1. **Write BFS and DFS for a graph**: What would be BFS and DFS traversal for the below graphs. Write the nodes for BFS and DFS. Start at node A.

A picture containing circle, line, screenshot

Description automatically generated

**BFS Traversal**

BFS traversal from node A is **{A, B, D, F, G, C, E }**

BFS (Breadth First Search) is a graph traversal algorithm that uses a queue data structure. It explores nodes in a level-by-level manner, visiting all adjacent nodes of the current level before moving to the next level.

To perform BFS traversal from node A:

Queue (Visited Nodes):

Result (BFS Traversal of all these nodes):



Queue: {**A**, B, D} #neighbors of A

**A picture containing circle, line, screenshot

Description automatically generated**

**Result: {A}**

Queue: {~~A~~, B, D} #done w/ A

A picture containing circle, clock, art

Description automatically generated {~~A~~, **B**, D} #now we go to B

**Result: {A, B}**

Queue: {A, **B**, D, F, G } #neighbors of B

{~~A~~, **B**, D, F, G } # But we are already covered A

A picture containing circle, clock, art

Description automatically generated



{~~A~~, ~~B~~, D, F, G } # Now we are done with node B

{~~A~~, ~~B~~, **D**, F, G} # So we go to the next element node D

A picture containing circle, clock

Description automatically generated

**Result: {A, B, D}**

Queue: {~~A~~, ~~B~~, **D**, F, G}

{~~A~~, ~~B~~, **D**, F, G C, E} #neighbors of D, but we already covered A & B

A picture containing circle

Description automatically generated

{~~A~~, ~~B~~, ~~D~~, **F**, G C, E} #now we’re done w/ D & next we go to node F

A picture containing circle, line, screenshot

Description automatically generated



**Result: {A, B, D, F}**

A picture containing circle, clock, design, art

Description automatically generatedQueue: {~~A~~, ~~B~~, ~~D~~, **F**, G C, E} #neighbors of F, but both already in Q



{~~A~~, ~~B~~, ~~D~~, ~~F~~, **G** C, E} #so are done w/ F & we go to G



A picture containing circle, line, screenshot

Description automatically generated



**Result: {A, B, D, F, G}**

Queue: {~~A~~, ~~B~~, ~~D~~, ~~F~~, **G** C, E} # G has no neighbors so we go we’re done w/ it

{~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~G~~ **C**, E}

A picture containing circle, line, screenshot

Description automatically generated



**Result: {A, B, D, F, G, C}**

**A picture containing circle, design

Description automatically generated**Queue: {~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~G~~ **C**, E} #C neighbors, but they’re covered/ already in queue

Queue: {~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~G~~ ~~C~~, **E**} #so we are done w C, move to next in queue: E

**Result: {A, B, D, F, G, C, E}**

Queue: {~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~G~~ ~~C~~, **E**}

A picture containing circle, line, screenshot

Description automatically generated



{~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~G~~ ~~C~~, **E**} # E has neighbors but they’ve been covered already

A picture containing circle, clock

Description automatically generated{~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~G~~ ~~C~~, ~~E~~} #so we are done and the graph is fully traversed.



**Result: {A, B, D, F, G, C, E}**

**DFS Traversal**

DFS traversal from node A is {**A, B, F, C, D, E, G}**

DFS means Depth First Search. Stack data structure is used to implement the DFS traversal. The Pushing order of nodes in stack is DFS traversal.

To perform DFS traversal from node A:

1. First push the node A

|  |
| --- |
| **A** |

2. From the node B and D are adjacent, according to alphabet /order push B to stack

|  |
| --- |
| **B** |
| **A** |

3. Now from B adjacent nodes are F and G, push F into stack

|  |
| --- |
| **F** |
| **B** |
| **A** |

Now, at this status, the DFS traversal is A B F

4. From F ,adjacent node is C, not yet entered or visited into stack. So that push C into stack

|  |
| --- |
| **C** |
| **F** |
| **B** |
| **A** |

5.From C adjacent nodes are D and E, according to order, Push D into stack

|  |
| --- |
| **D** |
| **C** |
| **F** |
| **B** |
| **A** |

6.From D, E is adjacent not yet visited into stack .push E into stack

|  |
| --- |
| **E** |
| **D** |
| **C** |
| **F** |
| **B** |
| **A** |

Now at this status, the DFS traversal is A B F C D E

7. Now from E , C and D are adjacent. But already visited into stack. So Pop the node E from stack

|  |
| --- |
| **D** |
| **C** |
| **F** |
| **B** |
| **A** |

from D also all adjacent nodes are visited into stack, So that pop D from stack

|  |
| --- |
| **C** |
| **F** |
| **B** |
| **A** |

from C also all adjacent nodes are visited into stack, So that pop C from stack

|  |
| --- |
| **F** |
| **B** |
| **A** |

from F also all adjacent nodes are visited into stack, So that pop F from stack

|  |
| --- |
| **B** |
| **A** |

8. From B there is a adjacent node G,which is not yet visited in stack.So that push the node G into stack

|  |
| --- |
| **G** |
| **B** |
| **A** |

Now at this staus, the DFS traversal is **A B F C D E G**

**9.** Pop G from stack

|  |
| --- |
| **B** |
| **A** |

pop B from stack

|  |
| --- |
| **A** |

Pop A from stack

|  |
| --- |
|  |

10. Pop all the nodes one by one , upto the stack is not empty.

DFS traversal from node A is **A B F C D E G**

**Finally,**

**BFS traversal from node A is --- A B D F G C E**

**DFS traversal from node A is ---- A B F C D E G**

**2. Apply BFS/DFS to solve a problem**

You are given a 3-D puzzle. The length and breadth of the puzzle is given by a 2D matrix puzzle[m][n]. The height of each cell is given by the value of each cell, the value of puzzle[row][column] give the height of the cell [row][column]. You are at [0][0] cell and you want to reach to the bottom right cell [m-1][n-1], the destination cell. You can move either up, down, left, or right. Write an algorithm to reach the destination cell with minimal effort.

How effort is defined: The effort of route is the maximum absolute difference between two consecutive cells.

If a route requires us to cross heights: 1, 3, 4, 6, 3, 1

The absolute differences between consecutive cells is: |1-3| = 2, |3-4|=1, |4-6|=2, |6-3|=3, |3-1|=2; this gives us the values: {2, 1, 2, 3, 2}. The maximum value of these absolute differences is 3. Hence the effort required on this path will be: 3.

Example:

Input: puzzle[][] = [[1, 3, 5], [2, 8, 3], [3, 4, 5]]

Output: 1

Explanation: The minimal effort route would be [1, 2, 3, 4, 5] which has an effort of value 1. This is better than other routes for instance, route [1, 3, 5, 3, 5] which has an effort of 2.

A picture containing screenshot, square, number

Description automatically generated

1. Implement the algorithm. Name your function **minEffort(puzzle)**; puzzle will be in the form of an 2D matrix as shown in the above example. Name your file **MinPuzzle.py**
2. b. What is the time complexity of your implementation?

The time complexity of this implementation is O(m \* n), where m is the number of rows and n is the number of columns in the puzzle. This is because we perform a BFS traversal, visiting each cell in the puzzle once. The worst-case scenario is visiting all cells in the puzzle.

3. **Analyze Dijkstra with negative edges**: Analyze with a sample graph and show why Dijkstra does not work with negative edges. Give the sample graph and write your explanation why Dijkstra would not work in this case.

**ANSWER:**

**Recall that in Dijkstra's algorithm, once a vertex is marked as "closed" (and out of the open set) - the algorithm found the shortest path to it, and will never have to develop this node again - it assumes the path developed to this path is the shortest.**

**But with negative weights - it might not be true. For example:**

A

/ \

/ \

/ \

5 2

/ \

B--(-10)-->C

V={A,B,C} ; E = {(A,C,2), (A,B,5), (B,C,-10)}

**Dijkstra from A will first develop C, and will later fail to find A->B->C**

**So starting out the values (*the distance from the source to the vertex*) initially assigned to each vertex are,**

**A= 0;
B= 20
C=0**

**We first extract the vertex in Q = [A,B,C] which has smallest value, i.e. A, after which Q = [B, C]. Note A has a directed edge to B and C, also both of them are in Q, therefore we update both of those values,**

**A=0; B= min(-0,5) = 5; C= min(-0,2) = 2**

**Now we extract C as (2<5), now Q = [B]. Note that C is connected to nothing, so line16 loop doesn't run.**

**A= 0;
0; B B = 5:
C= 2**

**4. (Extra Credit): What would be BFS and DFS traversal in below puzzle. Start at node A.**

A picture containing square, screenshot, number, text

Description automatically generated

BFS Traversal (starting from A):

1. Visit node A.
2. Visit the adjacent nodes of A in left-to-right and top-to-bottom order: B, C, and D.
3. Visit the adjacent nodes of B: E, F, and G.
4. Visit the adjacent nodes of C: H, I, and J.
5. Visit the adjacent nodes of D: empty cell (represented by '\_').
6. Visit the adjacent nodes of E: empty cell.
7. Visit the adjacent nodes of F: empty cell.
8. Visit the adjacent nodes of G: empty cell.
9. Visit the adjacent nodes of H: empty cell.
10. Visit the adjacent nodes of I: empty cell.
11. Visit the adjacent nodes of J: empty cell.

The BFS traversal starting from A is: A, B, C, D, E, F, G, H, I, J.

DFS Traversal (starting from A):

1. Visit node A.
2. Visit the adjacent nodes of A: B.
3. Visit the adjacent nodes of B: E.
4. Visit the adjacent nodes of E: F.
5. Visit the adjacent nodes of F: G.
6. Visit the adjacent nodes of G: C.
7. Visit the adjacent nodes of C: H.
8. Visit the adjacent nodes of H: I.
9. Visit the adjacent nodes of I: J.
10. Backtrack to the previous level: C.
11. Backtrack to the previous level: G.
12. Backtrack to the previous level: F.
13. Backtrack to the previous level: E.
14. Visit the next adjacent node of B: D.
15. Visit the adjacent nodes of D: empty cell.
16. Backtrack to the previous level: B.
17. Backtrack to the previous level: A.

The DFS traversal starting from A is: A, B, E, F, G, C, H, I, J, D.