Project 4

Due Date: Friday 30 April 2021 by 11:59 PM
Note: This assignment will not be accepted late!

General Guidelines.

The method signatures provided in the skeleton code indicate the required methods. You may need additional methods or classes that will not be directly tested but may be necessary to complete the assignment, and you are welcome to add those into the classes.

Unless otherwise stated in this handout, you are welcome to add to/alter any provided java files as well as create new java files as needed. Your solution must be coded in Java.

In general, you are not allowed to import any additional classes in your code without explicit permission from your instructor!

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Note on grading and provided tests: The provided tests are to help you as you implement the project and (in the case of auto-graded sections) to give you an idea of the number of points you are likely to receive. Please note that the points indicated when you run these tests locally are not your final grade. Your solution will be graded by the TAs after you submit. Please also note that these test cases are not likely to be exhaustive and find every possible error. Part of programming is learning to test and debug your own code, so if something goes wrong, we can help guide you in the debugging process but we will not debug your code for you.

Project Overview.

In this project, you will use some Graph algorithms to solve problems related to Grids.

Part 1. Word Search

In this part, the grid you are given will contain individual letters represented as Strings, and you must implement a word search. A word can occur as any path from the first letter to the last where a step in the path can go up, right, down, or left. (No diagonals). Also, it is possible for the same letter to be used more than once in the same word. (See ARIZONA example below.)

	0	1	2	3	4	5	6	7	8	9
0	A	N	O	D	E	F	G	H	I	J
1	R	O	N	N	O	P	Q	W	E	R
2	I	Z	A	E	R	P	A	S	D	F
3	G	H	J	V	S	L	G	H	J	K
4	A	S	W	S	R	F	F	G	H	J
5	B	A	M	N	I	V	C	X	Z	A
6	S	W	Y	U	R	E	V	T	B	Y
7	N	D	C	I	S	R	P	O	G	H
8	I	L	A	T	Y	S	C	V	R	F
9	W	S	T	A	S	D	F	G	H	E

The output for this search should be a String representation of the locations in the path as shown below.

Examples:
```
ARIZONA: (0, 0)(0, 1)(0, 2)(1, 2)(1, 1)(1, 0)(0, 1)
UNIVERSITY: (6, 3)(5, 3)(5, 4)(5, 5)(6, 5)(7, 5)(7, 4)(7, 3)(8, 3)(8, 
4)
WILDCATS: (9, 0)(8, 0)(8, 1)(7, 1)(7, 2)(8, 2)(9, 2)(9, 1)
```

Additionally, the search function will take two integer parameters that indicate the row and column that your search will start from.

Your implementation should follow these guidelines.

- Use a BFS algorithm to find the first character in the word. This search will not be included in the final path that is printed out, but it is included in the overall grid

access counts, so it is important to use BFS in order to find the closest option first and go on from there.
- Once you find a possible starting place for the word, use a DFS algorithm to search for the word itself.
- Always search a locations neighbors in the following order: UP, RIGHT, DOWN, LEFT. If you don't, you may get an alternate route that does not match the test cases.

Example: Let's say we are searching the above grid for ARIZONA starting at (2, 3). First we search for the closest "A" using a BFS algorithm, which is at (2, 2). We then use DFS to search for the word. When we don't find it, we continue the BFS for the next closest "A". Eventually, we find the correct "A" and find the path as noted above.

Testing.
- Test cases and test code is provided for you. Keep in mind that these may not necessarily catch all possible errors, so it's up to you to make sure you are handling any corner cases.
- For each word that is searched, you get 1 point for getting the correct path and 1 point for using an appropriate number of grid accesses. These are calculated for you.
- To avoid any differences in String output, use the *toString* method provided in *Loc.java* to print out the locations. Do not add anything in between them.

Provided Code.
Besides the test code, you are provided with the *Grid.java* class. You should NOT change this code as it will not be included in your submission. However, you may change the *Loc.java* class, and you should include your version of *Loc.java* in your submission even if you don't change it. You are also provided with the *Deque.java* and the *EmptyDequeException.java* class, which you are allowed to change as well. Or, if you prefer, you can use your own implementation from Project 1. You are not allowed to use other imported structures, but a Deque should be enough for your BFS and DFS implementations as it can function as both a Stack and a Queue.

API.
Implement your solution in a class called *Puzzle.java*. The required methods are listed below.

Method Signature	**Description**
`public Puzzle(Grid grid)`	`constructor`
`public String find(String word, int r, int c)`	`find and return (as a String) the path containing` *`word`*`; start the search at (r, c)`

Part 2. Shortest Path

In this part, you will implement an algorithm for finding the shortest path from one location in a grid to another location in the grid. Your algorithm should be a variation of Dijkstra's Algorithm. Note: In this part, all grids will contain non-negative integer values.

We define the shortest path from one location to another as the path with the smallest total sum. A step in a path can go UP, RIGHT, DOWN, or LEFT, and you should always search a location's neighbors in that order to ensure that you get the same path as the test cases. Your algorithm should follow the overall idea of Dijkstra's Algorithm.

Example

Consider the grid below.

	0	1	2	3	4
0	4	1	9	0	5
1	0	1	7	2	7
2	8	4	6	5	3
3	4	1	0	9	5
4	6	2	3	4	7

We will search for the shortest path from (0, 2) to (2, 1). To do this, I will use a grid to show the values of the current shortest distances and the previous values. I don't necessarily recommend this in your implementation, as it is possible to do this without the extra grid by updating the Loc class, but it helps as we visualize the algorithm.

- The starting values are below.

	0	1	2	3	4
0	d=∞ p=null	d=∞ p=null	d=9 p=null	d=∞ p=null	d=∞ p=null
1	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
2	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (0, 2) and update the neighbors.

	0	1	2	3	4
0	d=∞ p=null	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=∞ p=null
1	d=∞ p=null	d=∞ p=null	d=16 p=(0,2)	d=∞ p=null	d=∞ p=null
2	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (0, 3) and update the neighbors.

	0	1	2	3	4
0	d=∞ p=null	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=∞ p=null	d=∞ p=null	d=16 p=(0,2)	d=11 p=(1,2)	d=∞ p=null

2	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (0, 1) and update the neighbors.

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=∞ p=null	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=∞ p=null
2	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (1, 3) and update the neighbors.*

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=∞ p=null	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=18 p=(1,3)
2	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=16 p=(1,2)	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (1, 1) and update the neighbors.

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=11 p=(1,1)	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=18 p=(1,3)
2	d=∞ p=null	d=15 p=(1,1)	d=∞ p=null	d=16 p=(1,2)	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (1, 0) and update the neighbors.

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=11 p=(1,1)	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=18 p=(1,3)
2	d=19 p=(1,0)	d=15 p=(1,1)	d=∞ p=null	d=16 p=(1,2)	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (0, 4) and update the neighbors.

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=11 p=(1,1)	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=18 p=(1,3)
2	d=19 p=(1,0)	d=15 p=(1,1)	d=∞ p=null	d=16 p=(1,2)	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (0, 0) and update the neighbors.

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)
1	d=11 p=(1,1)	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=18 p=(1,3)
2	d=19 p=(1,0)	d=15 p=(1,1)	d=∞ p=null	d=16 p=(1,2)	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- Get the minimum distance (2, 1) and since this location is the target, we are done.

	0	1	2	3	4
0	d=14 p=(0,1)	d=10 p=(0,2)	d=9 p=null	d=9 p=(0,2)	d=14 p=(0,3)

1	d=11 p=(1,1)	d=11 p=(0,1)	d=16 p=(0,2)	d=11 p=(1,2)	d=18 p=(1,3)
2	d=19 p=(1,0)	d=15 p=(1,1)	d=∞ p=null	d=16 p=(1,2)	d=∞ p=null
3	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null
4	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null	d=∞ p=null

- The shortest path is: (0,2)(0,1)(1,1)(2,1).

*Note that there were two locations that were tied for the minimum at this step. This implementation chooses the one that was added to the queue first. The test cases use this implementation, so you should too.

Notes:
- Your output should be a string that represents the shortest path (similar to part 1).
- You are also tested based on access counts, so be sure to follow the guidelines.

Priority Queue Implementation.
You must implement your own priority queue to ensure that ties are handled according to what is expected. However, there are two options on this implementation, one of which can earn you some extra credit.
As discussed in class, a min-heap-based priority queue can do *insert* and *delMin* in O(logN) time. This is required for full credit. However, Dijkstra's Algorithm also requires you to be able to determine if something is in the priority queue and to update a (key, value) pair in the priority queue, and this is not typically O(logN) in a min heap. (Even Java's PriorityQueue implementation does not guarantee logarithmic runtime for these operations!). **If you implement your priority queue so that these two operations are also O(logN), you can earn up to 5 points of extra credit.** Otherwise, an implementation that does these things in O(N) time is sufficient.

Note: Please indicate that you are attempting the extra credit by adding a comment at the beginning of your ShortestPath.java file indicating that you believe your submission meets the EC requirements. If you do not do this, your submission may not be considered for the extra credit.

API.
Implement your solution in a class called *ShortestPath.java*. The required methods are listed below.

Method Signature	**Description**
`public ShortestPath(Grid grid)`	`constructor`
`public String getShortestPath(int sr, int sc, int tr, int tc)`	`find and return (as a String) the shortest path from (sr, sc) to (tr, tc) in the Grid`

Submission Procedure.

To submit your code, please upload the following files to **lectura** by using **turnin**. Once you log in to lectura, you can submit using the following command:

turnin csc345-spring21-p4 Puzzle.java Deque.java Loc.java ShortestPath.java

Upon successful submission, you will see this message:

```
Turning in:sed on access e
        Puzzle.java -- ok
        Deque.java-- ok
        Loc.java-- ok
        ShortestPath.java
All done.
```

Note: If your implementation includes other files, be sure to include those as well.
Note: Your code submission must be able to run on **lectura**, so make sure you give yourself enough time before the deadline to check that it runs and do any necessary debugging. I recommend finishing the project locally 24 hours before the deadline to make sure you have enough time to deal with any submission issues.

Grading.

The following rubric gives the basic breakdown of your grade for this Project.

Baseline

Item	**Max Points**

Part 1 Code	60
Part 2 Code	40 (+5 EC)
Total	**100 (+5 EC)**

Possible Deductions

Item	**Max Deduction Possible**
Bad Coding Style	5 points
Inefficient Solution	25% of total points for that part
Not following instructions	100% of the points you would have earned (i.e. automatic 0)
Not submitted correctly on lectura	100% of the points you would have earned (i.e. automatic 0)
Does not compile/run on lectura	100% of the points you would have earned (i.e. automatic 0)
Turned in 1 day late	10 points
Turned in 2 days late	10 points
Turned in more than 2 days late	100% of the points you would have earned (i.e. automatic 0)

Other notes about grading:

- If you do not follow the directions (e.g. importing classes without permission, not using an array for the Deque, etc.), you may not receive credit for that part of the assignment.
- Good Coding Style includes: using good indentation, using meaningful variable names, using informative comments, breaking out reusable functions instead of repeating them, etc.
- If you implement something correctly but in an inefficient way, you may not receive full credit.
- In cases where efficiency is determined by counting something like array accesses, it is considered academic dishonesty to *intentionally* try to avoid getting more access counts without actually improving the efficiency of the

solution, so if you are in doubt, it is always best to ask. If you are asking, then we tend to assume that you are trying to do the project correctly.

- If you have questions about your graded project, you may contact the TAs and set up a meeting to discuss your grade with them in person. Regrades on programs that do not work will only be allowed under limited circumstances, usually with a standard 20-point deduction. All regrade requests should be submitted according to the guidelines in the syllabus and within the allotted time frame.