[Python] Dijkstra / Binary Search + DFS / Union Find, complexity analysis

Olala_michelle ★ 165 Last Edit: October 31, 2019 3:47 AM 3.7K VIEWS

76

- Time: O(MN log MN), since for each element in matrix we have to do a heap push, which cost O(log # of element in the heap) times. The size of the heap can grow up to # of elemnts in the matrix.
- Space: O(MN). We need to keep track of the elements we have seen so far. Finally the size of seen will grow up to # of items in the matrix.

```
class Solution:
    def maximumMinimumPath(self, A: List[List[int]]) -> int:
        dire = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        R, C = len(A), len(A[0])

        maxHeap = [(-A[0][0], 0, 0)]
        seen = [[0 for _ in range(C)] for _ in range(R)]
        while maxHeap:
            val, x, y = heapq.heappop(maxHeap)
            # seen[x][y] = 1 # got TLE
            if x == R - 1 and y == C - 1: return -val
            for dx, dy in dire:
                nx, ny = x + dx, y + dy
                if 0 <= nx < R and 0 <= ny < C and not seen[nx][ny]:
                    seen[nx][ny] = 1 # passed
                    heapq.heappush(maxHeap, (max(val, -A[nx][ny]), nx, ny))
        return -1
```

Binary Search + DFS:

- Time: O(MN log MN)
- Space: O(MN)

Intuition:

- remove all the cells with value >= `min(begin,end)`
- sort all remaining unique values
- enumerate all remaining values to find a maximum value that is the minimum value in a path from the begin to the end; for each value, we use DFS to check weather there exists a path from begin to end such that this value is the the minimum among the values in that path.
  - if we find that value, we keep Binary Search to try to find a bigger value
  - we loose our search criatria and see if there is path from begin to end that all the values in that path >= a smaller value by moving the right pointer to the left

Why use DFS?
we use DFS to check if there exist a path from the being to the end

Why use Binary Search?
we use binary search to find the upper boundary. So when we find a valid value, we move left pointer to mid + 1 to keep finding a larger value. Just as what we did in finding the first bad version: if we find a bad version, we move right pointer to the mid - 1 to find a earlier bad product.

What are in the sorted array? / What are we binary search for?
Since we don't know weather a value 'val'. in this sorted array is the minimum, so we deliberately make it be, by only considering the cells with values that are larger than this value. And if this arrangement doesn't work (we cannot construct a path from begin to the end with 'val' being the minimum value seen) when BinarySearch('val') returns false.

```
class Solution:
    def maximumMinimumPath(self, A: List[List[int]]) -> int:
        dire = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        R, C = len(A), len(A[0])

        def check(val):
            memo = [[0 for _ in range(C)] for _ in range(R)]

            def dfs(x,y):
                if x == R - 1 and y == C - 1:
                    return True
                memo[x][y] = 1
                for d in dire:
                    nx = x + d[0]
                    ny = y + d[1]
                    if 0 <= nx < R and 0 <= ny < C and not memo[nx][ny] and A[nx][ny] >= val and dfs(nx,ny):
                        return True
                return False

            return dfs(0,0)

        unique = set()
        ceiling = min(A[0][0], A[-1][-1])
        for r in range(R):
            for c in range(C):
                if A[r][c] <= ceiling:
                    unique.add(A[r][c])

        arr = sorted(unique)
        l, r = 0, len(arr) - 1
        while l <= r:
            m = l + (r - l) // 2
            # if check(m):
            if check(arr[m]):
                # cause we're trying to find the MAXIMUM of 'minimum'
                l = m + 1
            else:
                r = m - 1
        return arr[r]
```

Union Find:
Idea is similar to LC788 Swim in Rising Water or percolation.

We want to find a path from (0,0) to (n-1, m-1) w/ max lower bound. So we just visit the cell in the order from largest to smallest, and use UF to connect all the **visited** cells. Once we make (0,0) and (n-1, m-1) connected, we know we get a path with max lower bound, which is just the value of the last visited cell.

1. sort all the points in a descending order
2. union the point with the explored points until `start` and `end` has the same parent

- Time: O(MN log MN)
- Space: O(MN)

```
class Solution:
    def maximumMinimumPath(self, A: List[List[int]]) -> int:
        R, C = len(A), len(A[0])
        parent = [i for i in range(R * C)]
        dire = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        seen = [[0 for _ in range(C)] for _ in range(R)]

        def find(x):
            if parent[x] != x:
                parent[x] = find(parent[x])
            return parent[x]

        def union(x, y):
            rx, ry = find(x), find(y)
            if rx != ry:
                parent[ry] = rx

        points = [(x, y) for x in range(R) for y in range(C)]
        points.sort(key = lambda x: A[x[0]][x[1]], reverse = True)

        for x, y in points:
            seen[x][y] = 1
            for dx, dy in dire:
                nx, ny = x + dx, y + dy
                if 0 <= nx < R and 0 <= ny < C and seen[nx][ny]:
                    union(x * C + y, nx * C + ny)
            if find(0) == find(R * C - 1):
                return A[x][y]
        return -1
```