Set 10

The source code for the AbstractGrid class is in Appendix D.

1. Where is the isValid method specified? Which classes provide an implementation of this method?
Answer: the method isValid is specified in Class Grid. The BoundedGrid class and UnboudedGrid class have it implemented.
2. Which AbstractGrid methods call the isValid method? Why don't the other methods need to call it?
Answer: the getValidAjacentLocations method calls the isValid method. The other methods don't need to call it because they call the getValidAjacentLocations method directly and indirectly.
3. Which methods of the Grid interface are called in the getNeighbors method? Which classes provide implementations of these methods?
Answer: the method of get and getOccupiedAjacentLocations in the Grid interface are called. The getOccupiedAjacentLocations method is implemented in AbstactGrid and the get method is implemented in both BoundedGrid and UnboudedGrid.
4. Why must the get method, which returns an object of type E, be used in the getEmptyAdjacentLocations method when this method returns locations, not objects of type E?
Answer: we first find out whether a location neighborLoc contains an actor. If doesn't, then we will put the location to the locations-list locs that we return in the function.
5. What would be the effect of replacing the constant Location.HALF_RIGHT with Location.RIGHT in the two places where it occurs in the getValidAdjacentLocations method?
Answer: we will get only four neighbors of the current location. That is, the neighbors except the four conner neighbors.

Set 11

The source code for the BoundedGrid class is in Appendix D.

1. What ensures that a grid has at least one valid location?
Answer:the constructor. If one of the number rows or cols is less than or equal to zero, it will throw an exception of IllegalArgumentException.
2. How is the number of columns in the grid determined by the getNumCols method? What assumption about the grid makes this possible?
Answer: occupantArray[0].length. The BoundedGrid have at least a row and a col.
3. What are the requirements for a Location to be valid in a BoundedGrid?
Answer: 0 <= Location.getRow() < BoundedGrid.getNumRows() && 0 <= Location.getCol() < BoundedGrid.getNumCols().

In the next four questions, let r = number of rows, c = number of columns, and n = number of occupied locations.

4. What type is returned by the getOccupiedLocations method? What is the time complexity (Big-Oh) for this method?
Answer: ArrayList<Location>. Time complexity is O(r*c).
5. What type is returned by the get method? What parameter is needed? What is the time complexity (Big-Oh) for this method?
Answer: type of E, a parameter of a location is needed. Time complexity O(1)
6. What conditions may cause an exception to be thrown by the put method? What is the time complexity (Big-Oh) for this method?
Answer: the location is not valid and the no object to put. Time complexity O(1)

7. What type is returned by the remove method? What happens when an attempt is made to remove an

item from an empty location? What is the time complexity (Big-Oh) for this method?

Answer: returns general type of E. Stores null and return null. Time Complexity O(1).

8. Based on the answers to questions 4, 5, 6, and 7, would you consider this an efficient implementation? Justify your answer.

Answer: Yes. The most inefficent method is getOccupiedLocations with time complexity O(r*c) while other method(get, put remove) with O(1). But it stores null in the unoccupied locations which is not so efficient and using a hashmap may be better.

Do You Know?

Set 12

The source code for the UnboundedGrid class is in Appendix D.

1. Which method must the Location class implement so that an instance of HashMap can be used for the map? What would be required of the Location class if a TreeMap were used instead? Does Location satisfy these requirements?

Answer:

the hashCode and the eqauls methods. When the equals method is called and tested true, the Hashcode method must return the samevalue for two locations.

The comparable interface need to be implemented. And so is the compareTo method. When the equals method is called and tested true, the compareTo method must return 0 for two locations. Also, the treeMap requires keys of the map to be Comparable.

The Location class satisfies all of these requirements.

2. Why are the checks for null included in the get, put, and remove methods? Why are no such checks included in the corresponding methods for the BoundedGrid?

Answer: BoundedGrid uses isValid methods to check if a location is valid, but the isValid method in UnboundedGrid return true which can't check a null location to an invalid location.

3. What is the average time complexity (Big-Oh) for the three methods: get, put, and remove? What would it be if a TreeMap were used instead of a HashMap?

Answer: the average time complexity is O(1). If a TreeMay were used instead, the average time complexity would be O(log n), where n is the number of occupied locations.

4. How would the behavior of this class differ, aside from time complexity, if a TreeMap were used instead of a HashMap?

Answer: the returned occupants will have a different order.

5. Could a map implementation be used for a bounded grid? What advantage, if any, would the two-dimensional array implementation that is used by the BoundedGrid class have over a map implementation?

Answer: Yes, a map implementation can be used for a bounded grid. For example, if a HashMap is used , the time complexity of the getOccupiedLocations method is O(n), where n is the number of existing items in the grid.

The two-dimensional array will use less memory than a HashMap when the grid is almost full because the HashMay stores locations and the items contained, while the array only store items.

Exercise

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?

Answer: for the UnboundedGrid is accomplished with a HashMap, we just change the valid condition judgement for the grid boundary is OK. The method getOccupiedLocations could be used without change.

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(c)	O(c)	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?

Answer: the get method Time complexity:O(1);

the put method Time complexity: O(1) needn't to resize and O(newDim^2) when need to resize to the new dimension newDim.