中山大學
SUN YAT-SEN UNIVERSITY

实训报告

作业名称	Part 5: Grid Data Structures					
院系	软件学院		班级	教务 2 班	姓名	李明宽
学号	11331173		完成日期	2013 年 07 月 07 日		
QQ	736459905		**E-mail**	limkuan@mail2.sysu.edu.cn		

The AbstractGrid Class

Two concrete implementations of the Grid interface are provided, one for a bounded grid that has a fixed number of rows and columns, and a second for an unbounded grid, for which any row and column values are valid. Rather than have repeated code in two classes, the AbstractGrid class defines five methods of the Grid interface that are common to both implementations.

Do You Know?

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 isValid method is declared in interface Grid, and is provided an implement of this method in class BoundedGrid and class UnboundedGrid. In the class BoundedGrid, the method returns whether the location is in grid, and in the class UnboundedGrid, the method always returns true.

2. Which AbstractGrid methods call the isValid method? Why don't the other methods need to call it?
   Answer: In method getValidAdjacentLocations calls the isValid method directly, and other methods don't call isValid method directly, but they call it indirectly. They call the method which is call isValid method to ensure the location is valid.

3. Which methods of the Grid interface are called in the getNeighbors method? Which classes provide implementations of these methods?
   Answer: getOccupiedAdjacentLocations and get method is call in the getNeighbors method in class AbstractGrid. These methods are declared in the Grid interface, and the getOccupiedAdjacentLocations method is implemented in the class AbstractGrid. The get method is implemented in the class BoundedGrid and the class UnboundedGrid. The class BoundedGrid and UnboundedGrid has different method to get an actor in given location.

4. Why must the get method, which returns an object of type E, be used in the getEmptyAdjacentLocations method when method returns locations, not objects of type E?
   Answer: We use get method and return the objects in neighborLoc, if the object is null, we know the location is empty and add the location to the empty locations list, others the location is occupied and we shouldn't add it to empty locations list.

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: it only test the location in North, South, East, and West, and return whether the location is valid. In other direction, the method will not test it and even it is valid, the method will not return it.

The BoundedGrid Class

A bounded grid has a fixed number of rows and columns. You can access only locations that are within the bounds of the grid. If you try to access an invalid location, a run-time exception will be thrown.

Do You Know?

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

1. What ensures that a grid has at least one valid location?
   Answer: In the constructed function, the method ensure that the number of rows and columns is larger than 0, if it small than 0, it will throw IllegalArgumentException. In this way, we can ensure the row and column is at least 1, and the grid has at least one valid location.

2. How is the number of columns in the grid determined by the getNumCols method? What assumption about the grid makes this possible?
   Answer: The grid is saving as a two-dimension array occupantArray, so the number of column in the grid is the number of the two-dimension array, we use occupantArray[0].length can get this number.

   The assumption in constructor about the number of row and column is larger than 0 make sure that the array is valid to get the length of rows and columns.

3. What are the requirements for a Location to be valid in a BoundedGrid?
   Answer: It must be sure the index of array is not out-of-range, which means that the row number of location is larger than 0 and smaller than the total number of rows, and the same with columns.

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: The getOccupiedLocations method returns a list of Locations. The time complexity for this method 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: The get method returns an Object E, which it can be an actor. And the time complexity of get method is 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: If the location is not valid, it will throw an IllegalArgumentException.

   If the Object which needs to be put in the grid is null, it will throw a NullPointerException.

   The time complexity for this method is 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: The remove method return an Object E which is the type in the location, and the object has been removed from the grid.

   If an attempt is made to remove an item from an empty location, it will set the array of that location to null and return null.

   The time complexity for this method is 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: On one hand, I think this is an efficient implementation, because we can complete get, put, remove and other methods in O(1) time complexity.
   On the other hand, the implementation is waste space. Because we must store lots of null object in two- dimension array, if the grid is big and the actor is less, this data structure is waste space.

The UnboundedGrid Class

In an unbounded grid, any location is valid, even when the row or column is negative or very large. Since there is no bound on the row or column of a location in this grid, the UnboundedGrid<E> class does not use a fixed size two-dimensional array. Instead, it stores occupants in a Map<Location, E>. The key type of the map is Location and the value type is E, the type of the grid occupants.

The numRows and numCols methods both return -1 to indicate that an unbounded grid does not have any specific number of rows or columns. The isValid method always returns true. The get, put, and remove methods simply invoke the corresponding Map methods. The getOccupiedLocations method returns the same locations that are contained in the key set for the map.

Do You Know?

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 method of the Location class implement so that an instance of HashMap can be used for the map because we use hashCode method to compute the hashCode of an object. And equals method of Location class must be implemented because finding in HashMap, we need the method to determine which object in the same hash code is the object we need.
   If we use TreeMap instead, the location class must implement interface Comparable and CompareTo method because the TreeMap is sorted by key of the map and the key of the TreeMap must be comparable.

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: Because if the location is null, the key of HashMap is null is not allowed. If in the put method, the Object is null means that the value of HashMap is null, the data structure allow this but it needn't store a null location, it is a waste of space. So we should check for null included in the get, put and remove methods.

   In BoundedGrid, just ensure that the locations are valid, and if the location is null it can't be valid, and if the object is null, it doesn't matter because place a null object in the location which is store a value null, it won't change anything. So there no such checks.

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: If we use HashMap, the average time complexity for the get, put and remove methods are O(1).

   But if we use TreeMap, the average time complexity for the get method is O(logN), the average time complexity for the put and remove methods also are O(logN).

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

   Answer: TreeMap is sorted by key in this project it means Location. And we use TreeMap, we can get some sorted locations, in some ways, it will be more convenience, but the time complexity is higher than using a HashMap.

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. We can use a HashMap to store the location and object as key and value, just attention that the location must be valid in the grid.

   If the grid has lots of objects, the two-dimensional array implementation is better than use a HashMap. The HashMap implementation may use more space than array while the grid is almost full. And the array is easy to implement and preserve.

Exercises

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.

   The "sparse array" is an array list of linked lists. Each linked list entry holds both a grid occupant and a column index. Each entry in the array list is a linked list or is null if that row is empty.

   Answer: I implement the linked list using a LinkedList<OccupantInCol> with a helper class.

   In this way, if the grid is sparse, it will save a lot of space. But if the grid is almost full, it will waste more time to get or put an object.

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?

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

   Answer: I implement the SparseBoundedGrid using HashMap. We can use the UnboundedGrid class to accomplish the task.

   Some methods of UnboundedGrid can use directly without change, such as getOccupiedLocations methods. We need to change the method getNumRows, getNumCols, and isValid method, and add judge statement whether the location is valid to get, put, and remove methods.

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

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

The getNeighbors, getEmptyAdjacentLocations, getOccupiedAdjacentLocations methods call get method, so the time complexity is same with get method.

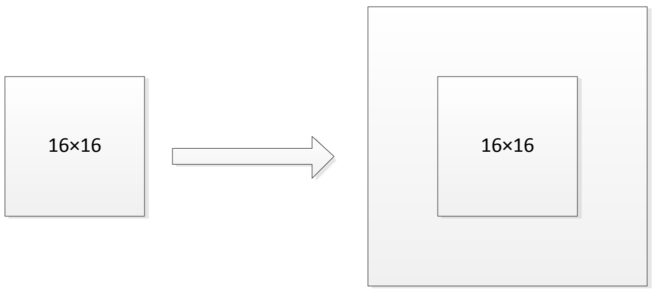
3. Consider an implementation of an unbounded grid in which all valid locations have non-negative row and column values. The constructor allocates a 16 x 16 array. When a call is made to the put method with a row or column index that is outside the current array bounds, double both array bounds until 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: Using a big square array to implement the UnboundedGrid, we need to use a dimension to determine the current dimension of array, if we need to put an actor outside the array, we need to double the array and multiply 2 to dimension.

Attention that the location of Grid will be negative, so if the array is n dimension, the origin of grid is the position (n/2-1, n/2-1) in array, it means the beginning of array (0, 0) indicate the location (1-n/2, 1-n/2) in the grid.

If the array needs to double, it will add dimension in the way show below.



Before doubling, the range is (-7, -7)~(7, 7) , After doubling, the range of array is added to (-15, -15)~(15, 15).

The source code of above three exercises has been checked.