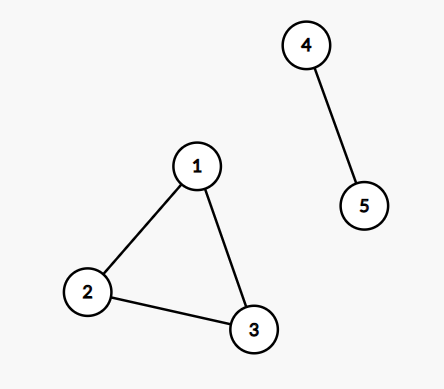
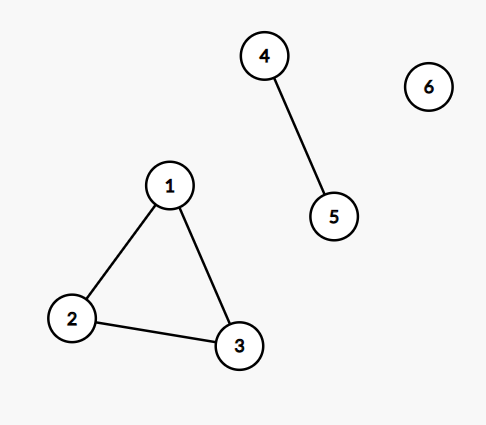
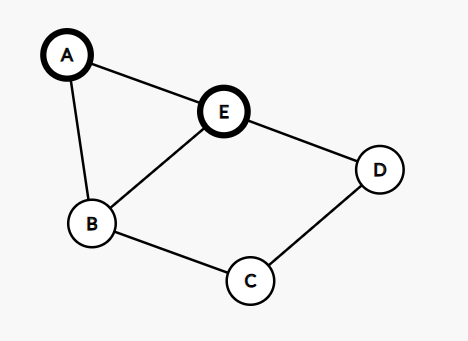
# Problems

1. Write a program that takes a **directed** graph as input and checks whether it is **bi-directionoal** which means for every pair of nodes where there is an edge **u -> v**, there should also be an edge **v -> u**.  
   Use this code to take the graph as input: <https://ideone.com/divsSy>  
   Which graph representation would you prefer? **15**
2. This is a follow up of problem 1. Trim all edges of the input graph where there is an edge from **u -> v** but not **v -> u**. Output the number of edges you have to remove. Use the same code to take input as **problem 1. 10**
3. Connected component in an **undirected** graph means a set of vertices in a graph that are linked to each other by paths. For example, **10**  
   - There are 2 connected components in the following graph  
     
   - There are 3 connected components in the following graph  
     
   Write a program to take an **undirected** graph as input and count the number of connected components in it.
4. Your friend was working with the following **undirected** graph and he was simulating **BFS** on it. But he accidentally selected both **node A** and **node E** as source. Which means he selected both nodes **A & E**, marked them as **visited** and pushed them into the queue initially. Then he went on with performing normal BFS. **15**  
     
   Simulate the resulting BFS for your friend. What is the resulting traversal order and what will the level of each node be. Assume that both nodes **A & E** are at level **0**. You don’t need to write code.  
   

1. The **int\_to\_binary()** function in this link takes a positive integer as parameter and returns its binary representation as string: <https://ideone.com/KzRVtu> **10**  
     
   Rewrite the function using **recursion**.
2. The **gcd()** function in this link calculates gcd of two positive integers: <https://ideone.com/tlqJ47>

Rewrite the function using **recursion**. **10**

1. Calculate time complexity of the following code snippet: **5**  
     
   **int count = 0;**

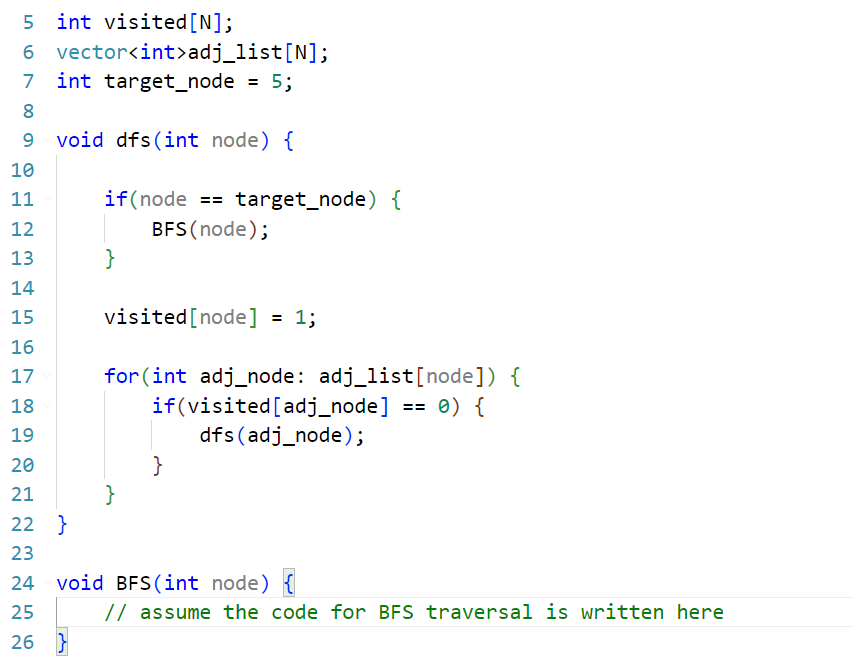
**for (int i = N; i > 0; i /= 2) {**

**for (int j = 0; j < i; j++) {**

**count += 1;**

**}**

**}**

1. Look at the following code snippet. Calculate the time and space complexity. **10**  
   

1. Will BFS or DFS act differently on graphs having **multi edge** and **self loops**? Why or why not? Explain. **5**
2. We want to perform **DFS** without recursion. Can you think of any data structure that we can use to do this? **10**  
   Show a simulation and print the traversal order for the following graph. Use **node 1** as source.   
   Compare your traversal order with the actual recursive DFS traversal order. You don’t need to write code.  
   