Ex. 1

```
A[(0, 0)] = 0

A[(1, 0)] = 0

A[(2, 0)] = 0

A[(3, 0)] = 0

A[(4, 0)] = 0

A[(5, 0)] = 0

A[(6, 0)] = 0

A[(7, 0)] = 0

A[(8, 0)] = 0

A[(0, 0)] = 0

A[(0, 1)] = 0

A[(0, 2)] = 0

A[(0, 3)] = 0

A[(0, 4)] = 0

A[(0, 5)] = 0

A[(0, 6)] = 0

A[(0, 7)] = 0

A[(0, 8)] = 0

A[(0, 9)] = 0

s[1] =  0 , z[1] =  1
are unequal, so A[(1, 1)] = max ( A[(0, 1)], A[(1, 0)] ) = max ((0, 0)) =
0

s[1] =  0 , z[2] =  0
 are equal, so A[(2, 1)] = A[(1, 0)] + 1 =  1

s[1] =  0 , z[3] =  0
 are equal, so A[(3, 1)] = A[(2, 0)] + 1 =  1

s[1] =  0 , z[4] =  1
```

```
are unequal, so A[(4, 1)] = max ( A[(3, 1)], A[(4, 0)] ) = max ((1, 0)) =
1

s[1] =  0 , z[5] =  0
 are equal, so A[(5, 1)] = A[(4, 0)] + 1 =  1

s[1] =  0 , z[6] =  1
are unequal, so A[(6, 1)] = max ( A[(5, 1)], A[(6, 0)] ) = max ((1, 0)) =
1

s[1] =  0 , z[7] =  0
 are equal, so A[(7, 1)] = A[(6, 0)] + 1 =  1

s[1] =  0 , z[8] =  1
are unequal, so A[(8, 1)] = max ( A[(7, 1)], A[(8, 0)] ) = max ((1, 0)) =
1

s[2] =  1 , z[1] =  1
 are equal, so A[(1, 2)] = A[(0, 1)] + 1 =  1

s[2] =  1 , z[2] =  0
are unequal, so A[(2, 2)] = max ( A[(1, 2)], A[(2, 1)] ) = max ((1, 1)) =
1

s[2] =  1 , z[3] =  0
are unequal, so A[(3, 2)] = max ( A[(2, 2)], A[(3, 1)] ) = max ((1, 1)) =
1

s[2] =  1 , z[4] =  1
 are equal, so A[(4, 2)] = A[(3, 1)] + 1 =  2

s[2] =  1 , z[5] =  0
are unequal, so A[(5, 2)] = max ( A[(4, 2)], A[(5, 1)] ) = max ((2, 1)) =
2

s[2] =  1 , z[6] =  1
 are equal, so A[(6, 2)] = A[(5, 1)] + 1 =  2

s[2] =  1 , z[7] =  0
are unequal, so A[(7, 2)] = max ( A[(6, 2)], A[(7, 1)] ) = max ((2, 1)) =
2

s[2] =  1 , z[8] =  1
 are equal, so A[(8, 2)] = A[(7, 1)] + 1 =  2

s[3] =  0 , z[1] =  1
are unequal, so A[(1, 3)] = max ( A[(0, 3)], A[(1, 2)] ) = max ((0, 1)) =
1

s[3] =  0 , z[2] =  0
 are equal, so A[(2, 3)] = A[(1, 2)] + 1 =  2

s[3] =  0 , z[3] =  0
 are equal, so A[(3, 3)] = A[(2, 2)] + 1 =  2
```

```
s[3] =  0 , z[4] =  1
are unequal, so A[(4, 3)] = max ( A[(3, 3)], A[(4, 2)] ) = max ((2, 2)) =
2

s[3] =  0 , z[5] =  0
 are equal, so A[(5, 3)] = A[(4, 2)] + 1 =  3

s[3] =  0 , z[6] =  1
are unequal, so A[(6, 3)] = max ( A[(5, 3)], A[(6, 2)] ) = max ((3, 2)) =
3

s[3] =  0 , z[7] =  0
 are equal, so A[(7, 3)] = A[(6, 2)] + 1 =  3

s[3] =  0 , z[8] =  1
are unequal, so A[(8, 3)] = max ( A[(7, 3)], A[(8, 2)] ) = max ((3, 2)) =
3

s[4] =  1 , z[1] =  1
 are equal, so A[(1, 4)] = A[(0, 3)] + 1 =  1

s[4] =  1 , z[2] =  0
are unequal, so A[(2, 4)] = max ( A[(1, 4)], A[(2, 3)] ) = max ((1, 2)) =
2

s[4] =  1 , z[3] =  0
are unequal, so A[(3, 4)] = max ( A[(2, 4)], A[(3, 3)] ) = max ((2, 2)) =
2

s[4] =  1 , z[4] =  1
 are equal, so A[(4, 4)] = A[(3, 3)] + 1 =  3

s[4] =  1 , z[5] =  0
are unequal, so A[(5, 4)] = max ( A[(4, 4)], A[(5, 3)] ) = max ((3, 3)) =
3

s[4] =  1 , z[6] =  1
 are equal, so A[(6, 4)] = A[(5, 3)] + 1 =  4

s[4] =  1 , z[7] =  0
are unequal, so A[(7, 4)] = max ( A[(6, 4)], A[(7, 3)] ) = max ((4, 3)) =
4

s[4] =  1 , z[8] =  1
 are equal, so A[(8, 4)] = A[(7, 3)] + 1 =  4

s[5] =  1 , z[1] =  1
 are equal, so A[(1, 5)] = A[(0, 4)] + 1 =  1

s[5] =  1 , z[2] =  0
are unequal, so A[(2, 5)] = max ( A[(1, 5)], A[(2, 4)] ) = max ((1, 2)) =
2
```

```
s[5] =  1 , z[3] =  0
are unequal, so A[(3, 5)] = max ( A[(2, 5)], A[(3, 4)] ) = max ((2, 2)) =
2

s[5] =  1 , z[4] =  1
 are equal, so A[(4, 5)] = A[(3, 4)] + 1 =  3

s[5] =  1 , z[5] =  0
are unequal, so A[(5, 5)] = max ( A[(4, 5)], A[(5, 4)] ) = max ((3, 3)) =
3

s[5] =  1 , z[6] =  1
 are equal, so A[(6, 5)] = A[(5, 4)] + 1 =  4

s[5] =  1 , z[7] =  0
are unequal, so A[(7, 5)] = max ( A[(6, 5)], A[(7, 4)] ) = max ((4, 4)) =
4

s[5] =  1 , z[8] =  1
 are equal, so A[(8, 5)] = A[(7, 4)] + 1 =  5

s[6] =  0 , z[1] =  1
are unequal, so A[(1, 6)] = max ( A[(0, 6)], A[(1, 5)] ) = max ((0, 1)) =
1

s[6] =  0 , z[2] =  0
 are equal, so A[(2, 6)] = A[(1, 5)] + 1 =  2

s[6] =  0 , z[3] =  0
 are equal, so A[(3, 6)] = A[(2, 5)] + 1 =  3

s[6] =  0 , z[4] =  1
are unequal, so A[(4, 6)] = max ( A[(3, 6)], A[(4, 5)] ) = max ((3, 3)) =
3

s[6] =  0 , z[5] =  0
 are equal, so A[(5, 6)] = A[(4, 5)] + 1 =  4

s[6] =  0 , z[6] =  1
are unequal, so A[(6, 6)] = max ( A[(5, 6)], A[(6, 5)] ) = max ((4, 4)) =
4

s[6] =  0 , z[7] =  0
 are equal, so A[(7, 6)] = A[(6, 5)] + 1 =  5

s[6] =  0 , z[8] =  1
are unequal, so A[(8, 6)] = max ( A[(7, 6)], A[(8, 5)] ) = max ((5, 5)) =
5

s[7] =  1 , z[1] =  1
 are equal, so A[(1, 7)] = A[(0, 6)] + 1 =  1

s[7] =  1 , z[2] =  0
are unequal, so A[(2, 7)] = max ( A[(1, 7)], A[(2, 6)] ) = max ((1, 2)) =
2

s[7] =  1 , z[3] =  0
are unequal, so A[(3, 7)] = max ( A[(2, 7)], A[(3, 6)] ) = max ((2, 3)) =
3

s[7] =  1 , z[4] =  1
 are equal, so A[(4, 7)] = A[(3, 6)] + 1 =  4

s[7] =  1 , z[5] =  0
are unequal, so A[(5, 7)] = max ( A[(4, 7)], A[(5, 6)] ) = max ((4, 4)) =
4

s[7] =  1 , z[6] =  1
 are equal, so A[(6, 7)] = A[(5, 6)] + 1 =  5

s[7] =  1 , z[7] =  0
are unequal, so A[(7, 7)] = max ( A[(6, 7)], A[(7, 6)] ) = max ((5, 5)) =
5

s[7] =  1 , z[8] =  1
 are equal, so A[(8, 7)] = A[(7, 6)] + 1 =  6

s[8] =  1 , z[1] =  1
 are equal, so A[(1, 8)] = A[(0, 7)] + 1 =  1

s[8] =  1 , z[2] =  0
are unequal, so A[(2, 8)] = max ( A[(1, 8)], A[(2, 7)] ) = max ((1, 2)) =
2

s[8] =  1 , z[3] =  0
are unequal, so A[(3, 8)] = max ( A[(2, 8)], A[(3, 7)] ) = max ((2, 3)) =
3

s[8] =  1 , z[4] =  1
 are equal, so A[(4, 8)] = A[(3, 7)] + 1 =  4

s[8] =  1 , z[5] =  0
are unequal, so A[(5, 8)] = max ( A[(4, 8)], A[(5, 7)] ) = max ((4, 4)) =
4

s[8] =  1 , z[6] =  1
 are equal, so A[(6, 8)] = A[(5, 7)] + 1 =  5

s[8] =  1 , z[7] =  0
are unequal, so A[(7, 8)] = max ( A[(6, 8)], A[(7, 7)] ) = max ((5, 5)) =
5

s[8] =  1 , z[8] =  1
 are equal, so A[(8, 8)] = A[(7, 7)] + 1 =  6

s[9] =  0 , z[1] =  1
are unequal, so A[(1, 9)] = max ( A[(0, 9)], A[(1, 8)] ) = max ((0, 1)) =
1
```

```
s[9] =  0 , z[2] =  0
 are equal, so A[(2, 9)] = A[(1, 8)] + 1 =  2

s[9] =  0 , z[3] =  0
 are equal, so A[(3, 9)] = A[(2, 8)] + 1 =  3

s[9] =  0 , z[4] =  1
are unequal, so A[(4, 9)] = max ( A[(3, 9)], A[(4, 8)] ) = max ((3, 4)) =
4

s[9] =  0 , z[5] =  0
 are equal, so A[(5, 9)] = A[(4, 8)] + 1 =  5

s[9] =  0 , z[6] =  1
are unequal, so A[(6, 9)] = max ( A[(5, 9)], A[(6, 8)] ) = max ((5, 5)) =
5

s[9] =  0 , z[7] =  0
 are equal, so A[(7, 9)] = A[(6, 8)] + 1 =  6

s[9] =  0 , z[8] =  1
are unequal, so A[(8, 9)] = max ( A[(7, 9)], A[(8, 8)] ) = max ((6, 6)) =
6
```

Final DP array:

```
j  0 1 2 3 4 5 6 7 8 9
i
0 [0 0 0 0 0 0 0 0 0 0]
1 [0 0 1 1 1 1 1 1 1 1]
2 [0 1 1 2 2 2 2 2 2 2]
3 [0 1 1 2 2 2 3 3 3 3]
4 [0 1 2 2 3 3 3 4 4 4]
5 [0 1 2 3 3 3 4 4 4 5]
6 [0 1 2 3 4 4 4 5 5 5]
7 [0 1 2 3 4 4 5 5 5 6]
8 [0 1 2 3 4 5 5 6 6 6]
```

length:

```
6
```

Retrieving the subsequence from the DP array:

```
A[(8, 9)] = A[(7, 8)] + 1, so prepend s[9] = 0
(i,j) = (7, 8)
```

```
A[(7, 8)] != A[(6, 7)] + 1, but A[(7, 8)] = A[(6, 8)], so i--
(i,j) = (6, 8)
A[(6, 8)] = A[(5, 7)] + 1, so prepend s[8] = 1
(i,j) = (5, 7)
A[(5, 7)] = A[(4, 6)] + 1, so prepend s[7] = 1
(i,j) = (4, 6)
A[(4, 6)] = A[(3, 5)] + 1, so prepend s[6] = 0
(i,j) = (3, 5)
A[(3, 5)] != A[(2, 4)] + 1, but A[(3, 5)] = A[(2, 5)], so i--
(i,j) = (2, 5)
A[(2, 5)] = A[(1, 4)] + 1, so prepend s[5] = 1
(i,j) = (1, 4)
A[(1, 4)] = A[(0, 3)] + 1, so prepend s[4] = 1
(i,j) = (0, 3)
[1, 1, 0, 1, 1, 0]
```

Ex. 2

The following (working) Python code implements the recovering of the LCS given the finished DP array. The explanations of the algorithm is in the comments. A note on indexing: in python, lists are indexed starting from 0. But in the lecture, list indices start at 1. To reflect this, the code prints that it is looking at s[j] when it actually looks at s[j-1] internally.

```
def retrieve_subseq(A, s, z):
    '''
    To retrieve afterwards what an LCS is corresponding to the found max
length of any LCS,
    we can keep track of how A[i,j] changes as we decrease i and j:

    starting at (i,j) = (n,m),

    EACH STEP CONSISTS OF TWO CHECKS:

    - if A[i,j] = A[i-1,j-1] + 1, we can construct a LCS by including
s[i],z[j]. So mark both s[i] and z[j] as included, and append this
        recursively to the subsequence of s[0..i-1], z[0..j-1] by moving to
A[i-1,j-1] and investigating that number.

    - if A[i,j] = A[i-1,j-1] + 1 does not hold, either:
        - A[i,j] = A[i,j-1]: this means that the LCS of s[0..i], z[0..j]
does not include z[j]. so decrement j by 1 and don't mark anything
        - A[i,j] = A[i-1,j]: this means that the LCS of s[0..i], z[0..j]
does not include s[i]. so decrement i by 1 and don't mark anything

    do this until either i = 0 or j = 0. In that case, we have considered
all elements of s, or z, respectively.
    So we can stop, because one of the considered sequences is empty, so
there were no other elements from the nonempty sublist included in the LCS

    The indices (i,j) in marked then give all the matching elements s[i],
z[j] in the LCS. This retrieval algorithm decreases at least i or j
```

```
    in each STEP, and one STEP is O(1) since it entails only O(1) conditional checks and O(1) assignments.

    so the retrieval algorithm runs in O(n+m). This is dominated by the O(nm) for the computation of A.
    '''

    lcs = []

    i,j = A.shape
    i-=1
    j-=1
    while i > 0 and j > 0:
        if A[i,j] == A[i-1,j-1] + 1:
            print(f'A[{i,j}] = A[{i-1,j-1}] + 1, so prepend s[{j}] = {s[j-1]}')
            i-= 1
            j-= 1
            lcs.append(s[j-1])
        elif A[i,j] == A[i-1,j]:
            print(f'A[{i,j}] != A[{i-1,j-1}] + 1, but A[{i,j}] = A[{i-1,j}], so i--')
            i-=1
        else:
            print(f'A[{i,j}] != A[{i-1,j-1}] + 1, but A[{i,j}] = A[{i-1,j}], so i--')
            j-=1
        print(f'(i,j) = {i,j}')

    return lcs
```

Ex. 3

a)

The above is a graph where the longest path should be

```
v1 -> v3 -> v4 -> v5
```

With length L = 3. We see that this path is the longest since there are only two paths possible: either via v2 or v3. And the one via v3 is of length 3, the one via v2 is of length 2.

But the algorithm will

```
start at v1: L = 0

pick vj such that j > 1 and j minimal: w=v1, j = 2: L += 1 so L = 1
pick vj such that j > 2 and j minimal: w=v5, j = 5: L += 1 so L = 2
there is no edge out of w: stop: return L = 2
```

b)

The idea is that we have the following recursion: call L[j] the length of the longest path from vj to vn. Then:

at node v_j, L[j] = 1 + max(L[i] : i > j and there is a connection from vj to vi) or, if this set is empty, -1 (a sentinel value) at node v_n, L[n] = 0 (Base case)

For line graphs, the convenient property is that we only need to consider vi with i > j in the max() function. This makes the recursion always arrive at a base case. We can even compute it bottom-up (in other words, "memoize" the recursive cases) by starting at v_n and computing L[i] for i= n-1, n-2, ... 1.

The below code assumes that the graph is passes as a dictionary of adjacency lists, i.e. with entries { vi : [neighbours of vi] }, and that the vertices are just integers, so v_1 = 1 as its name.

```
import numpy as np

def maxPathLineGraph(nbrs):
    n = len(nbrs)
    L = np.empty(shape=(n), dtype=np.int64)
    L[n-1] = 0
    for i in range(n-2, -1, -1):
        max = -1
        for j in nbrs[i]:
            if L[j] > max:
                max = L[j]
        L[i] = max + 1
    return L[0]

graph = dict()
graph[0] = [1,3]
graph[1] = [3,4]
graph[2] = [3]
graph[3] = [4]
graph[4] = []

print(maxPathLineGraph(graph))
```

If you run it, it will indeed show 3 in the terminal, just like we expect from figure 1. Note that my code uses indices starting at 0 for simplicity of implementing the array. The idea is just the same.

I included the computation of max(L[i] : i > j and there is a connection from vj to vi) explicitly, so that you can see that it practically explores all edges from each v in V. We only explore each edge only once, because we only explore it from its source node. We only explore each node once, from v_n-1 to v_1 (in the above code(, from v_n-2 to v_0). The total complexity therefore is O(V + E), which shows that by exploiting the line graph structure (i.e. the fact that there are only forward links from v_i to v_j where j > i), we are able to make a more efficient algorithm than Dijkstra (which may be applied by giving each edge the cost -1 (which is valid because the graph does not contain cycles, so Dijkstra should be able to handle this)), which is O(E + V log V).

Ex. 4

We distinguish the following recursion relation:

Let D[n][k] be defined as the number of distinct numbers of length n that can be dialed from key k in {0,...,9}. Then: since the starting position counts as being dialed: D[1][k] = 1 for all k in {0,...,9}. This gives the base case.

For D[n][k], we consider D[n-1][k'] for all k' that the knight can hop to from key k. The sum over all those numbers of distinct numbers of length n-1 that can be dialed from all reachable k', equals D[n][k] (since all these numbers D[n-1][k'] are counts of disjoint sets, so you can use the sum rule from combinatorics).

The recursion has been found, and we can see that it can be computed bottom-up by iterating over n first, and then for each n computing it for each k. We only need a table to see which keys k' can be reached from k (actually, an adjacency dict is exactly such a table): This can be precomputed, and is:

```
k       k'
--------------------
0       4,6
1       6,8
2       7,9
3       4,8
4       3,9,0
5
6       1,7,0
7       2,6
8       1,3
9       2,4
```

This is just the adjacency list of a symmetric graph, of course! Also, notice how 5 forms its own isolated component in this graph.

The program in python (which, to be fair, is just like pseudocode):

```python
import numpy as np

jumps = dict()
jumps[0]=       [4,6]
jumps[1]    =  [6,8]
jumps[2]     =  [7,9]
jumps[3]   =   [4,8]
```

```
jumps[4]    =   [3,9,0]
jumps[5]     =  []
jumps[6]      = [1,7,0]
jumps[7]     =  [2,6]
jumps[8]     =  [1,3]
jumps[9]     = [2,4]


def knightDialing(N, start):
    '''
    Computes the number of distinct numbers that a chess knight jumping on
the keyboard

    1   2   3
    4   5   6
    7   8   9
        0

    can dial when its first dialed number is `start` and it is allowed to
dial an N-digit number (i.e. it must still make N-1 jumps)
    '''
    D = np.empty(shape=(N+1,10), dtype = np.int64)
    for k in range(10):
        D[1,k] = 1
    for n in range(2,N+1):
        for k in range(10):
            num = 0
            for dest in jumps[k]:
                num += D[n-1, dest]
            D[n,k] = num
    return D[N,start]


# Try it yourself! Should print 5
print(knightDialing(3,1))
```

The recursion has a complexity of O(N), since we have to compute every D[n,k'] for 1 <= n <= N and k in {0,..9}. But since {0,..,9} is a constant, we can leave it out of the complexity analysis. The summation of num is also `O(|jump[k]|)` for each `k`, but we don't take this into account either because it is a constant bounded above by 3. So O(N) is the complexity.