Week 4 Live Coding Solutions

Week 4 - Live Coding Problem 1

Shortest Path

The express train routes are provided in the adjacency list `AList`, here you have to find the route from `start` to `end` with minimum number of possible. Write a function `minimumhops(AList, start, end)` to return the cities to be visited starting from `start` to `end`. Return a list with only `start` if the `end` is not reachable.

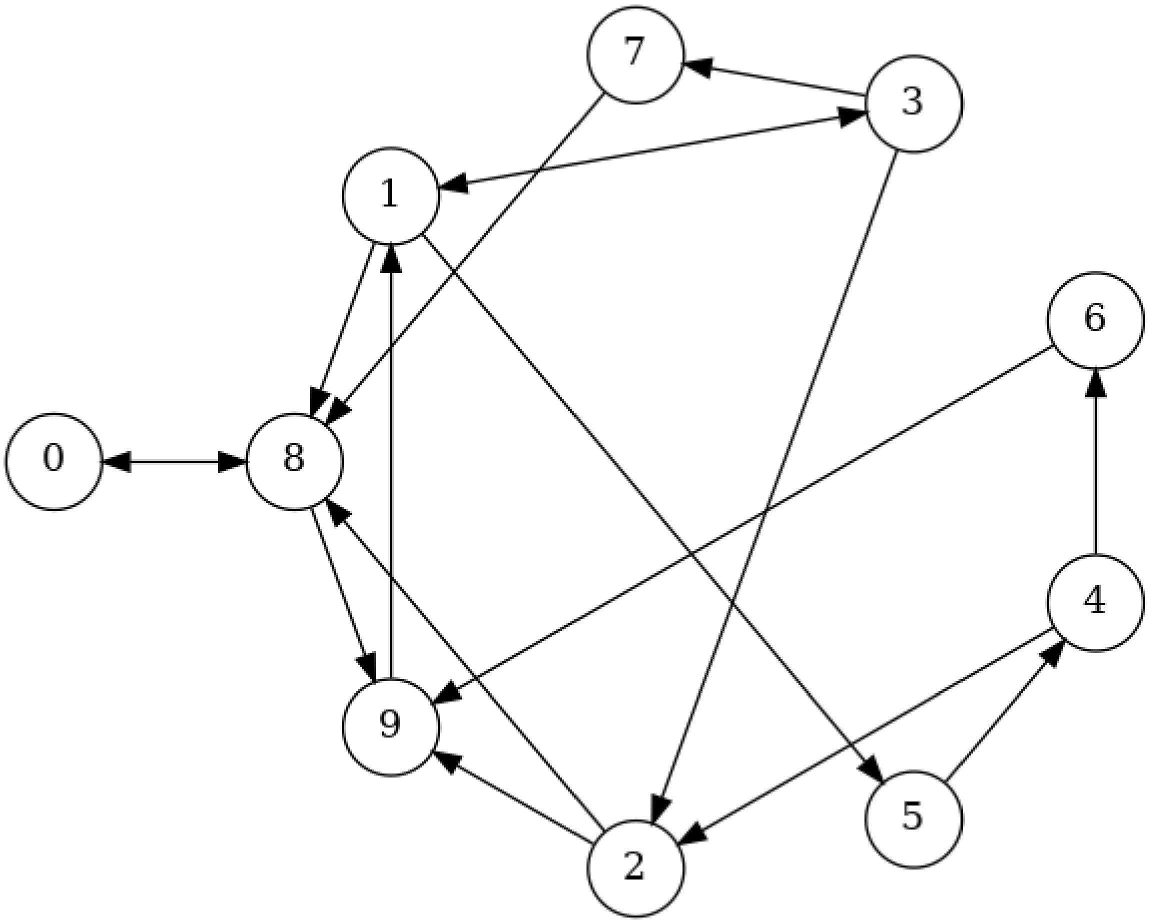
Sample Input

```
start = 8
end = 7
AList = {
            0: [8],
            8: [0, 9],
            1: [3, 5, 8],
            3: [1, 7, 2],
            5: [4],
            2: [8, 9],
            9: [1],
            7: [8],
            4: [2, 6],
            6: [9]
    }
```

Sample output

```
[8, 9, 1, 3, 7]
```

Graphical representation



Solution Code

```python
def BFSListPathLevel(AList, v):
    level, parent = {}, {}
    for i in AList.keys():
        level[i] = -1
        parent[i] = -1
    q = []

    level[v] = 0
    q.append(v)

    while len(q) > 0:
        j = q.pop(0)
        for k in AList[j]:
            if level[k] == -1:
                level[k] = level[j]+1
                parent[k] = j
                q.append(k)
    return level, parent

def minimumhops(AList, start, end):
    level, path = BFSListPathLevel(AList, start)
    shortestpath = []
    if level[end] != -1:
        shortestpath.append(end)
        while shortestpath[-1] != start:
            end = path[end]
            shortestpath.append(end)
    else:
        shortestpath.append(start)
    return shortestpath[::-1]

```

Suffix code(Visible)

```
start = int(input())
end = int(input())
AList = eval(input())
shortestpath = minimumhops(AList, start, end)
print(shortestpath)
```

Public Test case

Input 1

```
8
0
{0: [8], 8: [9], 1: [3, 5, 8], 3: [1, 7, 2], 5: [4], 2: [8, 9], 9: [1], 7: [8], 4: [2, 6], 6: [9]}
```

Output

```
[8]
```

Input 2

```
8
7
{0: [8], 8: [0, 9], 1: [3, 5, 8], 3: [1, 7, 2], 5: [4], 2: [8, 9], 9: [1], 7: [8], 4: [2, 6], 6: [9]}
```

Output

```
[8, 9, 1, 3, 7]
```

Input 3

```
0
1
{0: [1], 1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: [7], 7: [8], 8: [9], 9: [0]}
```

Output

```
[0, 1]
```

Private Test case

Input 1

```
1
0
{0: [1], 1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: [7], 7: [8], 8: [9], 9: [0]}
```

Output

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

Input 2

```
1
0
{0: [1], 1: [2, 3], 2: [3, 4], 3: [4, 5], 4: [5, 6], 5: [6, 7], 6: [7, 8], 7: [8, 9], 8: [9, 0], 9: [0]}
```

Output

```
[1, 2, 4, 6, 8, 0]
```

Input 3

```
1
0
{0: [1], 1: [2, 3, 4], 2: [3, 4, 5], 3: [4, 5, 6], 4: [5, 6, 7], 5: [6, 7, 8], 6: [7, 8, 9], 7: [8, 9, 0], 8: [9, 0], 9: [0]}
```

Output

```
[1, 4, 7, 0]
```

Week 4 - Live Coding Problem 2

Back and Forth

Write a function `backandforth(AList, end1, end2)` to return the maximum number of possible route between node `end1` and node `end2` in the undirected graph without going through the same node again with exception to `end1` and `end2`. The connectivity details between nodes are provided by the adjacency list `AList`.

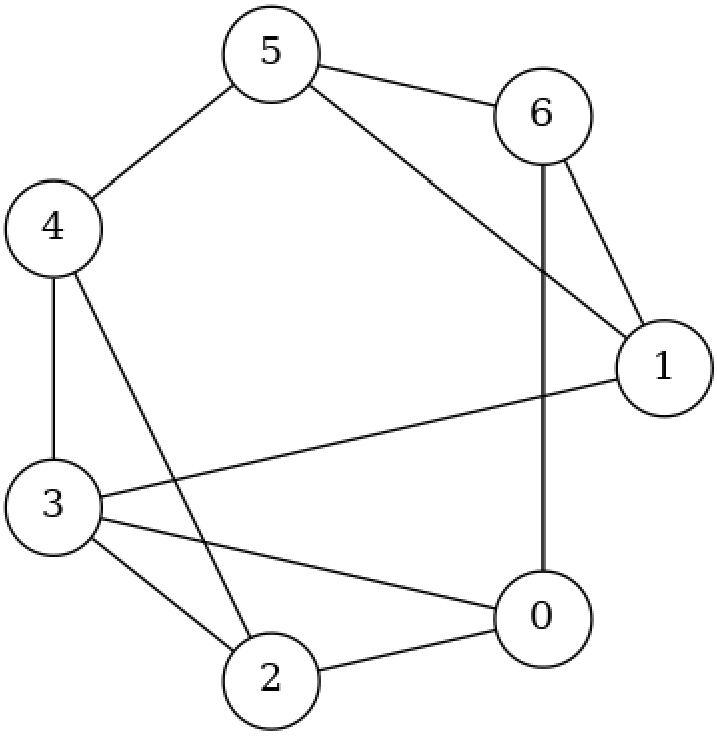
Sample Input

```
AList = {
 0: [2, 3, 6],
 2: [0, 3, 4],
 3: [4, 2, 0, 1],
 6: [1, 5, 0],
 1: [3, 6, 5],
 4: [2, 3, 5],
 5: [1, 4, 6]
 }
end1 = 0
end2 = 1
```

Sample Output

```
3
```

Graphical Representation



Explanation

The possible paths are [0, 3, 1], [1, 6, 0] and [0, 2, 4, 5, 1]. Hence the answer is 3.

Solution Code

```python
def BFSListPath(AList, v, preventionList):
  visited, parent = {}, {}
  for i in AList.keys():
    visited[i] = False
    parent[i] = -1
  q = []

  visited[v] = True
  q.append(v)

  while len(q) > 0:
    j = q.pop(0)
    for k in AList[j]:
      if not visited[k] and not k in preventionList:
        visited[k] = True
        parent[k] = j
        q.append(k)
  return visited, parent

def findpath(parent, start, end):
    L = []
    curr = parent[end]
    while curr != start:
        L.append(curr)
        curr = parent[curr]
    return L

def backandforth(AList, end1, end2):
    preventionList = []
    c = 0
    visited, parent = BFSListPath(AList, end1, preventionList)
    while visited[end2]:
        c += 1
        path = findpath(parent, end1, end2)
        preventionList.extend(path)
        visited, parent = BFSListPath(AList, end1, preventionList)
    return c

```

Suffix Code

```python
end1 = int(input())
end2 = int(input())

AList = {}

while True:
    line = input()
    if line.strip() == '':
        break
    u, vs = line.strip().split(':')
    u = int(u)
    AList[u] = []
    for v in vs.strip().split():
        v = int(v)
        if v not in AList:
            AList[v] = []
        AList[u].append(v)

print(backandforth(AList, end1, end2))
```

Public Test case

Input 1

```
0
1
0 : 2 3 6
1 : 3 5 6
2 : 0 3 4
3 : 0 1 2 4
4 : 2 3 5
5 : 1 4 6
6 : 0 1 5


```

Output

```
3
```

Input 2

```
0
1
0 : 2 3 6
1 : 3 5 6 8
2 : 0 3 4 7
3 : 0 1 2 4 7
4 : 2 3 5 7
5 : 1 4 6 8
6 : 0 1 5
7 : 2 3 4
8 : 1 5


```

Output

```
3
```

Input 3

```
0
1
0 : 2 3 4 5 6 7 8 9 10 11 12
1 : 2 3 4 5 6 7 8 9 10 11 12
2 : 0 1
3 : 0 1
4 : 0 1
5 : 0 1
6 : 0 1
7 : 0 1
8 : 0 1
9 : 0 1
10 : 0 1
11 : 0 1
12 : 0 1


```

Output

```
11
```

Private Test case

Input 1

```
0
1
0 : 2 3 4 5 6 7 8 9 10 11 12
1 : 2 3 4 5 6 7 8 9 10 11 12
2 : 0 1
3 : 0 1
4 : 0 1
5 : 0 1
6 : 0 1
7 : 0 1
8 : 0 1
9 : 0 1
10 : 0 1
11 : 0 1
12 : 0 1


```

Output

```
11
```

Input 2

```
0
1
0 : 2
1 : 12
2 : 0 3
3 : 2 4
4 : 3 5
5 : 4 6
6 : 5 7
7 : 6 8
8 : 7 9
9 : 8 10
10 : 9 11
11 : 10 12
12 : 1 11


```

Output

```
1
```

Input 3

```
0
1
0 : 2 3
1 : 11 12
2 : 0 3 12
3 : 0 2 4
4 : 3 5
5 : 4 6
6 : 5 7
7 : 6 8
8 : 7 9
9 : 8 10
10 : 9 11
11 : 1 10 12
12 : 1 2 11


```

Output

```
2
```

Week 4 - Live Coding Problem 3

Cool Worker

A group of workers have to complete a list of tasks, those tasks have dependencies within the task list. But the workers prefer some interesting task and hates to do some boring task. They always do the most interesting one among the available tasks to be done.

Write a function `coolWorkers(AList, preference)` to return the order in which the tasks will be done. `AList` is the adjacency list with the dependencies and `preference` is the tasks sorted in preferred order, in which task in index `0` is the most preferred and index `-1`(last element) be the least preferred.

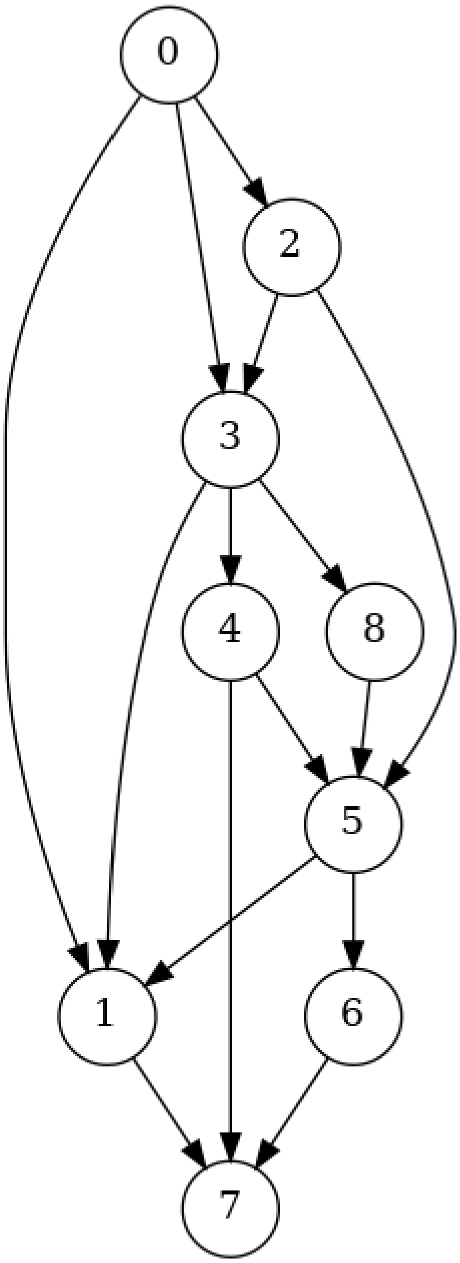
Sample Input

```
ALīst = {0: [1, 2, 3],
         1: [7],
         2: [3, 5],
         3: [4, 1, 8],
         7: [],
         5: [6, 1],
         4: [5, 7],
         8: [5],
         6: [7]}
preference = [1, 3, 2, 6, 8, 5, 4, 0, 7]
```

Sample Output

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

Graphical representation



Solution Code

```python
def coolWorkers(AList, preference):
    n = len(AList.keys())
    indegree = {}
    toposortlist = []
    for i in AList.keys():
        indegree[i] = 0
    for u in AList.keys():
        for v in AList[u]:
            indegree[v] = indegree[v] + 1

    for i in range(n):
        availableTasks = [k for k in AList if indegree[k] == 0]
        t = [(preference.index(i), i) for i in availableTasks]
        t.sort()
        j = t[0][1]
        toposortlist.append(j)
        indegree[j] = indegree[j]-1
        for k in AList[j]:
            indegree[k] -= 1
    return toposortlist
```

Suffix Code(Visible)

```
AList = eval(input())
preference = eval(input())
print(coolWorkers(AList, preference))
```

Public Test case

Input 1

```
{0: [1, 2, 3], 1: [7], 2: [3, 5], 3: [4, 1, 8], 7: [], 5: [6, 1], 4: [5, 7], 8: [5], 6: [7]}
[1, 3, 2, 6, 8, 5, 4, 0, 7]
```

Output

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

Input 2

```
{0: [1, 2, 3], 1: [7], 2: [3, 5], 3: [5, 1], 7: [], 5: [6, 1], 4: [5, 7], 6: [7], 8: [5]}
[0, 8, 7, 5, 6, 1, 3, 2, 4]
```

Output

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

Private Test case

Input 1

```
{0: [1, 2, 3], 1: [7], 2: [3, 5], 3: [5, 1], 7: [], 5: [6, 1], 4: [5, 7], 6: [7], 8: [5]}
[2, 4, 3, 1, 6, 0, 7, 5, 8]
```

Output

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

Input 2

```
{0: [8], 8: [], 1: [8], 2: [8], 3: [8], 4: [8], 5: [8], 6: [8], 7: [8]}
[5, 6, 4, 1, 0, 3, 7, 2, 8]
```

Output

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