COS10003 Assignment 3

Student Name: Vi Luan Dang

Student ID: 103802759

Counting:

Question 1

a/

i) There are 5 jobs during the day, from these activities we will need to form different activity patterns. As every activity is only undertaken at most once a day, and the order of activity will matter. Therefore, we will use Permutation, which is a way of arranging a set of $\boldsymbol{r}$ objects from a larger set of $\boldsymbol{n}$ objects that order will also matter.

⇨ $\boldsymbol{P(5,5) = \frac{5!}{(5-5)!} = \frac{5!}{0!} = \frac{5*4*3*2*1}{1} = 120\ (patterns)}$.

Therefore, there are 120 patterns that can be formed from those five activities

ii) Among the above patterns, if we are to choose patterns that only contain only three types (e.g., clean – sit – read). Meaning that we will need to choose pattern that arrange a set of **3** activities from a larger set of **5** activities that order will also matter. We will also use Permutation for this question.

⇨ $\boldsymbol{P(5,3) = \frac{5!}{(5-3)!} = \frac{5!}{2!} = \frac{5*4*3*2*1}{2*1} = 60\ (patterns)}$.

Therefore, there are 60 patterns that can be formed from only three activities.

iii) The number of patterns with at least four different activities that start with playing games would mean that the position of "playing games" is fixed, and we will have two scenarios:

In case that there are only four different activities and "playing games" is fixed, we will need to arrange a set of 3 activities from a larger set of 4 activities that order will also matter. We will use Permutation for this case:

⇨ $\boldsymbol{P(4,3) = \frac{4!}{(4-3)!} = \frac{4!}{1!} = \frac{4*3*2*1}{1} = 24\ (patterns)}$.

In case that there are five different activities and "playing games" is fixed, we will need to arrange a set of 4 activities from a set of 4 activities that order will also matter. We will use Permutation for this case:

⇨ $\boldsymbol{P(4,4) = \frac{4!}{(4-4)!} = \frac{4!}{0!} = \frac{4*3*2*1}{1} = 24\ (patterns)}$.

Therefore, the possible patterns with at least four different activities start with playing games will be: $P(4,3) + P(4,4)$ = *24 + 24 = 48 (patterns)*

b/

i) There are six ingredients in the fridge that need to be used up, this means that we will need a way of choosing a set of $\boldsymbol{r}$ objects from a set of $\boldsymbol{n}$ objects ;however, the arrangement of ingredients, or in other words, the order of ingredients will not matter in this case. Therefore, we will use combination for this question.

As the question does not mention how many times we are to cook these ingredients, there is only 1 way of combining all 6 ingredients in a combination when order does not matter, that is:

⇨ $\boldsymbol{C(6,6) = \frac{6!}{(6-6)!*6!} = \frac{6!}{0!*6!} = \frac{6!}{6!} = 1\ (way)}$.

Therefore, there is only 1 way of combining all the ingredients so that it is used up.

ii)

In this question we are asked to use only three ingredients, we will need a way of choosing a set of **3** ingredients from a set of **6** ingredients and the order of ingredients will not matter either. Therefore, we will use combination for this question.

⇨ $\boldsymbol{C(6,3) = \frac{6!}{(6-3)!*3!} = \frac{6!}{3!*3!} = \frac{1*2*3*4*5*6}{(1*2*3)*(1*2*3)} = \frac{720}{24} = 20\ (ways)}$.

Therefore, there are 20 ways to use three ingredients only.

c/

i) To sum up the information provided in this question, we have 12 different LEGO minifigures, we have 4 different locations around the house, and we are to place three minifigures at each location.

Although the minifigures are different, this will not affect the ordering of the figures themselves. Therefore, the order of figures will not matter. In order to solve this problem, we will need to find a way of choosing a set of 3 objects from a set of 12 objects and the order will not matter. We will use combination for this question, specifically:

The arrangements of minifigures in the first location are as follows:

$$\boldsymbol{C(12,3) = \frac{12!}{(12-3)! * 3!} = \frac{12!}{9! * 3!} = 220\ (ways).}$$

The arrangements of minifigures in the second location are as follows:

$$\boldsymbol{C(9,3) = \frac{9!}{(9-3)! * 3!} = \frac{9!}{6! * 3!} = 84\ (ways).}$$

Similarly, the arrangements of minifigures in the third and fourth location are as follows:

$$\boldsymbol{C(6,3) = \frac{6!}{(6-3)! * 3!} = \frac{6!}{3! * 3!} = 20\ (ways).}$$

$$\boldsymbol{C(3,3) = \frac{3!}{(3-3)! * 3!} = \frac{3!}{0! * 3!} = 1\ (way).}$$

⇨ **Therefore, there is a total of: 220 * 84 * 20 * 1= 369600 different ways the minifigures can be arranged.**

ii) The second approach to this problem would be to use the permutation formula for $\boldsymbol{n}$ identical objects of each type. We have 4 locations to place 3 minifigures into, therefore, we have a formula as follows:

⇨ $\boldsymbol{\frac{n!}{n_1! * n_2! * \ldots * n_k!} \Leftrightarrow \frac{12!}{3! * 3! * 3! * 3!}}$ **=> 369600 (ways).**

⇨ **Therefore, there is a total of 369600 different ways to arrange the minifigures.**

d/

From the information provided in this question, a study session will take two days to complete as you will study for a day and take a break the others. We have to study 10 sessions. Therefore, the minimum number of days it takes to finish 10 sessions without studying consecutively are as follows:

$$2 * 10 = 20\ (days)$$

As we only have 14 days to study, we are lacking three days to study and another three days to rest if we are not going to study consecutively.

Let's $\boldsymbol{m}$ be the number of days we have to study, which is 14 => $\boldsymbol{m}$ = 14. Let's $\boldsymbol{n}$ be the number of days we need to study 10 sessions, which is 20 => $\boldsymbol{n}$ = 20.

Pigeonhole principle states that where if there are n items to be put into m containers, and n > m then there will be at least one container that must carry one more item.

Applying Pigeonhole principle to our problem, we can clearly see that the "items" outnumbers the "container", or in other words, the number of days we need to study 10 sessions is greater than the number of days we must study, 20 > 14 (n > m). **Therefore, no matter how we plan and schedule our study in the next two weeks, we will still have to study consecutively at least once**.

Algorithms:

Question 2

a/

The flowchart that reads the file and calculates and prints the number of students who have logged in at least 50 times are as follows:

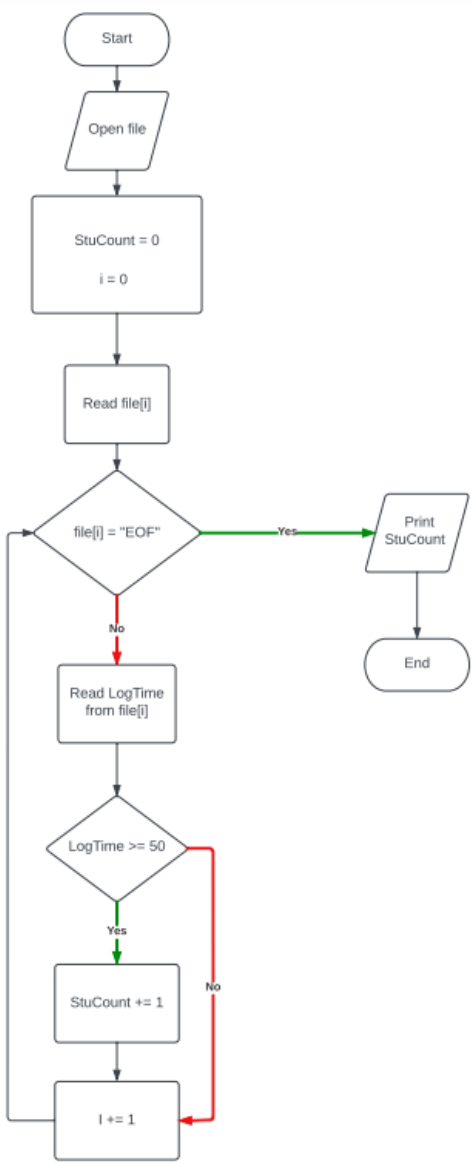

Figure 1: Flowchart that calculates number of students

b/

In order to find the complexity of the program using big-O notation, a pseudocode are as follows:

	Pseudocode	**Times complexity**
1	$Open\ file$	$O(1)$
2	$StuCount\ =\ 0$	$O(1)$
3	$i\ =\ 0$	$O(1)$
4	$Read\ file$	$O(1)$
5	$while\ file[i]\ !=\ "EOF"$	$O(n)$
6	$\quad Read\ LogTime\ from\ file[i]$	$O(n)$
7	$\quad If\ LogTime\ >=\ 50$	$O(n)$
8	$\quad\quad StuCount\ +=\ 1$	$O(n)$
9	$\quad i\ +=\ 1$	$O(n)$
10	$\boldsymbol{continue\ the\ while\ loop}$	$O(n)$
11	$print\ Stucount\ on\ the\ screen$	$O(1)$
12	end	

The time complexity of an algorithm can be calculated by analyzing the program's statements and determine the time it would take for the program to execute those statements, therefore:

Let T(n) be the total time in function of the input size n:

$$T(n)\ =\ 6n\ +\ 4$$

We can get to our conclusion that the time complexity of this program is O(n), $6n$ would not matter in this context as complexity qualification does not track scalar multipliers. The actual run time of a program or algorithm will depend on how fast computer instructions work, therefore, if we count scalar factors, our complexity calculation would be extremely context specific.

Question 3

a)

Although O(n^3) and O(n^2) are both disastrous in term of algorithm design**, I will still suggest them to use algorithm X with the runtime complexity of O(n^2).** A figure is provide

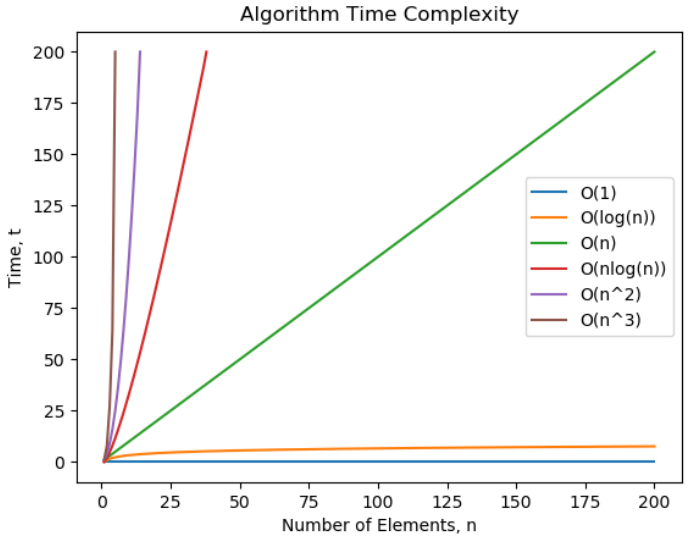

Figure 2: Comparison between Algorithm Time Complexity

As we can see from the figure, although the time complexity of both O(n^3) and O(n^2) are massive, with the same input size or number of elements, the time taken to complete algorithm with runtime complexity of O(n^2) is still less than that of O(n^3).

b)

Average-case time complexity refers to an expected number of steps of an algorithm given the input size of n. Although with the new information provided in question b, the average-case time complexity of Y is better than X, developers typically solve for the worst-case scenario, Big O, because we can not expect the algorithm or program to run in the best or even average case, expecting the worst-case scenario allows programmer to make analytical prediction for an algorithm.

Therefore, even with the new average time complexity, my advice remains the same, algorithm X is still overall more efficient than algorithm Y.

Question 4

The pseudocode for the recursive function sum_odd(n) are as follows:

Function sum_odd(n):

1. $\boldsymbol{if}\ n\ is\ equal\ to\ 0$
2. $\quad\quad\boldsymbol{return}\ 0$
3. $\boldsymbol{if}\ binary\_ones(n)\ mod\ 2\ equal\ to\ 1$
4. $\quad\quad\boldsymbol{return}\ n\ +\ sum\_odd(n-1)$
5. $\boldsymbol{return}\ sum\_odd(n-1)$

The above function will calculate a sum of the numbers $\boldsymbol{i}$ between 1 and $\boldsymbol{n}$ where the binary representation of $\boldsymbol{i}$ contains an odd number of 1s. The function will be designed based on recursion.

The first two lines of the pseudocode is a base case of our recursive function, without a base case the function will look infinitely. The initial value of $\boldsymbol{n}$ will be decreased over time and eventually comes to zero and the function will break. After that the function will return the sum of numbers that satisfies the requirements.

Graphs and trees

Question 5

a) According to the graph representation, the graph is drawn as follows:

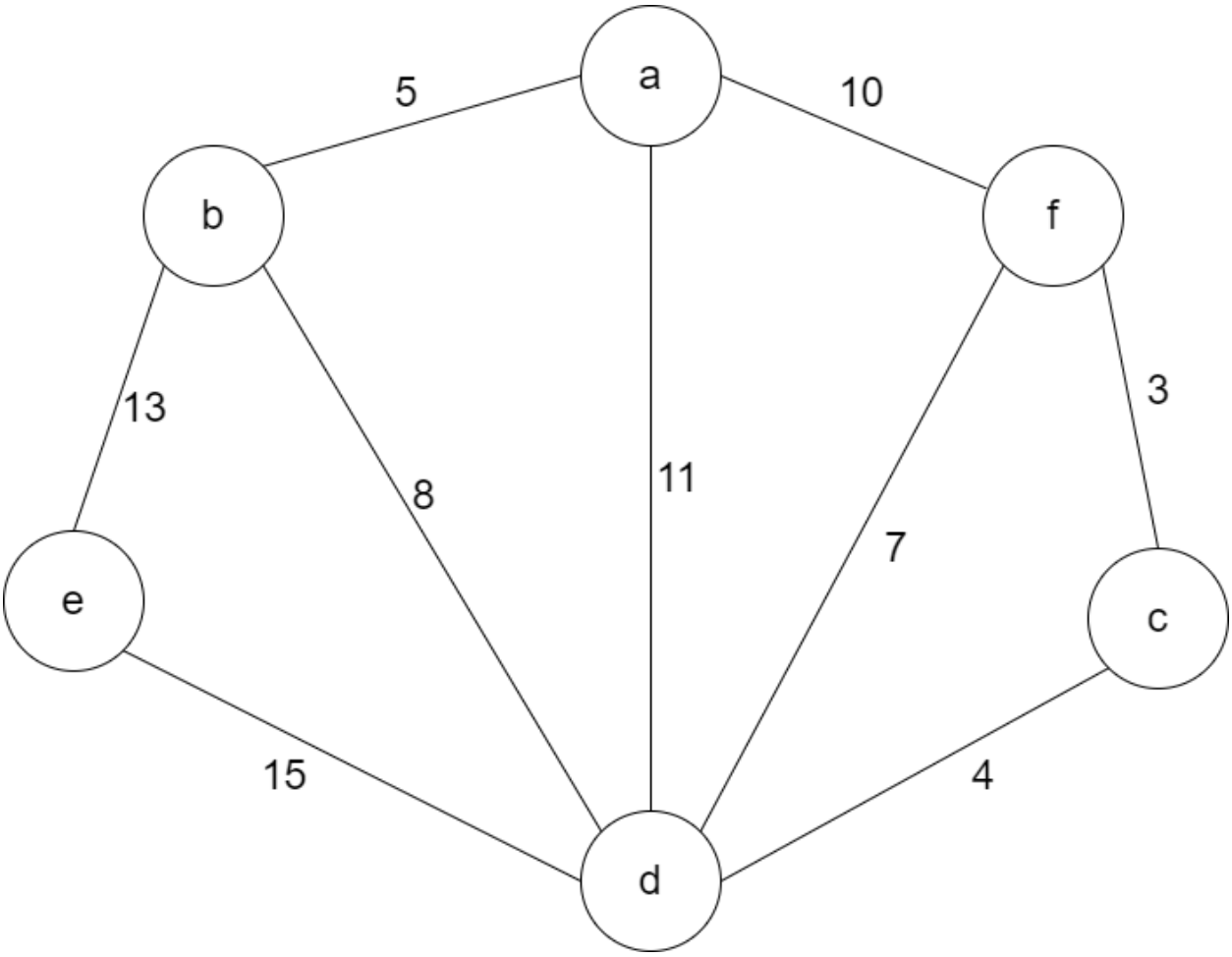

Figure 3: The graph including weights

b)

i) The order of edges and their status are as follows:

Order	Edge	Weight	Status
1	W(c, f)	3	Kept
2	W(c, d)	4	Kept
3	W(a, b)	5	Kept
4	W(d, f)	7	Discarded
5	W(b, d)	8	Kept
6	W(a, f)	10	Discarded
7	W(a, d)	11	Discarded
8	W(b, e)	13	Kept
9	W(d, e)	15	Discarded

Figure 4: Order and status table of the edges

ii)

Spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. Based on that explanation, a resulting spanning tree are as follows:

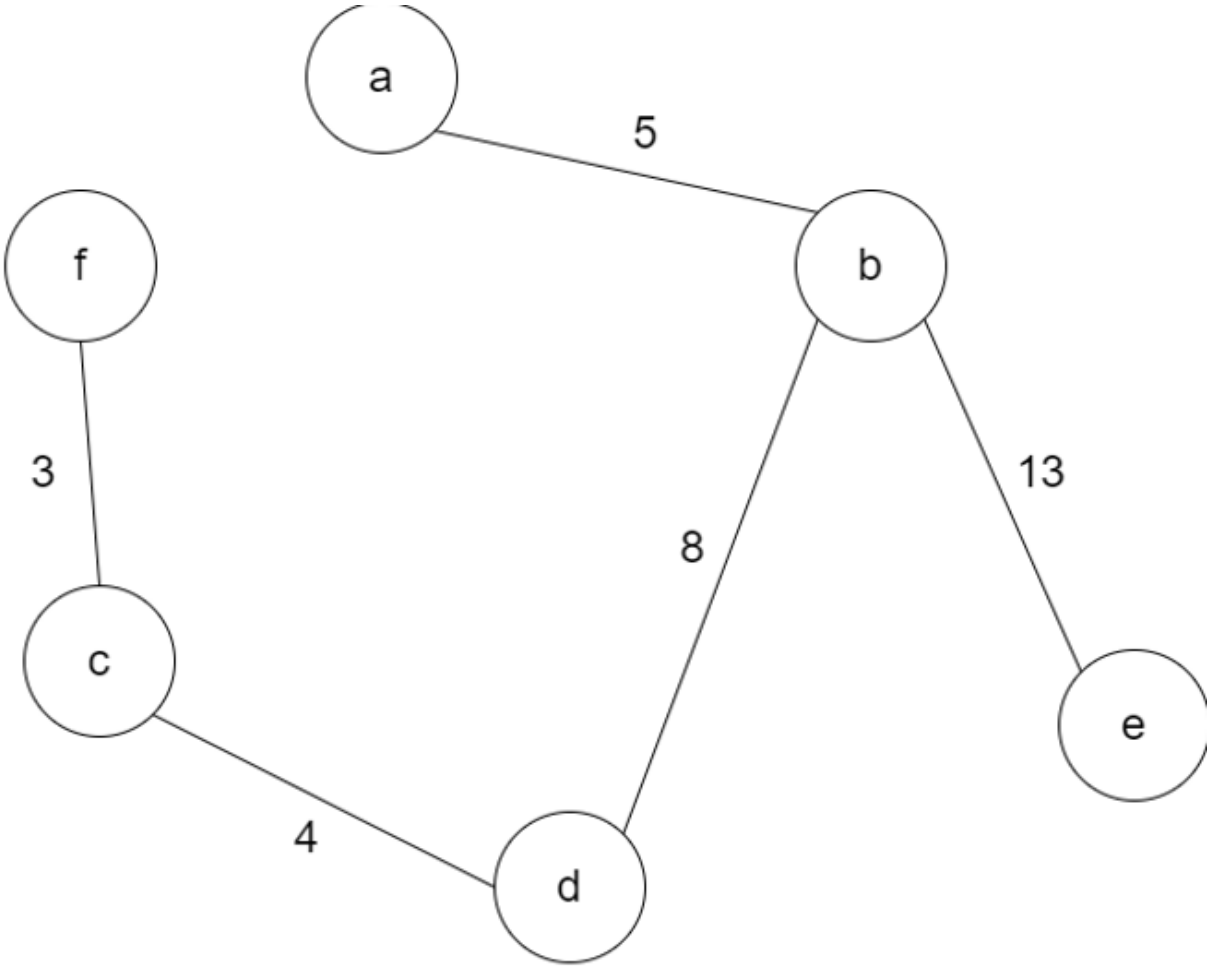

Figure 5: Spanning tree F

Question 6

According to the provided information, we would have the following graph:

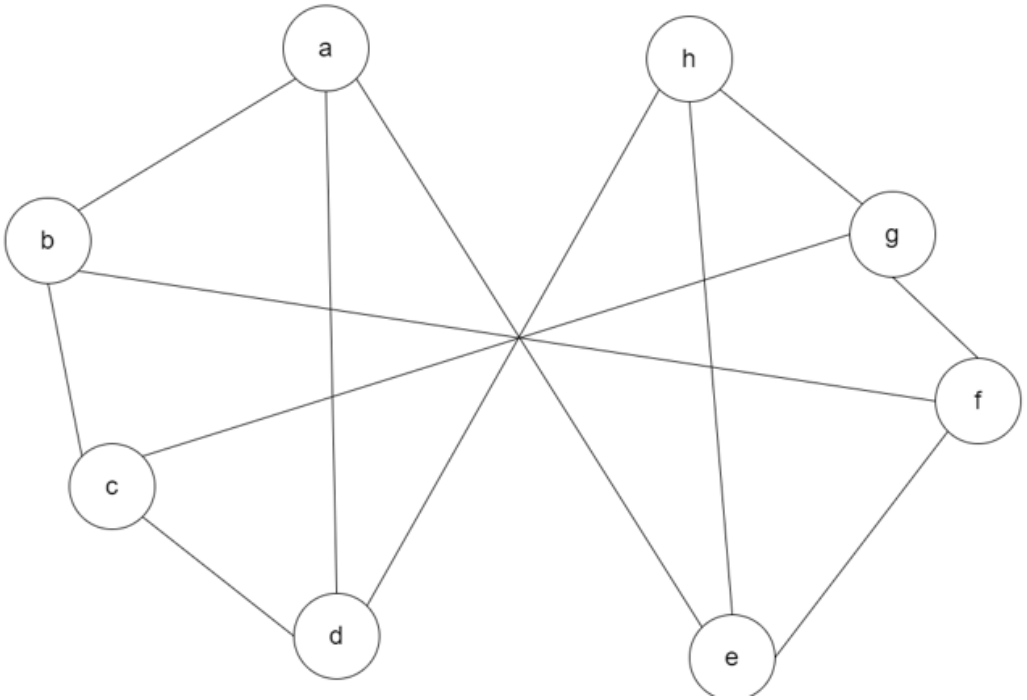

Figure 6: Graph for question 6

A graph can contain Eulerian cycles if and only if there is connected, and if the degree of each vertex is even. From the graph above, we can clearly see that each vertex has an odd degree of 3. Therefore, the graph cannot contain a Eulerian cycle.

A Hamiltonian cycle, on the other hand, is a cycle where each vertex is visited exactly once with no repeats, however, it does not have to start and end at the same vertex. The graph below is an example of a Hamiltonian cycle in the graph.

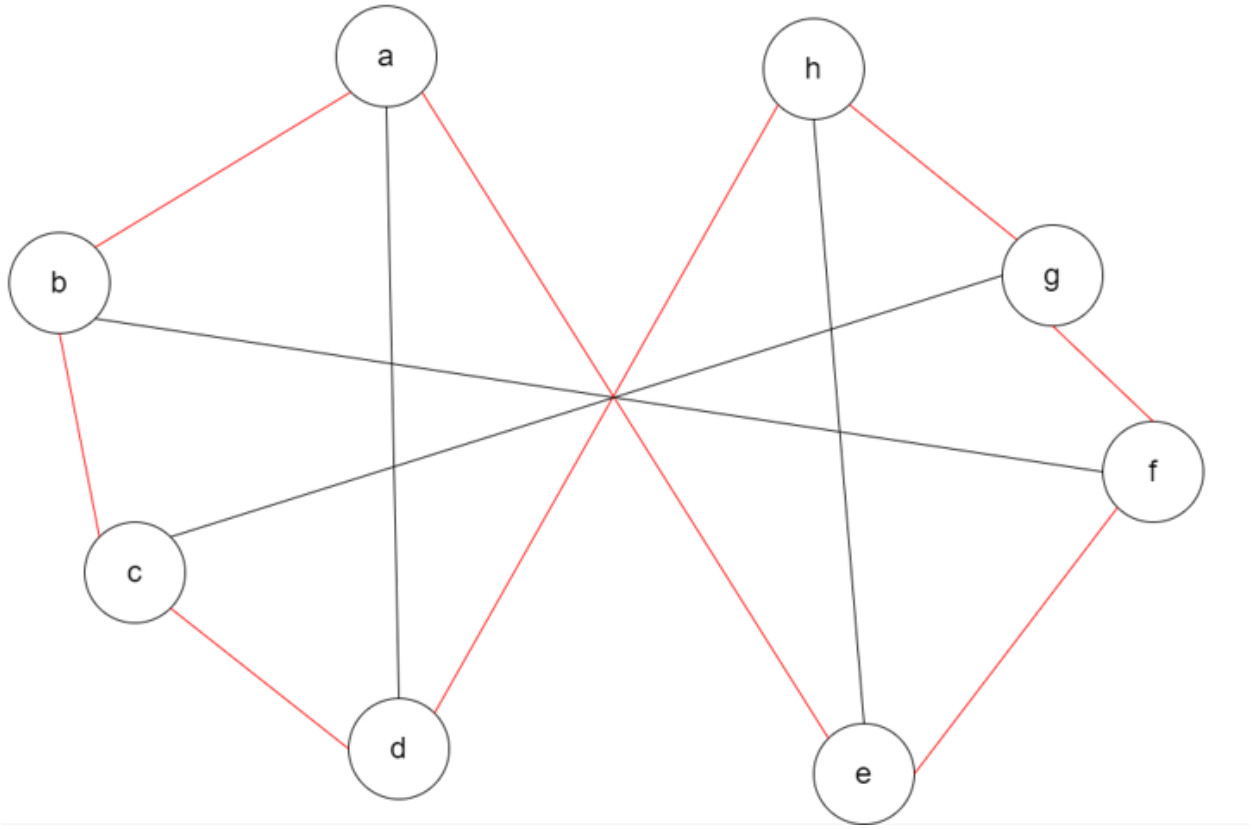

Figure 7: An example of a Hamiltonian cycle in the graph

The Hamiltonian cycle can be spotted as follows: a-> b -> c -> d -> h -> g -> f -> e -> a. The cycle goes from a and go through every other vertex with no repeat.

Therefore, the following graph only contains at least one Hamiltonian cycle but not any Eulerian cycle.