Notebook

August 2, 2024

[ ]:
```
from typing import List
```

[ ]:
```
#### Patching Array
#### https://leetcode.com/problems/patching-array/solutions/78492/
 ↪c-8ms-greedy-solution-with-explanation/?
 ↪envType=daily-question&envId=2024-06-16
```

[ ]:
```
#### https://leetcode.com/problems/minimum-cost-homecoming-of-a-robot-in-a-grid/
 ↪description/

"""
Input: startPos = [1, 0], homePos = [2, 3], rowCosts = [5, 4, 3], colCosts =␣
 ↪[8, 2, 6, 7]
Output: 18
Explanation: One optimal path is that:
Starting from (1, 0)
-> It goes down to (2, 0). This move costs rowCosts[2] = 3.
-> It goes right to (2, 1). This move costs colCosts[1] = 2.
-> It goes right to (2, 2). This move costs colCosts[2] = 6.
-> It goes right to (2, 3). This move costs colCosts[3] = 7.
The total cost is 3 + 2 + 6 + 7 = 18
"""

class Solution:
    def minCost(self, startPos: List[int], homePos: List[int], rowCosts:␣
 ↪List[int], colCosts: List[int]) -> int:
        ans = 0

        x, y = [startPos[0], homePos[0]], [startPos[1], homePos[1]]
        if x[0] > x[1]:
            x[0], x[1] = x[1], x[0]

        if y[0] > y[1]:
            y[0], y[1] = y[1], y[0]

        for i in range(x[0], x[1]+1):
            ans += rowCosts[i]
```

```
        for i in range(y[0], y[1]+1):
            ans += colCosts[i]

        return ans - rowCosts[startPos[0]] - colCosts[startPos[1]]
```

[ ]:
```
# https://leetcode.com/problems/find-valid-matrix-given-row-and-column-sums/
 ↪description/
# given rowSum and colSum, find a valid matrix
# greedily fill the current cell with the minimum of the rowSum[i] and␣
 ↪colSum[j] of that particular cell, i is the row index and j is the column␣
 ↪index
"""
Input: rowSum = [5,7,10], colSum = [8,6,8]
Output: [[0,5,0], [6,1,0], [2,0,8]]
"""

class Solution:
    def restoreMatrix(self, rowSum: List[int], colSum: List[int]) ->␣
 ↪List[List[int]]:
        m, n = len(rowSum), len(colSum)
        mat = [[0]*n for _ in range(m)]
        for i in range(m):
            for j in range(n):
                rsum, csum = rowSum[i], colSum[j]
                minn = min(rsum, csum)
                mat[i][j] = minn
                rowSum[i] -= minn
                colSum[j] -= minn
        return mat
```

[1]:
```
# https://leetcode.com/problems/reconstruct-a-2-row-binary-matrix/description/
# Similar to the above problem, but with a constraint that the sum of the ith␣
 ↪row should be upper[i] and lower[i]
# again check if the colsum[i] is 1, then fill the cell with 1, if upper >␣
 ↪lower, fill the cell with 1 in the upper row, else fill the cell with 1 in␣
 ↪the lower row

class Solution:
    def reconstructMatrix(self, upper: int, lower: int, colsum: List[int]) ->␣
 ↪List[List[int]]:
        n = len(colsum)
        upper_list = [0] * n
        lower_list = [0] * n

        for idx, val in enumerate(colsum):
```

```
            if val == 1:
                if upper > lower:
                    upper_list[idx] = 1
                    upper -= 1
                else:
                    lower_list[idx] = 1
                    lower -= 1
            elif val == 2:
                upper_list[idx] = lower_list[idx] = 1
                upper, lower = upper-1, lower-1

        return [upper_list, lower_list] if upper == lower == 0 else []
```

[ ]: