Laboratory Assignment 4

You can use “hashlib” or “collections” libraries. Please do not use any other libraries. All unspecified library lines will be suppressed and if your code fails, your submission will be graded “0” points

Problem 1 – Hashing [5 pts]

Write a function that takes a string that denotes a filename as an input and returns the hash table that hold the plate and properties of cars listed in the file with the filename. You can use `hashlib` library.

- Function parameter name does not matter, but the function should expect 1 parameter that is the name of the file.
- The lines of the file represent the plate, color, engine type, and capacity of the vehicles:
  ```
  '09 ES 715    yellow    diesel    1.0'
  ```
- You have to return a list that is the size of 4096, and the elements of the list should be AVL Trees. You have to use avlgtu.py and bstgtu.py files. The modules with same file name will be used to grade and test your functions. Do not change the name of the files and be sure your naming is correct. **These files shouldn’t be submitted as separate files or shouldn’t be included into your submitted file.** If you copy these files into your submission or use any other tree modules, your submission will not be graded.
- Every element of the AVL tree that is a BST node, should store the following information:
  ```
  BSTnode.key    = '09ES715'
  BSTnode.color  = 'yellow'
  BSTnode.fuel   = 'diesel'
  BSTnode.engine = '1.0'
  ```
  Note that key of the node is the plate without spaces, all variables are string.
- To determine the index of the list that the plate belongs to, you have to hash the plate without space using md5 and utf-8 encoding. Then take the least significant 3 hexadecimal digits of the hash, find the decimal conversion, insert the plate to the AVL tree as a node, set all information. For example,
- Plate : `'09ES715'`
- Hexadecimal hash: `'80bc8174b6c0859694f748b138c682ff'`
- Decimal conversion of last 3 digits: 767
- This plate should be insert to the AVL tree that is located in 767th element of the list.
- `input  :`
  - `filename = string`
- `output : x = { x : x is the list of size 4096 with elements AVL tree}`

```
>>> CarList=problem1('platelist.txt')
>>> print(len(CarList))
4096
```

```
>>> print(CarList[100])
20JL573
/     \
   40YC0658
   /      \
>>> node=CarList[100].root.right
>>> print(node)
<BST Node, key:40YC0658>
>>> print(node.color)
red
>>> print(node.engine)
3.0
>>> print(node.fuel)
diesel
>>> print(node.key)
40YC0658

Explanation: The md5 hash of '40YC0658' is 'e6f1cd2e6e4585613960a95b897ea064',
and last 3 digits of the hash is 100 in decimal representation.
```

Problem 2 – Graph - Undirected `[5 pts]`

Write a function that takes list of vertices and edges as inputs and returns the adjacency of an **undirected** graph as an object.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter is a list of vertices, e.g.
  `['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm']`
- Second parameter is a list of lists that represents an edge between vertices, e.g.
  `[['a', 'c'], ['c', 'i'], ['b', 'c'], ['i', 'k'], ['b', 'k'], ['k', 'm'], ['m', 'j'], ['j', 'f'], ['j', 'g'], ['f', 'd'], ['l', 'f'], ['l', 'h'], ['g', 'l'], ['l', 'e'], ['a', 'd']]`
- Function returns a graph object that has two features, i.e.
  - `self.add_edge(u,v)`: takes vertices of an edge u and `v,` and add them to a dictionary that holds adjacency relation as key and value pairs.
  - `self.adj[u]`: a dictionary that holds the neighbors of vertex u.
- Hint: You can take a look at the Recitation 13 document.

```
>>> Vertices= ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm']
>>> Edges=[['a', 'c'], ['c', 'i'], ['b', 'c'], ['i', 'k'], ['b', 'k'], ['k',
'm'], ['m', 'j'], ['j', 'f'], ['j', 'g'], ['f', 'd'], ['l', 'f'], ['l', 'h'],
['g', 'l'], ['l', 'e'], ['a', 'd']]
>>> G1=problem2(Vertices,Edges)
>>> print(G1.adj)
{'a': ['c', 'd'], 'c': ['a', 'i', 'b'], 'i': ['c', 'k'], 'b': ['c', 'k'], 'k':
['i', 'b', 'm'], 'm': ['k', 'j'], 'j': ['m', 'f', 'g'], 'f': ['j', 'd', 'l'],
'g': ['j', 'l'], 'd': ['f', 'a'], 'l': ['f', 'h', 'g', 'e'], 'h': ['l'], 'e':
['l']}
```

Problem 3 – Graph - Directed [5 pts]

Write a function that takes list of vertices and edges as inputs and returns the adjacency of a **directed** graph as an object.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter is a list of vertices, e.g.
  `['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm']`
- Second parameter is a list of lists that represents an edge between vertices, e.g.
  `[['a', 'c'], ['c', 'i'], ['b', 'c'], ['i', 'k'], ['b', 'k'], ['k', 'm'], ['m', 'j'], ['j', 'f'], ['j', 'g'], ['f', 'd'], ['l', 'f'], ['l', 'h'], ['g', 'l'], ['l', 'e'], ['a', 'd']]`
- Function returns a graph object that has two features, i.e.
  - `self.add_edge(u,v)`: takes node of an edge u and v, and add them to a dictionary that holds adjacency relation as key and value pairs.
  - `self.adj[u]`: a dictionary that holds the neighbors of node u.
- If a vertex does not points another vertex, then adjacency of the vertex should be `None`.
- Hint: You can take a look at the Recitation 13 document.

```
>>> G2=problem3(Vertices,Edges)
>>> print(G2.adj)
{'a': ['c', 'd'], 'c': ['i'], 'i': ['k'], 'b': ['c', 'k'], 'k': ['m'], 'm':
['j'], 'j': ['f', 'g'], 'f': ['d'], 'g': ['l'], 'd': None, 'l': ['f', 'h',
'e'], 'h': None, 'e': None}
```

Problem 4 – Breadth First Search [5 pts]

Write a function that takes an adjacency array of a graph object and a vertex as an input, and returns the breadth first tree as an object. Input graph can be both directed or undirected graph.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter is graph object.
- Second parameter is a vertex in that graph.
- Function returns a breadth first tree object that has two features, i.e.
  - `self.level[u]`: a dictionary that holds the level of the vertex u w.r.t. input vertex.
  - `self.parent[u]`: a dictionary that holds the parent of vertex u.
- If a vertex does not have a parent, then parent of the vertex should be `None`.
- Hint: You can take a look at the Recitation 13 document.
- You can use `deque` from `collections` module.

```
>>> B1 = problem4(G1,'a')
>>> print(B1.level)
{'a': 0, 'c': 1, 'd': 1, 'i': 2, 'b': 2, 'f': 2, 'k': 3, 'j': 3, 'l': 3, 'm':
4, 'g': 4, 'h': 4, 'e': 4}
>>> print(B1.parent)
```

```
{'a': None, 'c': 'a', 'd': 'a', 'i': 'c', 'b': 'c', 'f': 'd', 'k': 'i', 'j':
'f', 'l': 'f', 'm': 'k', 'g': 'j', 'h': 'l', 'e': 'l'}

>>> B2 = problem4(G2,'a')
>>> print(B2.level)
{'a': 0, 'c': 1, 'd': 1, 'i': 2, 'k': 3, 'm': 4, 'j': 5, 'f': 6, 'g': 6, 'l':
7, 'h': 8, 'e': 8}
>>> print(B2.parent)
{'a': None, 'c': 'a', 'd': 'a', 'i': 'c', 'k': 'i', 'm': 'k', 'j': 'm', 'f':
'j', 'g': 'j', 'l': 'g', 'h': 'l', 'e': 'l'}
```

Problem 5 – Path Finder `[5 pts]`

Write a function that takes list of vertices and edges, a source vertex, an observation vertex, and a flag for directed/undirected graph, as inputs, and returns the ordered vertices that is the path between the source vertex and observation vertex as a list.

- Function parameter names do not matter, but the function should expect 5 parameters.
- First parameter is a list of vertices.
- Second parameter is a list of lists that represents an edge between vertices.
- Third parameter is the source vertex `s`, which is already in the list of vertices.
- Fourth parameter is the observation vertex `o`, which is already in the list of vertices.
- Fifth parameter is a string, `'u'` for undirected graph (default), `'d'` for directed graph. If it is not one of `'u'` and `'d'`, function should return `None`.
- Returns a list that has the vertices ordered from `'s'` to `'o'` (including them).
- Notes:
    - Function does not necessarily have to find the shortest path between the source and observation vertices.
    - The separate graphs won’t be tested.
    - If there is no path between the source and observation vertices, function has to return `'INF212'`.
    - The graph type should be undirected by default.
    - The order should start from `'s'` and ends with `'o'`.
- You can use `deque` from `collections` module.

```
>>> P1 = problem5(Vertices,Edges,'a','j','u')
>>> print(P1)
['a', 'd', 'f', 'j']
>>> P2 = problem5(Vertices,Edges,'a','j','d')
>>> print(P2)
['a', 'c', 'i', 'k', 'm', 'j']
>>> P3 = problem5(Vertices,Edges,'a','j','s')
>>> print(P3)
None
>>> P4 = problem5(Vertices,Edges,'d','a','d')
>>> print(P4)
INF212
```