1. Use kmp to find substring. We will use kmp to check whether there is matched substring. For KMP, the key is to get lps array. For the pattern we want to match, we need to compare letter later to previous letter. It they are the same, we will add 1 and set current lps value as the current idx we could match. We will always find the same letter until it come to 0, then we will set this lps as zero. We will repeat this process until the end. Then we will take advantage of this lps array, we will fail to compare current letter, we will move m to lps[m - 1], until to zero. We will acculate string’s index. When we find our destination, we will return True. O(n)

```python
def stringMatch2(strs, match):
    result = []
    length = len(match)

    for value in strs:
        for v in value["bussinessName"].split(' '):
            if kmpSearch(v.lower(), match):
                result.append(value["bussinessName"])
                break

    return result

def kmpSearch(dest, match):
    length = len(dest)
    length_match = len(match)
    lps = [0] * length_match

    lpsCreate(match, lps)

    i, j = 0, 0
    while i < length:
        if dest[i] == match[j]:
            j += 1
            i += 1
            if j == length_match:
                return True
        else:
            if j == 0:
                i += 1
            else:
                j = lps[j - 1]
    return False



def lpsCreate(str, lps):
    idx = 0
    length = len(str)
    i = 1

    while i < length:
        if str[i] == str[idx]:
            idx += 1
            lps[i] = idx
            i += 1
        else:
            if idx == 0:
                lps[i] = 0
                i += 1
            else:
                idx = lps[idx - 1]
```

2. Random match users
We will check whether users is even, if not , we will append a None. Then, we will use random. Sample to get two elements to pair. Then we will replace these two with last two elements of the array and deduct/subtract length by 2. TC is O(n)

```
import random
def matchUsers2(users):
    length = len(users)
    if length % 2 == 1:
        users.append(None)
        length += 1
    result = []

    while length > 0:
        idx1, idx2 = random.sample(range(0, length), 2)
        result.append((users[idx1], users[idx2]))
```

```
       users[idx1], users[length - 1] = users[length - 1],
users[idx1]
       users[idx2], users[length - 2] = users[length - 2],
users[idx2]
       length -= 2

   return result
```

3. Love messages:
We will iterate through the whole array and use a memo to store every receiver’s message, to prevent cheat, we will use set as our value. In the end, we will use heapq to heappop k most frequent receivers. TC is O(nlogk)

```
from collections import defaultdict
from heapq import *
def getMaxiMessage(messages):
   memo = defaultdict(int)
   result = []
   h = []
   for message in messages:
       memo[message['receiver']] += 1

   for k, v in memo.items():
       heappush(h, (-v, k))
   for _ in range(k):
       result.append(heappop(h)[1])
   return result
```

4. Unique Path:
We will use a 2-D array to memorize the sum of paths. We will add left one and top one as current number of path to this cell. In the end, The last cell’s number is what we want. TC is O(mn):

```
def Unique_Path(m, n):
   path = [[0 for j in range(n)] for i in range(m)]
   for i in range(m):
       path[i][0] = 1
   for j in range(n):
       path[0][j] = 1

```

```
    for i in range(1, n):
        for j in range(1, m):
            path[i][j] = path[i - 1][j] + path[i][j - 1]
    return path[-1][-1]
```

5. Design a data structure to store catagory/sub/subsub

```
class Catagory:
    def __init__(self, nextCatagory):
        self.nextCatagory
        self.subCatories = set()
    def addSubCatory(self, catagory):
        self.subCatories.add(catagory)
```