1.

It is specified in the Grid interface:

//@file:info/gridworld/grid/Grid.java

//@line:50

boolean isValid(Location loc);

The BoundedGrid and the UnBounded class implements the method.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:62~63

return 0 <= loc.getRow() && loc.getRow() < getNumRows()

&& 0 <= loc.getCol() && loc.getCol() < getNumCols();

//@file:info/gridworld/grid/UnBoundedGrid.java

//@line:55

return true;

2.

The getValidAdjacentLocations() method calls the isValid method.Because the getEmptyAdjacentLocations() and getOccupiedAdjacentLocations() method could call the getValidAdjacentLocations() to get all the valid and adjacent locations, them dont't need to call the isValid() method directly.The getNeighbors() method calls the getOccupiedAdjacentLocations() method, so it also don't need to call the isValid() method directly.

//@file:info/gridworld/grid/AbstractGrid.java

//@line:44

if (isValid(neighborLoc))

3.

The getOccupiedAdjacentLocations() and the get() method.The AbstractGrid class implements the getOccupiedAdjacentLocations() method and the BoundedGrid and the UnBoundedGrid class implements the get() method.

//@file:info/gridworld/grid/AbstractGrid.java

//@line:62

public ArrayList<Location> getOccupiedAdjacentLocations(Location loc)

//@file:info/gridworld/grid/BoundedGrid.java

//@line:85

public E get(Location loc)

//@file:info/gridworld/grid/UnBoundedGrid.java

//@line:66

public E get(Location loc)

4.

Because we need the get() method to konw whether the location is empty or occupied.If the get() method returns null, then the location is empty, otherwise the location is occupied.If the location is empty, then add it to the ArrayList<Locations> that the getEmptyAdjacentLocations() method will return.

//@file:info/gridworld/grid/AbstractGrid.java

//@line:56~57

if (get(neighborLoc) == null)

locs.add(neighborLoc);

5.

Then the getValidAdjacentLocations() method will only return the valid locations in the north, south, east and west direction of the location.That is, the for loop will only loop for 4 times.

//@file:info/gridworld/grid/AbstractGrid.

//@line:41~47

for (int i = 0; i < Location.FULL\_CIRCLE / Location.HALF\_RIGHT; i++)

{

Location neighborLoc = loc.getAdjacentLocation(d);

if (isValid(neighborLoc))

locs.add(neighborLoc);

d = d + Location.HALF\_RIGHT;

}

6.

The BoundedGrid class will check the validity of the numrows and numcols in it's constructor, if one of the two parameters is less than or equal to 0, then it will throw an IllegalArgumentException.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:41~44

if (rows <= 0)

throw new IllegalArgumentException("rows <= 0");

if (cols <= 0)

throw new IllegalArgumentException("cols <= 0");

7.

return occupantArray[0].length;

//@file:info/gridworld/grid/BoundedGrid.java

//@line:57

return occupantArray[0].length;

It assumes that occupantArray[0] is not null, because the constructor has checked the validity of the numrows and numcols.

8.

\*\*0 <= row < numrows and 0 <= col < numcols\*\*.row and col are the value of the Row and Col of the location.numrows and numcols are the value of the number of rows and cols in grid.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:62~63

return 0 <= loc.getRow() && loc.getRow() < getNumRows()

&& 0 <= loc.getCol() && loc.getCol() < getNumCols();

9.

ArrayList<Location>.The time complexity (Big-Oh) for this method is O(r \* c) since it uses two nesting for loops.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:68~82

ArrayList<Location> theLocations = new ArrayList<Location>();

// Look at all grid locations.

for (int r = 0; r < getNumRows(); r++)

{

for (int c = 0; c < getNumCols(); 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;

10.

Type E.E is a generic type of object, it represents the entry in the occupantArray.

It need a location for parameter.

It's time complexity (Big-Oh) is O(1), since use array to get entry is efficient.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:85~91

public E get(Location loc)

{

if (!isValid(loc))

throw new IllegalArgumentException("Location " + loc

+ " is not valid");

return (E) occupantArray[loc.getRow()][loc.getCol()]; // unavoidable warning

}

11.

When the location is not valid the put method() will throw an IllegalArgumentException, and when the object is null, it will throw a NullPointerException.

It's time complexity is O(1), the same as the get() method.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:95~99 and 103

if (!isValid(loc))

throw new IllegalArgumentException("Location " + loc

+ " is not valid");

if (obj == null)

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

occupantArray[loc.getRow()][loc.getCol()] = obj;

12.

Type E.E is a generic type of object, it represents the entry in the occupantArray.

It will return null.

It's time complexity is O(1), the same as the get() and put() method.All of them are operating on an array.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:114~115

E r = get(loc);

occupantArray[loc.getRow()][loc.getCol()] = null;

13.

I think it is very efficient, because the get(), put() and remove() method have time complexity O(1), it is the most effcifient way to get, put and remove.Although the getOccupiedLocations() method is not efficient, but in general, it's an an efficient implementation.

O(1):

//@file:info/gridworld/grid/BoundedGrid.java

//@line:90 and 103 and 115

return (E) occupantArray[loc.getRow()][loc.getCol()]

occupantArray[loc.getRow()][loc.getCol()] = obj;

occupantArray[loc.getRow()][loc.getCol()] = null;

14.

The Location class must implement the \*\*hashcode() and equals()\*\* method to ensure HashMap can be used for the map, because Location is the key for the HashMap, and hashmap need to use a hashnode to find it's entry.

//@file:info/gridworld/grid/Location.java

//@line:205~212 and 218~221

public boolean equals(Object other)

{

if (!(other instanceof Location))

return false;

Location otherLoc = (Location) other;

return getRow() == otherLoc.getRow() && getCol() == otherLoc.getCol();

}

public int hashCode()

{

return getRow() \* 3737 + getCol();

}

If a TreeMap were used instead, then the Location class must implement the compareTo() method to ensure that two Location can be compared.Because all the entries in the treemap have an order, so it is important to compare the key.

//@file:info/gridworld/grid/Location.java

//@line:234~246

public int compareTo(Object other)

{

Location otherLoc = (Location) other;

if (getRow() < otherLoc.getRow())

return -1;

if (getRow() > otherLoc.getRow())

return 1;

if (getCol() < otherLoc.getCol())

return -1;

if (getCol() > otherLoc.getCol())

return 1;

return 0;

}

\*\* Location satisfy all these requirements.\*\*

15.

Because the UnBoundedGrid class was implemented by using a HashMap, and only a valid Location could be a key of the HashMap, which means the Location can't be null to be a key.So we must check for null in the get, put, and remove methods.

//@file:info/gridworld/grid/UnBoundedGrid.java

//@line:40

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

Because the BoundedGrid classs has a isValid method which check whether the location is in the grid, and the get, put, and remove method call the isValid method to check the location.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:60~64

public boolean isValid(Location loc)

{

return 0 <= loc.getRow() && loc.getRow() < getNumRows()

&& 0 <= loc.getCol() && loc.getCol() < getNumCols();

}

16.

The three methods simply invoke the corresponding Map methods, so the average time complexity would be O(1) since it is a HashMap.

If it is a TreeMap, then the average time complexity would be O(logn), n is the number of occupants in the UnBoundedGrid.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:70 and 79 and 86

return occupantMap.get(loc);

return occupantMap.put(loc, obj);

return occupantMap.remove(loc);

17.

All the entries in the TreeMap are ordered, while the HashMap is not.So the getOccupiedLocations() method will returns an ordered result if a TreeMap were used instead of a HashMap.

//@file:info/gridworld/grid/BoundedGrid.java

//@line:58~64

public ArrayList<Location> getOccupiedLocations()

{

ArrayList<Location> a = new ArrayList<Location>();

for (Location loc : occupantMap.keySet())

a.add(loc);

return a;

}

18.

Yes, a map implementation could be used for a bounded grid.The advantage is that a HashMap implementation can have a time complexity O(n) for the getOccupiedLocations() method, it's more efficient.

//@file:info/gridworld/grid/UnBoundedGrid.java

//@line:58~64

public ArrayList<Location> getOccupiedLocations()

{

ArrayList<Location> a = new ArrayList<Location>();

for (Location loc : occupantMap.keySet())

a.add(loc);

return a;

}

19.

Methods | SparseGridNode version | LinkedList<OccupantInCol> version | HashMap version | TreeMap version |

--- | --- | --- | --- | --- |

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(1) | 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) |

For the SparseGridNode version, only the getOccupiedLocations has Big-Oh efficiencies O(r + n), others all have O(c), because the getOccupiedLocations need to scan all the occupants, it need loop for r times, and every time it need scan n occupants.The others need only first to find the row and then scan n occupants, so it's O(n).

//@file:Part5/1/SparseBoundedGrid.java

//@line:75~83

for (int r = 0; r < getNumRows(); r++)

{

SparseGridNode row = occupantArray[r];

while (row != null) {

Location loc = new Location(r, row.getCol());

theLocations.add(loc);

row = row.getNext();

}

}

For the TreeMap version, only the getOccupiedLocations has Big-Oh efficiencies O(n), others all have O(logn), because the getOccupiedLocation method need only to find all the keys in the Map's keySet(since the location is the key).The others al have the same Big-Oh efficiencies as the search efficiencies for a tree, which is O(logn).

//@file:Part5/2/SparseBoundedGrid.java

//@line:75~77

for (Location loc : occupantMap.keySet()) {

a.add(loc);

}

20.

The Big-Oh efficiency of the get method is O(1) since we use array to find the entry.

//@file:Part5/3/UnBoundedGrid.java

//@line:92

return (E) occupantArray[loc.getRow()][loc.getCol()]; // unavoidable warning

The efficiency of the put method when don't need resize is O(1) since we just put the entry into an array

//@file:Part5/3/UnBoundedGrid.java

//@line:111

occupantArray[loc.getRow()][loc.getCol()] = obj;

The efficiency when the array needs to be resized is O(len \* len), len is the current size for the array.Because we need to resize the array, and the resize() method uses two nesting for loops.

//@file:Part5/3/UnBoundedGrid.java

//@line:138~142

for (int r = 0; r < len; ++r) {

for (int c = 0; c < len; ++c) {

newArray[r][c] = occupantArray[r][c];

}

}