Part 5:Grid Data Structures

Set 10

1.The isValid method is specified in the Grid interface.The BounderGrid and UnboundedGrid class implement this method.
2.Method getValidAdjacentLocation calls the isValid method.
Because other methods such as getEmptyAdjacentLocations and getOccupiedAdjacentLocations call getValidAdjacentLocation method, so they call the isValid method indirectly.And method getNeighbors call getOccupierAdjacentLocations, it also call the isValid method indirectly.
3.get() and getOccupiedAdjacentLocations().
The AbstractGrid class implements the getOccupiedLocations method, and the get method is implemented in the BoundedGrid and UnboundedGrid classes.
4.The get method is the only method to test whether or not a given location is empty or occupied.
5.The number of possible valid location would decrease from eight to four.The valid locations would be north, south,east and west of the given location.

Set 11

1.The constructor for the BounderGrid will throw an IllegalArguementException if the number of rows <= 0 or the number of cols <= 0.
2.occupantArray[0].length //return the number of columns in row 0
And according to the constructor precondition, numRows() > 0, so theGrid[0] is non-null.
3.(0 <= loc.getRow() <= getNumRows() && 0 <= loc.getCol() <= getNumCols())
4.ArrayList<Location> is returned. O(r*c)
5.E is returned, which is whatever types is stored in the occupantArray, so it return a Location object.
O(1)
6.If the location is not valid or the obj is null. O(1)
7.The generic type E: whatever type is actually stored in the BounderGrid object.
If the location is empty, null is returned.It is not an error to call remove on an empty location.
O(1)
8.Yes.The only method that is inefficient is the getOccupiedLocation method, O(r*c).Other methods are O(1).
A BoundedGrid stores more than just its occupants.It also stores null values.A BoundedGrid that only stored the occupants is more efficient, which also has O(1) access the occupants.

Set 12

1.The Location class must implement the hashCode and the equals methods.
The TreeMap requires keys of the map to be comparable, and the Location class implements the Comparable interface, so it must implement compareTo method.
Yes, the Location class saticfies all of these requirements.
2.Because in a Map object, null is a legal value, but in an UnboundedGrid object, null is illegal.So check for null is nessesary.
Because isValid method is always true for an UnboundedGrid object.
3.O(1).
If a TreeMap were used instead of HashMap, the average time complexity would be O(log n), where n is the number of occupied locations in the grid.
4.Most of the time, the getOccupiedLocation method will return the occupants in a different order.

HashMap — depends on where it is located in the hash table.
TreeMap — depends on the order defined by Location's compareTo method.

5.Yes, a map could be used to implement a bounded grid.
If the bounded grid were almost full, the map would use more memory than two-dimensional array.

Exercise 1

1.For a large grid which frequently calls the getOccupiedLocations method, the time-complexity for this implementation is O(r+n), where n is the number of this node items in the grid.For BoundedGrid implementatio, the time-complexity is O(r*c);

2.Consider using a HashMap or TreeMap to implement the SparseBoundedGrid. How could you use the UnboundedGrid class to accomplish this task? Which methods of UnboundedGrid could be used without change?

Most of the code in the UnboundedGrid class could be used without modification to create a SparseBoundedGrid class.

Such as: getOccupiedLocations(), get(), put(), remove().

Fill in the following chart to compare the expected Big-Oh efficiencies for each implementation of the SparseBoundedGrid.

Let r = number of rows, c = number of columns, and n = number of occupied locations

Methods	SparseGridNode version	LinkedList<OccupantInCol> version	HashMap version	TreeMap version
getNeighbors	O(c)	O(c)	O(1)	O(log n)
getEmptyAdjacentLocations	O(c)	O(c)	O(1)	O(log n)
getOccupiedAdjacentLocations	O(c)	O(c)	O(1)	O(log n)
getOccupiedLocations	O(r+n)	O(r+n)	O(n)	O(n)
get	O(c)	O(c)	O(1)	O(log n)
put	O(1)	O(1)	O(1)	O(log n)
remove	O(c)	O(c)	O(1)	O(log n)

3. Consider an implementation of an unbounded grid in which all valid locations have non-negative row and column values. The constructor allocates a 16 x 16 array. When a call is made to the put method with a row or column index that is outside the current array bounds, double both array bounds until they are large enough, construct a new square array with those bounds, and place the existing

occupants into the new array.

Implement the methods specified by the Grid interface using this data structure. What is the Big-Oh efficiency of the get method? What is the efficiency of the put method when the row and column index values are within the current array bounds? What is the efficiency when the array needs to be resized?

Big-Oh of get: O(1)

Big-Oh of put: O(1) when row and column values are within the current array bounds

O(size * size) when array needs to be resized.