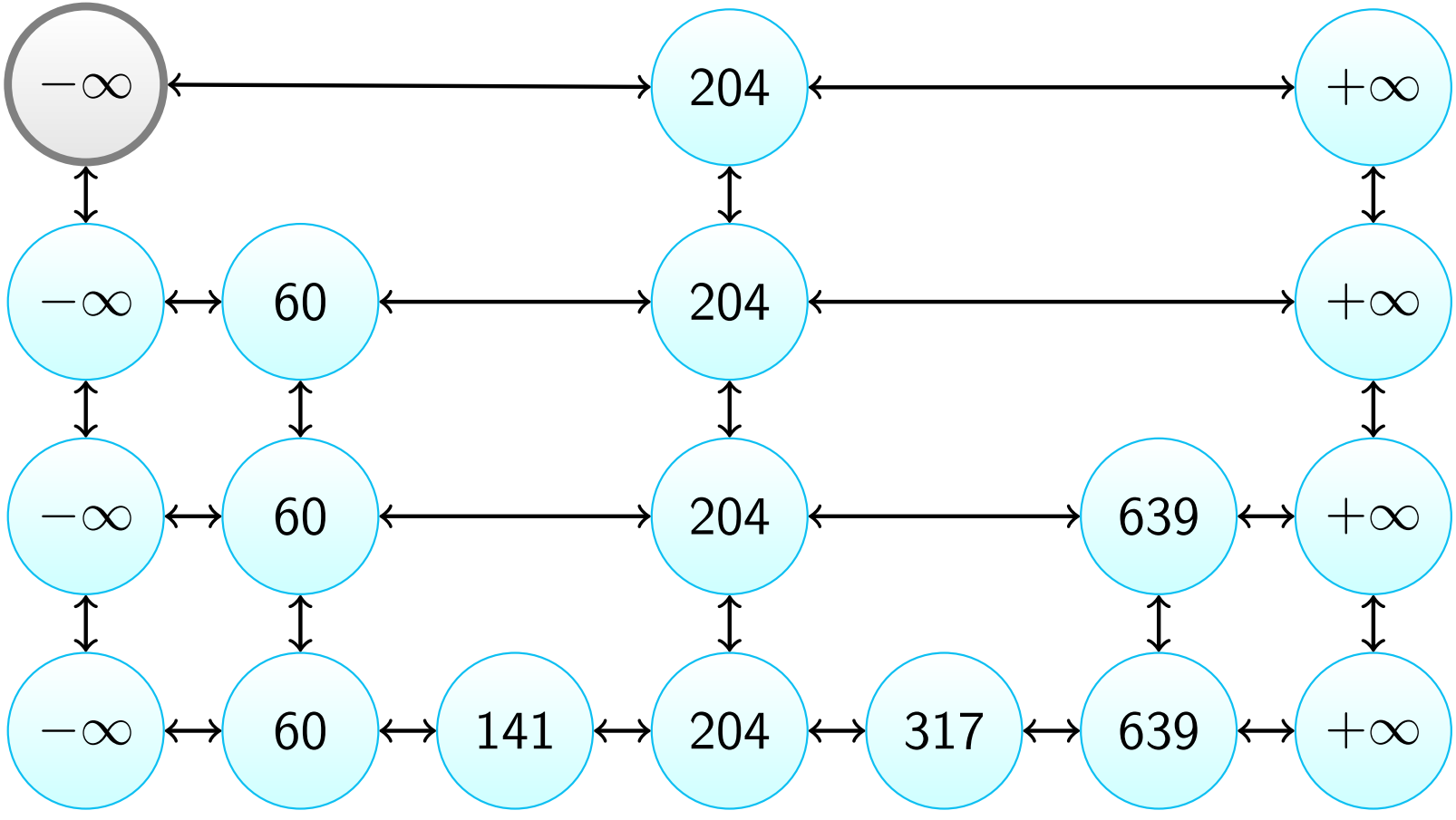
Searching

For example, if you were searching for 405:



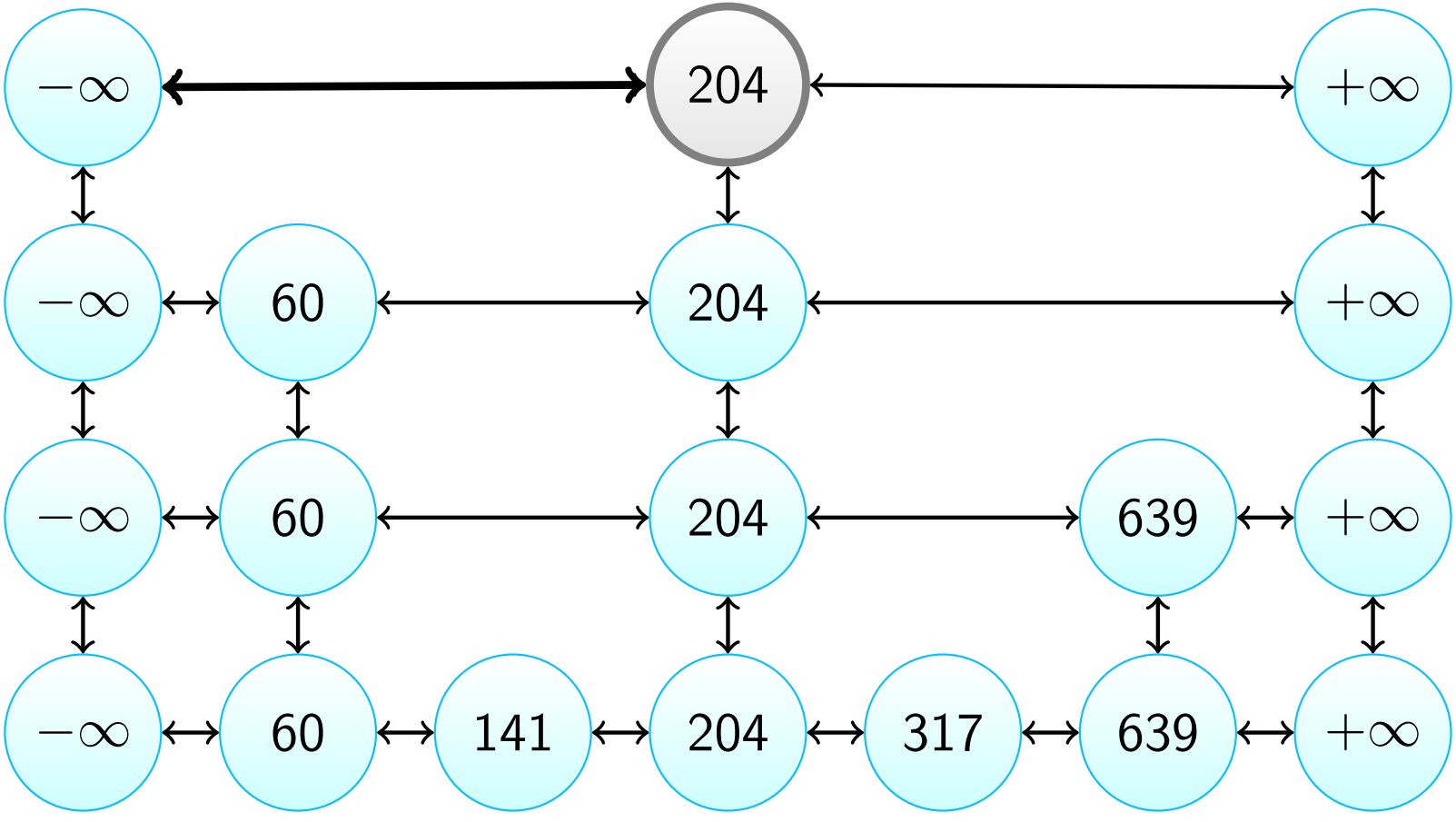
Searching

For example, if you were searching for 405:



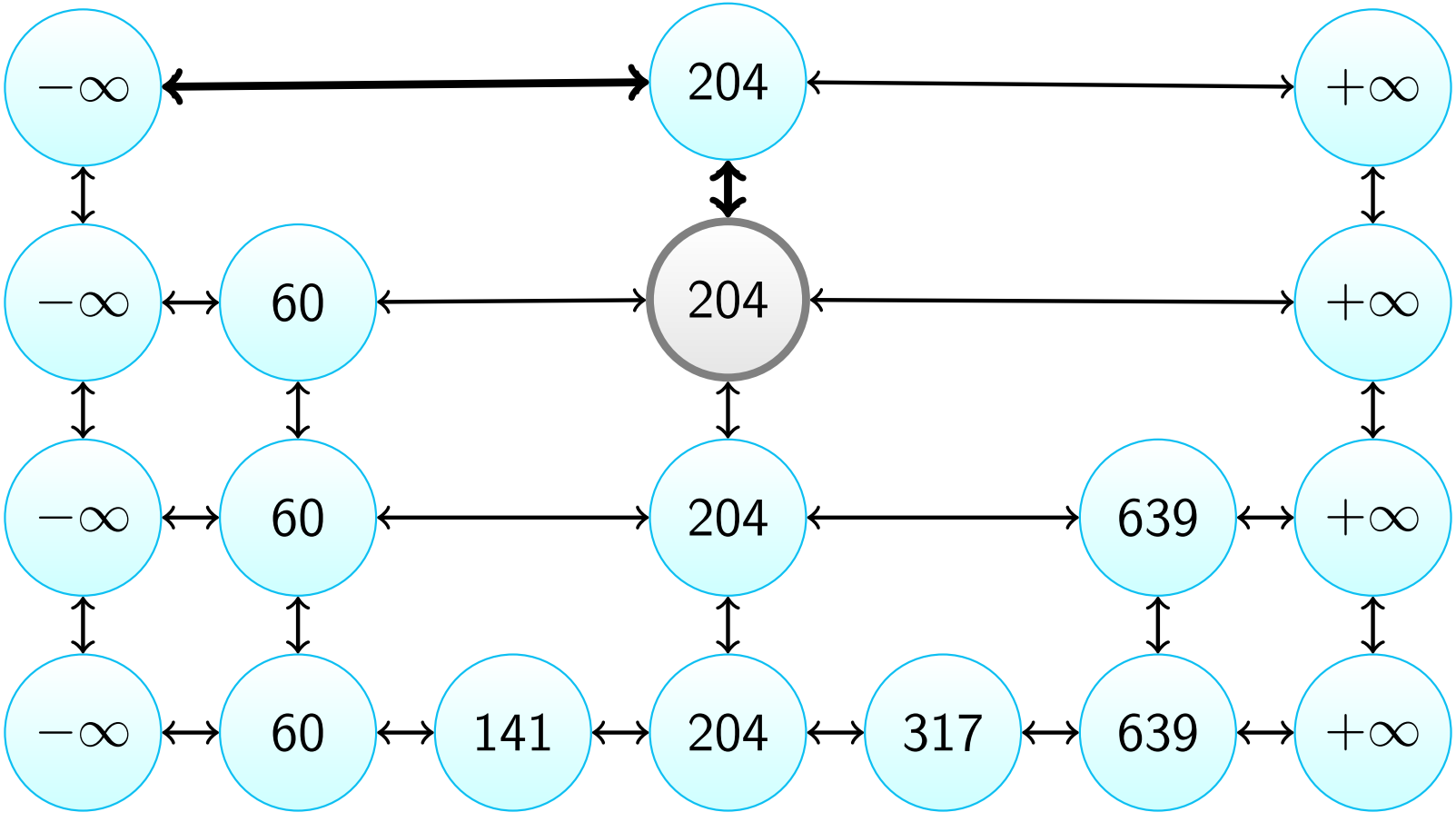
Searching

For example, if you were searching for 405:



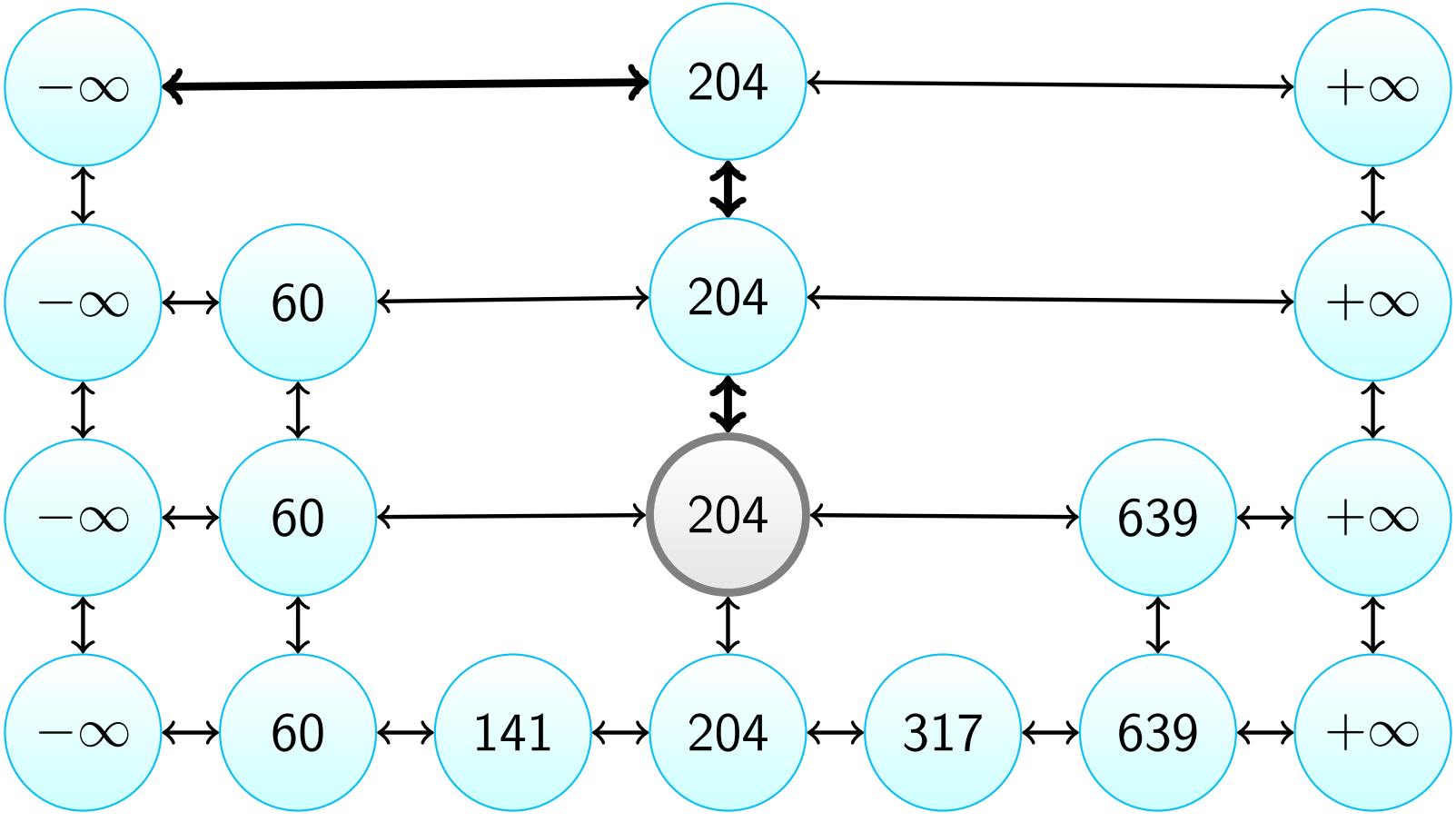
Searching

For example, if you were searching for 405:



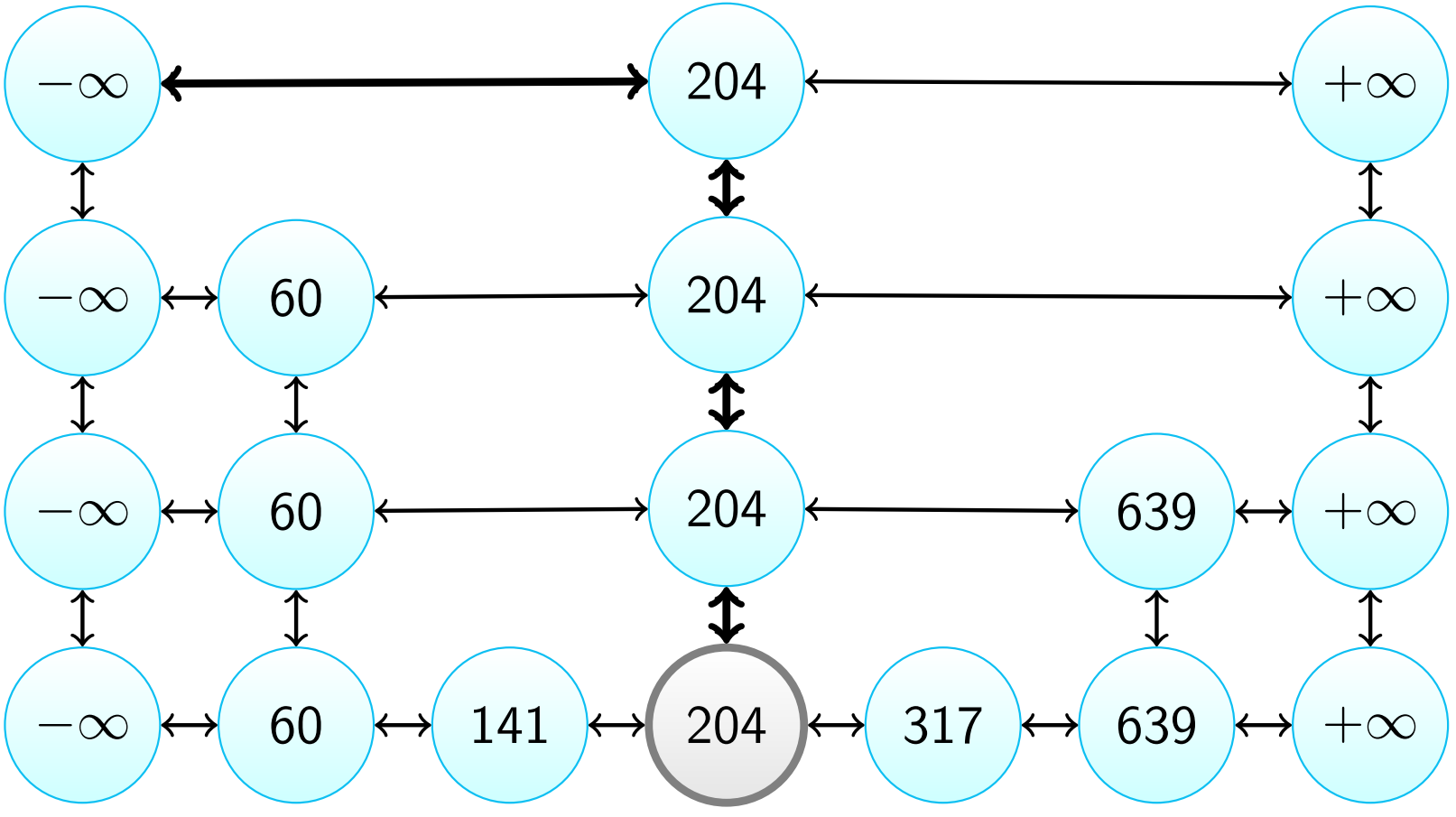
Searching

For example, if you were searching for 405:



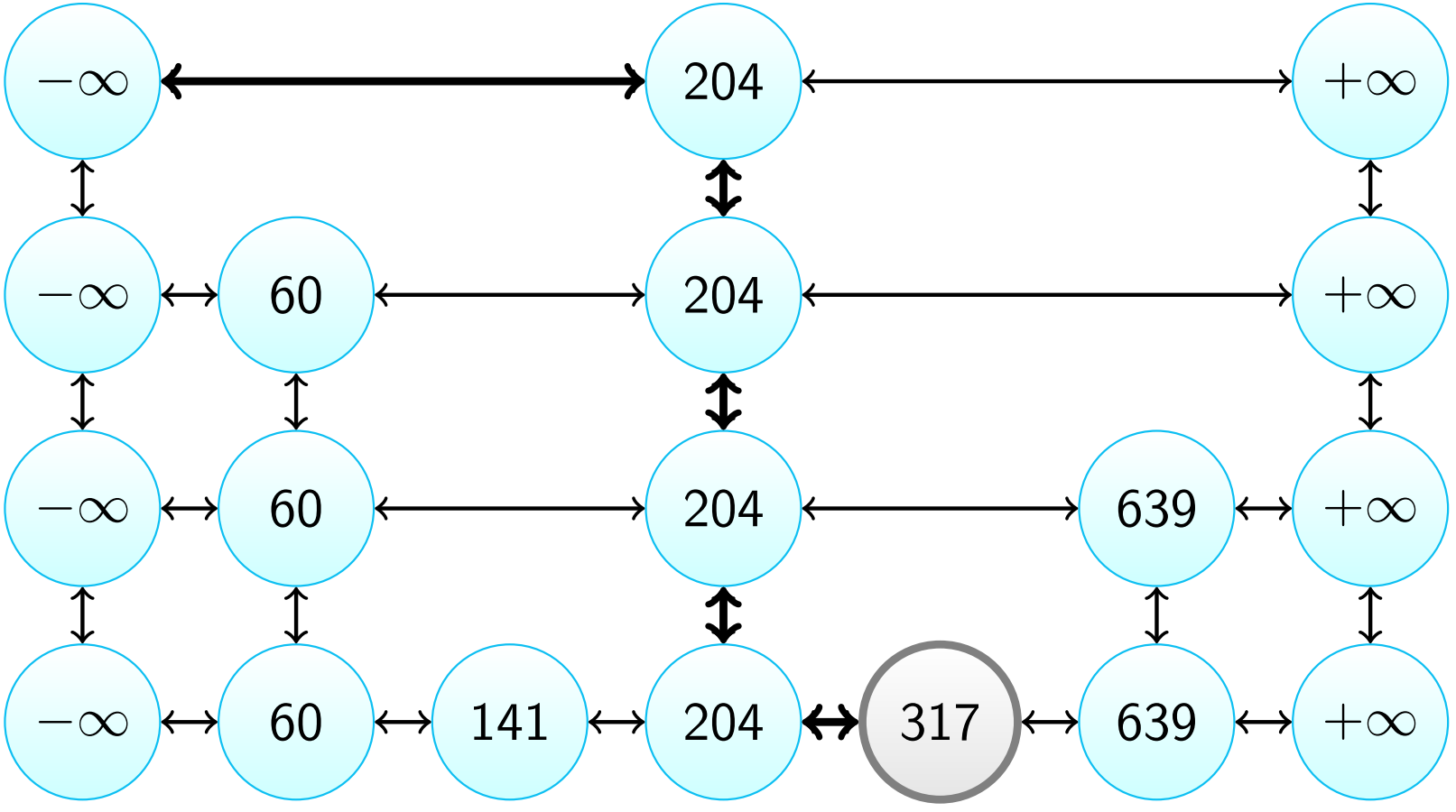
Searching

For example, if you were searching for 405:



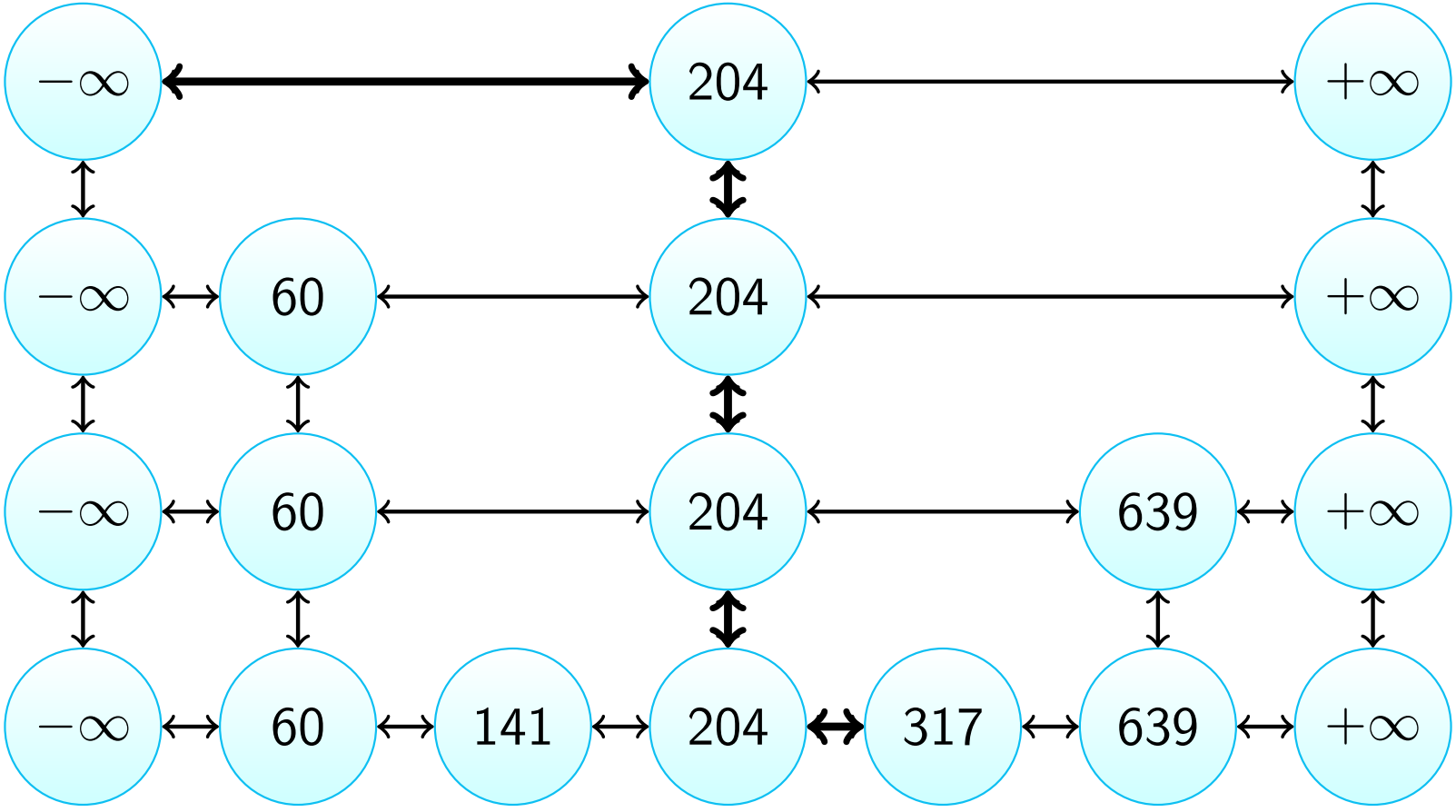
Searching

For example, if you were searching for 405:



Searching

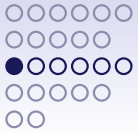
For example, if you were searching for 405:



405 is not in the skip list.

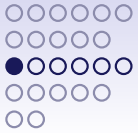
Searching

```
procedure SEARCH(data, node)
    if node is not valid then
        return FALSE
    else
        while data > node.next.data do
            node ← node.next
        end while
        if data = node.next.data then
            return TRUE
        else
            return SEARCH(data, node.down)
        end if
    end if
end procedure
```



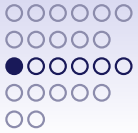
Adding

- Use a coin toss to see how many times the new item will be promoted (i.e. the number of levels where it resides).



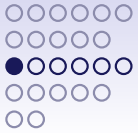
Adding

- Use a coin toss to see how many times the new item will be promoted (i.e. the number of levels where it resides).
- If there aren't enough levels in the skip list, then create the new levels.



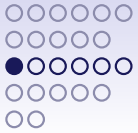
Adding

- Use a coin toss to see how many times the new item will be promoted (i.e. the number of levels where it resides).
- If there aren't enough levels in the skip list, then create the new levels.
- Traverse the skip list as if you're searching for the item. Insert the item on the level, as determined by the coin toss, while moving down to the next level.



Adding

- Use a coin toss to see how many times the new item will be promoted (i.e. the number of levels where it resides).
- If there aren't enough levels in the skip list, then create the new levels.
- Traverse the skip list as if you're searching for the item. Insert the item on the level, as determined by the coin toss, while moving down to the next level.
- Repeat the previous step until you reach the bottom level.



Adding

- Use a coin toss to see how many times the new item will be promoted (i.e. the number of levels where it resides).
- If there aren't enough levels in the skip list, then create the new levels.
- Traverse the skip list as if you're searching for the item. Insert the item on the level, as determined by the coin toss, while moving down to the next level.
- Repeat the previous step until you reach the bottom level.
- What if there are duplicates?

Adding Duplicates

- There are multiple approaches to handling duplicates when adding a new item.

Adding Duplicates

- There are multiple approaches to handling duplicates when adding a new item.
- One implementation is to search and add the item as you move down the skiplist. Placing the new item on any levels in the skip list as noted from by the coin toss, if the item is not already there.

Adding Duplicates

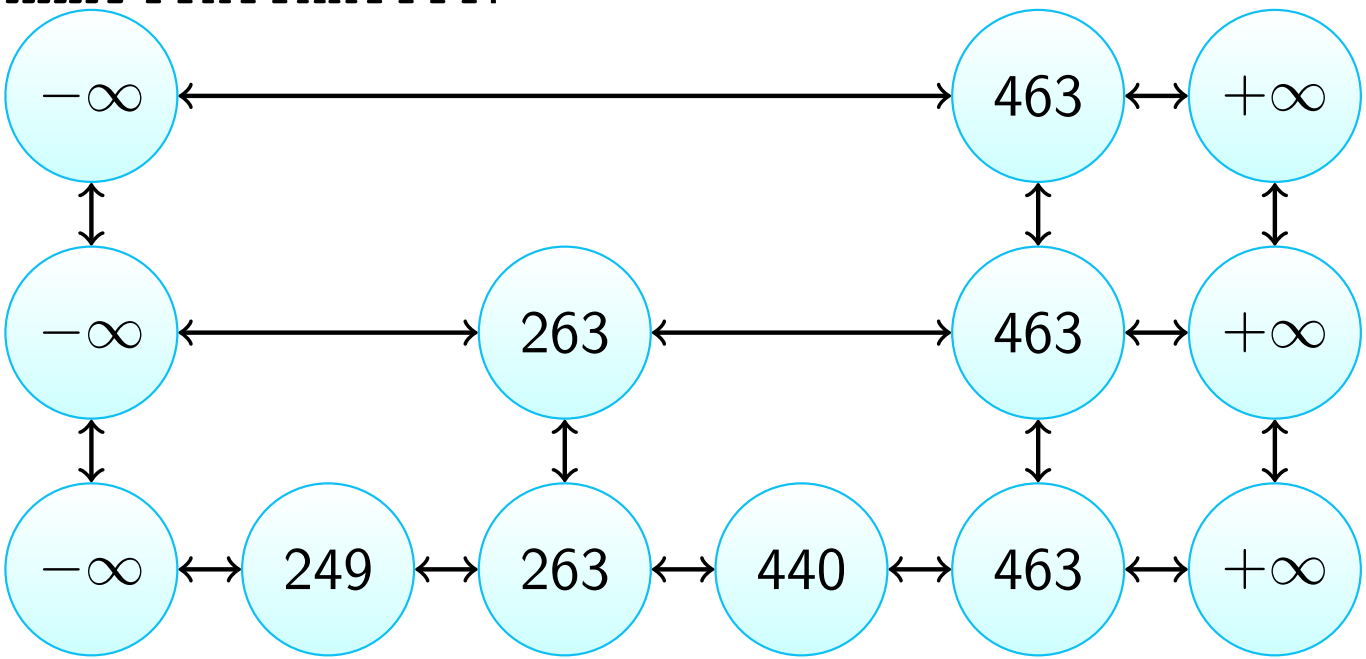
- There are multiple approaches to handling duplicates when adding a new item.
- One implementation is to search and add the item as you move down the skiplist. Placing the new item on any levels in the skip list as noted from by the coin toss, if the item is not already there.
- Another implementation is to search down the skiplist to see if a duplicate exists, regardless of the level. If there is a duplicate, then return without adding the item. Otherwise, add the new item to each level as denoted by the coin toss.

Adding Duplicates

- There are multiple approaches to handling duplicates when adding a new item.
- One implementation is to search and add the item as you move down the skiplist. Placing the new item on any levels in the skip list as noted from by the coin toss, if the item is not already there.
- Another implementation is to search down the skiplist to see if a duplicate exists, regardless of the level. If there is a duplicate, then return without adding the item. Otherwise, add the new item to each level as denoted by the coin toss.
- The former implementation is used in the diagrams and pseudocode. Both implementations are useful, and prepare to implement either one.

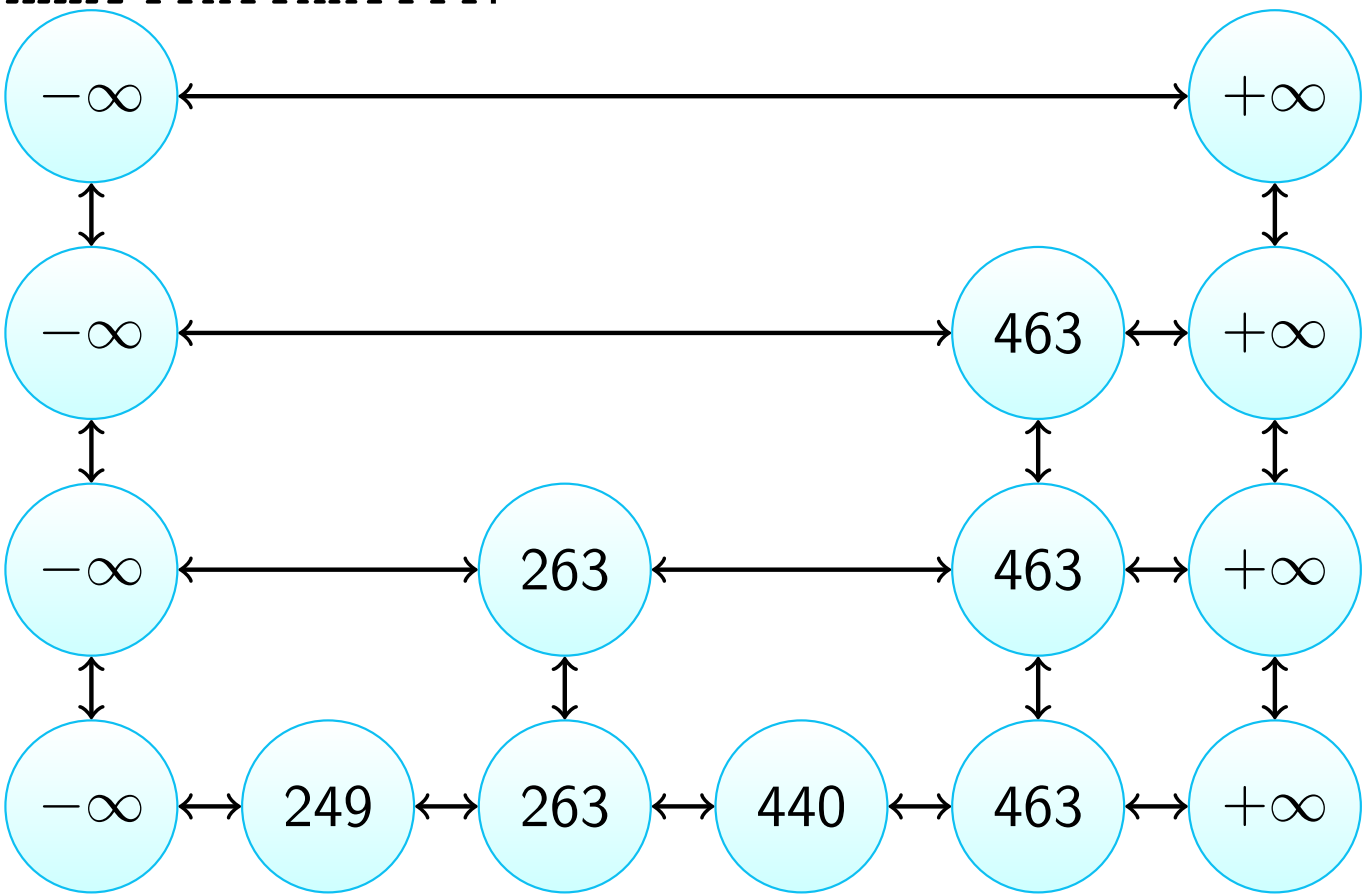
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



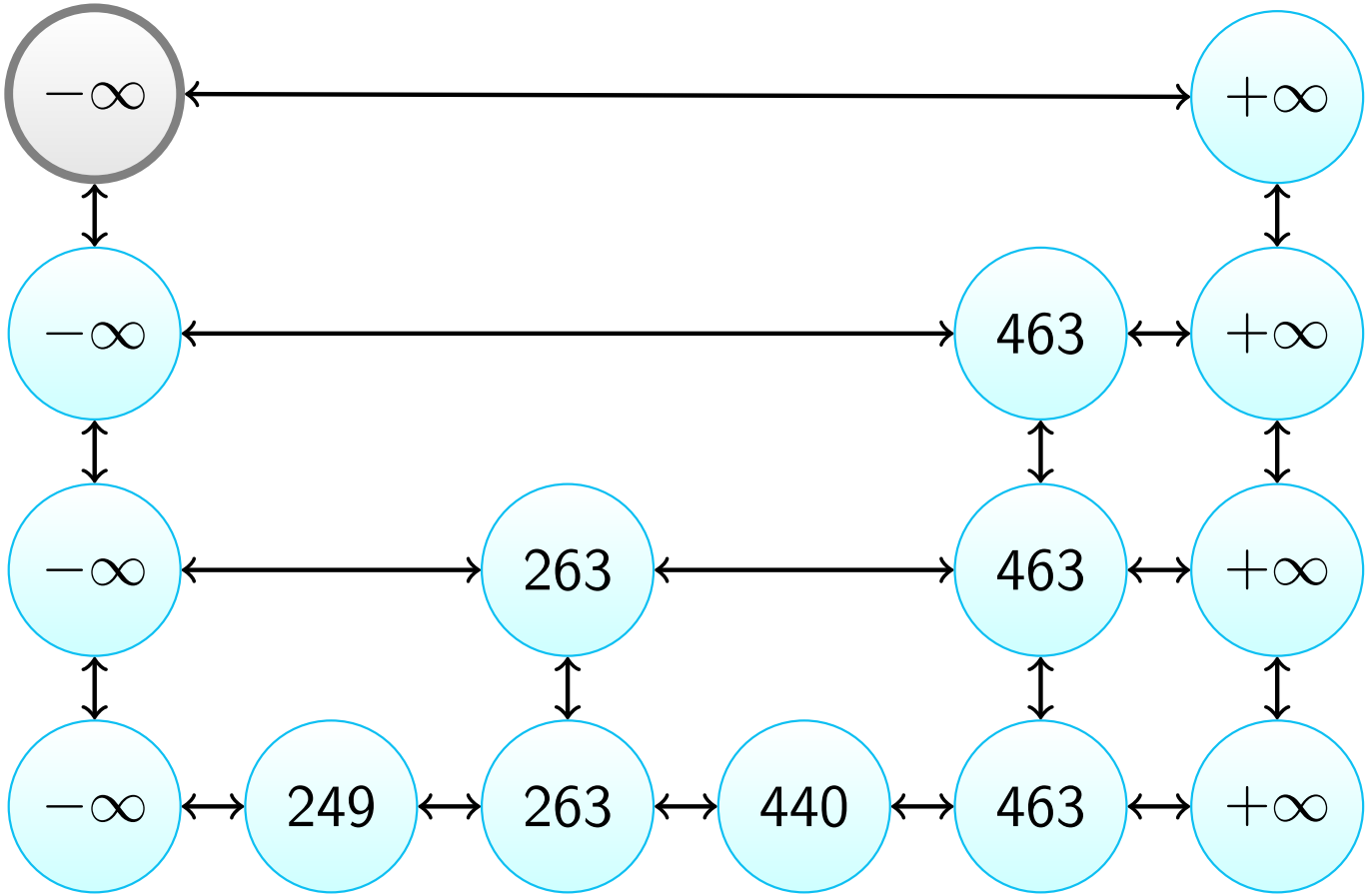
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



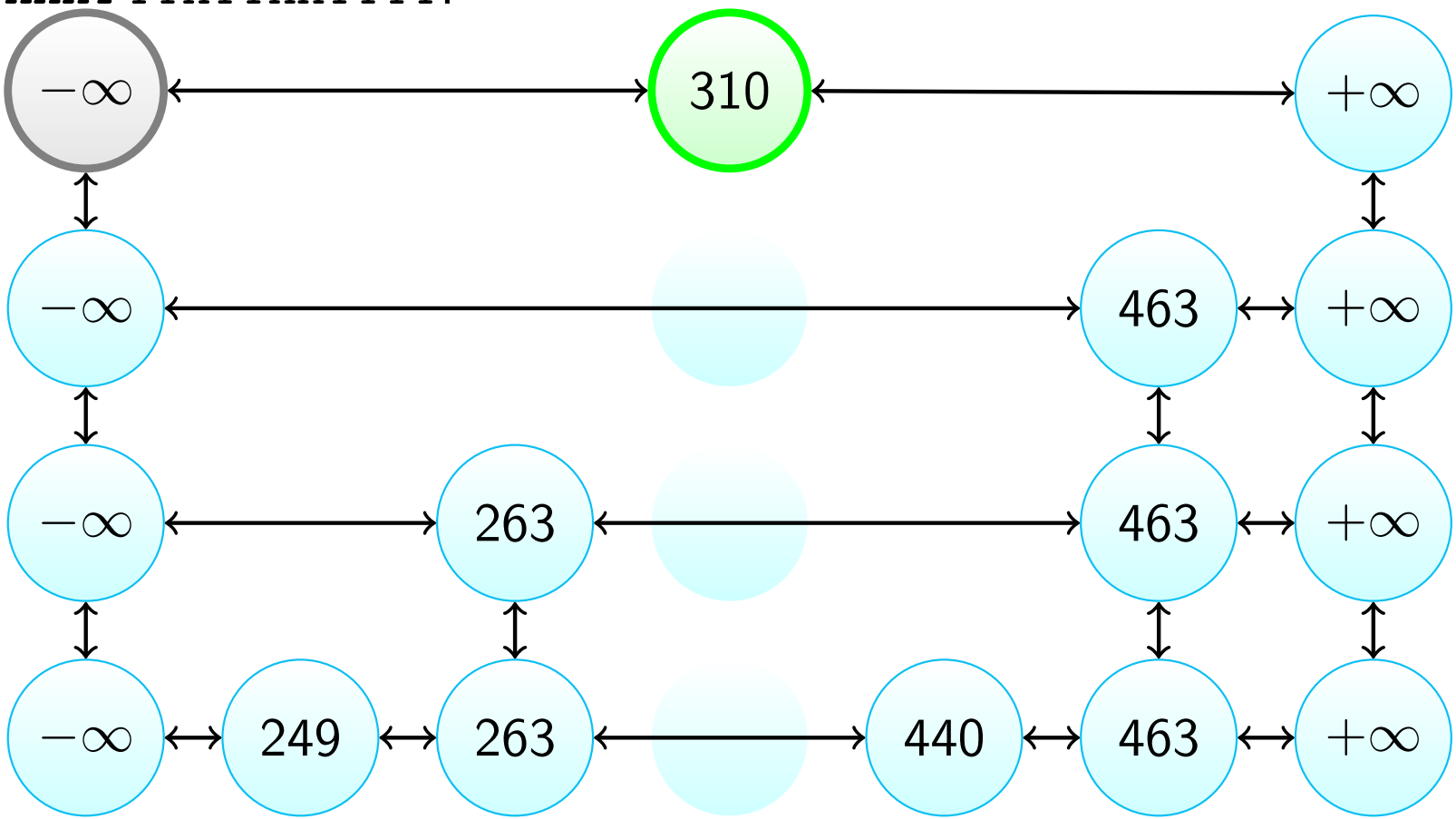
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



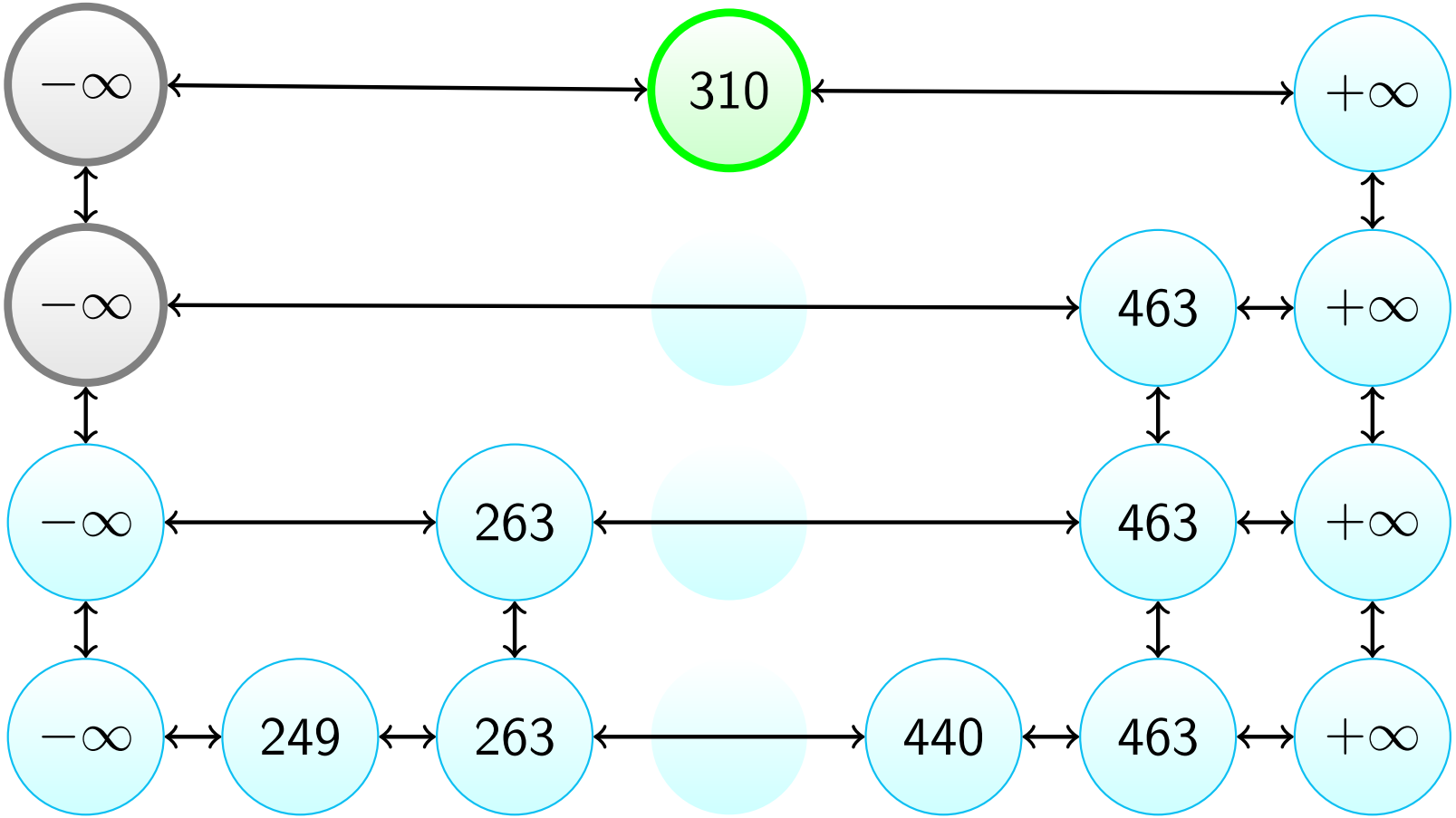
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



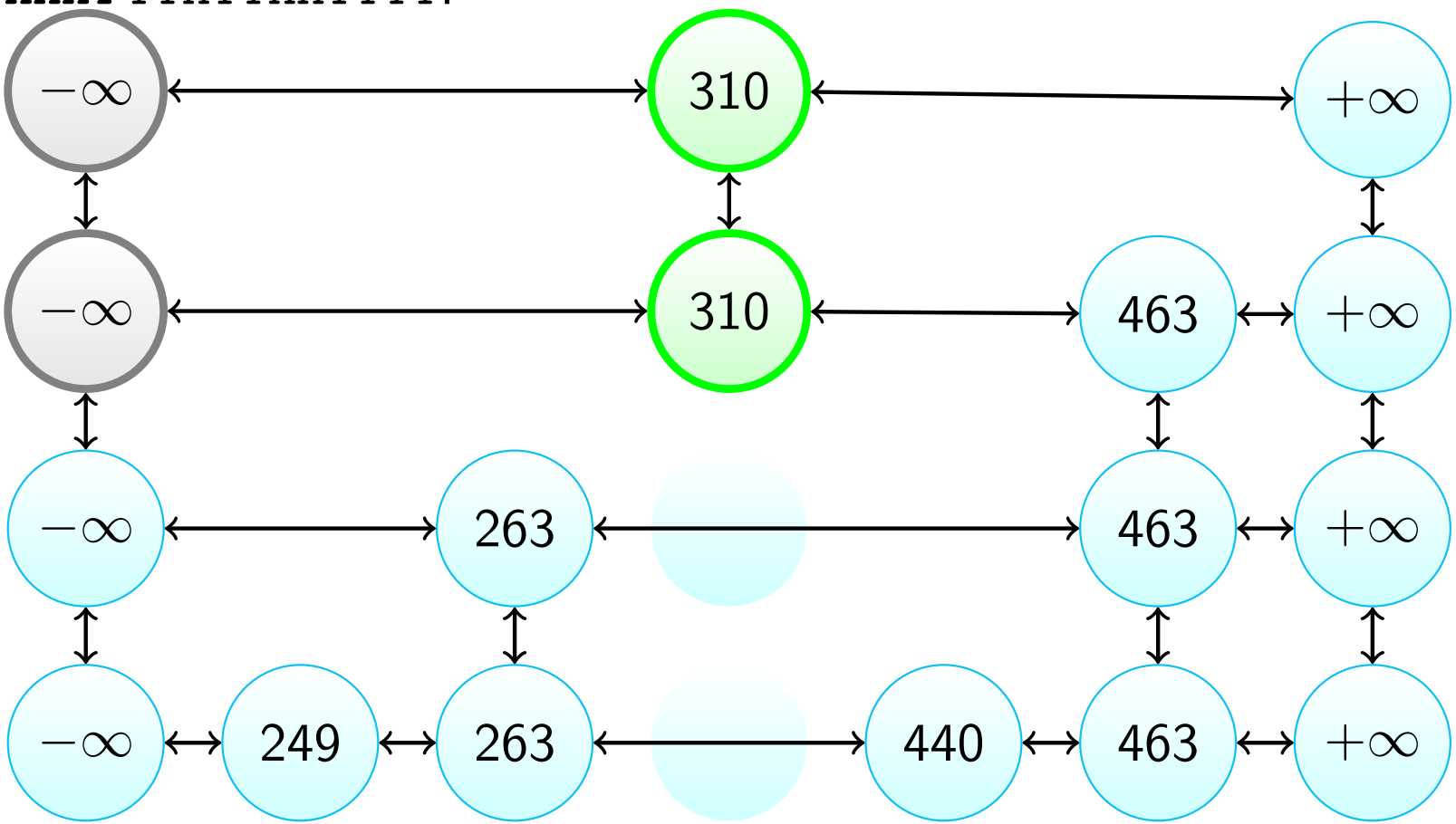
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



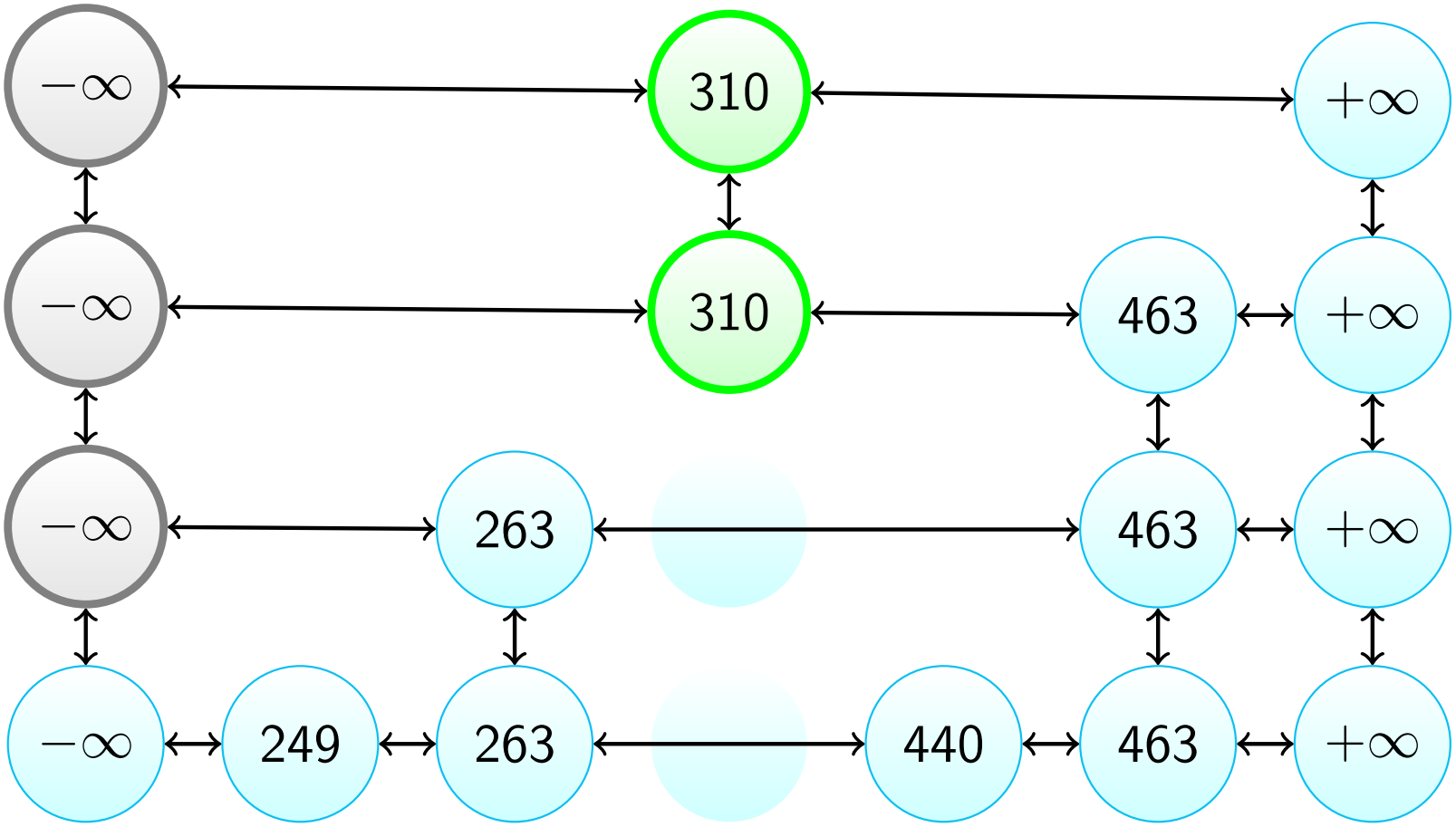
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



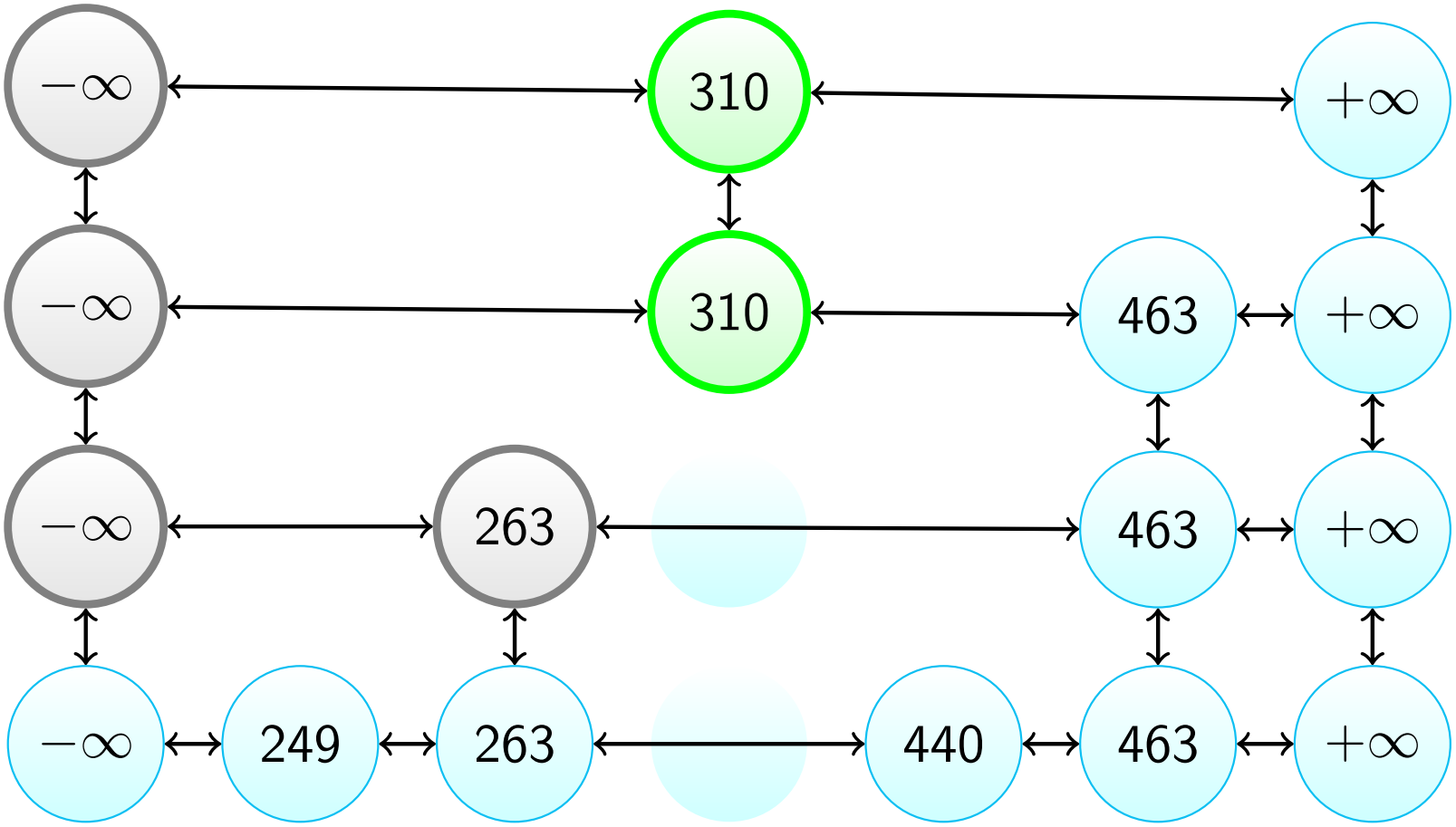
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



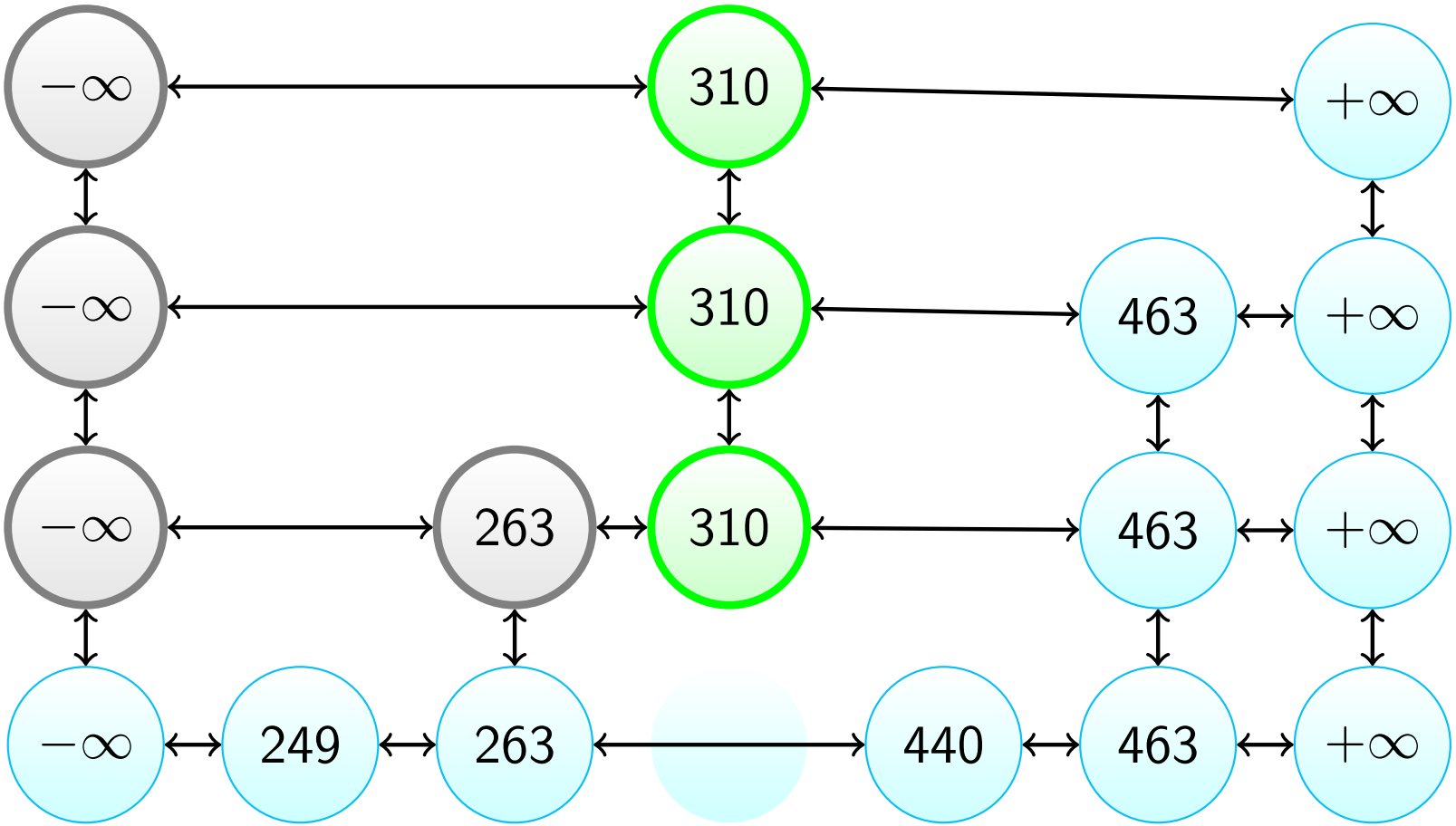
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



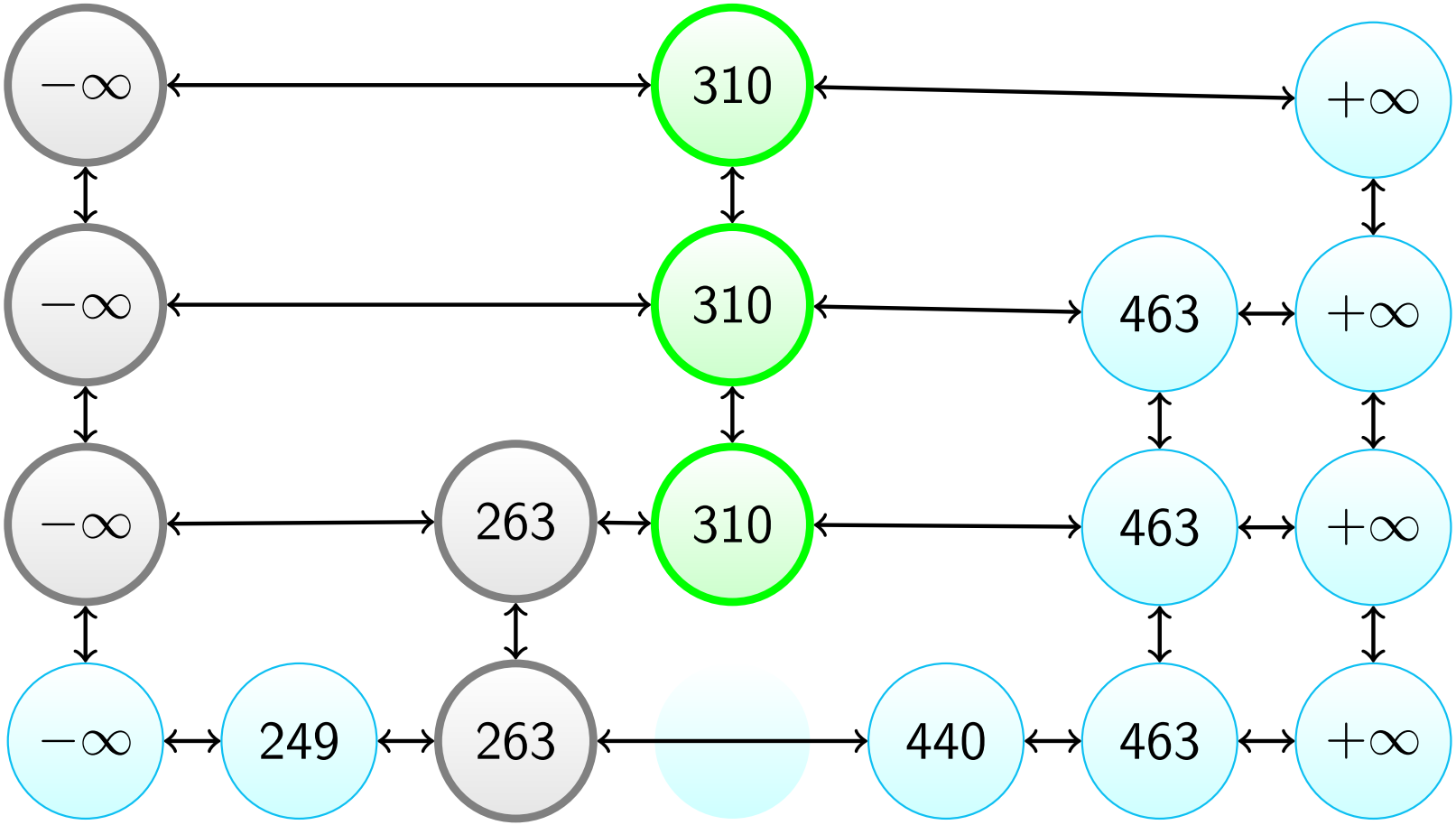
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



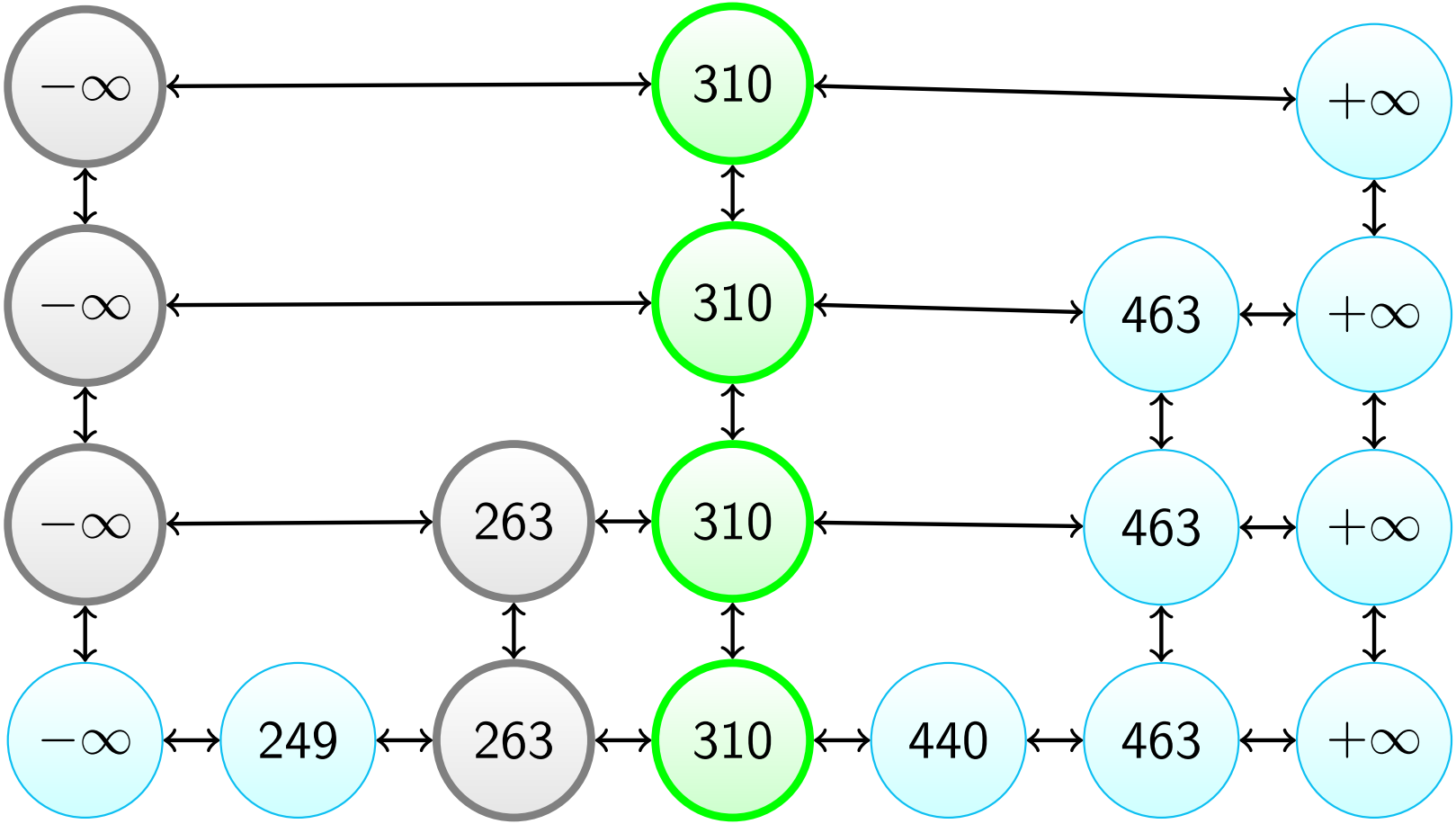
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



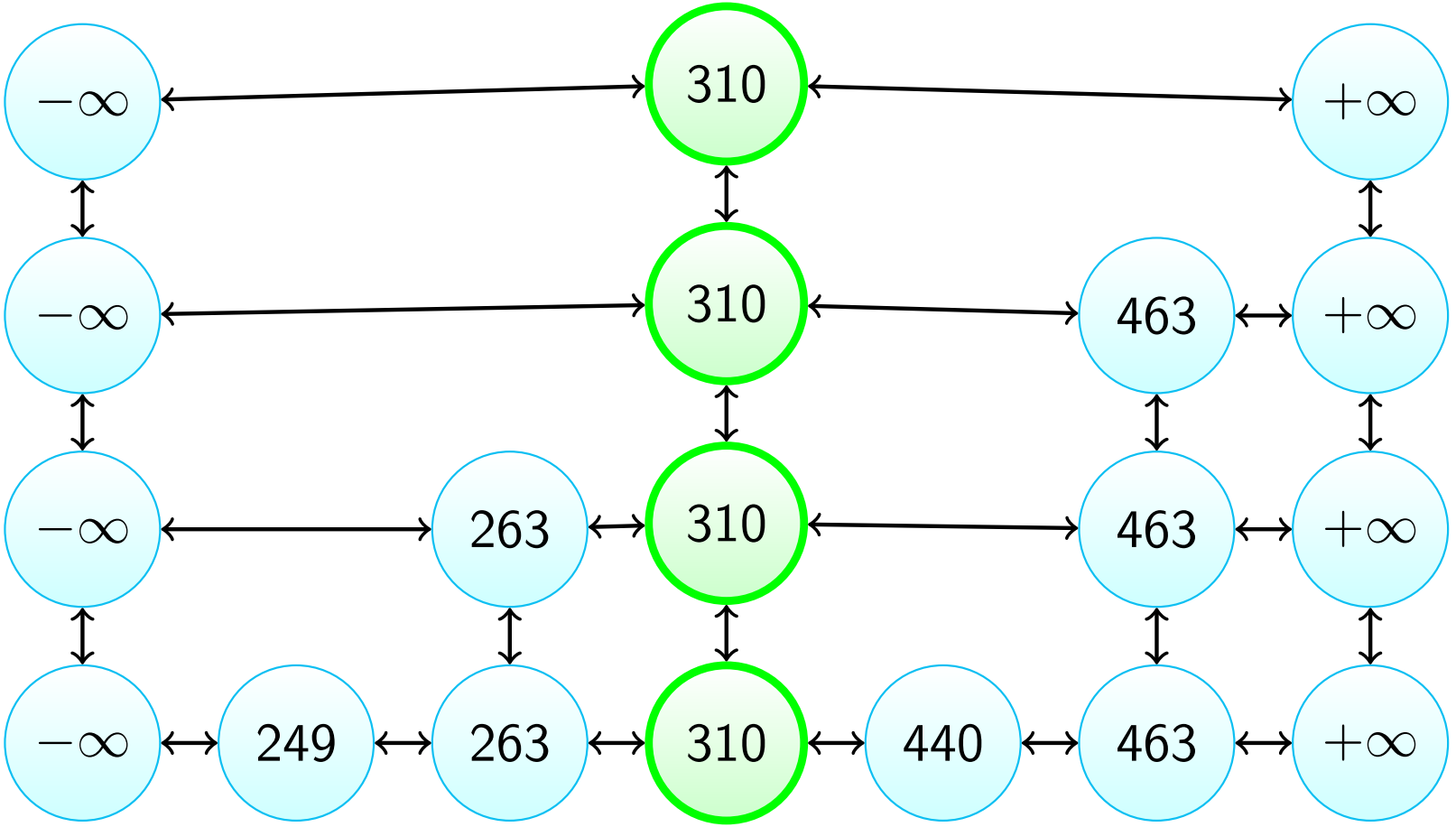
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



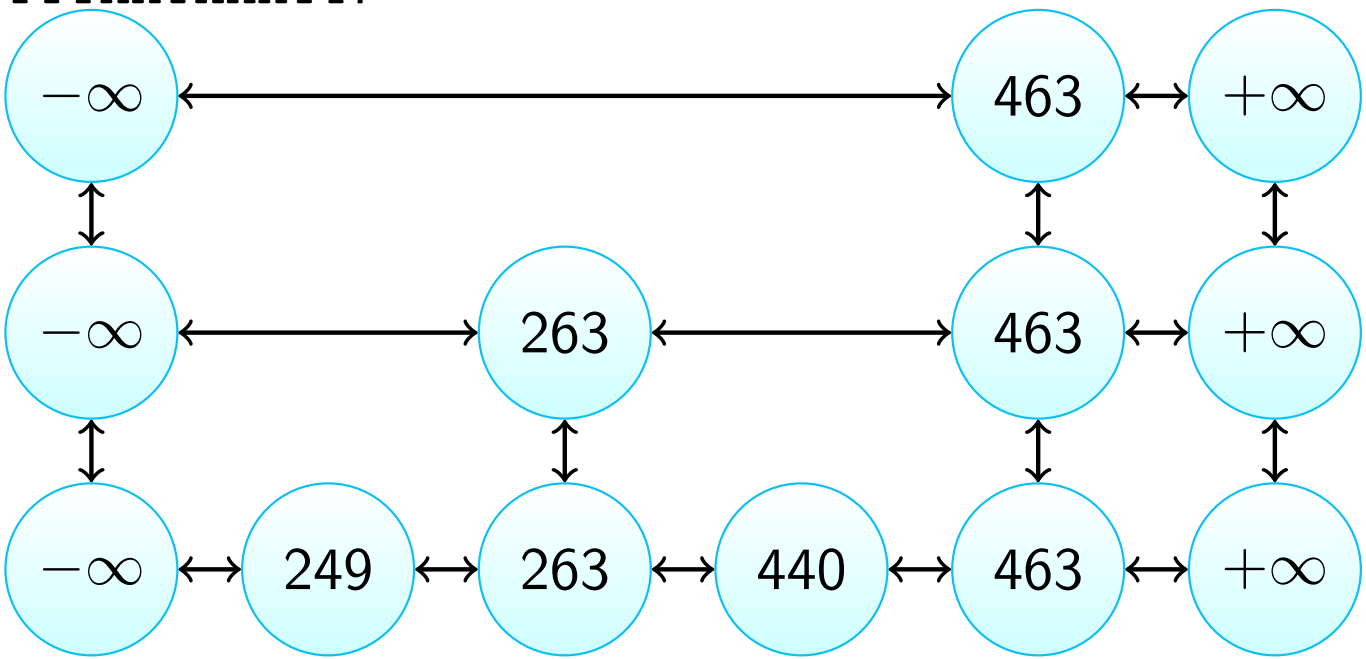
Adding

For example, if you were adding 310, and the coin toss gave ***HHHT***TTHTTHHTTTT:



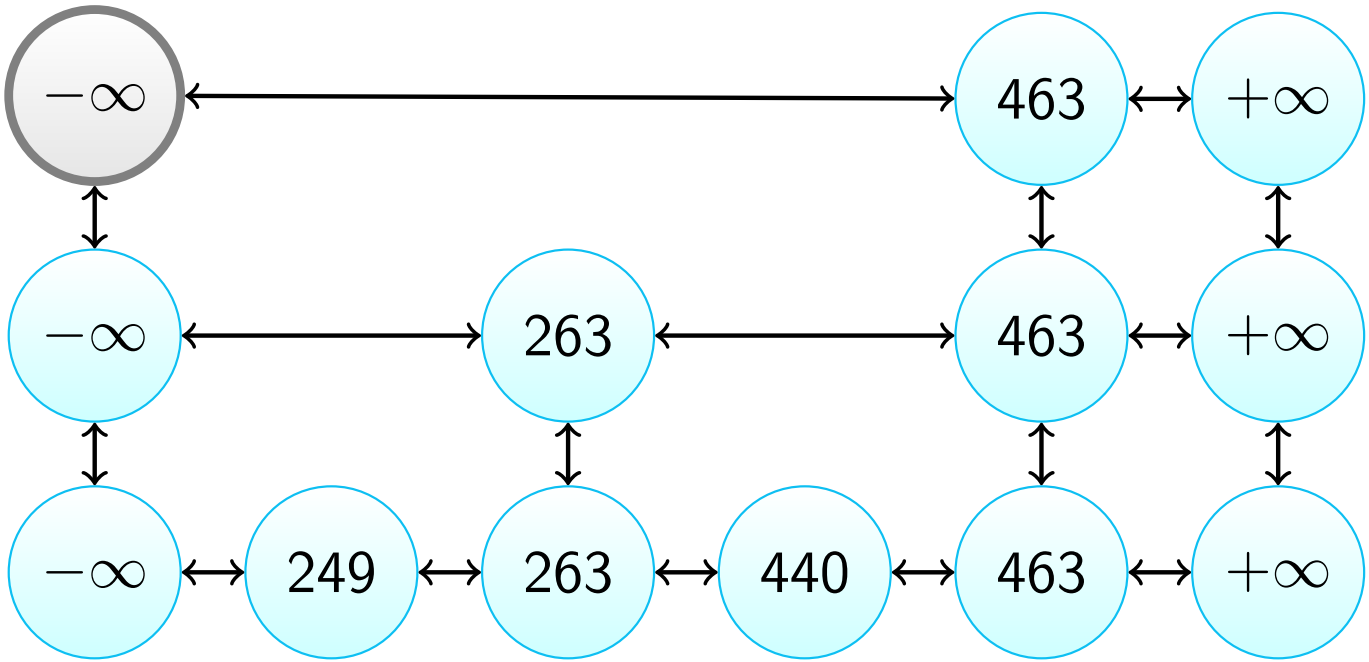
Adding

For example, if you were adding 118, and the coin toss gave TTTHHTHHHTT:



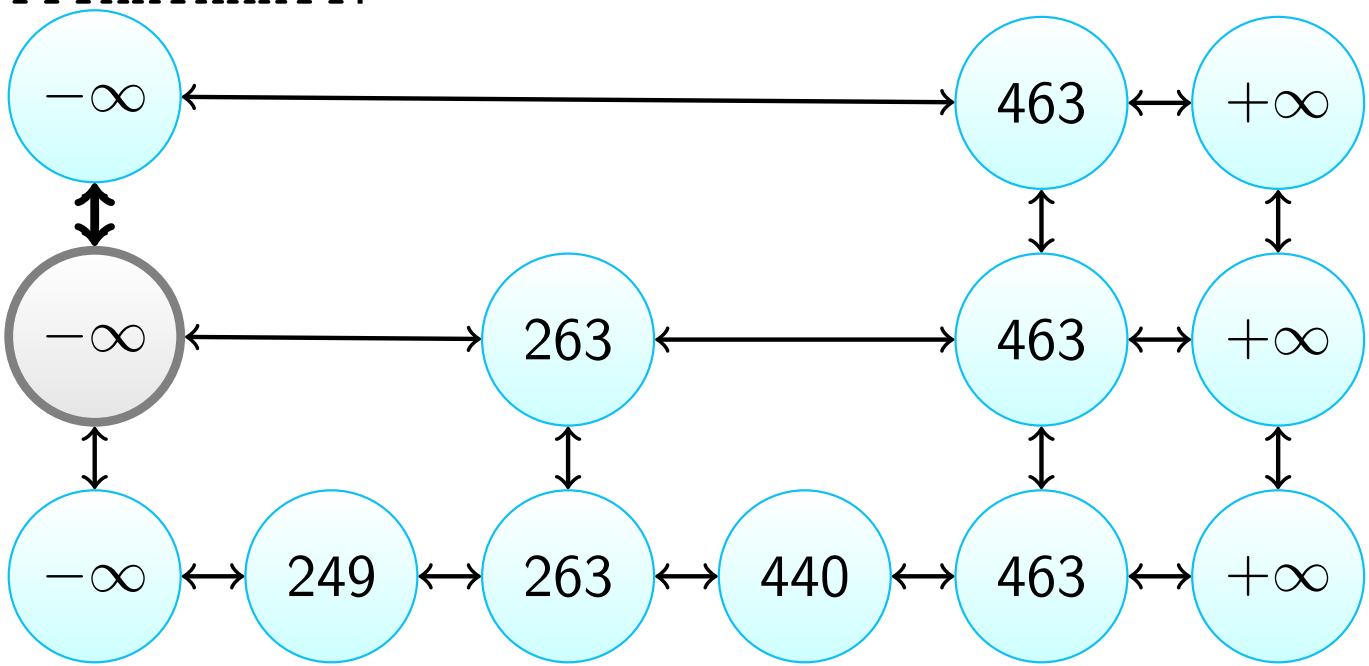
Adding

For example, if you were adding 118, and the coin toss gave TTTHHTHHHTT:



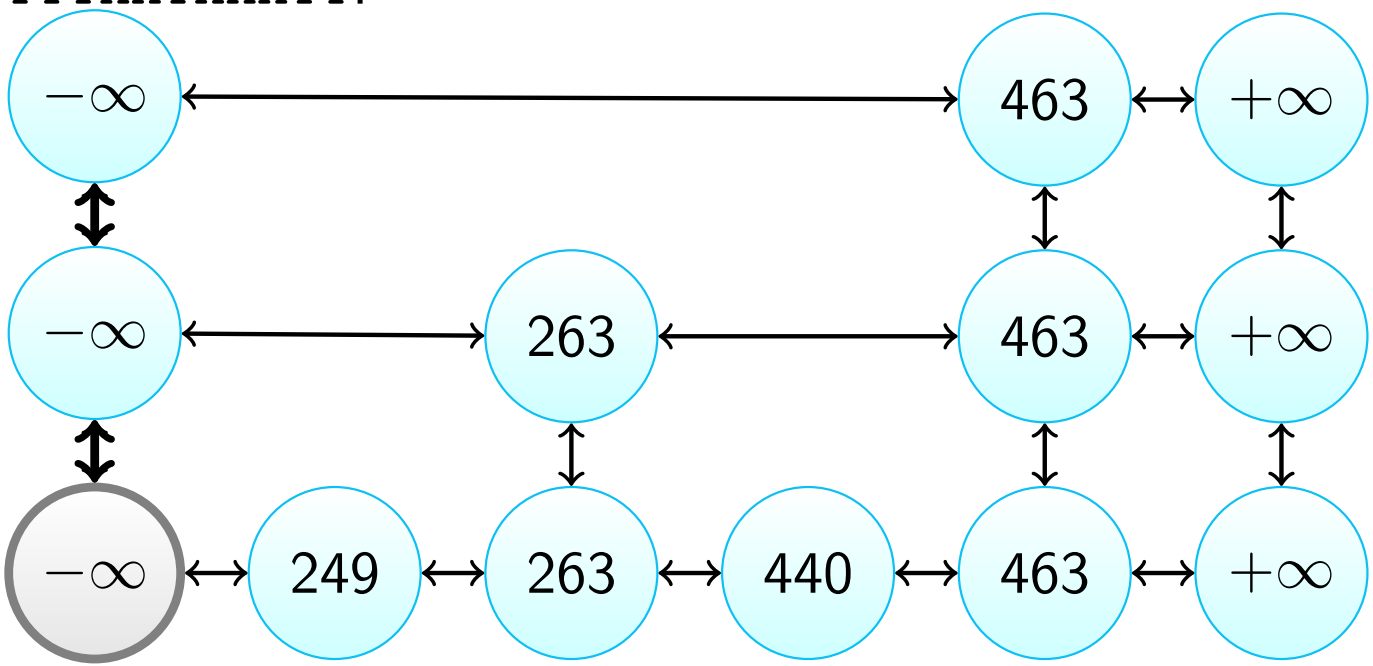
Adding

For example, if you were adding 118, and the coin toss gave TTTHHTHHHTT:



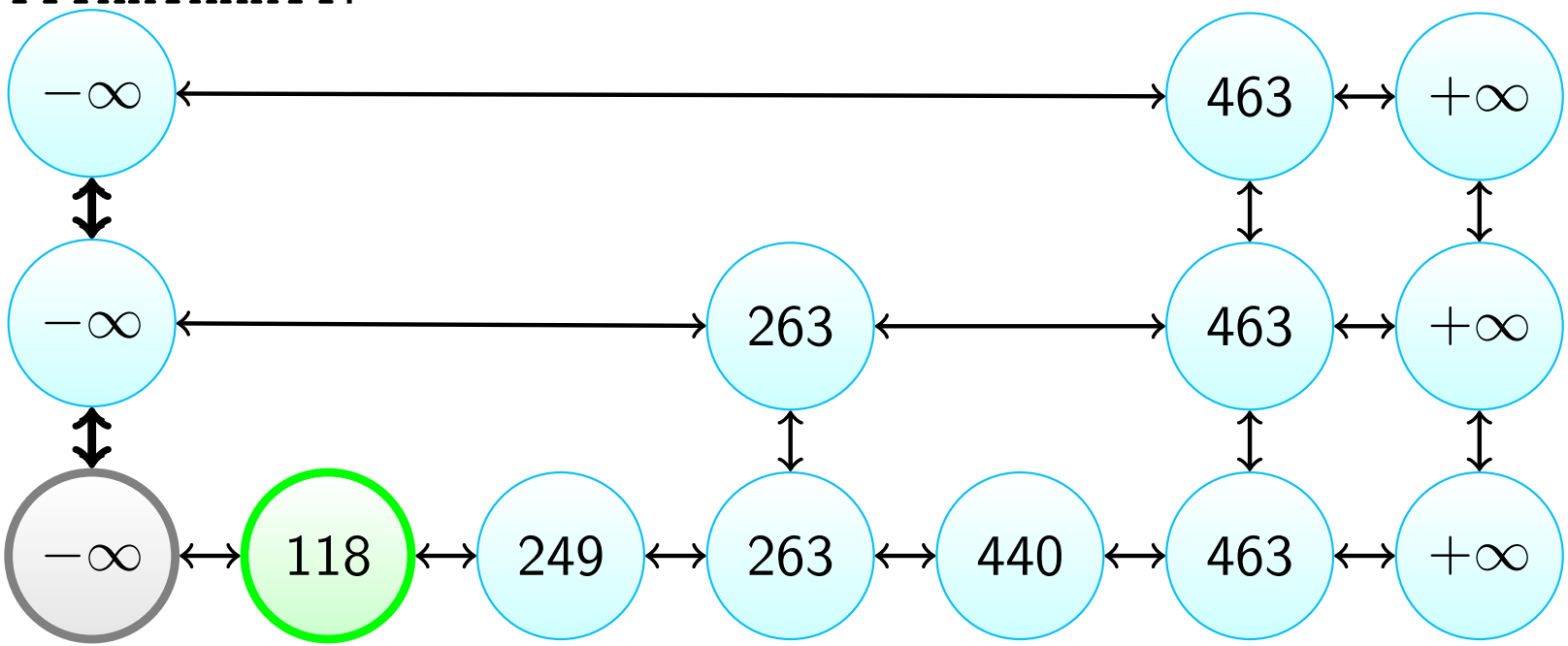
Adding

For example, if you were adding 118, and the coin toss gave TTTHHTHHHTT:



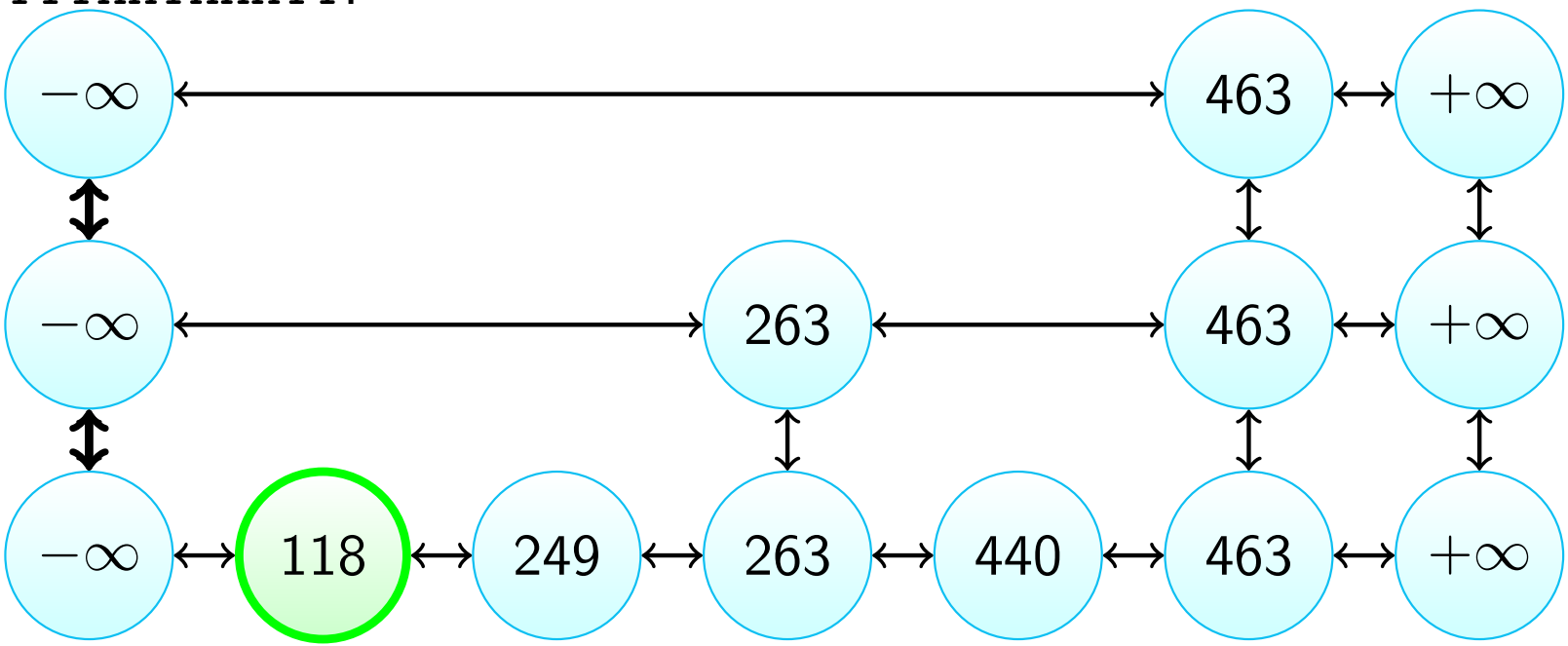
Adding

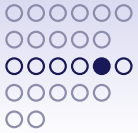
For example, if you were adding 118, and the coin toss gave TTTHHTHHHTT:



Adding

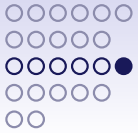
For example, if you were adding 118, and the coin toss gave TTTHHTHHHTT:





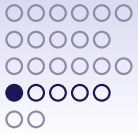
Adding

procedure ADD(*data*)
    datalevels ← highest level the data resides on (based on coin toss)
    while current level < *datalevels* **do**
        create another level
    end while
    ADD(*data*, *datalevels*, top-left phantom node, `NULL`)
end procedure



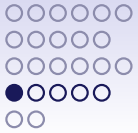
Adding

```
procedure ADD(data, levels, node, upperNode)
    if node is not valid then
        return
    else
        while data > node.next.data do
            node ← node.next
        end while
        if node.level <= levels then
            Add new node containing data after node
            node.up ← upperNode
            upperNode ← newly-added node
        end if
        ADD(data, levels, node.down, upperNode)
    end if
end procedure
```



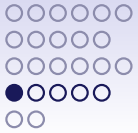
Removing

- Follow the same steps as searching until you find the item in the skip list or go off of the skip list (in which case the data is not there).



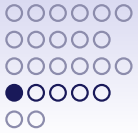
Removing

- Follow the same steps as searching until you find the item in the skip list or go off of the skip list (in which case the data is not there).
- Disconnect the node from the rest of the nodes on that level, and move down to the level below.



Removing

- Follow the same steps as searching until you find the item in the skip list or go off of the skip list (in which case the data is not there).
- Disconnect the node from the rest of the nodes on that level, and move down to the level below.
- Repeat the previous step until you go off of the skip list.

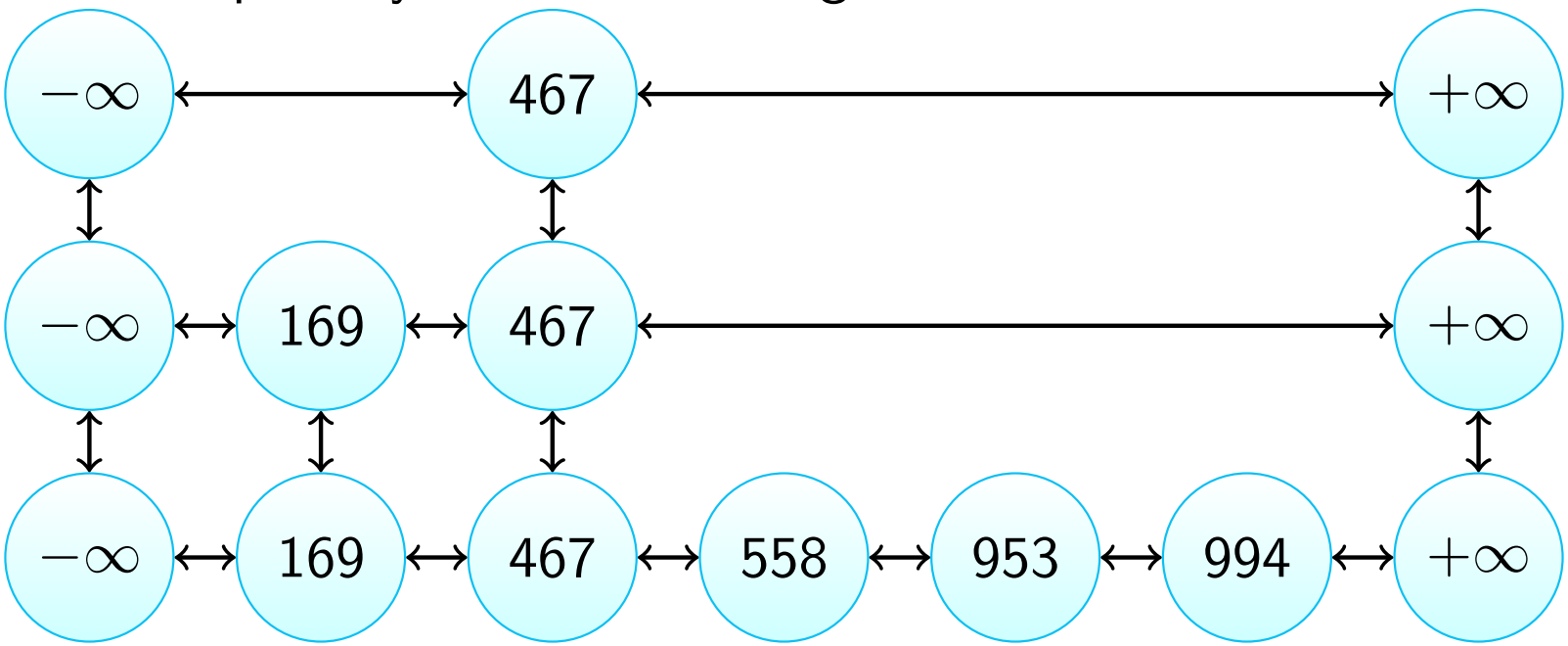


Removing

- Follow the same steps as searching until you find the item in the skip list or go off of the skip list (in which case the data is not there).
- Disconnect the node from the rest of the nodes on that level, and move down to the level below.
- Repeat the previous step until you go off of the skip list.
- Remove any empty levels in the skip list.

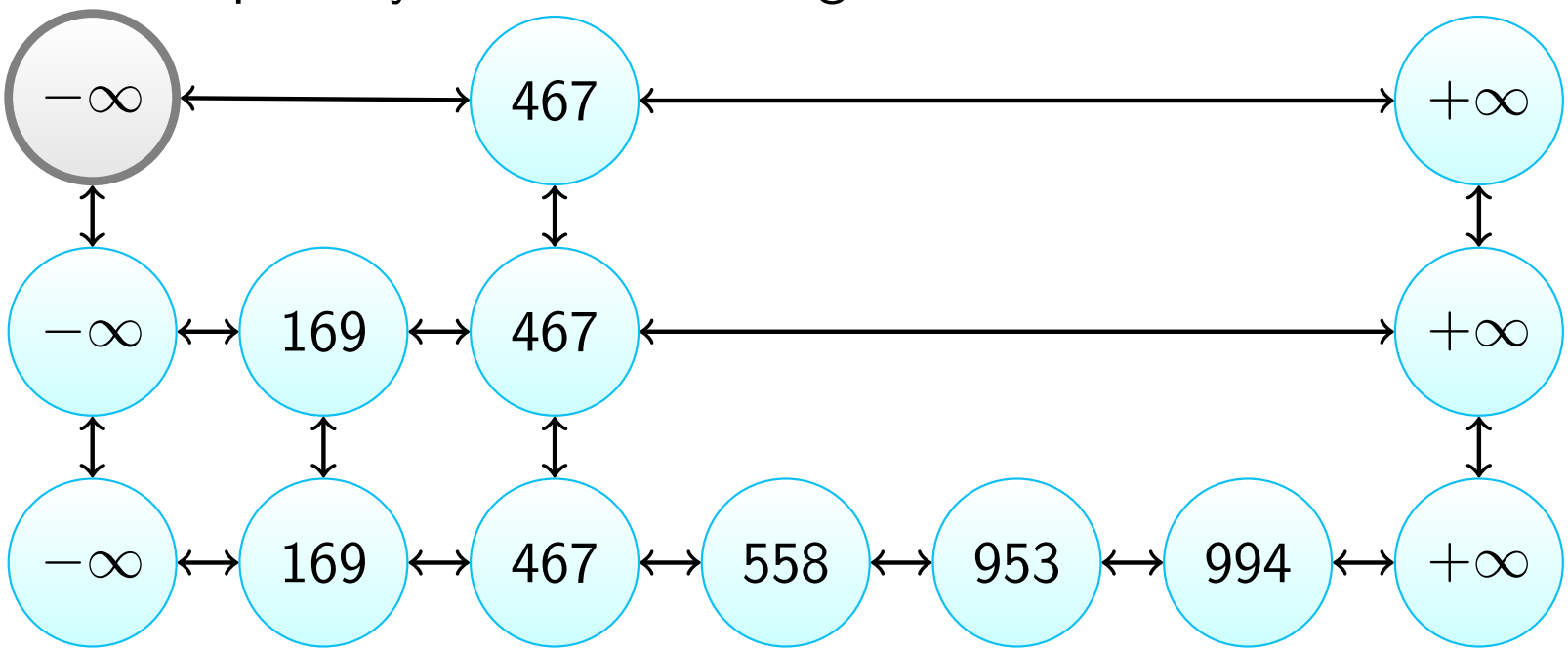
Removing

For example, if you were removing 467:



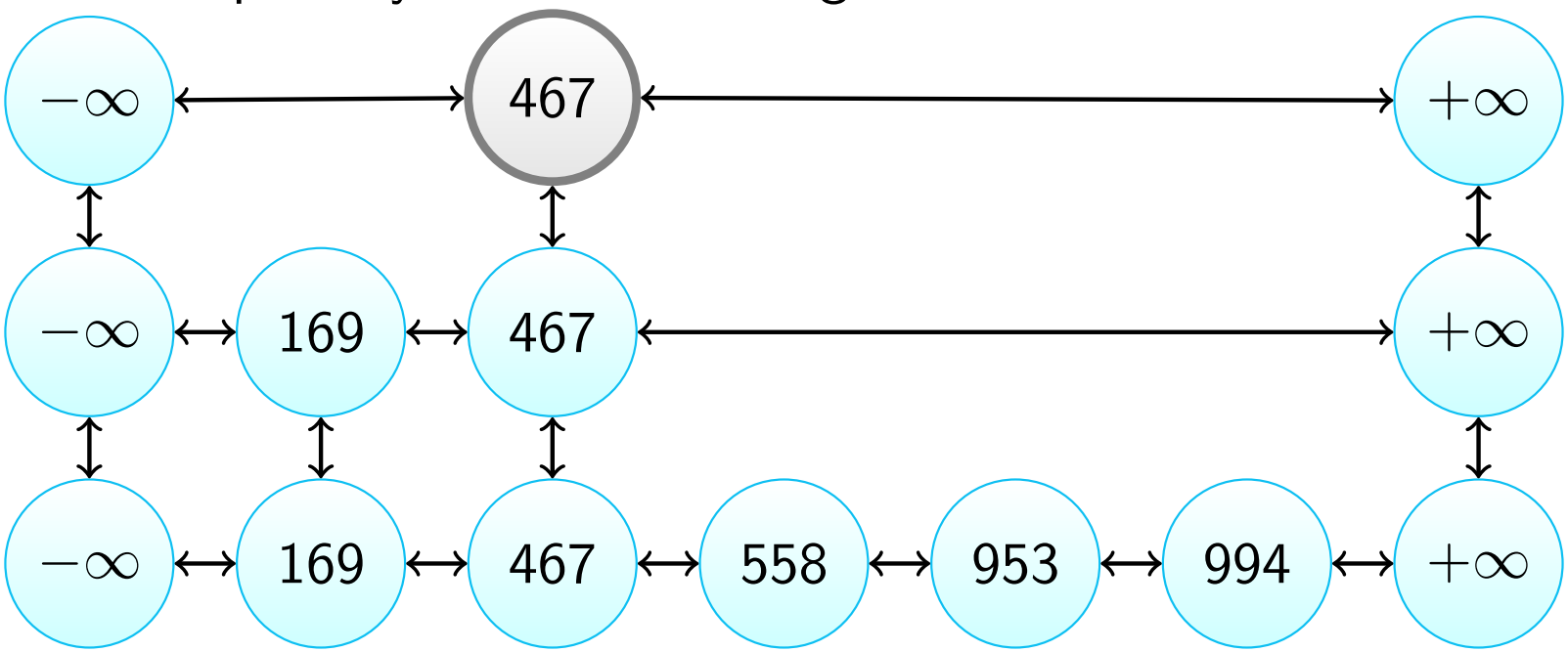
Removing

For example, if you were removing 467:



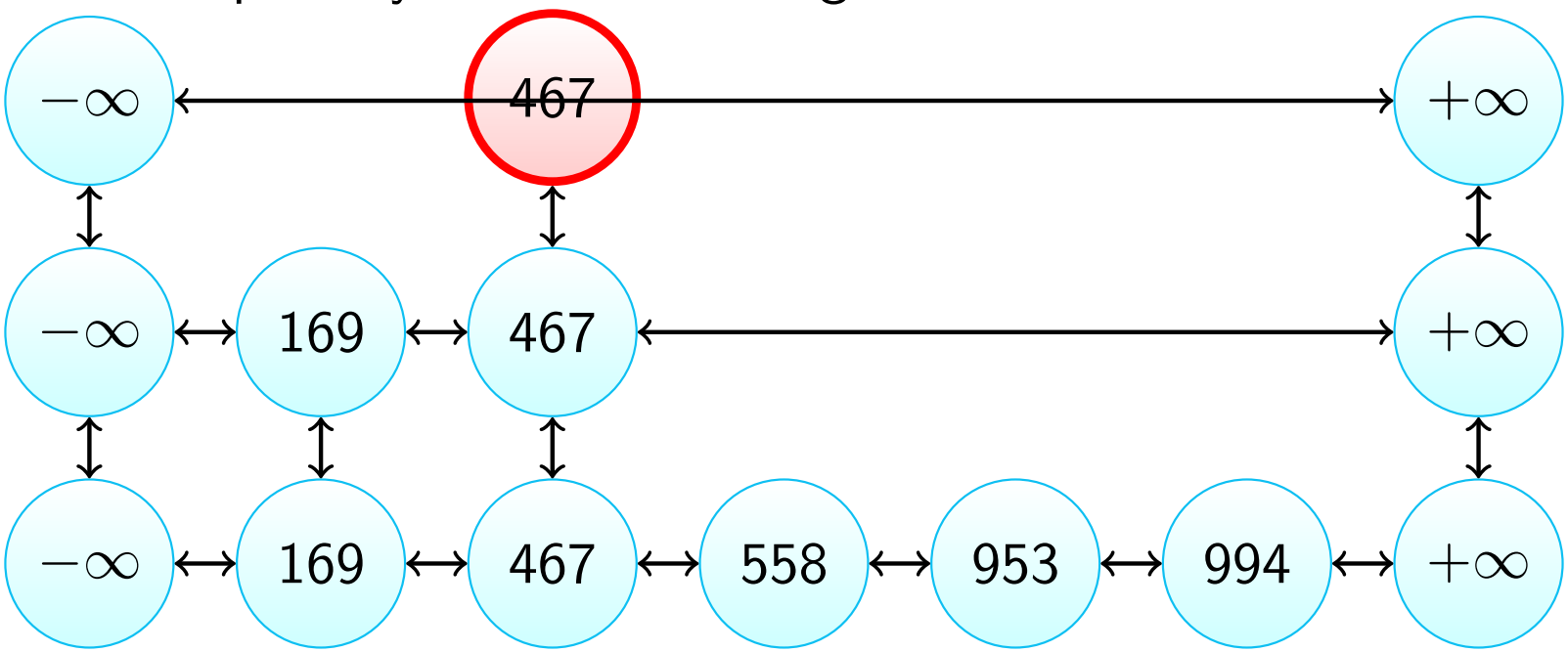
Removing

For example, if you were removing 467:



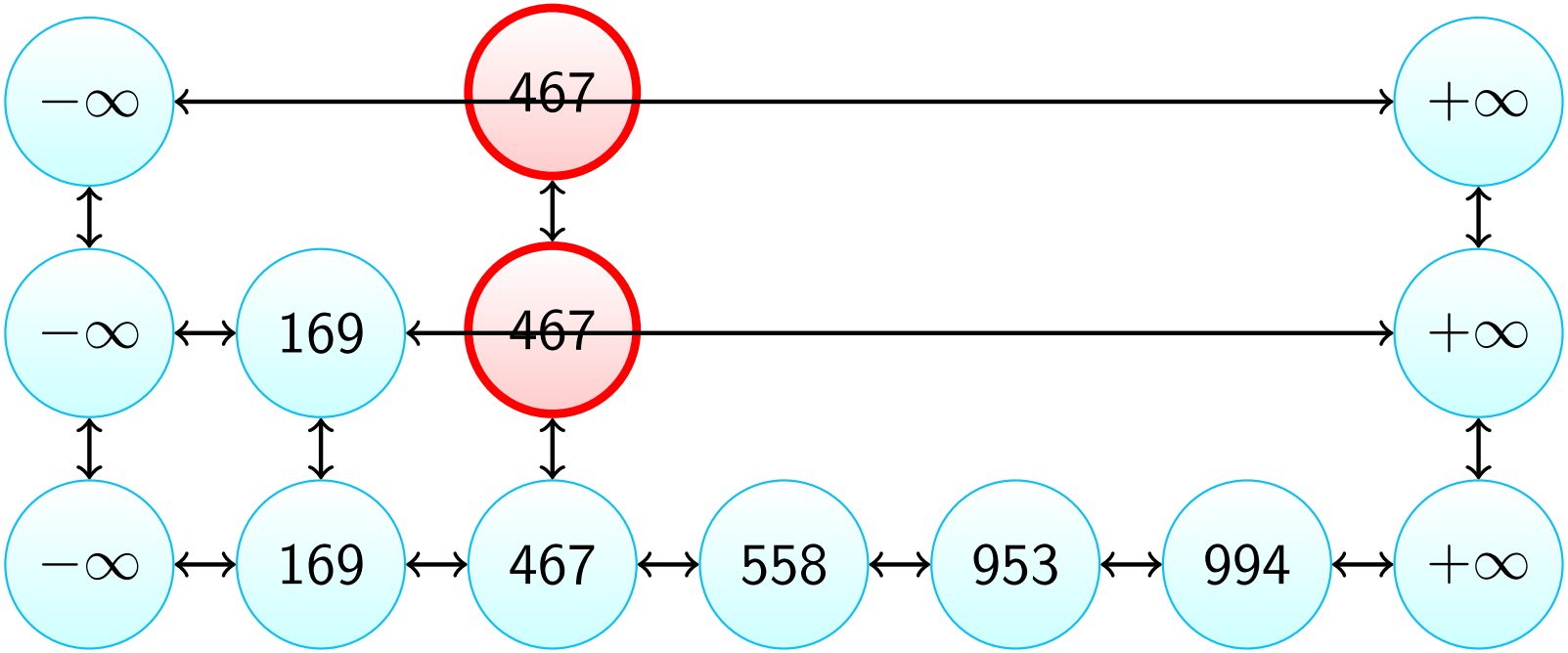
Removing

For example, if you were removing 467:



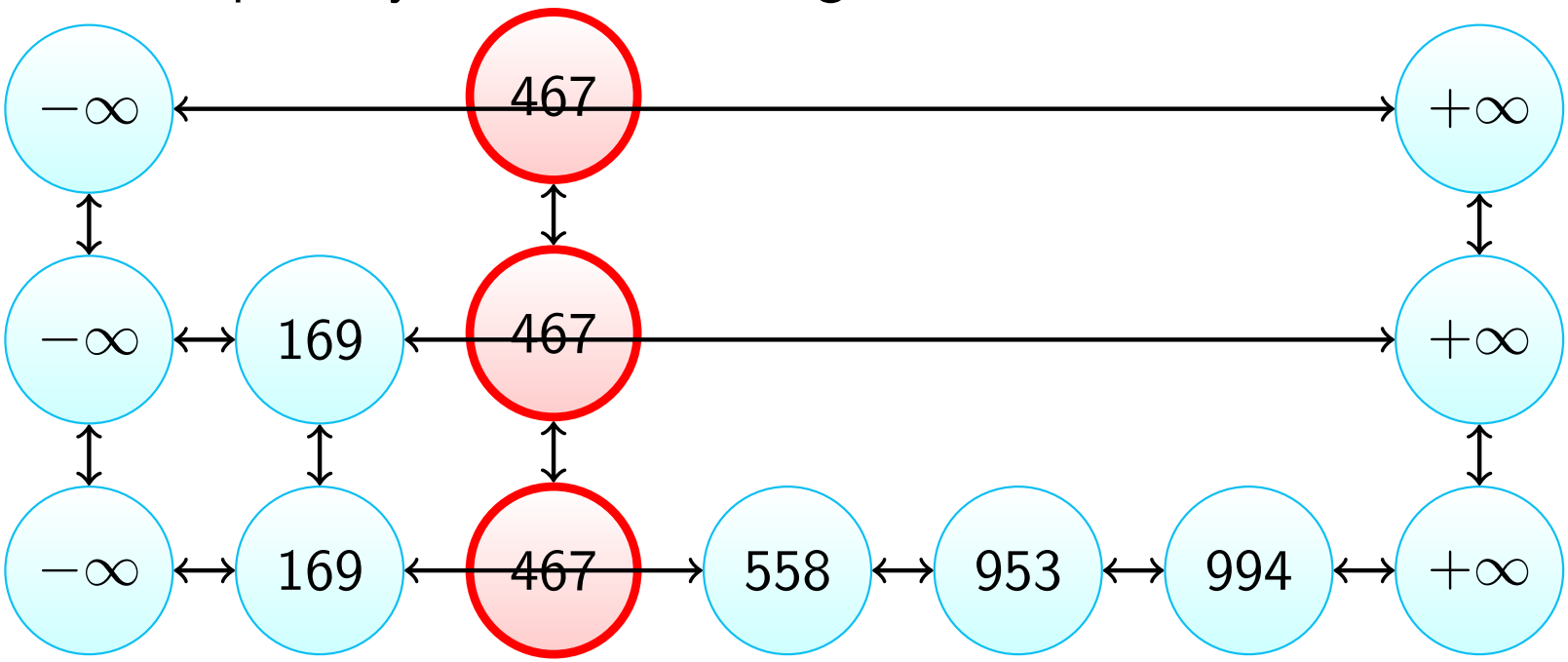
Removing

For example, if you were removing 467:



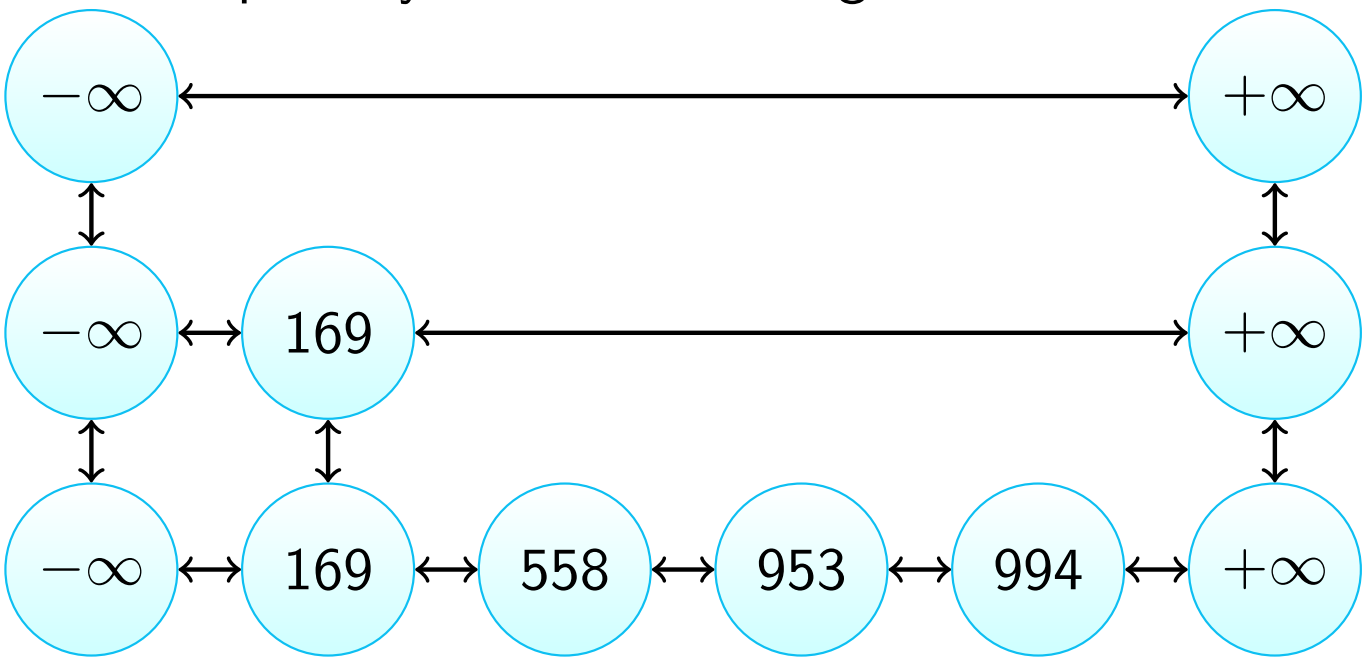
Removing

For example, if you were removing 467:



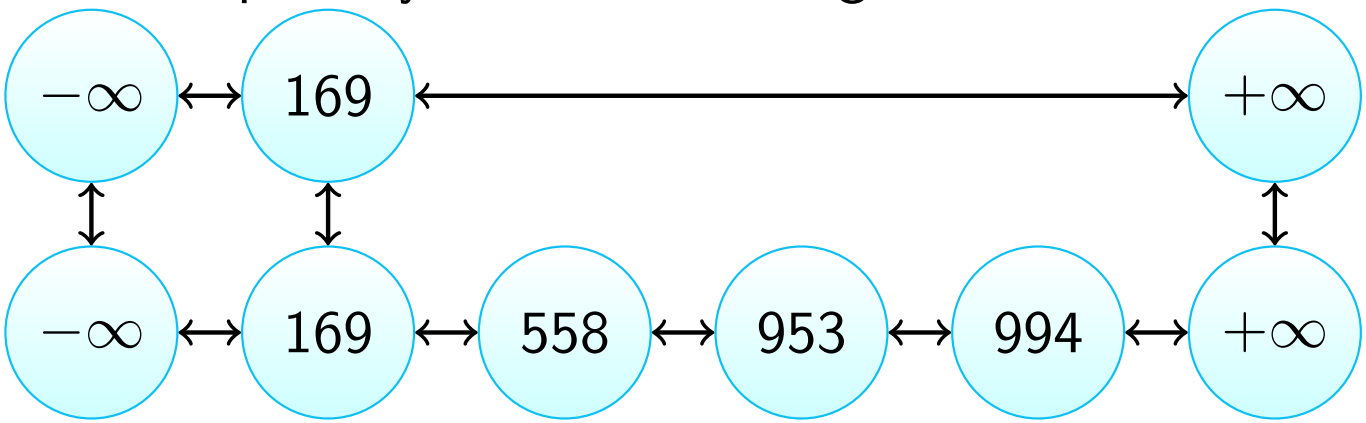
Removing

For example, if you were removing 467:



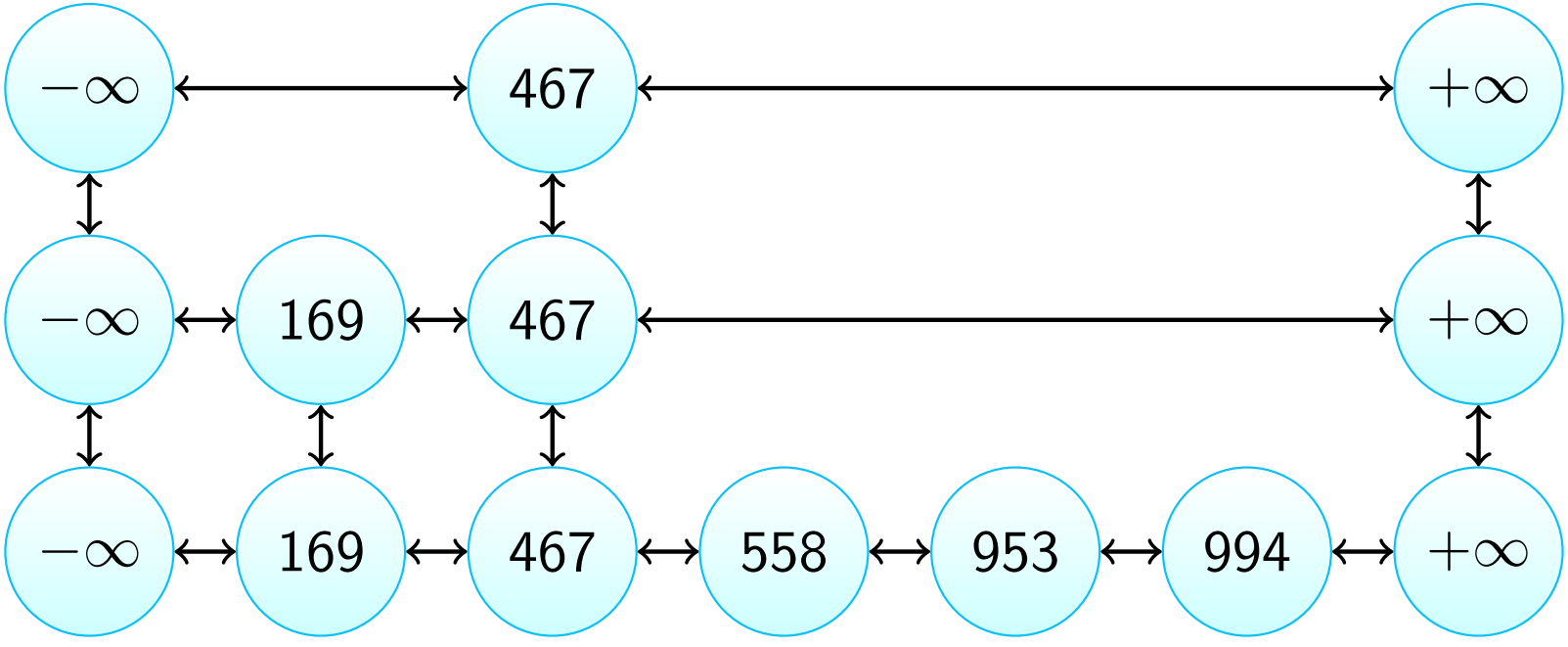
Removing

For example, if you were removing 467:



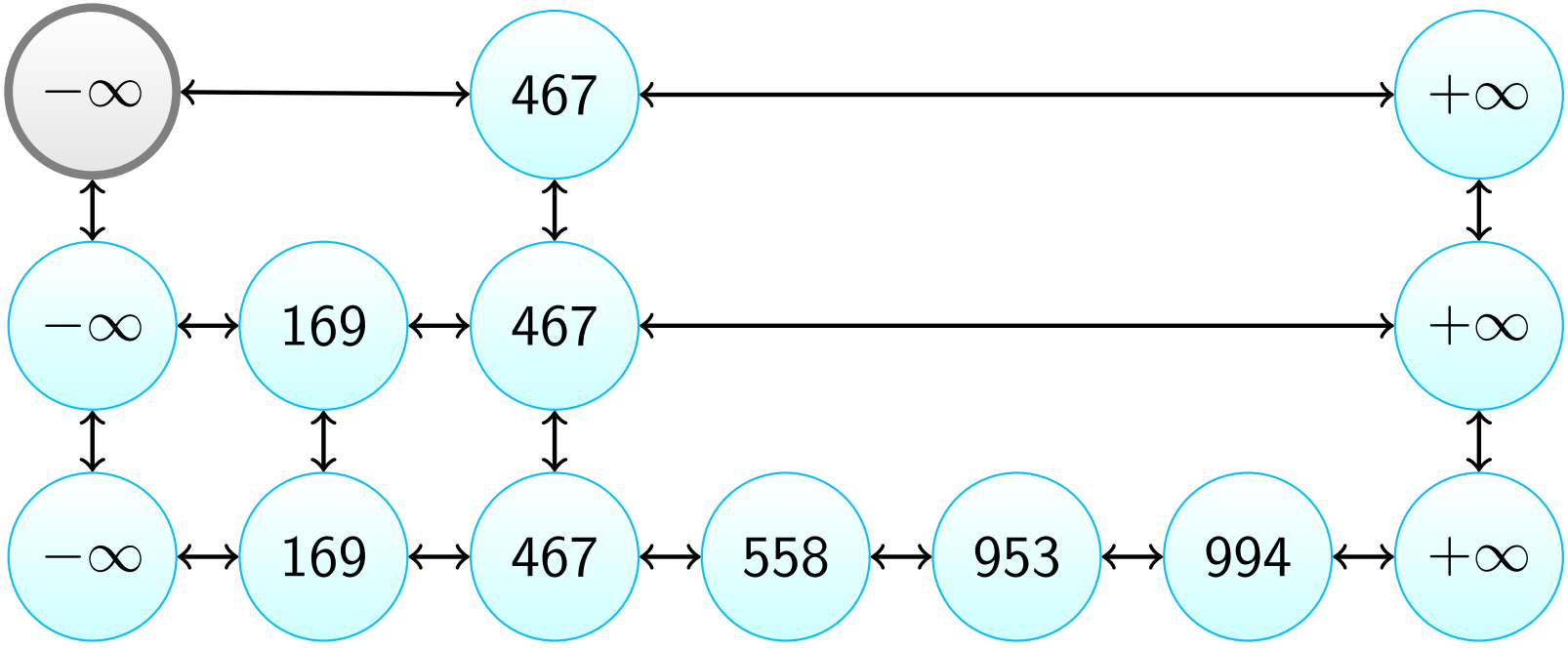
Removing

For example, if you were removing 169:



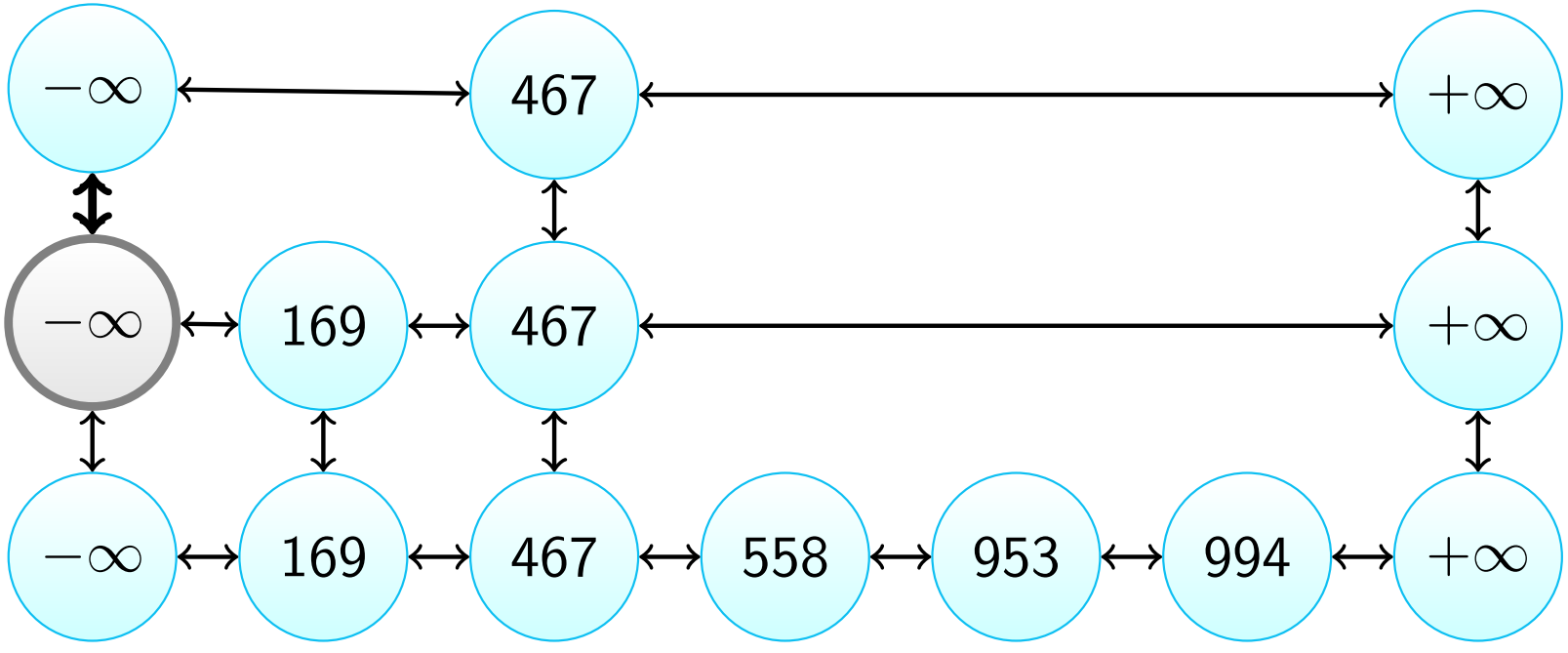
Removing

For example, if you were removing 169:



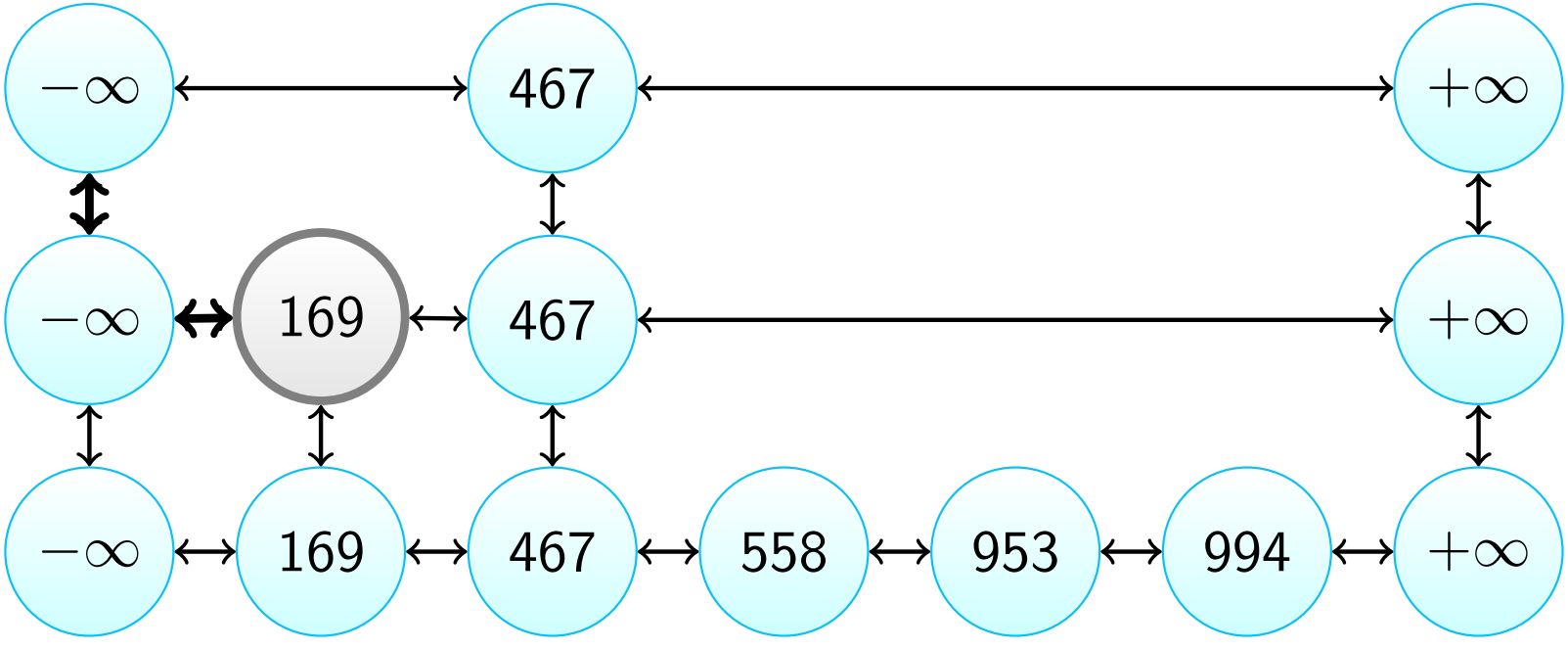
Removing

For example, if you were removing 169:



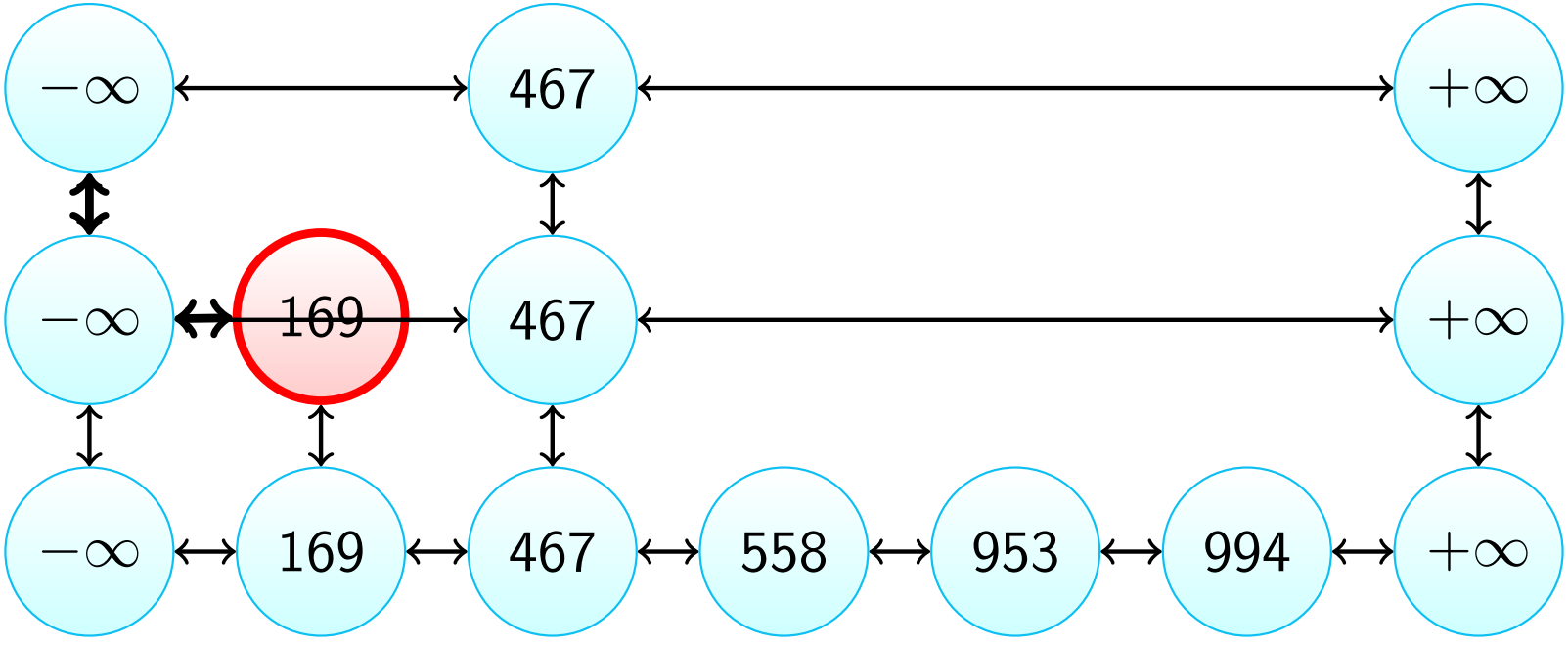
Removing

For example, if you were removing 169:



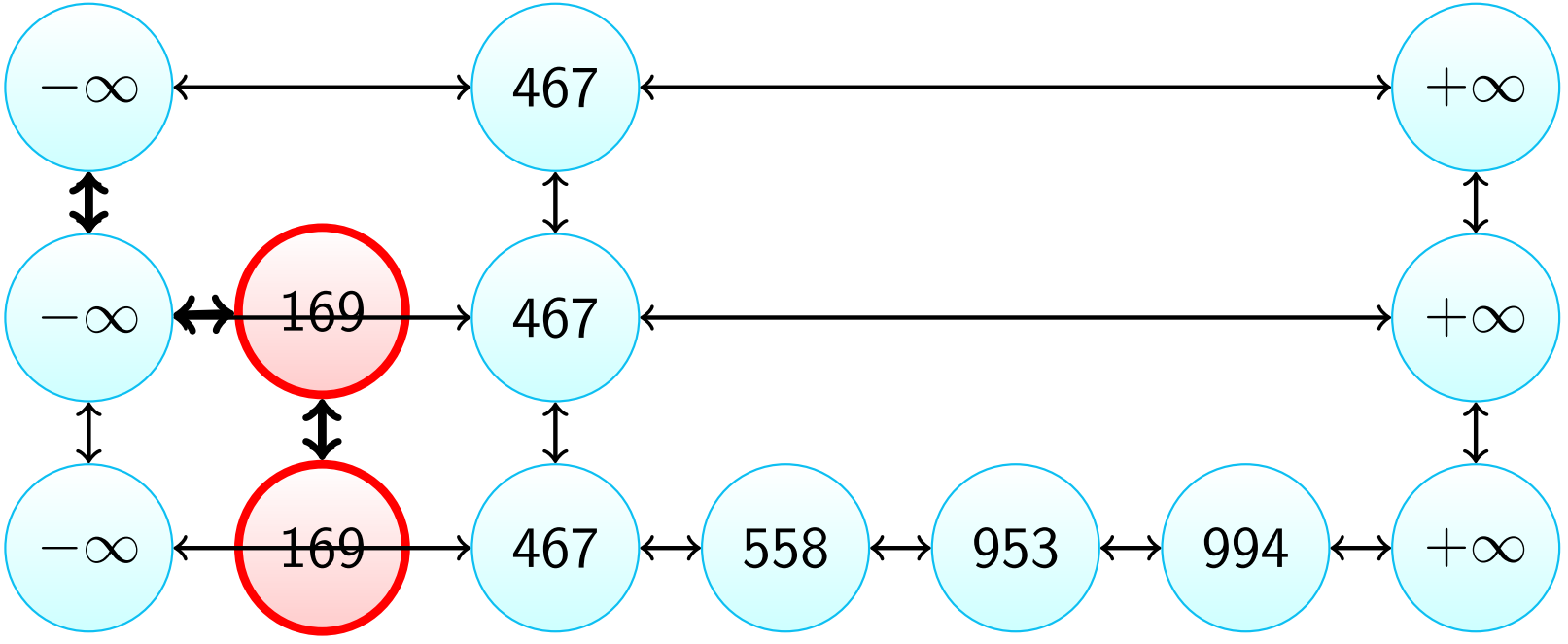
Removing

For example, if you were removing 169:



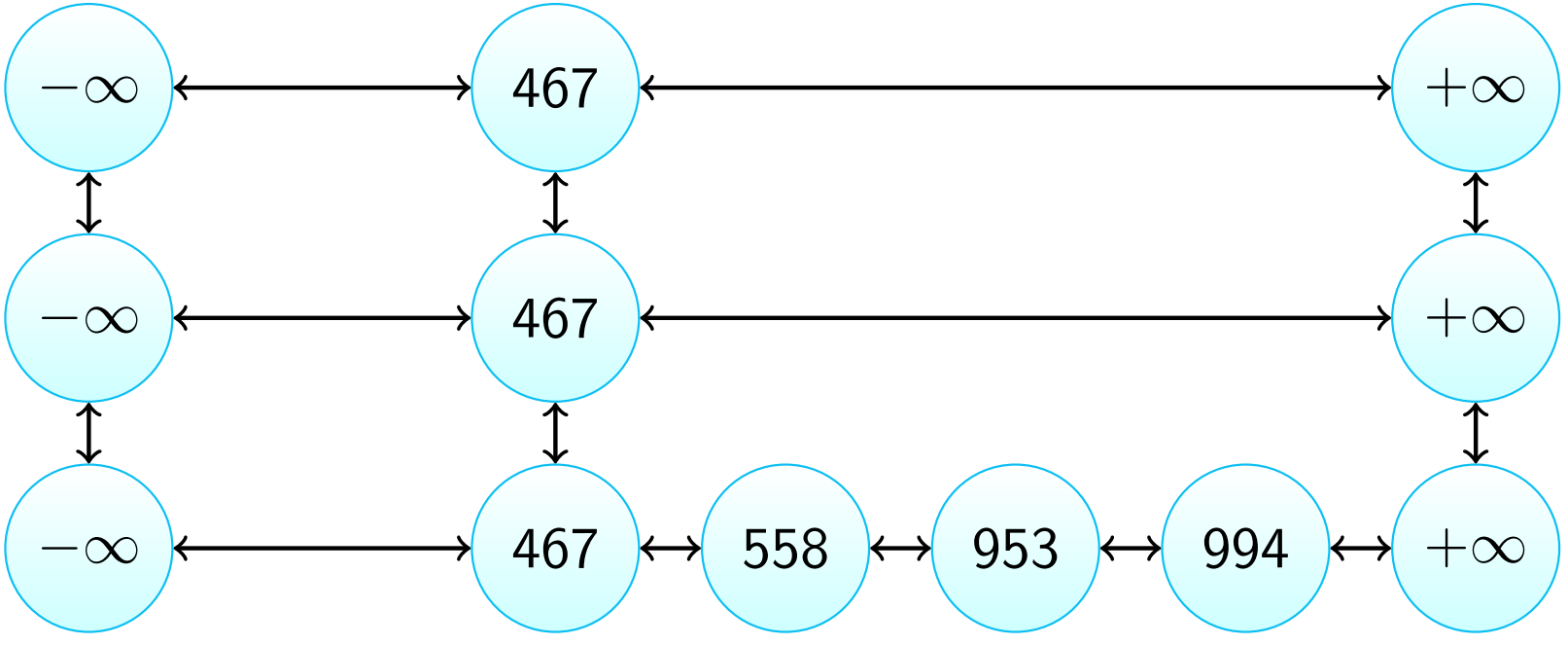
Removing

For example, if you were removing 169:



Removing

For example, if you were removing 169:



Removing

```
procedure REMOVE(data, node)
    if node is not valid then
        return
    else
        while data > node.next.data do
            node ← node.next
        end while
        if data = node.next.data then
            REMOVE(node.next)
            remove any empty levels
        else
            REMOVE(data, node.down)
        end if
    end if
end procedure
```

Removing

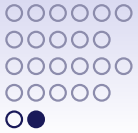
procedure REMOVE(*node*)
    while *node* is valid **do**
        disconnect/remove *node* from this level
        node $\leftarrow$ *node.down*
    end while
end procedure

Performance

- In the best case, a truly random coin toss is used, the skip list has $\log n$ levels (where n is the number of data items), and each level has half of the items in the level below. In this case, adding, searching, and removing are $O(\log n)$.

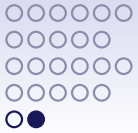
Performance

- In the best case, a truly random coin toss is used, the skip list has $\log n$ levels (where n is the number of data items), and each level has half of the items in the level below. In this case, adding, searching, and removing are $O(\log n)$.
- The average case for all three operations are $O(\log n)$ as well, because each level will contain roughly half of the items on the level below it.



Performance

- In the worst case, the coin toss used is not truly random, or it just so happens that all of the items added into the skip list are at the same level. In this case, adding, searching, and removing are $O(n)$.



Performance

- In the worst case, the coin toss used is not truly random, or it just so happens that all of the items added into the skip list are at the same level. In this case, adding, searching, and removing are $O(n)$.
- Note that unlike all of the data structures we've seen, because each item is stored multiple times in a skip list, the space complexity (the rate at which the space used increases) is $O(n \log n)$ in the worst case (on average, it is rougly $O(n)$). All of the other data structures we've seen have a space complexity of $O(n)$.