Complementary Assignment 2

Please Note: This assignment is **not compulsory** and will **not be graded**.The answers will be released later for your reference.

Question 1: Graph

A graph in computer science is a structure consisting of vertices and edges. Figure. 1 shows an example of a graph, where the nodes “a”, “b”, “c”, “d”, “e”, “f” are vertices and the lines connect the nodes are the edges, written as {a, b}, {a,c}, {a, d}, {b,d}, {c, d} and {e,f}.

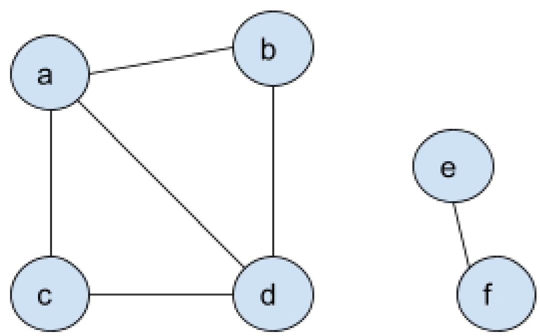

Fig. 1 An undirected graph which has 6 nodes and 6 edges.

Graphs are widely used in modeling. For example, a social network can be represented by a graph. Each person is represented by a vertice, and there is an edge between two vertices if the two corresponding persons know each other.

We want to create a class `Graph` which represents an undirected graph. In general, this class has two parts: the information about a graph, which is stored in the form of a Python dictionary, and some operations on the graph such as adding vertex or edges. Let’s take Fig.1 as an example. In the `Graph` class, the graph in Fig.1 is stored in a dictionary, like

```
fig1 = {"a":["b", "c", "d"],
        "b":["a", "d"],
        "c":["a", "d"],
        "d":["a", "b","c"],
        "e":["f"],
        "f":["e"]}
```

The keys are the nodes of the graph, and the values are lists of nodes that are connected to the keys by one edge. You need to complete the `graph_student.py`. (Note: the examples given are continued, which means the second example is based on the first one, and so on.)

(a) The Graph class contains the following methods. (5 points per method)

__init__()	The code is given. It initializes an instance of `Graph`, assigning the input graph to `self._graph_dict`. Note: you are not allowed to modify this method. `In [59]: graph_fig1 = Graph(fig1)` `In [60]: graph_fig1._graph_dict` `Out[60]:` `{'a': ['b', 'c', 'd'],` ` 'b': ['a', 'd'],` ` 'c': ['a', 'd'],` ` 'd': ['a', 'b', 'c', 'e'],` ` 'e': ['f', 'd'],` ` 'f': ['e']}`
vertices()	You need to implement this method. It returns a list of all vertices. `In [61]: graph_fig1.vertices()` `Out[61]: ['a', 'b', 'c', 'd', 'e', 'f']`
neighbors()	You need to implement this method. It takes a vertex as the argument and returns a list of all the vertices that connect to it by one edge. If the vertex is not in the graph, it prints "the vertex is not in the graph." and returns nothing. (see the following example). `In [11]: graph_fig1.neighbors("a")` `Out[11]: ['b', 'c', 'd']` `In [12]: graph_fig1.neighbors("z")` `z is not in the graph.`
edges()	You need to implement this method. It returns a list of all edges; each edge is a **set** of two vertices. Note: {'a', 'b'} is equal to {'b', 'a'}, and in your output, the edges may have different order to the following example, which is fine. `In [83]: graph_fig1.edges()` `Out[83]: [{'a', 'b'}, {'a', 'c'}, {'a', 'd'},` `{'b', 'd'}, {'c', 'd'}, {'e', 'f'}]`
add_vertex()	You need to implement this method. It takes a vertex as the argument and adds it into `self._graph_dict` if it is not in the graph. If the vertex has been in the graph already, it prints "the vertex has already been in the graph.".(see the following example).

	`In [69]: graph_fig1.add_vertex("g")` `In [70]: graph_fig1.vertices()` `Out[70]: ['a', 'b', 'c', 'd', 'e', 'f', 'g']` `In [71]: graph_fig1.add_vertex("a")` `a has already been in the graph.`
add_edge()	You need to implement this method. It takes two vertices as arguments and adds them in to `self._graph_dict` as values to the corresponding keys. If any of the vertices is not in the graph, it will first add the vertex into the graph and then add the edge. `In [84]: graph_fig1.add_edge("d", "e")` `In [85]: graph_fig1.edges()` `Out[85]:` `[{'a', 'b'},` ` {'a', 'c'},` ` {'a', 'd'},` ` {'b', 'd'},` ` {'c', 'd'},` ` {'d', 'e'},` ` {'e', 'f'}]` `In [6]: # When "g" is not in the graph,` `In [7]: graph_fig1.add_edge("f", "g")` `In [8]: graph_fig1.edges()` `Out[8]:` `[{'a', 'b'},` ` {'a', 'c'},` ` {'a', 'd'},` ` {'b', 'd'},` ` {'c', 'd'},` ` {'d', 'e'},` ` {'e', 'f'},` ` {'f', 'g'}]`
remove_edge()	You need to implement this method. It takes two vertices as arguments and removes the edge between them. If the edge does not exist, then it does nothing.

	`` In [22]: graph_fig1.remove_edge("f", "g") In [23]: graph_fig1.edges() Out[23]: [{'a', 'b'}, {'a', 'c'}, {'a', 'd'}, {'b', 'd'}, {'c', 'd'}, {'d', 'e'}, {'e', 'f'}] In [92]: graph_fig1.remove_edge("d", "e") In [93]: graph_fig1.edges() Out[93]: [{'a', 'b'}, {'a', 'c'}, {'a', 'd'}, {'b', 'd'}, {'c', 'd'}, {'e', 'f'}] In [31]: #When there is no edge between two vertices In [32]: graph_fig1.remove_edge("d", "g") In [33]: graph_fig1.edges() Out[33]: [{'a', 'b'}, {'a', 'c'}, {'a', 'd'}, {'b', 'd'}, {'c', 'd'}, {'e', 'f'}] ```
remove_vertex()	You need to implement this method. It takes a vertex as the argument and removes it from the graph. Note: it also removes edges containing the vertex. If the vertex is not in the graph, then it does nothing. ``` In [36]: #"g"is an isolated vertex In [37]: graph_fig1.remove_vertex('g') In [38]: graph_fig1.vertices() Out[38]: ['a', 'b', 'c', 'd', 'e', 'f'] In [39]: graph_fig1.edges() Out[39]: [{'a', 'b'}, {'a', 'c'}, {'a', 'd'}, {'b', 'd'}, {'c', 'd'}, {'e', 'f'}] ```

	`In [40]: #"a" is connected to other vertices` `In [41]: graph_fig1.remove_vertex('a')` `In [42]: graph_fig1.vertices()` `Out[42]: ['b', 'c', 'd', 'e', 'f']` `In [43]: graph_fig1.edges()` `Out[43]: [{'b', 'd'}, {'c', 'd'}, {'e', 'f'}]` `In [51]: #When 'z' is not in the graph` `In [52]: graph_fig1.remove_vertex('z')` `In [53]: graph_fig1.vertices()` `Out[53]: ['b', 'c', 'd', 'e', 'f']`
__str__()	The code is given. It overrides the inherited __str__() method, printing the nodes and their neighbors.

(b) Contact tracing is an effective public health measure for the control of COVID-19. Graph models are often used in this task. The file `contacts.txt` contains the records of the contacts of a local community. We want to build an undirected graph model for tracing contacts. In the graph, each member is represented as a vertice, and if two members have contacts, then, there is an edge between them. You need to write **two functions**.

(i) The first one is `load_graph()` which loads the records in `contacts.txt` and returns an instance of `Graph` class. Note: In `contacts.txt`, each line represents the contacts of a person. The first character is the person's ID, and the following characters are the IDs of persons whom he/she contacted with. (5 points)

(ii) The second one is `trace_contact()`. Given a graph **g** and vertex **v**, this function returns all vertices in **g** that are connected with **v** by edges (including **v** itself). Let's take Fig.2 as an example. Given a vertex `"b"`, the function returns `["a", "b", "c", "d"]`, and given a vertex `"e"`, it returns `["e", "f", "g", "h"]`. Such a set of vertices is also called a component. (10 points)

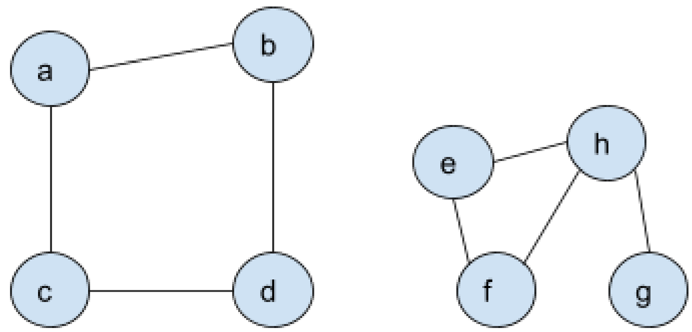

Fig. 2 An undirected graph which has 8 nodes and 8 edges.

We can trace the contacts of a member “x” by the following steps:
Step 1. Put “x” into the component.
Step 2. Find the neighbors of “x”;
Step 3. Trace the contacts of every neighbor: If a neighbor “y” is not in the component, then go to Step 1, replace “x” by “y” and go on with Step 2 to find the neighbors of “y”, and so on. Otherwise, pick another neighbor and check if it is in the component or not.
Step 4. Stop if every neighbor of “x” is traced, and now, the component is filled by all the members that connect to “x” via one (or several) edge(s).

load_graph()	You need to implement this function. It takes a filename as the argument and returns an instance of Graph class. `In [10]: graph_fig2 = load_graph("contacts.txt")` `In [11]: print(graph_fig2)` `{'a': ['b', 'c'], 'b': ['a', 'd'], 'c': ['a', 'd'], 'd': ['b', 'c'], 'e': ['f', 'h'], 'f': ['e', 'h'], 'h': ['e', 'f', 'g'], 'g': ['h']}`
trace_contact()	You need to implement this function. It takes one graph, one vertex, and an empty list (named as `component` in the starting code) as the arguments and returns the `component` which should contain all the vertices that connect to the vertex.

```
In [9]: component = []

In [10]: trace_contact(graph_fig2, "a", component)
Out[10]: ['a', 'b', 'd', 'c']

In [11]: graph_fig2.add_edge("d", "e")

In [12]: component = []

In [13]: trace_contact(graph_fig2, "a", component)
Out[13]: ['a', 'b', 'd', 'c', 'e', 'f', 'h', 'g']
```

Question 2: Maximizing the profit

You want to invest 4 million dollars in the securities market. There are three portfolios (P1, P2, and P3) available and their investment/profits are listed in the following table. For example, if you invest all the 4 million dollars in P1, you will have 19 million.

Portfolio	Amount of investment and profit (million)				
	0	1	2	3	4
P1	0	13	16	17	19
P2	0	12	14	16	18
P3	0	18	19	20	20

(a) What is the maximum profit you can obtain? [Hint: you may use recursion, or dynamic programming]