6. Advanced Graphs Notes

LinkedIn: Md. Ziaul Karim

Find me here:

Topics:

#advanced-graphs #graphs #dijkstras_algorithm #heap #bfs #dfs #bellman-ford-algorithm #backtracking #DAG #topological-sort #prims_algorithm #minimum_spanning_tree #mst #directed-acyclic-graphs

Contents of Advanced Graph Algorithms

---

1. Reconstruct Itinerary

Link

1.1. Problem Statement

Hard #DFS #graphs #backtracking

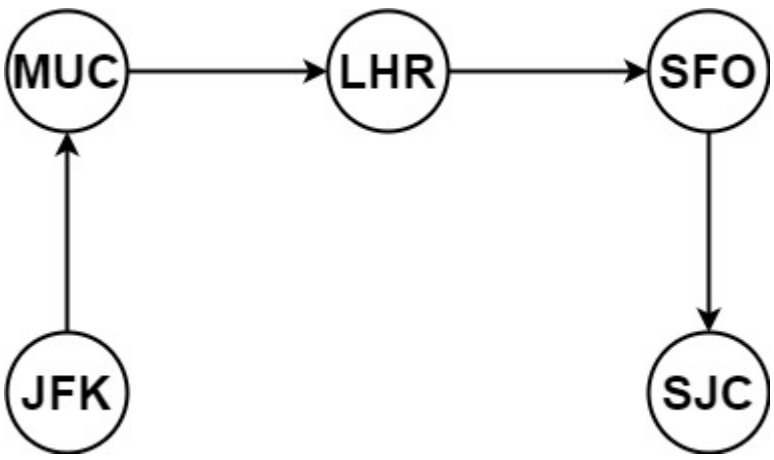
You are given a list of airline `tickets` where `tickets[i] = [fromi, toi]` represent the departure and the arrival airports of one flight. Reconstruct the itinerary in order and return it.

All of the tickets belong to a man who departs from `"JFK"`, thus, the itinerary must begin with `"JFK"`. If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string.

- For example, the itinerary `["JFK", "LGA"]` has a smaller lexical order than `["JFK", "LGB"]`.

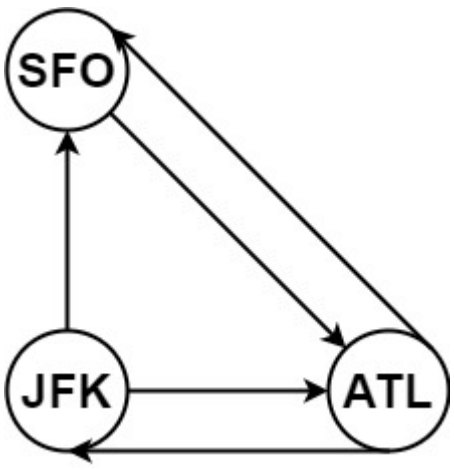
You may assume all tickets form at least one valid itinerary. You must use all the tickets once and only once.

Example 1:




```
Input: tickets = [["MUC","LHR"],["JFK","MUC"],["SFO","SJC"],["LHR","SFO"]]
Output: ["JFK","MUC","LHR","SFO","SJC"]
```

Example 2:




```
Input:
tickets = [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

Output:
["JFK","ATL","JFK","SFO","ATL","SFO"]
```

Explanation: Another possible reconstruction is `["JFK","SFO","ATL","JFK","ATL","SFO"]` but it is larger in lexical order.

Constraints:

- `1 <= tickets.length <= 300`
- `tickets[i].length == 2`
- `fromi.length == 3`
- `toi.length == 3`
- `fromi` and `toi` consist of uppercase English letters.
- `fromi != toi`

1.2. Code

```python
class Solution:
    def findItinerary(self, tickets):
        adj = {start: [] for start, destination in tickets}
        tickets.sort()
        for start, destination in tickets:
            adj[start].append(destination)
```

```
        res = ["JFK"]
        def dfs(start):
            if len(res) == len(tickets)+1:
                return True
            if start not in adj:
                return False
            temp = list(adj[start])
            for index, value in enumerate(temp):
                res.append(value)
                adj[start].pop(index)
                if dfs(value): return True
                res.pop()
                adj[start].insert(index, value)
            return False
        dfs("JFK")
        return res

if __name__ == '__main__':
    s = Solution()
    tickets = [["MUC","LHR"],["JFK","MUC"],["SFO","SJC"],["LHR","SFO"]]
    print("output: ",s.findItinerary(tickets))
```

1.3. Explanation

```
def findItirinary(self, [['MUC', 'LHR'], ['JFK', 'MUC'], ['SFO', 'SJC'], ['LHR', 'SFO']]):
        adj = {start: [] for start, destination in tickets} >>  adj = {'MUC': [], 'JFK': [], 'SFO': [], 'LHR': []}
        tickets.sort() >> tickets = [['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']]
        for 'JFK', 'MUC' in [['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']]: >>
adj['JFK'].append('MUC') >> {'MUC': [], 'JFK': ['MUC'], 'SFO': [], 'LHR': []}
        for 'LHR', 'SFO' in [['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']]: >>
adj['LHR'].append('SFO') >> {'MUC': [], 'JFK': ['MUC'], 'SFO': [], 'LHR': ['SFO']}
        for 'MUC', 'LHR' in [['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']]: >>
adj['MUC'].append('LHR') >> {'MUC': ['LHR'], 'JFK': ['MUC'], 'SFO': [], 'LHR': ['SFO']}
        for 'SFO', 'SJC' in [['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']]: >>
adj['SFO'].append('SJC') >> {'MUC': ['LHR'], 'JFK': ['MUC'], 'SFO': ['SJC'], 'LHR': ['SFO']}
        res = ["JFK"] >> res = ['JFK']
        dfs("JFK")
        def dfs({start}):
                if len(['JFK']) == len([['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']])+1: -->
False
                if JFK not in {'MUC': ['LHR'], 'JFK': ['MUC'], 'SFO': ['SJC'], 'LHR': ['SFO']}: --> False
                        temp = list(['MUC']) >> temp = ['MUC']
                        for (0, 'MUC') in enumerate(['MUC']):
                                ['JFK', 'MUC'].append(MUC) >> res = ['JFK', 'MUC']
                                adj['JFK'].pop(0) >> adj = {'MUC': ['LHR'], 'JFK': [], 'SFO': ['SJC'],
'LHR': ['SFO']}
                                if dfs('MUC'):
        #recursive block 1
        dfs('MUC'):
                if len(['JFK', 'MUC']) == len([['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO', 'SJC']])+1: -
-> False
                if 'MUC' not in {'MUC': ['LHR'], 'JFK': [], 'SFO': ['SJC'], 'LHR': ['SFO']}: --> False
                        temp = list(['LHR']) >> temp = ['LHR']
                        for (0, 'LHR') in enumerate(['LHR']):
                                ['JFK', 'MUC', 'LHR'].append('LHR') >> res = ['JFK', 'MUC', 'LHR']
                                adj['MUC'].pop(0) >> adj = {'MUC': [], 'JFK': [], 'SFO': ['SJC'], 'LHR':
['SFO']}
                                if dfs('LHR'):
        #recursive block 2
        dfs('LHR'):
                if len(['JFK', 'MUC', 'LHR']) == len([['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO',
'SJC']])+1: --> False
                if 'LHR' not in {'MUC': [], 'JFK': [], 'SFO': ['SJC'], 'LHR': ['SFO']}: --> False
```

```
                        temp = list(['SFO']) >> temp = ['SFO']
                                for (0, 'SFO') in enumerate(['SFO']):
                                        ['JFK', 'MUC', 'LHR', 'SFO'].append('SFO') >> res = ['JFK', 'MUC', 'LHR',
'SFO']
                                        adj['LHR'].pop(0) >> adj = {'MUC': [], 'JFK': [], 'SFO': ['SJC'], 'LHR': []}
                                        if dfs('SFO'):
        #recursive block 3
        dfs('SFO'):
                if len(['JFK', 'MUC', 'LHR', 'SFO']) == len([['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'], ['SFO',
'SJC']])+1: --> False
                if 'SFO' not in {'MUC': [], 'JFK': [], 'SFO': ['SJC'], 'LHR': []}: --> False
                        temp = list(['SJC']) >> temp = ['SJC']
                                for (0, 'SJC') in enumerate(['SJC']):
                                        ['JFK', 'MUC', 'LHR', 'SFO', 'SJC'].append('SJC') >> res = ['JFK', 'MUC',
'LHR', 'SFO', 'SJC']
                                        adj['SFO'].pop(0) >> adj = {'MUC': [], 'JFK': [], 'SFO': [], 'LHR': []}
                                        if dfs('SJC'):
        #recursive block 4
        dfs('SJC'):
                if len(['JFK', 'MUC', 'LHR', 'SFO', 'SJC']) == len([['JFK', 'MUC'], ['LHR', 'SFO'], ['MUC', 'LHR'],
['SFO', 'SJC']])+1: --> True
                if 'SJC' not in {'MUC': [], 'JFK': [], 'SFO': [], 'LHR': []}: --> True
                        return True
                        #Base Case
                                return True # recursive block 4
                                return True # recursive block 3
                                return True # recursive block 2
                                return True # recursive block 1
        return ['JFK', 'MUC', 'LHR', 'SFO', 'SJC']
```

2. Min Cost to Connect All Points

Link

2.1. Problem Statement

Medium #prims_algorithm #mst #minimum_spanning_tree #graphs #heap

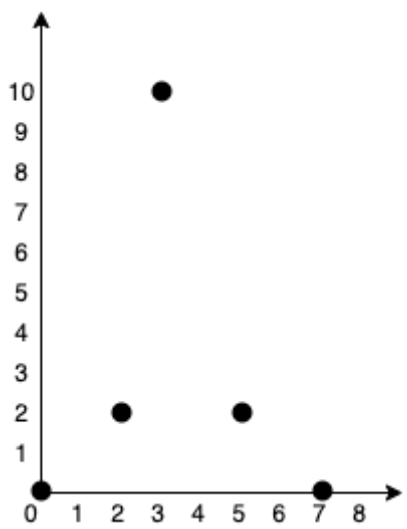
If you're not familiar with Prim's algorithm checkout: This Link

You are given an array `points` representing integer coordinates of some points on a 2D-plane, where `points[i] = [xi, yi]`.

The cost of connecting two points `[xi, yi]` and `[xj, yj]` is the **manhattan distance** between them: `|xi - xj| + |yi - yj|`, where `|val|` denotes the absolute value of `val`.

Return *the minimum cost to make all points connected.* All points are connected if there is **exactly one** simple path between any two points.

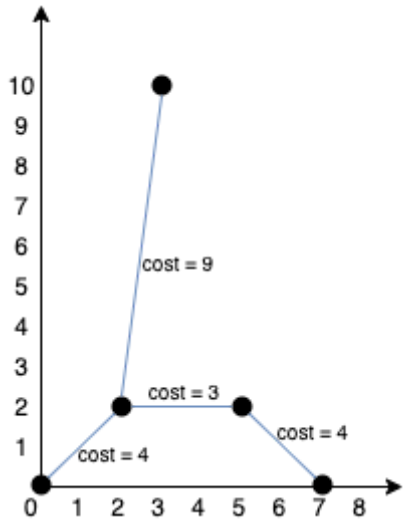
Example 1:



Input: points = `[[0,0],[2,2],[3,10],[5,2],[7,0]]`

Output: 20

Explanation:



We can connect the points as shown above to get the minimum cost of 20.
Notice that there is a unique path between every pair of points.

Example 2:

Input: points = `[[3,12],[-2,5],[-4,1]]`
Output: 18

Constraints:

- `1 <= points.length <= 1000`
- `-106 <= xi, yi <= 106`
- All pairs `(xi, yi)` are distinct.

2.2. Code

```
import heapq
class Solution:
    def minCostConnectPoints(self, points):
        N = len(points)
        adj = { i:[] for i in range(N) } # i : list of [cost, node]
        for i in range(N):
            x1, y1 = points[i]
            for j in range(i + 1, N):
                x2, y2 = points[j]
                dist = abs (x1 - x2) + abs(y1 - y2)
                adj[i].append([dist, j])
                adj[j].append([dist, i])
        # Prim's
        res = 0
        visit = set()
        minH = [[0, 0]] # [cost, point]
        while len(visit) < N:
            cost, i = heapq.heappop(minH)
            if i in visit:
                continue
            res += cost
            visit.add(i)
            for neiCost, nei in adj[i]:
                if nei not in visit:
                    heapq.heappush (minH, [neiCost, nei])
        return res
if __name__ == '__main__':
```

```
    s = Solution()
    points = [[0,0],[2,2],[3,10],[5,2],[7,0]]
    print("Output: ", s.minCostConnectPoints(points))
```

2.3. Explanation

```
Adjacency List:
adj = {
 "0": [[4, 1], [13, 2], [7, 3], [7, 4]],
 "1": [[4, 0], [9, 2], [3, 3], [7, 4]],
 "2": [[13, 0], [9, 1], [10, 3], [14, 4]],
 "3": [[7, 0], [3, 1], [10, 2], [4, 4]],
 "4": [[7, 0], [7, 1], [14, 2], [4, 3]]
 }

# Prim's

#0 < 5 --> True
Visited nodes: set()
Processing node 0 with cost 0
minH = []
minH = [[4, 1]]
minH = [[4, 1], [13, 2]]
minH = [[4, 1], [13, 2], [7, 3]]
minH = [[4, 1], [7, 4], [7, 3], [13, 2]]
Visited nodes: {0}
Current result: 0

#1 < 5 --> True
Visited nodes: {0}
Processing node 1 with cost 4
minH = [[7, 3], [7, 4], [13, 2]]
minH = [[7, 3], [7, 4], [13, 2], [9, 2]]
minH = [[3, 3], [7, 3], [13, 2], [9, 2], [7, 4]]
minH = [[3, 3], [7, 3], [7, 4], [9, 2], [7, 4], [13, 2]]
Visited nodes: {0, 1}
Current result: 4

#2 < 5 --> True
Visited nodes: {0, 1}
Processing node 3 with cost 3
minH = [[7, 3], [7, 4], [7, 4], [9, 2], [13, 2]]
minH = [[7, 3], [7, 4], [7, 4], [9, 2], [13, 2], [10, 2]]
minH = [[4, 4], [7, 4], [7, 3], [9, 2], [13, 2], [10, 2], [7, 4]]
Visited nodes: {0, 1, 3}
Current result: 7

#3 < 5 --> True
Visited nodes: {0, 1, 3}
Processing node 4 with cost 4
minH = [[7, 3], [7, 4], [7, 4], [9, 2], [13, 2], [10, 2]]
minH = [[7, 3], [7, 4], [7, 4], [9, 2], [13, 2], [10, 2], [14, 2]]
Visited nodes: {0, 1, 3, 4}
Current result: 11

#4 < 5 --> True
Visited nodes: {0, 1, 3, 4}
Processing node 3 with cost 7
minH = [[7, 4], [7, 4], [10, 2], [9, 2], [13, 2], [14, 2]]
3 is visited
continue

#4 < 5 --> True
Visited nodes: {0, 1, 3, 4}
```

```
Processing node 4 with cost 7
minH = [[7, 4], [9, 2], [10, 2], [14, 2], [13, 2]]
4 is visited
continue

#4 < 5 --> True
Visited nodes: {0, 1, 3, 4}
Processing node 4 with cost 7
minH = [[9, 2], [13, 2], [10, 2], [14, 2]]
4 is visited
continue

#4 < 5 --> True
Visited nodes: {0, 1, 3, 4}
Processing node 2 with cost 9
minH = [[10, 2], [13, 2], [14, 2]]
Visited nodes: {0, 1, 2, 3, 4}
Current result: 20

# 5 < 5 --> False
break loop

Final result: 20
Output: 20
```

3. Network Delay Time

Link

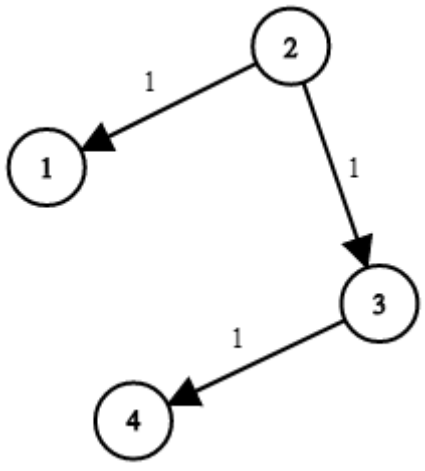
3.1. Problem Statement

Medium #dijkstras_algorithm #bfs #heap #graphs

You are given a network of `n` nodes, labeled from `1` to `n`. You are also given `times`, a list of travel times as directed edges `times[i] = (ui, vi, wi)`, where `ui` is the source node, `vi` is the target node, and `wi` is the time it takes for a signal to travel from source to target.

We will send a signal from a given node `k`. Return *the **minimum** time it takes for all the* `n` *nodes to receive the signal*. If it is impossible for all the `n` nodes to receive the signal, return `-1`.

Example 1:



Input: `times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2`
Output: 2

Example 2:

Input: `times = [[1,2,1]], n = 2, k = 1`
Output: 1

Example 3:

Input: `times = [[1,2,1]], n = 2, k = 2`
Output: -1

Constraints:

- `1 <= k <= n <= 100`
- `1 <= times.length <= 6000`
- `times[i].length == 3`
- `1 <= ui, vi <= n`
- `ui != vi`
- `0 <= wi <= 100`
- All the pairs `(ui, vi)` are **unique**. (i.e., no multiple edges.)

3.2. Code

```
import heapq
import collections
class Solution:
    def networkDelayTime (self, times, n, k):
        edges = collections.defaultdict(list)
        for u, v, w in times:
            edges [u].append((v, w))
        minHeap = [(0, k)]
        visit = set()
        t = 0
        while minHeap:
            w1, n1 = heapq.heappop(minHeap)
            if n1 in visit:
                continue
            visit.add(n1)
            t = max(t, w1)
            for n2, w2 in edges [n1]:
                if n2 not in visit:
                    heapq.heappush(minHeap, (w1 + w2, n2))
        return t if len(visit) == n else -1

if __name__ == '__main__':
    s = Solution()
    times = [[2,1,1],[2,3,1],[3,4,1]]
    n = 4 # total visited node
    k = 2 # start node
    print("Output: ", s.networkDelayTime(times, n, k))
```

3.3. Explanation

```
def networkDelayTime(self, times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]], n = 4, k = 2):
        edges = collections.defaultdict(list)
        >> edges = defaultdict(<class 'list'>, {})

        for (2, 1, 1)in [[2, 1, 1], [2, 3, 1], [3, 4, 1]]:
        >> edges[2].append(((1, 1)))
        >> edges = defaultdict(<class 'list'>, {2: [(1, 1)]})

        for (2, 3, 1)in [[2, 1, 1], [2, 3, 1], [3, 4, 1]]:
        >> edges[2].append(((3, 1)))
        >> edges = defaultdict(<class 'list'>, {2: [(1, 1), (3, 1)]})

        for (3, 4, 1)in [[2, 1, 1], [2, 3, 1], [3, 4, 1]]:
        >> edges[3].append(((4, 1)))
        >> edges = defaultdict(<class 'list'>, {2: [(1, 1), (3, 1)], 3: [(4, 1)]})
```

```
minHeap = [(0, 2)] >> minHeap = [(0, 2)]
visit = set() >> visit = {}
t=0

while [(0, 2)]:
        w1, n1 = heapq.heappop([]) >> w1 = 0, n1 = 2
        if 2 in set(): --> False
        {2}.add(2) >> visit = {2}
        t = max(0,0) >> t = 0
        for (1, 1) in edges[2]: >> if 1 not in {2}: --> True
                heapq.heappush([], (0 + 1, 1)) >> minHeap = [(1, 1)]

        for (3, 1) in edges[2]: >> if 3 not in {2}: --> True
                heapq.heappush([(1, 1)], (0 + 1, 3)) >> minHeap = [(1, 1), (1, 3)]

        Visited node 2 with time 0
        Current visited nodes: {2}
        Current maximum time: 0
        Current minHeap: [(1, 1), (1, 3)]

while [(1, 1), (1, 3)]:
        w1, n1 = heapq.heappop([(1, 3)]) >> w1 = 1, n1 = 1
        if 1 in {2}: --> False
        {1, 2}.add(1) >> visit = {1, 2}
        t = max(0,1) >> t = 1
        Visited node 1 with time 1
        Current visited nodes: {1, 2}
        Current maximum time: 1
        Current minHeap: [(1, 3)]

while [(1, 3)]:
        w1, n1 = heapq.heappop([]) >> w1 = 1, n1 = 3
        if 3 in {1, 2}: --> False
        {1, 2, 3}.add(3) >> visit = {1, 2, 3}
        t = max(1,1) >> t = 1
        for (4, 1) in edges[3]: >> if 4 not in {1, 2, 3}: --> True
                heapq.heappush([], (1 + 1, 4)) >> minHeap = [(2, 4)]

        Visited node 3 with time 1
        Current visited nodes: {1, 2, 3}
        Current maximum time: 1
        Current minHeap: [(2, 4)]

while [(2, 4)]:
        w1, n1 = heapq.heappop([]) >> w1 = 2, n1 = 4
        if 4 in {1, 2, 3}: --> False
        {1, 2, 3, 4}.add(4) >> visit = {1, 2, 3, 4}
        t = max(1,2) >> t = 2
        Visited node 4 with time 2
        Current visited nodes: {1, 2, 3, 4}
        Current maximum time: 2
        Current minHeap: []
        #End of while loop
    return 2 if len({1, 2, 3, 4}) == 4 else -1 --> return 2
Output: 2
```

4. Swim In Rising Water

Link

4.1. Problem Statement

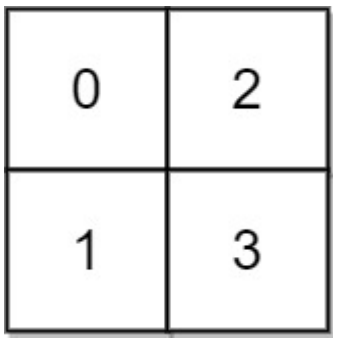
Hard #dijkstras_algorithm #heap #bfs

You are given an `n x n` integer matrix `grid` where each value `grid[i][j]` represents the elevation at that point `(i, j)`.

The rain starts to fall. At time `t`, the depth of the water everywhere is `t`. You can swim from a square to another 4-directionally adjacent square if and only if the elevation of both squares individually are at most `t`. You can swim infinite distances in zero time. Of course, you must stay within the boundaries of the grid during your swim.

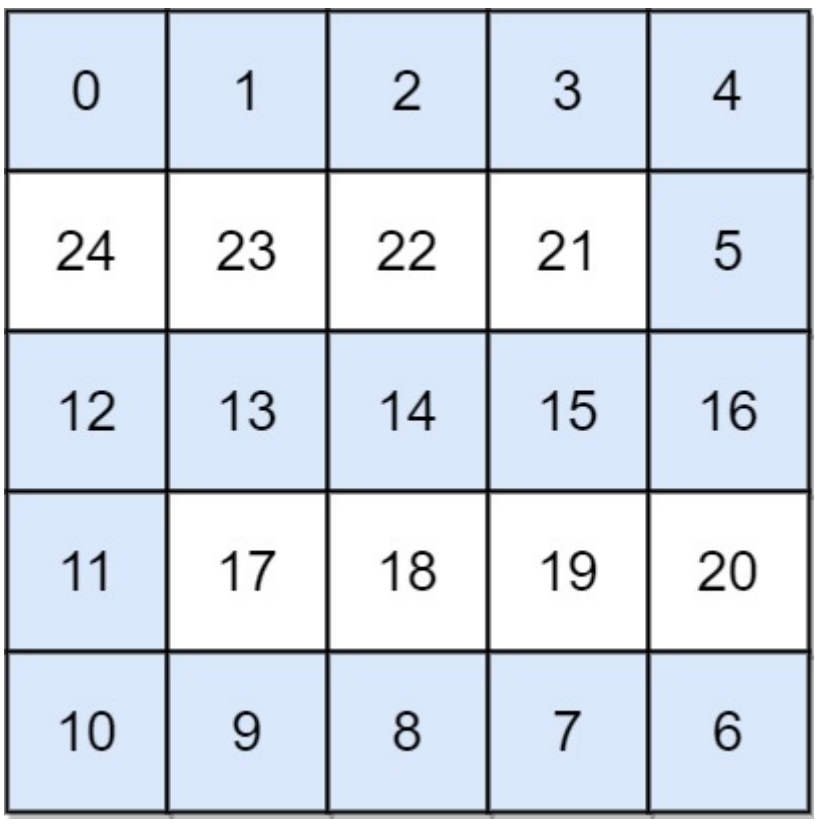
Return *the least time until you can reach the bottom right square* `(n - 1, n - 1)` *if you start at the top left square* `(0, 0)`.

Example 1:



Input: grid = `[[0,2],[1,3]]`
Output: 3
Explanation:
At time 0, you are in grid location (0, 0).
You cannot go anywhere else because 4-directionally adjacent neighbors have a higher elevation than t = 0.
You cannot reach point (1, 1) until time 3.
When the depth of water is 3, we can swim anywhere inside the grid.

Example 2:



Input: grid = `[[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,6]]`

Output: 16
Explanation: The final route is shown.
We need to wait until time 16 so that (0, 0) and (4, 4) are connected.

Constraints:

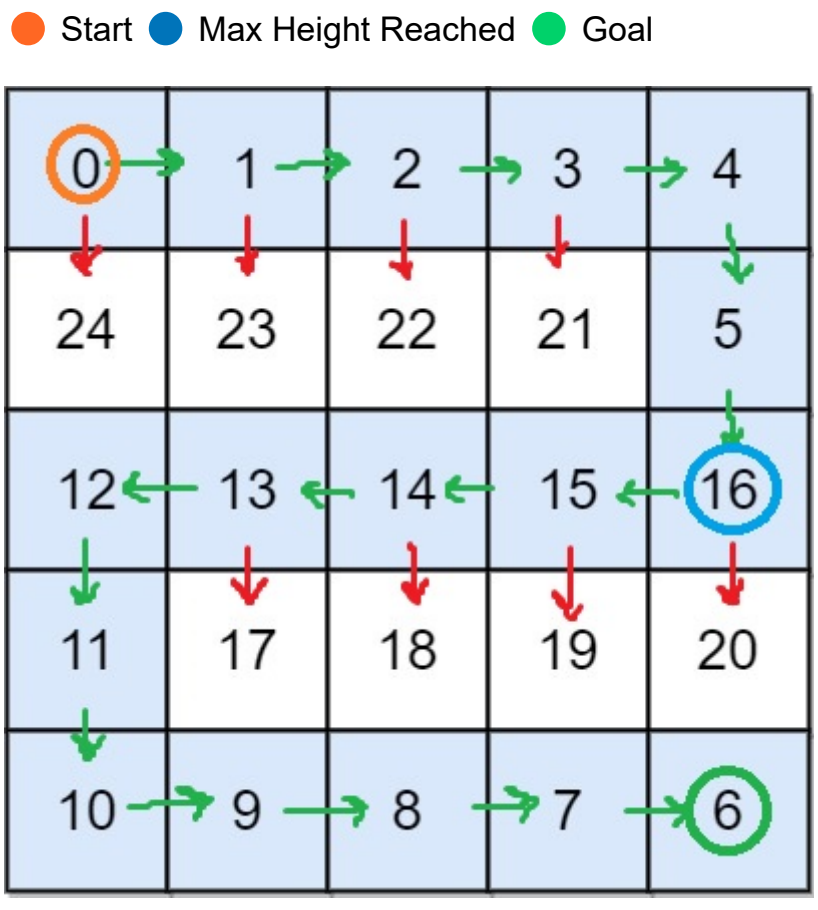
- `n == grid.length`
- `n == grid[i].length`
- `1 <= n <= 50`
- `0 <= grid[i][j] < n2`
- Each value `grid[i][j]` is **unique**.

4.2. Code

```
import heapq
class Solution:
    def swimInWater(self, grid):
        N = len(grid)
        visit = set()
        minH = [[grid[0][0], 0, 0]] # (time/max-height, r, c)
        directions = [[0, 1], #up
                      [0, -1], #down
                      [1, 0], #right
                      [-1,0] #left
                      ]
        visit.add((0, 0))
        while minH:
            t, r, c = heapq.heappop (minH)
            if r == N - 1 and c == N - 1:
                return t
            for dr, dc in directions:
                neiR, neiC = r + dr, c + dc
                if (neiR < 0 or neiC < 0 or neiR == N or neiC == N or (neiR, neiC) in visit):
                    continue
                visit.add((neiR, neiC))
                heapq.heappush(minH, [max(t, grid[neiR] [neiC]), neiR, neiC])

if __name__ == '__main__':
    s = Solution()
    grid = [[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,6]]
    print("Output: ", s.swimInWater(grid))
```

4.3. Explanation



Green arrow depicts the path in favor, red arrow depicts the path avoided.

```
def swimInWater(self, grid):
    N = len(grid) >> N = 5
    visit = set() >> visit = {}
    minH = [[grid[0][0], 0, 0]] >> minH = [[0, 0, 0]]
    directions = [
        [0, 1], #up
        [0, -1], #down
```

```
        [1, 0], #right
        [-1,0] #left
        ]
    visit.add((0, 0)) >> visit = {(0, 0)}
    while [[0, 0, 0]]:
        t, r, c = heapq.heappop ([])
        >> t = 0, r = 0, c = 0, minH = []

        if 0 == 5 - 1 and 0 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 0 + 0, 0 + 1 >> neiR = 0, neiC = 1,
            #visit = {(0, 0)}
            if (0 < 0 or 1 < 0 or 0 == 5 or 1 == 5 or (0, 1) in visit): --> False
            visit.add(((0, 1)))
            >> visit = {(0, 1), (0, 0)}
            heapq.heappush(minH, [max(0, grid[0] [1]), 0, 1])
            >> minH = [[1, 0, 1]]

        for 0, -1 in directions:
            neiR, neiC = 0 + 0, 0 + -1 >> neiR = 0, neiC = -1,
            #visit = {(0, 1), (0, 0)}
            if (0 < 0 or -1 < 0 or 0 == 5 or -1 == 5 or (0, -1) in visit): --> True
                continue

        for 1, 0 in directions:

            neiR, neiC = 0 + 1, 0 + 0
            >> neiR = 1, neiC = 0,
            #visit = {(0, 1), (0, 0)}
            if (1 < 0 or 0 < 0 or 1 == 5 or 0 == 5 or (1, 0) in visit): --> False
            visit.add(((1, 0)))
            >> visit = {(0, 1), (1, 0), (0, 0)}
            heapq.heappush(minH, [max(0, grid[1] [0]), 1, 0])
            >> minH = [[1, 0, 1], [24, 1, 0]]

        for -1, 0 in directions:

            neiR, neiC = 0 + -1, 0 + 0 >> neiR = -1, neiC = 0,

            #visit = {(0, 1), (1, 0), (0, 0)}
            if (-1 < 0 or 0 < 0 or -1 == 5 or 0 == 5 or (-1, 0) in visit): --> True
                continue

================================================================================

    while [[1, 0, 1], [24, 1, 0]]:
        t, r, c = heapq.heappop ([[24, 1, 0]])
        >> t = 1, r = 0, c = 1, minH = [[24, 1, 0]]

        if 0 == 5 - 1 and 1 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 0 + 0, 1 + 1 >> neiR = 0, neiC = 2,
            #visit = {(0, 1), (1, 0), (0, 0)}
            if (0 < 0 or 2 < 0 or 0 == 5 or 2 == 5 or (0, 2) in visit): --> False
            visit.add(((0, 2)))
            >> visit = {(0, 1), (1, 0), (0, 2), (0, 0)}
            heapq.heappush(minH, [max(1, grid[0] [2]), 0, 2])
            >> minH = [[2, 0, 2], [24, 1, 0]]

        for 0, -1 in directions:
            neiR, neiC = 0 + 0, 1 + -1 >> neiR = 0, neiC = 0,
            #visit = {(0, 1), (1, 0), (0, 2), (0, 0)}
            if (0 < 0 or 0 < 0 or 0 == 5 or 0 == 5 or (0, 0) in visit): --> True
                continue
```

```
        for 1, 0 in directions:
            neiR, neiC = 0 + 1, 1 + 0
            >> neiR = 1, neiC = 1,
            #visit = {(0, 1), (1, 0), (0, 2), (0, 0)}
            if (1 < 0 or 1 < 0 or 1 == 5 or 1 == 5 or (1, 1) in visit): --> False
            visit.add(((1, 1)))
            >> visit = {(0, 1), (0, 0), (1, 1), (0, 2), (1, 0)}
            heapq.heappush(minH, [max(1, grid[1] [1]), 1, 1])
            >> minH = [[2, 0, 2], [24, 1, 0], [23, 1, 1]]

        for -1, 0 in directions:
            neiR, neiC = 0 + -1, 1 + 0 >> neiR = -1, neiC = 1,
            #visit = {(0, 1), (0, 0), (1, 1), (0, 2), (1, 0)}
            if (-1 < 0 or 1 < 0 or -1 == 5 or 1 == 5 or (-1, 1) in visit): --> True
                continue

==========================================================================================
    while [[2, 0, 2], [24, 1, 0], [23, 1, 1]]:
        t, r, c = heapq.heappop ([[23, 1, 1], [24, 1, 0]])
        >> t = 2, r = 0, c = 2, minH = [[23, 1, 1], [24, 1, 0]]
        if 0 == 5 - 1 and 2 == 5 - 1: --> False

    for 0, 1 in directions:
        neiR, neiC = 0 + 0, 2 + 1
        >> neiR = 0, neiC = 3,
        #visit = {(0, 1), (0, 0), (1, 1), (0, 2), (1, 0)}
        if (0 < 0 or 3 < 0 or 0 == 5 or 3 == 5 or (0, 3) in visit): --> False
        visit.add(((0, 3)))
        >> visit = {(0, 1), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
        heapq.heappush(minH, [max(2, grid[0] [3]), 0, 3])
        >> minH = [[3, 0, 3], [24, 1, 0], [23, 1, 1]]

    for 0, -1 in directions:
        neiR, neiC = 0 + 0, 2 + -1 >> neiR = 0, neiC = 1,
        #visit = {(0, 1), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
        if (0 < 0 or 1 < 0 or 0 == 5 or 1 == 5 or (0, 1) in visit): --> True
            continue

    for 1, 0 in directions:
        neiR, neiC = 0 + 1, 2 + 0
        >> neiR = 1, neiC = 2,
        #visit = {(0, 1), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
        if (1 < 0 or 2 < 0 or 1 == 5 or 2 == 5 or (1, 2) in visit): --> False
        visit.add(((1, 2)))
        >> visit = {(0, 1), (1, 2), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
        heapq.heappush(minH, [max(2, grid[1] [2]), 1, 2])
            >> minH = [[3, 0, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]]

        for -1, 0 in directions:
            neiR, neiC = 0 + -1, 2 + 0 >> neiR = -1, neiC = 2,
            #visit = {(0, 1), (1, 2), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
            if (-1 < 0 or 2 < 0 or -1 == 5 or 2 == 5 or (-1, 2) in visit): --> True
                continue

==========================================================================================
    while [[3, 0, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]]:
        t, r, c = heapq.heappop ([[22, 1, 2], [24, 1, 0], [23, 1, 1]])
        >> t = 3, r = 0, c = 3, minH = [[22, 1, 2], [24, 1, 0], [23, 1, 1]]
        if 0 == 5 - 1 and 3 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 0 + 0, 3 + 1
            >> neiR = 0, neiC = 4,
            #visit = {(0, 1), (1, 2), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
            if (0 < 0 or 4 < 0 or 0 == 5 or 4 == 5 or (0, 4) in visit): --> False
            visit.add(((0, 4)))
            >> visit = {
```

```
            >> (0, 1), (1, 2),
            >> (0, 4), (0, 0),
            >> (1, 1), (0, 3),
            >> (0, 2), (1, 0)
            >> }
            heapq.heappush(minH, [max(3, grid[0] [4]), 0, 4])
            >> minH = [
            >> [4, 0, 4], [22, 1, 2],
            >> [23, 1, 1], [24, 1, 0]
            >> ]

        for 0, -1 in directions:
            neiR, neiC = 0 + 0, 3 + -1
            >> neiR = 0, neiC = 2,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
            if (0 < 0 or 2 < 0 or 0 == 5 or 2 == 5 or (0, 2) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 0 + 1, 3 + 0
            >> neiR = 1, neiC = 3,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0)}
            if (1 < 0 or 3 < 0 or 1 == 5 or 3 == 5 or (1, 3) in visit): --> False
            visit.add(((1, 3)))
            >> visit = {
            >> (0, 1), (1, 2), (0, 4),
            >> (0, 0), (1, 1), (0, 3),
            >> (0, 2), (1, 0), (1, 3)
            >> }
            heapq.heappush(minH, [max(3, grid[1] [3]), 1, 3])
            >> minH = [[4, 0, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]

        for -1, 0 in directions:
            neiR, neiC = 0 + -1, 3 + 0
            >> neiR = -1, neiC = 3,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0), (1, 3)}
            if (-1 < 0 or 3 < 0 or -1 == 5 or 3 == 5 or (-1, 3) in visit): --> True
                continue

================================================================================
    while [[4, 0, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]:
        t, r, c = heapq.heappop ([[21, 1, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]])
        >> t = 4, r = 0, c = 4,
        >> minH = [
        >> [21, 1, 3], [22, 1, 2],
        >> [23, 1, 1], [24, 1, 0]
        >> ]
        if 0 == 5 - 1 and 4 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 0 + 0, 4 + 1
            >> neiR = 0, neiC = 5,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0), (1, 3)}
            if (0 < 0 or 5 < 0 or 0 == 5 or 5 == 5 or (0, 5) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 0 + 0, 4 + -1
            >> neiR = 0, neiC = 3,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0), (1, 3)}
            if (0 < 0 or 3 < 0 or 0 == 5 or 3 == 5 or (0, 3) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 0 + 1, 4 + 0
            >> neiR = 1, neiC = 4,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (0, 2), (1, 0), (1, 3)}
```

```
            if (1 < 0 or 4 < 0 or 1 == 5 or 4 == 5 or (1, 4) in visit): --> False
            visit.add(((1, 4)))
            >> visit = {
            >> (0, 1), (1, 2), (0, 4),
            >> (0, 0), (1, 1), (0, 3),
            >> (1, 4), (0, 2), (1, 0),
            >> (1, 3)
            >> }
            heapq.heappush(minH, [max(4, grid[1] [4]), 1, 4])
            >> minH = [[5, 1, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]

        for -1, 0 in directions:
            neiR, neiC = 0 + -1, 4 + 0 >> neiR = -1, neiC = 4,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (-1 < 0 or 4 < 0 or -1 == 5 or 4 == 5 or (-1, 4) in visit): --> True
                continue

================================================================================
    while [[5, 1, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]:
        t, r, c = heapq.heappop ([[21, 1, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]])
        >> t = 5, r = 1, c = 4,
        >> minH = [[21, 1, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]]
        if 1 == 5 - 1 and 4 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 1 + 0, 4 + 1
            >> neiR = 1, neiC = 5,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (1 < 0 or 5 < 0 or 1 == 5 or 5 == 5 or (1, 5) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 1 + 0, 4 + -1
            >> neiR = 1, neiC = 3,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (1 < 0 or 3 < 0 or 1 == 5 or 3 == 5 or (1, 3) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 1 + 1, 4 + 0
            >> neiR = 2, neiC = 4,
#visit = {(0, 1), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (2 < 0 or 4 < 0 or 2 == 5 or 4 == 5 or (2, 4) in visit): --> False
            visit.add(((2, 4)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (0, 0), (1, 1),
            >> (0, 3), (1, 4), (0, 2),
            >> (1, 0), (1, 3)
            >> }
            heapq.heappush(minH, [max(5, grid[2] [4]), 2, 4])
            >> minH = [[16, 2, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]

        for -1, 0 in directions:
            neiR, neiC = 1 + -1, 4 + 0
            >> neiR = 0, neiC = 4,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (0 < 0 or 4 < 0 or 0 == 5 or 4 == 5 or (0, 4) in visit): --> True
                continue

================================================================================
    while [[16, 2, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]:
        t, r, c = heapq.heappop ([[21, 1, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]])
        >> t = 16, r = 2, c = 4,
        >> minH = [[21, 1, 3], [22, 1, 2], [23, 1, 1], [24, 1, 0]]
        if 2 == 5 - 1 and 4 == 5 - 1: --> False
        for 0, 1 in directions:
```

```
            neiR, neiC = 2 + 0, 4 + 1
            >> neiR = 2, neiC = 5,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (2 < 0 or 5 < 0 or 2 == 5 or 5 == 5 or (2, 5) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 2 + 0, 4 + -1
            >> neiR = 2, neiC = 3,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (0, 2), (1, 0), (1, 3)}
            if (2 < 0 or 3 < 0 or 2 == 5 or 3 == 5 or (2, 3) in visit): --> False
            visit.add(((2, 3)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (0, 0), (1, 1),
            >> (0, 3), (1, 4), (2, 3),
            >> (0, 2), (1, 0), (1, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[2] [3]), 2, 3])
            >> minH = [[16, 2, 3], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]

        for 1, 0 in directions:
            neiR, neiC = 2 + 1, 4 + 0 >> neiR = 3, neiC = 4,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (1, 0), (1, 3)}
            if (3 < 0 or 4 < 0 or 3 == 5 or 4 == 5 or (3, 4) in visit): --> False
            visit.add(((3, 4)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (3, 4), (0, 0), (1, 1),
            >> (0, 3), (1, 4), (2, 3),
            >> (0, 2), (1, 0), (1, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[3] [4]), 3, 4])
            >> minH = [
            >> [16, 2, 3], [21, 1, 3],
            >> [20, 3, 4], [24, 1, 0],
            >> [22, 1, 2], [23, 1, 1]
            >> ]

        for -1, 0 in directions:
            neiR, neiC = 2 + -1, 4 + 0 >> neiR = 1, neiC = 4,

            #visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (1,
0), (1, 3)}
            if (1 < 0 or 4 < 0 or 1 == 5 or 4 == 5 or (1, 4) in visit): --> True
                continue


================================================================================

    while [[16, 2, 3], [21, 1, 3], [20, 3, 4], [24, 1, 0], [22, 1, 2], [23, 1, 1]]:
        t, r, c = heapq.heappop ([
        [20, 3, 4], [21, 1, 3],
        [23, 1, 1], [24, 1, 0], [22, 1, 2]
        ])
        >> t = 16, r = 2, c = 3,
        >> minH = [[20, 3, 4], [21, 1, 3], [23, 1, 1], [24, 1, 0], [22, 1, 2]]
        if 2 == 5 - 1 and 3 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 2 + 0, 3 + 1
            >> neiR = 2, neiC = 4,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (1, 0), (1, 3)}
            if (2 < 0 or 4 < 0 or 2 == 5 or 4 == 5 or (2, 4) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 2 + 0, 3 + -1
```

```
            >> neiR = 2, neiC = 2,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (1, 0), (1, 3)}
            if (2 < 0 or 2 < 0 or 2 == 5 or 2 == 5 or (2, 2) in visit): --> False
            visit.add(((2, 2)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (3, 4), (0, 0),
            >> (1, 1), (0, 3), (1, 4),
            >> (2, 3), (0, 2), (2, 2),
            >> (1, 0), (1, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[2] [2]), 2, 2])
            >> minH = [
            >> [16, 2, 2], [21, 1, 3],
            >> [20, 3, 4], [24, 1, 0],
            >> [22, 1, 2], [23, 1, 1]
            >> ]

        for 1, 0 in directions:
            neiR, neiC = 2 + 1, 3 + 0
            >> neiR = 3, neiC = 3,
            #visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (2,
2), (1, 0), (1, 3)}
            if (3 < 0 or 3 < 0 or 3 == 5 or 3 == 5 or (3, 3) in visit): --> False
            visit.add(((3, 3)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (3, 4), (0, 0),
            >> (1, 1), (0, 3), (1, 4),
            >> (2, 3), (0, 2), (3, 3),
            >> (2, 2), (1, 0), (1, 3)
            >> }

            heapq.heappush(minH, [max(16, grid[3] [3]), 3, 3])
            >> minH = [
            >> [16, 2, 2], [21, 1, 3],
            >> [19, 3, 3], [24, 1, 0],
            >> [22, 1, 2], [23, 1, 1],
            >> [20, 3, 4]
            >> ]

        for -1, 0 in directions:
            neiR, neiC = 2 + -1, 3 + 0
            >> neiR = 1, neiC = 3,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3), (2, 2),
(1, 0), (1, 3)}
            if (1 < 0 or 3 < 0 or 1 == 5 or 3 == 5 or (1, 3) in visit): --> True
                continue

================================================================================
    while [
    [16, 2, 2], [21, 1, 3],
    [19, 3, 3], [24, 1, 0], [22, 1, 2],
    [23, 1, 1], [20, 3, 4]
    ]:
        t, r, c = heapq.heappop ([
        [19, 3, 3], [21, 1, 3],
        [20, 3, 4], [24, 1, 0],
        [22, 1, 2], [23, 1, 1]
        ])
        >> t = 16, r = 2, c = 2,
        >> minH = [
        >> [19, 3, 3], [21, 1, 3],
        >> [20, 3, 4], [24, 1, 0],
        >> [22, 1, 2], [23, 1, 1]
        >> ]
        if 2 == 5 - 1 and 2 == 5 - 1: --> False
```

```
        for 0, 1 in directions:
            neiR, neiC = 2 + 0, 2 + 1
            >> neiR = 2, neiC = 3,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3), (2, 2),
(1, 0), (1, 3)}
            if (2 < 0 or 3 < 0 or 2 == 5 or 3 == 5 or (2, 3) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 2 + 0, 2 + -1
            >> neiR = 2, neiC = 1,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3), (2, 2),
(1, 0), (1, 3)}
            if (2 < 0 or 1 < 0 or 2 == 5 or 1 == 5 or (2, 1) in visit): --> False
            visit.add(((2, 1)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (3, 4), (2, 1),
            >> (0, 0), (1, 1), (0, 3),
            >> (1, 4), (2, 3), (0, 2),
            >> (3, 3), (2, 2), (1, 0),
            >> (1, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[2] [1]), 2, 1])
            >> minH = [
            >> [16, 2, 1], [21, 1, 3],
            >> [19, 3, 3], [24, 1, 0],
            >> [22, 1, 2], [23, 1, 1],
            >> [20, 3, 4]
            >> ]

        for 1, 0 in directions:
            neiR, neiC = 2 + 1, 2 + 0 >> neiR = 3, neiC = 2,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (2, 1), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3),
(2, 2), (1, 0), (1, 3)}
            if (3 < 0 or 2 < 0 or 3 == 5 or 2 == 5 or (3, 2) in visit): --> False
            visit.add(((3, 2)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2),
            >> (0, 4), (3, 4), (2, 1),
            >> (0, 0), (1, 1), (0, 3),
            >> (1, 4), (2, 3), (0, 2),
            >> (3, 3), (2, 2), (1, 0),
            >> (3, 2), (1, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[3] [2]), 3, 2])
            >> minH = [
            >> [16, 2, 1], [18, 3, 2],
            >> [19, 3, 3], [21, 1, 3],
            >> [22, 1, 2], [23, 1, 1],
            >> [20, 3, 4], [24, 1, 0]
            >> ]

        for -1, 0 in directions:
            neiR, neiC = 2 + -1, 2 + 0
            >> neiR = 1, neiC = 2,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (2, 1), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3),
(2, 2), (1, 0), (3, 2), (1, 3)}
            if (1 < 0 or 2 < 0 or 1 == 5 or 2 == 5 or (1, 2) in visit): --> True
                continue

================================================================================
    while [
    [16, 2, 1], [18, 3, 2],
    [19, 3, 3], [21, 1, 3],
    [22, 1, 2], [23, 1, 1],
```

```
    [20, 3, 4], [24, 1, 0]
    ]:
        t, r, c = heapq.heappop ([
        [18, 3, 2], [21, 1, 3],
        [19, 3, 3], [24, 1, 0],
        [22, 1, 2], [23, 1, 1],
        [20, 3, 4]
        ])
        >> t = 16, r = 2, c = 1,
        >> minH = [
        >> [18, 3, 2], [21, 1, 3],
        >> [19, 3, 3], [24, 1, 0],
        >> [22, 1, 2], [23, 1, 1],
        >> [20, 3, 4]
        >> ]
        if 2 == 5 - 1 and 1 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 2 + 0, 1 + 1
            >> neiR = 2, neiC = 2,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (2, 1), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3),
(2, 2), (1, 0), (3, 2), (1, 3)}
            if (2 < 0 or 2 < 0 or 2 == 5 or 2 == 5 or (2, 2) in visit): --> True
                continue

================================================================================
        for 0, -1 in directions:
            neiR, neiC = 2 + 0, 1 + -1
            >> neiR = 2, neiC = 0,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (2, 1), (0, 0), (1, 1), (0, 3), (1, 4), (2, 3), (0, 2), (3, 3),
(2, 2), (1, 0), (3, 2), (1, 3)}
            if (2 < 0 or 0 < 0 or 2 == 5 or 0 == 5 or (2, 0) in visit): --> False
            visit.add(((2, 0)))
            >> visit = {
            >> (0, 1), (2, 4), (1, 2), (0, 4),
            >> (3, 4), (2, 1), (0, 0), (1, 1),
            >> (0, 3), (2, 0), (1, 4), (2, 3),
            >> (0, 2), (3, 3), (2, 2), (1, 0),
            >> (3, 2), (1, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[2] [0]), 2, 0])
            >> minH = [
            >> [16, 2, 0], [18, 3, 2], [19, 3, 3], [21, 1, 3],
            >> [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0]
            >> ]

        for 1, 0 in directions:
            neiR, neiC = 2 + 1, 1 + 0
            >> neiR = 3, neiC = 1,
#visit = {(0, 1), (2, 4), (1, 2), (0, 4), (3, 4), (2, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3), (0, 2),
(3, 3), (2, 2), (1, 0), (3, 2), (1, 3)}
            if (3 < 0 or 1 < 0 or 3 == 5 or 1 == 5 or (3, 1) in visit): --> False
            visit.add(((3, 1)))
            >> visit = {
            >> (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 3),
            >> (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2), (0, 0),
            >> (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)
            >> }
            heapq.heappush(minH, [max(16, grid[3] [1]), 3, 1])
            >> minH = [
            >> [16, 2, 0], [17, 3, 1], [19, 3, 3], [18, 3, 2],
            >> [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0], [21, 1, 3]
            >> ]

        for -1, 0 in directions:
            neiR, neiC = 2 + -1, 1 + 0
            >> neiR = 1, neiC = 1,
```

```
#visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
(0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (1 < 0 or 1 < 0 or 1 == 5 or 1 == 5 or (1, 1) in visit): --> True
                continue

================================================================================
    while [
    [16, 2, 0], [17, 3, 1], [19, 3, 3], [18, 3, 2],
    [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0], [21, 1, 3]
    ]:
        t, r, c = heapq.heappop ([
        [17, 3, 1], [18, 3, 2], [19, 3, 3], [21, 1, 3],
        [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0]
        ])
        >> t = 16, r = 2, c = 0,
        >> minH = [
        >> [17, 3, 1], [18, 3, 2], [19, 3, 3], [21, 1, 3],
        >> [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0]
        >> ]
        if 2 == 5 - 1 and 0 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 2 + 0, 0 + 1
            >> neiR = 2, neiC = 1,
#visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
(0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (2 < 0 or 1 < 0 or 2 == 5 or 1 == 5 or (2, 1) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 2 + 0, 0 + -1
            >> neiR = 2, neiC = -1,
#visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
(0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (2 < 0 or -1 < 0 or 2 == 5 or -1 == 5 or (2, -1) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 2 + 1, 0 + 0
            >> neiR = 3, neiC = 0,
 #visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
(0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or 0 < 0 or 3 == 5 or 0 == 5 or (3, 0) in visit): --> False
            visit.add(((3, 0)))
            >> visit = {
            >> (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0),
            >> (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
            >> (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}


            heapq.heappush(minH, [max(16, grid[3] [0]), 3, 0])
            >> minH = [[16, 3, 0], [17, 3, 1], [19, 3, 3], [18, 3, 2],
            >> [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0], [21, 1, 3]]

        for -1, 0 in directions:
            neiR, neiC = 2 + -1, 0 + 0
            >> neiR = 1, neiC = 0,
#visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1),
(3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (1 < 0 or 0 < 0 or 1 == 5 or 0 == 5 or (1, 0) in visit): --> True
                continue

================================================================================

    while [[16, 3, 0], [17, 3, 1], [19, 3, 3], [18, 3, 2],
    [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0], [21, 1, 3]]:
        t, r, c = heapq.heappop ([
```

```
        [17, 3, 1], [18, 3, 2], [19, 3, 3], [21, 1, 3],
        [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0]
        ])
        >> t = 16, r = 3, c = 0,
        >> minH = [[17, 3, 1], [18, 3, 2], [19, 3, 3], [21, 1, 3],
        >> [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0]]

        if 3 == 5 - 1 and 0 == 5 - 1: --> False
        for 0, 1 in directions:
            neiR, neiC = 3 + 0, 0 + 1
            >> neiR = 3, neiC = 1,
 #visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1),
(3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or 1 < 0 or 3 == 5 or 1 == 5 or (3, 1) in visit): --> True
                continue

        for 0, -1 in directions:
            neiR, neiC = 3 + 0, 0 + -1
            >> neiR = 3, neiC = -1,
#visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1),
(3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or -1 < 0 or 3 == 5 or -1 == 5 or (3, -1) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 3 + 1, 0 + 0
            >> neiR = 4, neiC = 0,
#visit = {(3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4), (2, 1),
(3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 0 < 0 or 4 == 5 or 0 == 5 or (4, 0) in visit): --> False
            visit.add(((4, 0)))
            >> visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0),
            >> (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
            >> (2, 1), (3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}

            heapq.heappush(minH, [max(16, grid[4] [0]), 4, 0])
            >> minH = [[16, 4, 0], [17, 3, 1], [19, 3, 3],
            >> [18, 3, 2], [22, 1, 2], [23, 1, 1], [20, 3, 4],
            >> [24, 1, 0], [21, 1, 3]]

        for -1, 0 in directions:
            neiR, neiC = 3 + -1, 0 + 0
            >> neiR = 2, neiC = 0,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
(2, 1), (3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (2 < 0 or 0 < 0 or 2 == 5 or 0 == 5 or (2, 0) in visit): --> True
                continue

==========================================================================================
    while [
    [16, 4, 0],
    [17, 3, 1],
    [19, 3, 3],
    [18, 3, 2],
    [22, 1, 2],
    [23, 1, 1],
    [20, 3, 4],
    [24, 1, 0],
    [21, 1, 3]
    ]:
        t, r, c = heapq.heappop ([
        [17, 3, 1], [18, 3, 2],
        [19, 3, 3], [21, 1, 3],
        [22, 1, 2], [23, 1, 1],
        [20, 3, 4], [24, 1, 0]
        ])
        >> t = 16, r = 4, c = 0,
```

```
        >> minH = [
        >> [17, 3, 1], [18, 3, 2],
        >> [19, 3, 3], [21, 1, 3],
        >> [22, 1, 2], [23, 1, 1],
        >> [20, 3, 4], [24, 1, 0]
        >> ]

        if 4 == 5 - 1 and 0 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 4 + 0, 0 + 1
            >> neiR = 4, neiC = 1,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
(2, 1), (3, 2), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 1 < 0 or 4 == 5 or 1 == 5 or (4, 1) in visit): --> False
            visit.add(((4, 1)))
            >> visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2),
            >> (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4),
            >> (1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (0, 0),
            >> (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            heapq.heappush(minH, [max(16, grid[4] [1]), 4, 1])
            >> minH = [[16, 4, 1], [17, 3, 1], [19, 3, 3], [18, 3, 2],
            >> [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0], [21, 1, 3]]

        for 0, -1 in directions:
            neiR, neiC = 4 + 0, 0 + -1
            >> neiR = 4, neiC = -1,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
(2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or -1 < 0 or 4 == 5 or -1 == 5 or (4, -1) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 4 + 1, 0 + 0
            >> neiR = 5, neiC = 0,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
(2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (5 < 0 or 0 < 0 or 5 == 5 or 0 == 5 or (5, 0) in visit): --> True
                continue

        for -1, 0 in directions:
            neiR, neiC = 4 + -1, 0 + 0
            >> neiR = 3, neiC = 0,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
(2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or 0 < 0 or 3 == 5 or 0 == 5 or (3, 0) in visit): --> True
                continue

================================================================================
    while [
    [16, 4, 1], [17, 3, 1], [19, 3, 3],
    [18, 3, 2], [22, 1, 2], [23, 1, 1],
    [20, 3, 4], [24, 1, 0], [21, 1, 3]
    ]:
        t, r, c = heapq.heappop ([
        [17, 3, 1], [18, 3, 2], [19, 3, 3],
        [21, 1, 3], [22, 1, 2], [23, 1, 1],
        [20, 3, 4], [24, 1, 0]])
        >> t = 16, r = 4, c = 1,
        >> minH = [[17, 3, 1], [18, 3, 2], [19, 3, 3],
        >> [21, 1, 3], [22, 1, 2], [23, 1, 1], [20, 3, 4], [24, 1, 0]]

        if 4 == 5 - 1 and 1 == 5 - 1: --> False
        for 0, 1 in directions:
            neiR, neiC = 4 + 0, 1 + 1
            >> neiR = 4, neiC = 2,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2), (0, 4),
```

```
(2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 2 < 0 or 4 == 5 or 2 == 5 or (4, 2) in visit): --> False
            visit.add(((4, 2)))
            >> visit = {(4, 0), (3, 4), (3, 1),
            >> (0, 2), (2, 2), (1, 0), (1, 3), (4, 2),
            >> (3, 0), (3, 3), (0, 1), (2, 4), (1, 2),
            >> (0, 4), (2, 1), (3, 2), (4, 1), (0, 0),
            >> (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            heapq.heappush(minH, [max(16, grid[4] [2]), 4, 2])
            >> minH = [[16, 4, 2], [17, 3, 1], [19, 3, 3],
            >> [18, 3, 2], [22, 1, 2], [23, 1, 1], [20, 3, 4],
            >>  [24, 1, 0], [21, 1, 3]]

        for 0, -1 in directions:
            neiR, neiC = 4 + 0, 1 + -1
            >> neiR = 4, neiC = 0,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2),
(0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 0 < 0 or 4 == 5 or 0 == 5 or (4, 0) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 4 + 1, 1 + 0
            >> neiR = 5, neiC = 1,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2),
(0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (5 < 0 or 1 < 0 or 5 == 5 or 1 == 5 or (5, 1) in visit): --> True
                continue

        for -1, 0 in directions:
            neiR, neiC = 4 + -1, 1 + 0
            >> neiR = 3, neiC = 1,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2),
(0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or 1 < 0 or 3 == 5 or 1 == 5 or (3, 1) in visit): --> True
                continue

================================================================================
    while [
    [16, 4, 2], [17, 3, 1], [19, 3, 3],
    [18, 3, 2], [22, 1, 2], [23, 1, 1],
    [20, 3, 4], [24, 1, 0], [21, 1, 3]
    ]:
        t, r, c = heapq.heappop ([
        [17, 3, 1], [18, 3, 2], [19, 3, 3],
        [21, 1, 3], [22, 1, 2], [23, 1, 1],
        [20, 3, 4], [24, 1, 0]
        ])
        >> t = 16, r = 4, c = 2,
        >> minH = [
        >> [17, 3, 1], [18, 3, 2], [19, 3, 3],
        >> [21, 1, 3], [22, 1, 2], [23, 1, 1],
        >> [20, 3, 4], [24, 1, 0]]

        if 4 == 5 - 1 and 2 == 5 - 1: --> False
        for 0, 1 in directions:
            neiR, neiC = 4 + 0, 2 + 1
            >> neiR = 4, neiC = 3,
#visit = {(4, 0), (3, 4), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4), (1, 2),
(0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 3 < 0 or 4 == 5 or 3 == 5 or (4, 3) in visit): --> False

            visit.add(((4, 3)))
            >> visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2),
            >> (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3),
            >> (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
            >> (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
```

```
            heapq.heappush(minH, [max(16, grid[4] [3]), 4, 3])
            >> minH = [[16, 4, 3], [17, 3, 1], [19, 3, 3],
            >> [18, 3, 2], [22, 1, 2], [23, 1, 1], [20, 3, 4],
            >> [24, 1, 0], [21, 1, 3]]

        for 0, -1 in directions:
            neiR, neiC = 4 + 0, 2 + -1
            >> neiR = 4, neiC = 1,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 1 < 0 or 4 == 5 or 1 == 5 or (4, 1) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 4 + 1, 2 + 0
            >> neiR = 5, neiC = 2,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (5 < 0 or 2 < 0 or 5 == 5 or 2 == 5 or (5, 2) in visit): --> True
                continue

        for -1, 0 in directions:
            neiR, neiC = 4 + -1, 2 + 0
            >> neiR = 3, neiC = 2,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or 2 < 0 or 3 == 5 or 2 == 5 or (3, 2) in visit): --> True
                continue

================================================================================
    while [
    [16, 4, 3], [17, 3, 1], [19, 3, 3],
    [18, 3, 2], [22, 1, 2], [23, 1, 1],
    [20, 3, 4], [24, 1, 0], [21, 1, 3]
    ]:
        t, r, c = heapq.heappop ([
        [17, 3, 1], [18, 3, 2], [19, 3, 3],
        [21, 1, 3], [22, 1, 2], [23, 1, 1],
        [20, 3, 4], [24, 1, 0]])
        >> t = 16, r = 4, c = 3,
        >> minH = [[17, 3, 1], [18, 3, 2], [19, 3, 3],
        >> [21, 1, 3], [22, 1, 2], [23, 1, 1],
        >> [20, 3, 4], [24, 1, 0]]

        if 4 == 5 - 1 and 3 == 5 - 1: --> False

        for 0, 1 in directions:
            neiR, neiC = 4 + 0, 3 + 1
            >> neiR = 4, neiC = 4,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 4 < 0 or 4 == 5 or 4 == 5 or (4, 4) in visit): --> False
            visit.add(((4, 4)))
            >> visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2),
            >> (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3),
            >> (0, 1), (2, 4), (1, 2), (0, 4), (2, 1), (3, 2),
            >> (4, 1), (4, 4), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            heapq.heappush(minH, [max(16, grid[4] [4]), 4, 4])
            >> minH = [[16, 4, 4], [17, 3, 1], [19, 3, 3],
            >> [18, 3, 2], [22, 1, 2], [23, 1, 1], [20, 3, 4],
            >> [24, 1, 0], [21, 1, 3]]
            >>
        for 0, -1 in directions:
            neiR, neiC = 4 + 0, 3 + -1
            >> neiR = 4, neiC = 2,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
```

```
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (4, 4), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (4 < 0 or 2 < 0 or 4 == 5 or 2 == 5 or (4, 2) in visit): --> True
                continue

        for 1, 0 in directions:
            neiR, neiC = 4 + 1, 3 + 0
            >> neiR = 5, neiC = 3,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (4, 4), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (5 < 0 or 3 < 0 or 5 == 5 or 3 == 5 or (5, 3) in visit): --> True
                continue

        for -1, 0 in directions:
            neiR, neiC = 4 + -1, 3 + 0
            >> neiR = 3, neiC = 3,
#visit = {(4, 0), (3, 4), (4, 3), (3, 1), (0, 2), (2, 2), (1, 0), (1, 3), (4, 2), (3, 0), (3, 3), (0, 1), (2, 4),
(1, 2), (0, 4), (2, 1), (3, 2), (4, 1), (4, 4), (0, 0), (1, 1), (0, 3), (2, 0), (1, 4), (2, 3)}
            if (3 < 0 or 3 < 0 or 3 == 5 or 3 == 5 or (3, 3) in visit): --> True
                continue

================================================================================
    while [
    [16, 4, 4], [17, 3, 1],
    [19, 3, 3], [18, 3, 2],
    [22, 1, 2], [23, 1, 1],
    [20, 3, 4], [24, 1, 0],
    [21, 1, 3]]:
        t, r, c = heapq.heappop ([
        [17, 3, 1], [18, 3, 2],
        [19, 3, 3], [21, 1, 3],
        [22, 1, 2], [23, 1, 1],
        [20, 3, 4], [24, 1, 0]
        ])
        >> t = 16, r = 4, c = 4,
        >> minH = [
        >> [17, 3, 1], [18, 3, 2],
        >> [19, 3, 3], [21, 1, 3],
        >> [22, 1, 2], [23, 1, 1],
        >> [20, 3, 4], [24, 1, 0]
        >> ]

        if 4 == 5 - 1 and 4 == 5 - 1: --> True
            return 16
================================================================================
Output:  16
```

5. Alien Dictionary

Link

Hard #toplogical_sort #DAG #graphs #dfs #directed-acyclic-graphs

5.1. Problem Statement

There is a new alien language which uses the Latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where words are **sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

1. You may assume all letters are in lowercase.
2. The dictionary is invalid, if string a is prefix of string b and b is appear before a.
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return the smallest in normal lexicographical order.
5. The letters in **one** string are of the same rank by default and are sorted in Human dictionary order.

Example 1:

```
Input: ["wrt","wrf","er","ett","rftt"]
Output: "wertf"
Explanation:
from "wrt"and"wrf" ,we can get 't'<'f'
from "wrt"and"er" ,we can get 'w'<'e'
from "er"and"ett" ,we can get 'r'<'t'
from "ett"and"rftt" ,we can get 'e'<'r'
So return "wertf"
```

Example 2:

```
Input: ["z","x"]
Output: "zx"
Explanation:
from "z" and "x", we can get 'z' < 'x'
So return "zx"
```

5.2. Code

```python
class Solution:
        def alienOrder(self, words):
                adj = { c: set() for w in words for c in w}
                for i in range(len(words) -1):
                        w1, w2 = words[i], words[i + 1]
                        minLen = min(len(w1), len(w2))
                        if len(w1) > len(w2) and w1[:minLen] == w2[:minLen]:
                                return ""
                        for j in range(minLen):
                                if w1[j] != w2[j]:
                                        adj[w1[j]].add(w2[j])
                                        break
                visit = {} # False = visited, True = current path
                res = []
                def dfs(c):
                        if c in visit:
                                return visit[c]
                        visit[c] = True
                        for neighbor in adj[c]:
                                if dfs(neighbor):
                                        return True
                        visit[c] = False
                        res.append(c)
                for c in adj:
                        if dfs(c):
                                return ""
                return "".join(res[::-1])
if __name__ == "__main__":
        s = Solution()
        words = ["wrt","wrf","er","ett","rftt"]
        result = s.alienOrder(words)
        print("Output: ", result)
```

5.3. Explanation

```
def alienOrder(self, ['wrt', 'wrf', 'er', 'ett', 'rftt']):
```

```
                adj = {'w': set(), 'r': set(), 't': set(), 'f': set(), 'e': set()}

    for 0 in range(0, 4):
        w1, w2 = 'wrt', 'wrf' >> w1 = 'wrt', w2 = 'wrf'
        minLen = 3
        if 3 > 3 and 'wrt' == 'wrf': --> False

        for 0 in range(0, 3):
            if w != w: --> False

        for 1 in range(0, 3):
            if r != r: --> False

        for 2 in range(0, 3):
            if t != f: --> True
                adj[t].add(f)
                >> adj = {'w': set(), 'r': set(),
                >> 't': {'f'}, 'f': set(), 'e': set()}
                break

    for 1 in range(0, 4):
        w1, w2 = 'wrf', 'er' >> w1 = 'wrf', w2 = 'er'
        minLen = 2
        if 3 > 2 and 'wr' == 'er': --> False

        for 0 in range(0, 2):
            if w != e: --> True
                adj[w].add(e)
                >> adj = {'w': {'e'}, 'r': set(),
                >> 't': {'f'}, 'f': set(), 'e': set()}
                break

    for 2 in range(0, 4):
        w1, w2 = 'er', 'ett' >> w1 = 'er', w2 = 'ett'
        minLen = 2
        if 2 > 3 and 'er' == 'et': --> False

        for 0 in range(0, 2):
            if e != e: --> False

        for 1 in range(0, 2):
            if r != t: --> True
                adj[r].add(t)
                >> adj = {'w': {'e'}, 'r': {'t'},
                >> 't': {'f'}, 'f': set(), 'e': set()}
                break

    for 3 in range(0, 4):
        w1, w2 = 'ett', 'rftt' >> w1 = 'ett', w2 = 'rftt'
        minLen = 3
        if 3 > 4 and 'ett' == 'rft': --> False

        for 0 in range(0, 3):
            if e != r: --> True
                adj[e].add(r)
                >> adj = {'w': {'e'}, 'r': {'t'},
                >> 't': {'f'}, 'f': set(), 'e': {'r'}}
                break

    visit = {}
    res = []

    for 'w' in {'w': {'e'}, 'r': {'t'}, 't': {'f'}, 'f': set(), 'e': {'r'}}:
        if dfs('w'): --> None:
        ⬇️⬇️⬇️⬇️⬇️⬇️

    visit['w'] = True
```

```
        >> visit = {'w': True}


        visit['e'] = True
        >> visit = {'w': True, 'e': True}


        visit['r'] = True
        >> visit = {'w': True, 'e': True, 'r': True}


        visit['t'] = True
        >> visit = {'w': True, 'e': True, 'r': True, 't': True}


        visit['f'] = True
        >> visit = {'w': True, 'e': True, 'r': True, 't': True, 'f': True}


        visit['f'] = False
        >> visit = {'w': True, 'e': True, 'r': True, 't': True, 'f': False}
        res.append('f') >> res = ['f']


        visit['t'] = False
        >> visit = {'w': True, 'e': True, 'r': True, 't': False, 'f': False}
        res.append('t') >> res = ['f', 't']


        visit['r'] = False
        >> visit = {'w': True, 'e': True, 'r': False, 't': False, 'f': False}
        res.append('r') >> res = ['f', 't', 'r']


        visit['e'] = False
        >> visit = {'w': True, 'e': False, 'r': False, 't': False, 'f': False}
        res.append('e') >> res = ['f', 't', 'r', 'e']


        visit['w'] = False
        >> visit = {'w': False, 'e': False, 'r': False, 't': False, 'f': False}
        res.append('w') >> res = ['f', 't', 'r', 'e', 'w']

                ⬅ ⬅ ⬅ ⬅ ⬅ ⬅ ⬅ ⬅
                END END END END END END END END

        for 'r' in {'w': {'e'}, 'r': {'t'}, 't': {'f'}, 'f': set(), 'e': {'r'}}:
                if dfs('r'): --> False:

        for 't' in {'w': {'e'}, 'r': {'t'}, 't': {'f'}, 'f': set(), 'e': {'r'}}:
                if dfs('t'): --> False:

        for 'f' in {'w': {'e'}, 'r': {'t'}, 't': {'f'}, 'f': set(), 'e': {'r'}}:
                if dfs('f'): --> False:

        for 'e' in {'w': {'e'}, 'r': {'t'}, 't': {'f'}, 'f': set(), 'e': {'r'}}:
                if dfs('e'): --> False:

        return 'wertf'

Output:  'wertf'
```

6. Cheapest Flights Within K Stops

Link

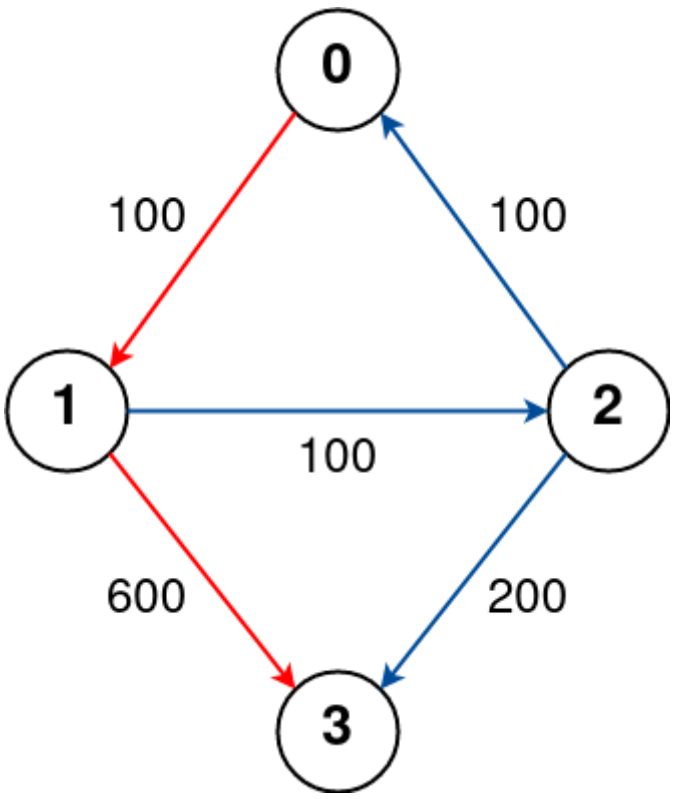
6.1. Problem Statement

Medium #bellman-ford-algorithm #graphs #bfs

There are `n` cities connected by some number of flights. You are given an array `flights` where `flights[i] = [fromi, toi, pricei]` indicates that there is a flight from city `fromi` to city `toi` with cost `pricei`.

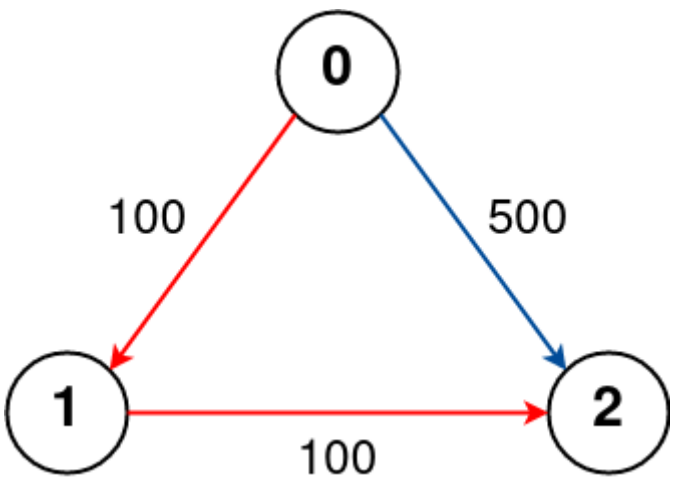
You are also given three integers `src`, `dst`, and `k`, return ***the cheapest price*** *from* `src` *to* `dst` *with at most* `k` *stops.* If there is no such route, return `-1`.

Example 1:



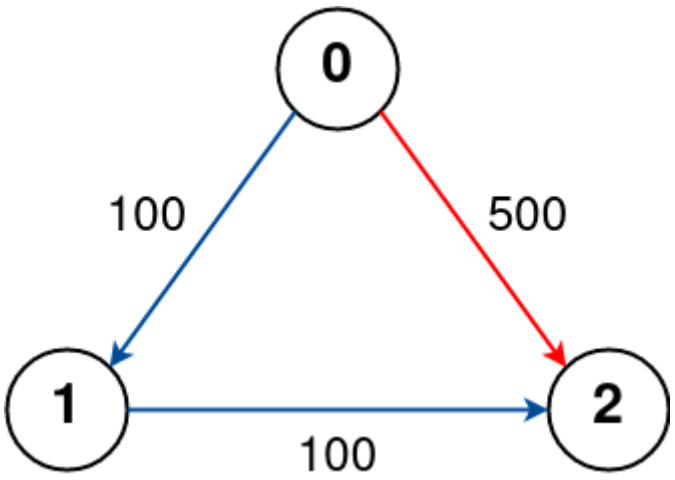
Input: n = 4, flights = `[[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]]`, src = 0, dst = 3, k = 1
Output: 700
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700.
Note that the path through cities `[0,1,2,3]` is cheaper but is invalid because it uses 2 stops.

Example 2:



Input: `n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1`
Output: 200
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 = 200.

Example 3:



Input: `n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 0`
Output: 500
Explanation:
The graph is shown above.
The optimal path with no stops from city 0 to 2 is marked in red and has cost 500.

Constraints:

- `1 <= n <= 100`
- `0 <= flights.length <= (n * (n - 1) / 2)`
- `flights[i].length == 3`
- `0 <= fromi, toi < n`
- `fromi != toi`
- `1 <= pricei <= 104`
- There will not be any multiple flights between two cities.
- `0 <= src, dst, k < n`
- `src != dst`

6.2. Code

```
class Solution:
        def findCheapestPrice(self, n, flights, src, dst, k):
                prices = [float("inf")] *n
                prices[src] = 0
                for i in range(k+1):
                        tmpPrices = prices.copy()
                        for s, d, p in flights: # s = source, d = destination, p = price
                                if prices[s] == float("inf"):
                                        continue
                                if prices[s] + p < tmpPrices[d]:
                                        tmpPrices[d] = prices[s] + p
                        prices = tmpPrices
                return -1 if prices[dst] == float('inf') else prices[dst]
if __name__ == '__main__':
        s = Solution()
        n = 4
        flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]]
        src = 0
        dst = 3
        k = 1
        result = s.findCheapestPrice(n, flights, src, dst, k)
        print(f"Output: {result}")
```

6.3. Explanation

```
def findCheapestPrice(self, n = 4, flights = [[0, 1, 100], [1, 2, 100], [2, 0, 100], [1, 3, 600], [2, 3, 200]], src
= 0, dst = 3, k = 1):
        prices = [float("inf")] *4
        >> prices = [0, inf, inf, inf]
```

```
        prices[src] = 0
        >> prices = [0, inf, inf, inf]

        for 0 in range(0, 2):
            tmpPrices = [0, inf, inf, inf].copy()
            >> tmpPrices = [0, inf, inf, inf]

            for [0, 1, 100] in [[0, 1, 100], [1, 2, 100], [2, 0, 100],
                                              [1, 3, 600], [2, 3, 200]]:
                if prices[0] == float("inf"): --> False
                if prices[0] + 100 < tmpPrices[1]: --> True
                # prices[0] + 100 = 100, tmpPrices[1] = inf
                    tmpPrices[1] = prices[0] + 100
                    >> tmpPrices = [0, 100, inf, inf]

            for [1, 2, 100] in [[0, 1, 100], [1, 2, 100],
                                              [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[1] == float("inf"): --> True
                if prices[1] + 100 < tmpPrices[2]: --> False
                # prices[1] + 100 = inf, tmpPrices[2] = inf
                    continue

            for [2, 0, 100] in [[0, 1, 100], [1, 2, 100],
                                              [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[2] == float("inf"): --> True
                if prices[2] + 100 < tmpPrices[0]: --> False
                # prices[2] + 100 = inf, tmpPrices[0] = 0
                    continue

            for [1, 3, 600] in [[0, 1, 100], [1, 2, 100],
                                              [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[1] == float("inf"): --> True
                if prices[1] + 600 < tmpPrices[3]: --> False
                # prices[1] + 600 = inf, tmpPrices[3] = inf
                    continue

            for [2, 3, 200] in [[0, 1, 100], [1, 2, 100], [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[2] == float("inf"): --> True
                if prices[2] + 200 < tmpPrices[3]: --> False
                # prices[2] + 200 = inf, tmpPrices[3] = inf
                    continue

            prices = [0, 100, inf, inf]

        for 1 in range(0, 2):
            tmpPrices = [0, 100, inf, inf].copy()
            >> tmpPrices = [0, 100, inf, inf]

            for [0, 1, 100] in [[0, 1, 100], [1, 2, 100],
                                              [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[0] == float("inf"): --> False
                if prices[0] + 100 < tmpPrices[1]: --> False
                # prices[0] + 100 = 100, tmpPrices[1] = 100

            for [1, 2, 100] in [[0, 1, 100], [1, 2, 100],
                                              [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[1] == float("inf"): --> False
                if prices[1] + 100 < tmpPrices[2]: --> True
                # prices[1] + 100 = 200, tmpPrices[2] = inf
                    tmpPrices[2] = prices[1] + 100
                    >> tmpPrices = [0, 100, 200, inf]

            for [2, 0, 100] in [[0, 1, 100], [1, 2, 100], [2, 0, 100],
                                              [1, 3, 600], [2, 3, 200]]:
                if prices[2] == float("inf"): --> True
                if prices[2] + 100 < tmpPrices[0]: --> False
                # prices[2] + 100 = inf, tmpPrices[0] = 0
```

```
                    continue

            for [1, 3, 600] in [[0, 1, 100], [1, 2, 100], [2, 0, 100],
                                                [1, 3, 600], [2, 3, 200]]:
                if prices[1] == float("inf"): --> False
                if prices[1] + 600 < tmpPrices[3]: --> True
                # prices[1] + 600 = 700, tmpPrices[3] = inf
                    tmpPrices[3] = prices[1] + 600
                    >> tmpPrices = [0, 100, 200, 700]

            for [2, 3, 200] in [[0, 1, 100], [1, 2, 100], [2, 0, 100], [1, 3, 600], [2, 3, 200]]:
                if prices[2] == float("inf"): --> True
                if prices[2] + 200 < tmpPrices[3]: --> False
                # prices[2] + 200 = inf, tmpPrices[3] = 700
                    continue
            prices = [0, 100, 200, 700]

        return 700

Output: 700
```

End