13331235 TanXiao

1 Part 5 - Set10

1.1 Where is the isValid method specified? Which classes provide an implementation of this method?

It is specified in the Grid Class and implemented by the BoundedGrid class and the UnboundedGrid class.

1.2 Which AbstractGrid methods call the isValid method? Why don't the other methods need to call it?

The getValidAdjacentLocations method.

Because other methods such as the getOccupiedAdjacentLocations method call the getValidAdjacentLocations method and the getNeighbors method call the getOccupiedAdjacentLocations method.

1.3 Which methods of the Grid interface are called in the getNeighbors method? Which classes provide implementations of these methods?

It calls the get method and the getOccupiedAdjacentLocations method.

The getOccupiedAdjacentLocations method is implemented by the AbstractGrid class.

The get method is implemented by the BoundedGrid class and the UnboundedGrid class.

1.4 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?

When a location has an object, the get method returns the reference of this object. When a location is empty, the get method returns null. That is, we can judge a location is empty by the get method. That's why the getEmptyAdjacentLocations method use the get method.

1.5 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?

The method will only judge current location's North, South, East, West directions' locations instead of eight directions.

2 Part5 - Set11

2.1 What ensures that a grid has at least one valid location?

The Constructor of this class. When the initial number of row or column <= 0, it will throw an IllegalArgumentException.

2.2 How is the number of columns in the grid determined by the getNumCols method? What assumption about the grid makes this possible?

The number of columns determined by the getNumCols method is occupantArray[0].length.

From the previous question, the constructor ensures that the number of row and column are both > 0, so that occupantArray[0].length has a valid value.

2.3 What are the requirements for a Location to be valid in a BoundedGrid?

This location's row number is belong to [0, getNumRows())

This location's column number is belong to [0, getNumCols())

2.4 What type is returned by the getOccupiedLocations method? What is the time complexity (Big-Oh) for this method?

ArrayList<Location>

O(r * c)

2.5 What type is returned by the get method? What parameter is needed? What is the time complexity (Big-Oh) for this method?

Type E.

A Location type paremeter.

O(1)

2.6 What conditions may cause an exception to be thrown by the put method? What is the time complexity (Big-Oh) for this method?

If the Location parameter is not valid or the <E> parameter is null.

O(1)

2.7 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?

Type E.

It will throw an IllegalArgumentException.

O(1)

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

Yes. We can use a Map like the UnboundedGrid class to store the occupied objects so that we can reduce the time complexity of the getOccupiedLocations method to O(n).

The other methods' time complexity are all O(1), so we can't give efficient implementations.

3 Part5 – Set12

3.1 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?

The hashCode method and the equals method.

The compareTo method.

Location satisfy these requirements.

3.2 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?

Because the isValid method in the UnboundedGrid is always return true. What else, the null is valid for HashMap and is invalid for the UnboundedGrid.

In the BoundedGrid, the isValid method will call the getRow and the getCol methods, which will throw a NullPointException when the Location is null.

3.3 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?

O(1)

TreeMap: O(log (getOccupiedLocations().size()))

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

The order of the getOccupiedLocations method's result.

3.5 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?

Yes. When the BoundedGrid use the HashMap, it's getOccupiedLocations method's time complexity will be reduce to O(n).

The two-dimensional array implementation's advantage: if there is many objects in the BoundedGrid that they are almost occupied the whole grid, the HashMap will use more memory than the two-dimensional array.

4 Part5 - Exercises

4.1 Suppose that a program requires a very large bounded grid that contains very few objects and that the program frequently calls the getOccupiedLocations method (as, for example, ActorWorld). Create a class SparseBoundedGrid that uses a "sparse array"

implementation. Your solution need not be a generic class; you may simply store occupants of type Object.

Implement the methods specified by the Grid interface using this data structure. Why is this a more time-efficient implementation than BoundedGrid?

Because every time the BoundedGrid's getOccupiedLocations method is called, it will always search each location in the grid. If the grid is large and the number of occupied objects is small, it will waste a lot of time. The time complexity of the getOccupiedLocations method in SparseBoundedGrid is O(n), n is the number of objects in the grid.

```
import java.util.ArrayList;
import java.util.LinkedList;

import info.gridworld.grid.AbstractGrid;
import info.gridworld.grid.Location;

public class SparseBoundedGrid<E> extends AbstractGrid<E> {
   private ArrayList<LinkedList<OccupantInCol>> occupantArray;
   private int numRows;
   private int numCols;

   /**
    * Constructs an empty sparse bounded grid with the given dimensions.
    * (Precondition: <code>rows > 0</code> and <code>cols > 0</code>.)
    *
    * @param rows
    *           number of rows in BoundedGrid
    * @param cols
    *           number of columns in BoundedGrid
    */
   public SparseBoundedGrid(int rows, int cols) {
      if (rows <= 0) {
         throw new IllegalArgumentException("rows <= 0");
      }
```

```
    if (cols <= 0) {
        throw new IllegalArgumentException("cols <= 0");
    }
    numCols = cols;
    numRows = rows;
    occupantArray = new ArrayList<LinkedList<OccupantInCol>>();
    for (int i = 0; i < numRows; i++) {
        occupantArray.add(new LinkedList<OccupantInCol>());
    }
    // System.out.println(occupantArray.size());
}

@Override
public int getNumRows() {
    return numRows;
}

@Override
public int getNumCols() {
    return numCols;
}

@Override
public boolean isValid(Location loc) {
    return 0 <= loc.getRow() && loc.getRow() < getNumRows()
            && 0 <= loc.getCol() && loc.getCol() < getNumCols();
}

@Override
public ArrayList<Location> getOccupiedLocations() {
    ArrayList<Location> theLocations = new ArrayList<Location>();

    // Look at all grid locations.
    for (int r = 0; r < getNumRows(); r++) {
        // get the linked list of the row
```

```
        LinkedList<OccupantInCol> everyRow = occupantArray.get(r);
        // if everyRow is not null, it must has at least one of occupants.
        if (everyRow != null) {
            for (OccupantInCol occupant : everyRow) {
                theLocations.add(new Location(r, occupant.getCol()));
            }
        }
    }
    return theLocations;
}

@Override
public E get(Location loc) {
    if (!isValid(loc))
        throw new IllegalArgumentException(
                "Location " + loc + " is not valid");
    // get the linked list of the row, like the getOccupiedLocations method
    LinkedList<OccupantInCol> tarRow = occupantArray.get(loc.getRow());
    // if tarRow is not null, we can use a foreach to find it.
    if (tarRow != null) {
        for (OccupantInCol occupant : tarRow) {
            if (occupant.getCol() == loc.getCol()) {
                return (E) occupant.getOccupant();
            }
        }
    }
    // if can't find, we have to return a null.
    return null;
}

@Override
public E put(Location loc, E obj) {
    if (!isValid(loc))
        throw new IllegalArgumentException(
                "Location " + loc + " is not valid");
```

```
    if (obj == null)
        throw new NullPointerException("obj == null");

    // Add the object to the grid.
    E oldOccupant = get(loc);
    // if oldOccupant have an object, we should remove it
    if (oldOccupant != null) {
        remove(loc);
    }
    occupantArray.get(loc.getRow())
            .add(new OccupantInCol(obj, loc.getCol()));
    return oldOccupant;
}

@Override
public E remove(Location loc) {
    if (!isValid(loc))
        throw new IllegalArgumentException(
                "Location " + loc + " is not valid");

    // Remove the object from the grid.
    E ret = get(loc);
    // if there is nothing in this location, we should return null.
    if (ret == null) {
        return ret;
    }
    // get the linked list of the row, like the getOccupiedLocations method
    LinkedList<OccupantInCol> tarRow = occupantArray.get(loc.getRow());
    for (OccupantInCol occupant : tarRow) {
        if (occupant.getCol() == loc.getCol()) {
            tarRow.remove(occupant);
            return ret;
        }
    }
    return null;
```

}

4.2 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?

I use HashMap. Actually almost all of methods in the UnboundedGrid can be used in this class.

List of methods(the same as UnboundedGrid):

getOccupiedLocations

get

put

remove

Code:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import info.gridworld.grid.AbstractGrid;
import info.gridworld.grid.Location;

public class SparseBoundedGrid2<E> extends AbstractGrid<E> {
   private Map<Location, E> occupantMap;
   private int numRows;
   private int numCols;

   /**
    * Constructs an empty sparse bounded grid with the given dimensions.
    * (Precondition: <code>rows > 0</code> and <code>cols > 0</code>.)
    *
    * @param rows
    *           number of rows in BoundedGrid
    * @param cols
    *           number of columns in BoundedGrid
    */
   public SparseBoundedGrid2(int rows, int cols) {
      numCols = cols;
      numRows = rows;
```

```
    occupantMap = new HashMap<Location, E>();
}

@Override
public int getNumRows() {
    return numRows;
}

@Override
public int getNumCols() {
    return numCols;
}

@Override
public boolean isValid(Location loc) {
    return 0 <= loc.getRow() && loc.getRow() < getNumRows()
            && 0 <= loc.getCol() && loc.getCol() < getNumCols();
}

@Override
public ArrayList<Location> getOccupiedLocations() {
    ArrayList<Location> a = new ArrayList<Location>();
    for (Location loc : occupantMap.keySet())
        a.add(loc);
    return a;
}

@Override
public E get(Location loc) {
    if (loc == null)
        throw new NullPointerException("loc == null");
    return occupantMap.get(loc);
}

@Override
```

```
public E put(Location loc, E obj) {
    if (loc == null)
        throw new NullPointerException("loc == null");
    if (obj == null)
        throw new NullPointerException("obj == null");
    return occupantMap.put(loc, obj);
}

@Override
public E remove(Location loc) {
    if (loc == null)
        throw new NullPointerException("loc == null");
    return occupantMap.remove(loc);
}
```

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)

4.3 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?

get method: O(1)

put method: 1). If index values are within the current array bounds: O(1)

2). If the array needs to be resized: O(size^2), size is current array's size(Row or Col, because Row = Col).

```
// Code:
import java.util.ArrayList;

import info.gridworld.grid.AbstractGrid;
import info.gridworld.grid.Location;

/**
 * An <code>UnboundedGrid</code> is a rectangular grid with an unbounded number
 * of rows and columns. <br />
 * The implementation of this class is testable on the AP CS AB exam.
 */
public class UnboundedGrid2<E> extends AbstractGrid<E> {
   private Object[][] occupantArray; // the array storing the grid elements
   private int size; // the grid's current size

   /**
    * Constructs an empty unbounded grid with the default 16 dimensions.
    * (Precondition: <code>rows > 0</code> and <code>cols > 0</code>.)
    */
   public UnboundedGrid2() {
      size = 16;
      occupantArray = new Object[size][size];
   }

   @Override
   public E get(Location loc) {
      if (!isValid(loc))
         throw new IllegalArgumentException(
               "Location " + loc + " is not valid");
```

```
    // isValid 只能判断位置是否合法,不能判断位置是不是在当前的 size 范围内
    if (loc.getCol() >= size || loc.getRow() >= size) {
        // System.out.println("null!\n");
        return null;
    }
    return (E) occupantArray[loc.getRow()][loc.getCol()]; // unavoidable
                                                          // warning
}

@Override
public int getNumCols() {
    return -1;
}

@Override
public int getNumRows() {
    return -1;
}

@Override
public ArrayList<Location> getOccupiedLocations() {
    ArrayList<Location> theLocations = new ArrayList<Location>();
    // Look at all grid locations.
    for (int r = 0; r < size; r++) {
        for (int c = 0; c < size; c++) {
            // If there's an object at this location, put it in the array.
            Location loc = new Location(r, c);
            if (get(loc) != null)
                theLocations.add(loc);
        }
    }
    return theLocations;
}

@Override
```

```
public boolean isValid(Location loc) {
    // row and col must >= 0
    return (0 <= loc.getRow() && 0 <= loc.getCol());
}

@Override
public E put(Location loc, E obj) {
    if (!isValid(loc))
        throw new IllegalArgumentException(
                "Location " + loc + " is not valid");
    if (obj == null)
        throw new NullPointerException("obj == null");
    // 位置超出当前范围,调用 resize
    int minSize = Math.max(loc.getCol(), loc.getRow()) + 1;
    if (size < minSize) {
        resize(minSize);
    }
    // Add the object to the grid.
    E oldOccupant = get(loc);
    occupantArray[loc.getRow()][loc.getCol()] = obj;
    return oldOccupant;
}

@Override
public E remove(Location loc) {
    if (!isValid(loc))
        throw new IllegalArgumentException(
                "Location " + loc + " is not valid");
    // isValid 只能判断位置是否合法,不能判断位置是不是在当前的 size 范围内
    if (loc.getCol() >= size || loc.getRow() >= size) {
        return null;
    }
    // Remove the object from the grid.
    E r = get(loc);
    occupantArray[loc.getRow()][loc.getCol()] = null;
```

```
        return r;
    }

    /**
     * resize the matrix
     *
     * @param minSize:
     *            size must >= minSize
     */
    public void resize(int minSize) {
        // System.out.println("resize!\n");
        /*
         * store previous array & objects Hint: must use the old size to get
         * occupied locations!
         */
        Object[][] temp = occupantArray.clone();
        ArrayList<Location> locs = new ArrayList<Location>();
        locs = getOccupiedLocations();
        // double size until it is satisfied with the parameter
        while (size < minSize) {
            size *= 2;
        }
        occupantArray = new Object[size][size];
        for (Location location : locs) {
            // copy objects to new array
            occupantArray[location.getRow()][location
                    .getCol()] = temp[location.getRow()][location.getCol()];
        }
    }
}
```