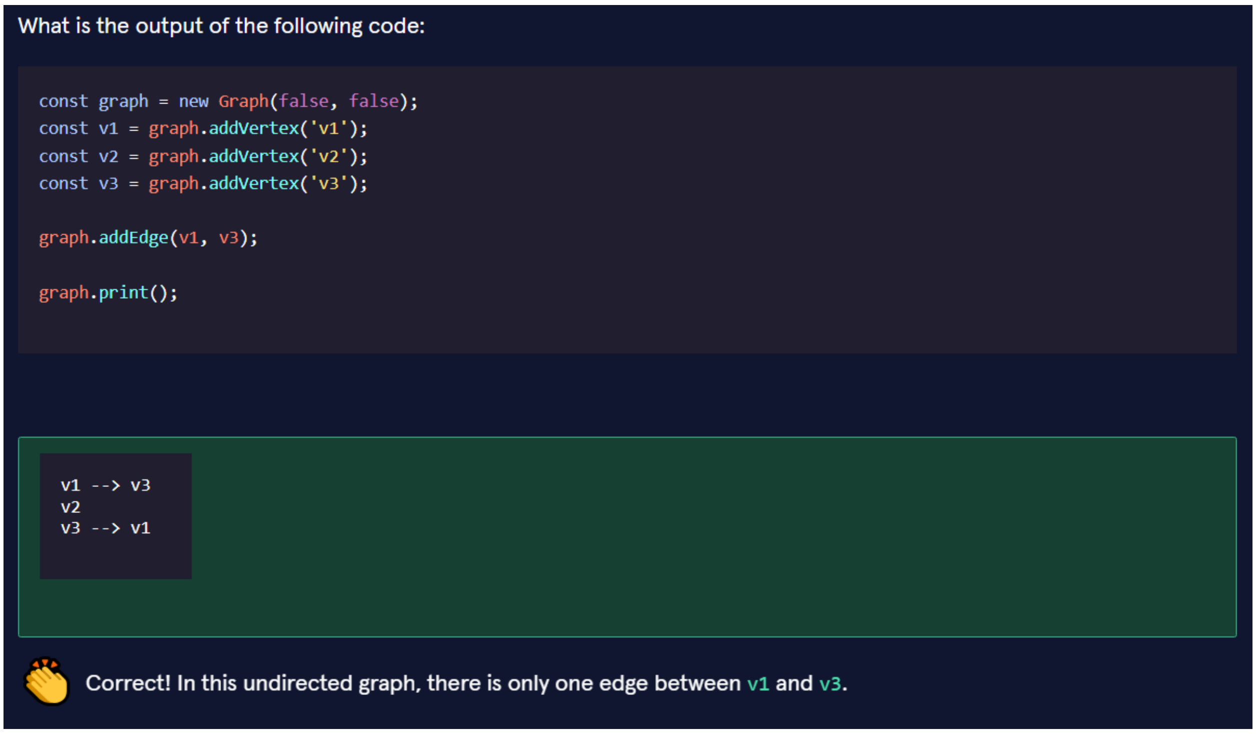
QUIZ

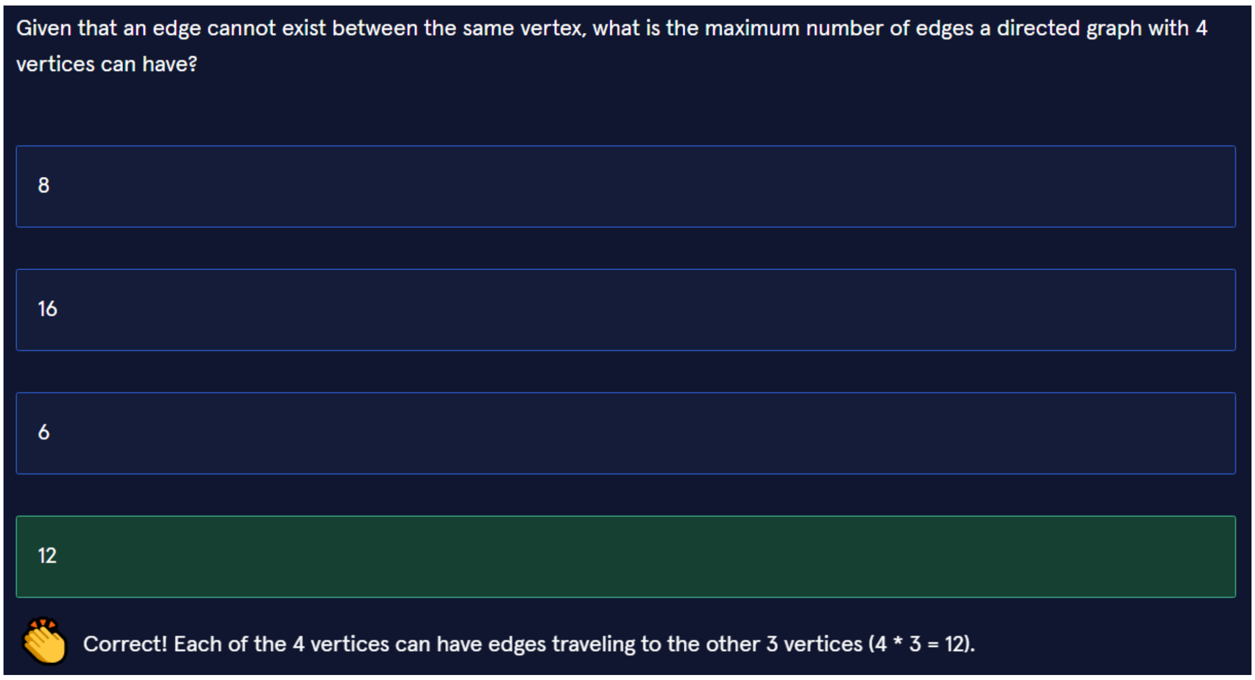


What is the output of the following code:

```
const graph = new Graph(false, false);
const v1 = graph.addVertex('v1');
const v2 = graph.addVertex('v2');
const v3 = graph.addVertex('v3');

graph.addEdge(v1, v3);

graph.print();
```

```
v1 --> v3
v2
v3 --> v1
```

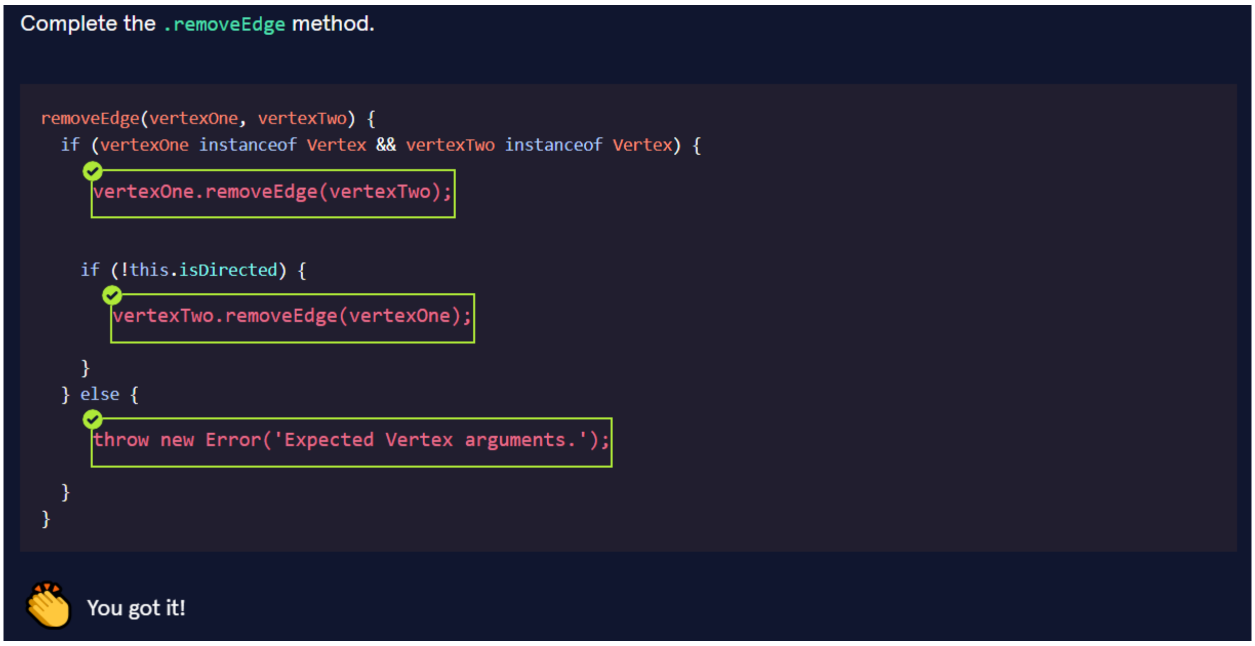
Correct! In this undirected graph, there is only one edge between `v1` and `v3`.



Given that an edge cannot exist between the same vertex, what is the maximum number of edges a directed graph with 4 vertices can have?

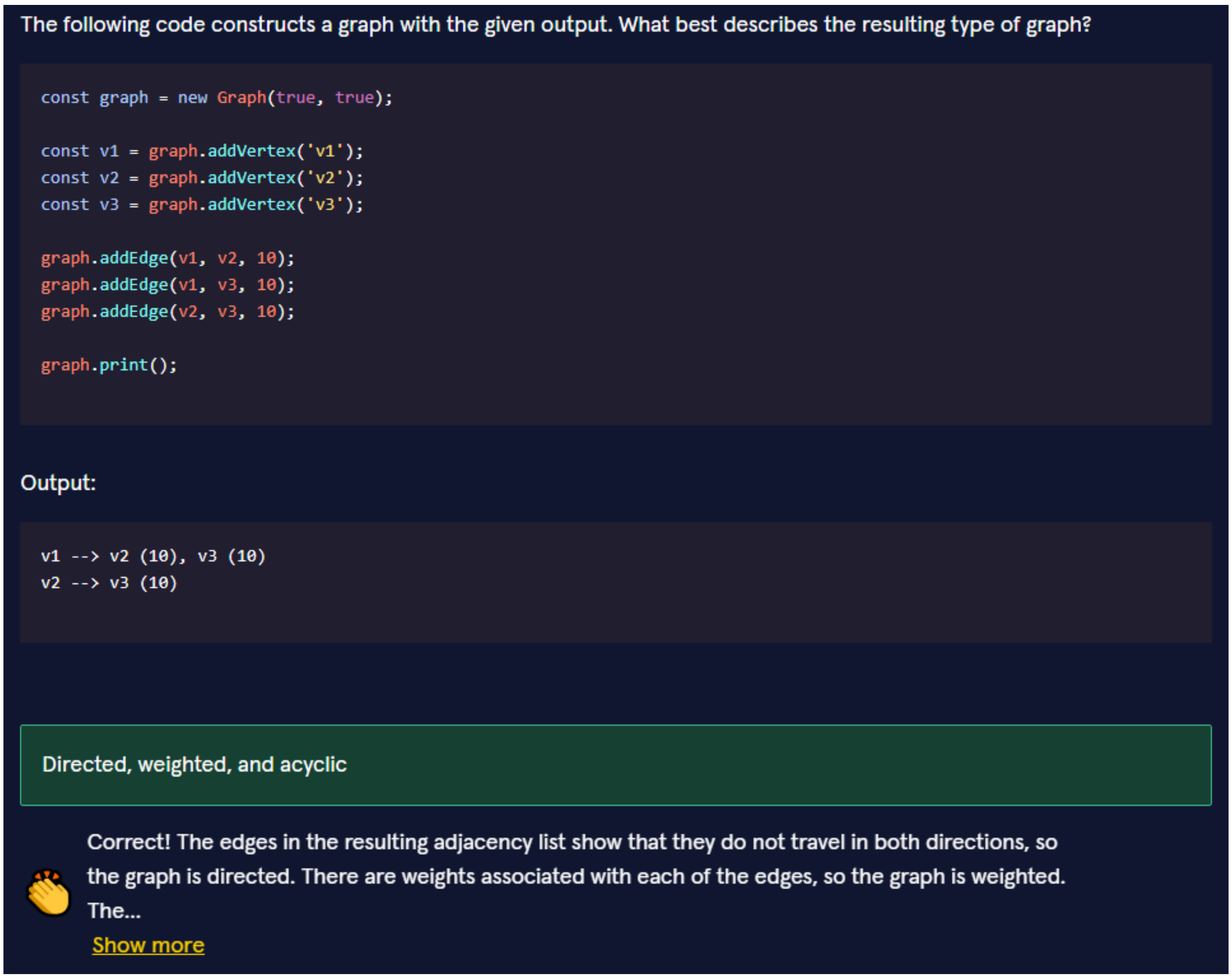
- 8
- 16
- 6
- [✓ 12]

Correct! Each of the 4 vertices can have edges traveling to the other 3 vertices (4 * 3 = 12).


Complete the `.removeEdge` method.

```
removeEdge(vertexOne, vertexTwo) {
  if (vertexOne instanceof Vertex && vertexTwo instanceof Vertex) {
    vertexOne.removeEdge(vertexTwo);

    if (!this.isDirected) {
      vertexTwo.removeEdge(vertexOne);
    }
  } else {
    throw new Error('Expected Vertex arguments.');
  }
}
```

You got it!


The following code constructs a graph with the given output. What best describes the resulting type of graph?

```
const graph = new Graph(true, true);

const v1 = graph.addVertex('v1');
const v2 = graph.addVertex('v2');
const v3 = graph.addVertex('v3');

graph.addEdge(v1, v2, 10);
graph.addEdge(v1, v3, 10);
graph.addEdge(v2, v3, 10);

graph.print();
```

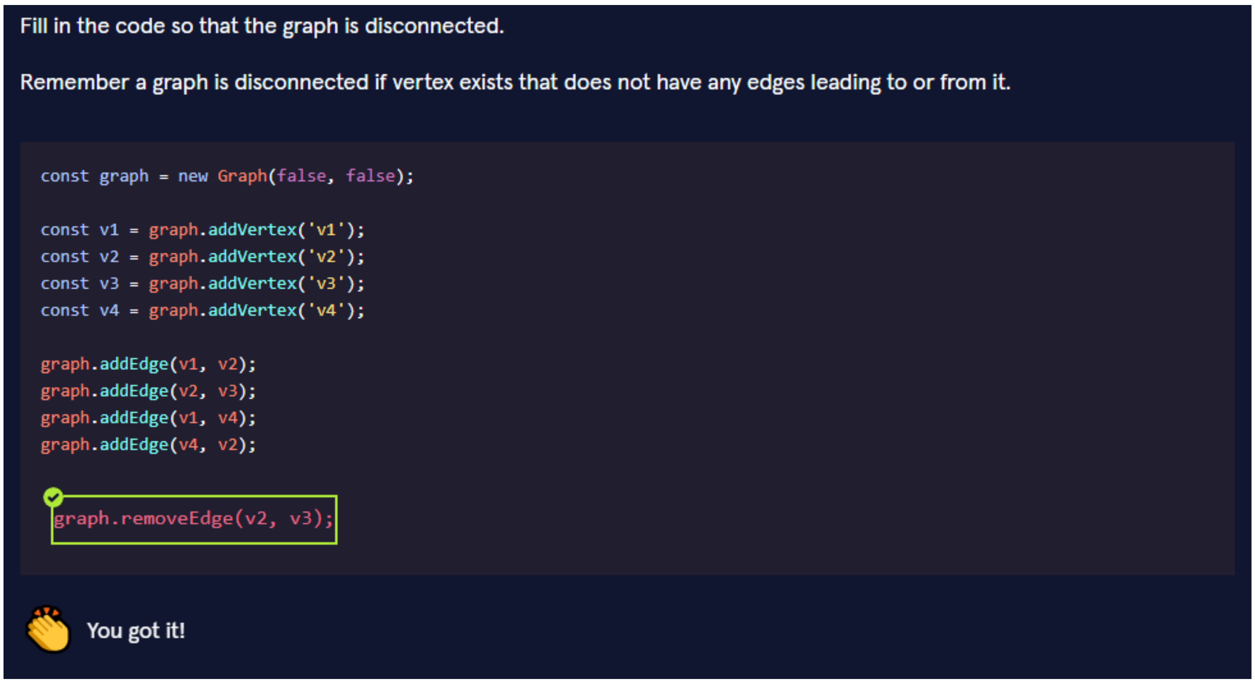
Output:

```
v1 --> v2 (10), v3 (10)
v2 --> v3 (10)
```

Directed, weighted, and acyclic

Correct! The edges in the resulting adjacency list show that they do not travel in both directions, so the graph is directed. There are weights associated with each of the edges, so the graph is weighted. The...

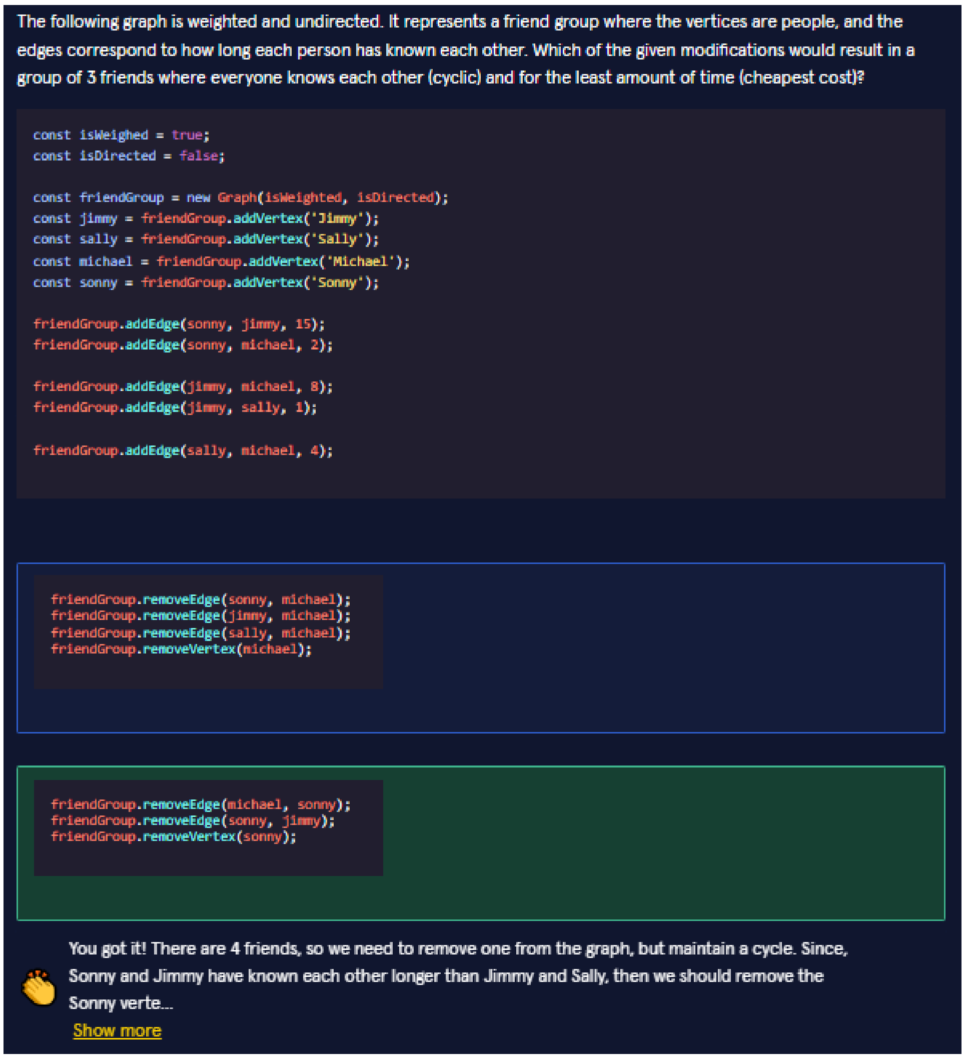
Show more


Fill in the code so that the graph is disconnected.

Remember a graph is disconnected if vertex exists that does not have any edges leading to or from it.

```
const graph = new Graph(false, false);

const v1 = graph.addVertex('v1');
const v2 = graph.addVertex('v2');
const v3 = graph.addVertex('v3');
const v4 = graph.addVertex('v4');

graph.addEdge(v1, v2);
graph.addEdge(v2, v3);
graph.addEdge(v1, v4);
graph.addEdge(v4, v2);

graph.removeEdge(v2, v3);
```

You got it!

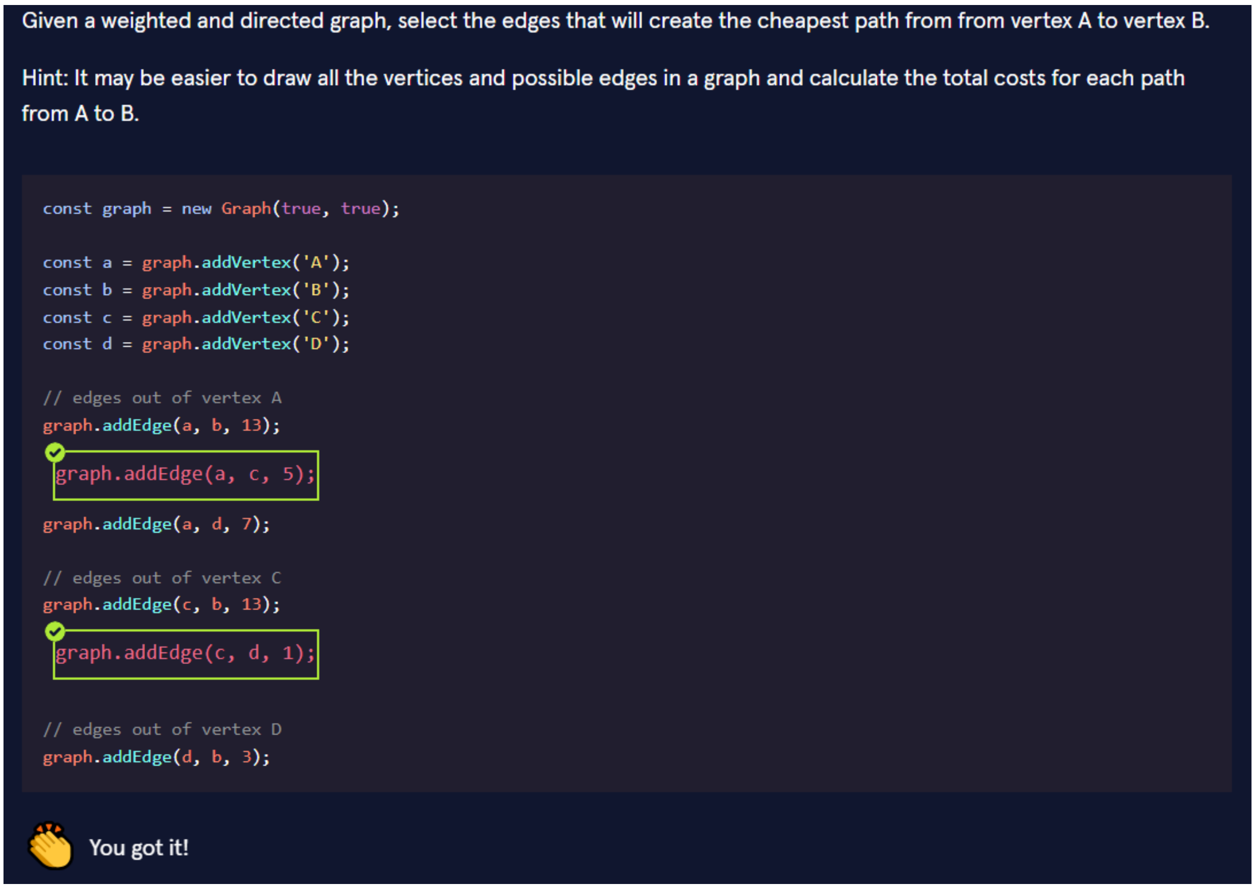

The following graph is weighted and undirected. It represents a friend group where the vertices are people, and the edges correspond to how long each person has known each other. Which of the given modifications would result in a group of 3 friends where everyone knows each other (cyclic) and for the least amount of time (cheapest cost)?

```
const isWeighed = true;
const isDirected = false;

const friendGroup = new Graph(isWeighted, isDirected);
const jimmy = friendGroup.addVertex('Jimmy');
const sally = friendGroup.addVertex('Sally');
const michael = friendGroup.addVertex('Michael');
const sonny = friendGroup.addVertex('Sonny');

friendGroup.addEdge(sonny, jimmy, 15);
friendGroup.addEdge(sonny, michael, 2);

friendGroup.addEdge(jimmy, michael, 8);
friendGroup.addEdge(jimmy, sally, 1);

friendGroup.addEdge(sally, michael, 4);
```

```
friendGroup.removeEdge(sonny, michael);
friendGroup.removeEdge(jimmy, michael);
friendGroup.removeEdge(sally, michael);
friendGroup.removeVertex(michael);
```

```
friendGroup.removeEdge(michael, sonny);
friendGroup.removeEdge(sonny, jimmy);
friendGroup.removeVertex(sonny);
```

You got it! There are 4 friends, so we need to remove one from the graph, but maintain a cycle. Since, Sonny and Jimmy have known each other longer than Jimmy and Sally, then we should remove the Sonny verte...
Show more

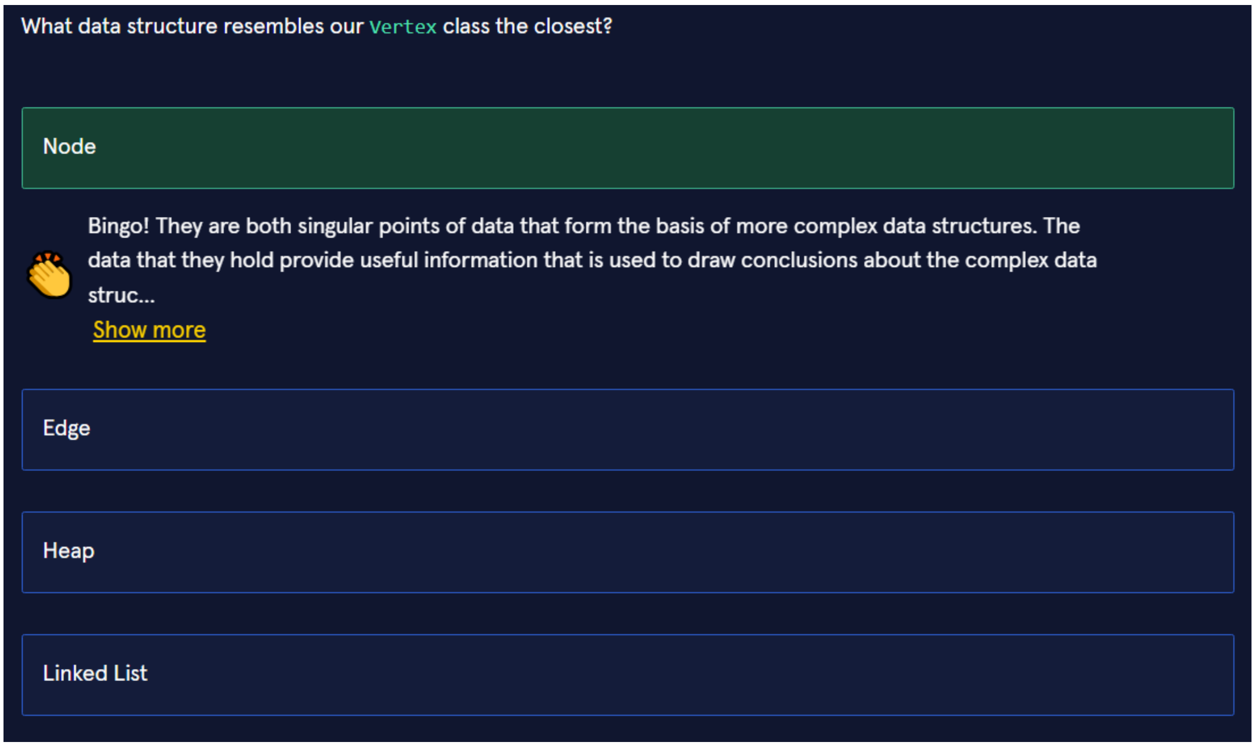


Given a weighted and directed graph, select the edges that will create the cheapest path from from vertex A to vertex B.

Hint: It may be easier to draw all the vertices and possible edges in a graph and calculate the total costs for each path from A to B.

```
const graph = new Graph(true, true);

const a = graph.addVertex('A');
const b = graph.addVertex('B');
const c = graph.addVertex('C');
const d = graph.addVertex('D');

// edges out of vertex A
graph.addEdge(a, b, 13);
[✓ graph.addEdge(a, c, 5);]
graph.addEdge(a, d, 7);

// edges out of vertex C
graph.addEdge(c, b, 13);
[✓ graph.addEdge(c, d, 1);]

// edges out of vertex D
graph.addEdge(d, b, 3);
```

👏 You got it!



What data structure resembles our `Vertex` class the closest?

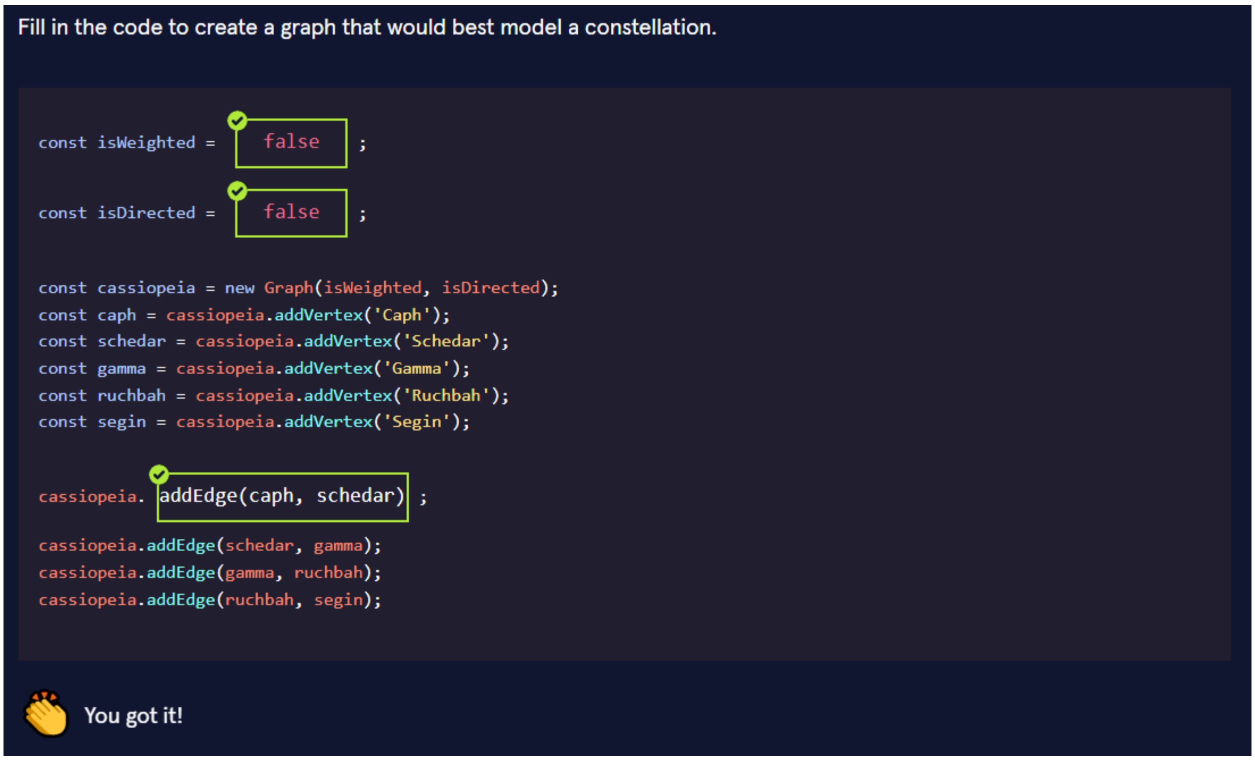
[✓ Node]

👏 Bingo! They are both singular points of data that form the basis of more complex data structures. The data that they hold provide useful information that is used to draw conclusions about the complex data struc...
Show more

Edge

Heap

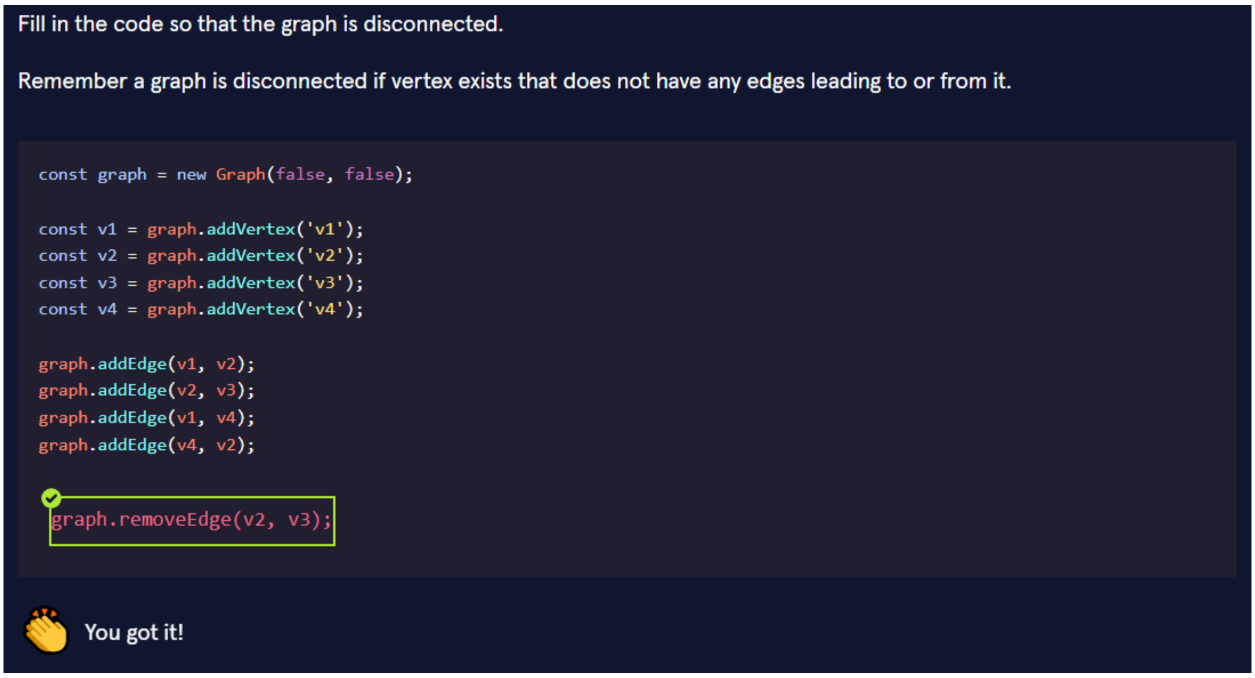
Linked List



Fill in the code to create a graph that would best model a constellation.

```
const isWeighted = false ;

const isDirected = false ;

const cassiopeia = new Graph(isWeighted, isDirected);
const caph = cassiopeia.addVertex('Caph');
const schedar = cassiopeia.addVertex('Schedar');
const gamma = cassiopeia.addVertex('Gamma');
const ruchbah = cassiopeia.addVertex('Ruchbah');
const segin = cassiopeia.addVertex('Segin');

cassiopeia. addEdge(caph, schedar) ;

cassiopeia.addEdge(schedar, gamma);
cassiopeia.addEdge(gamma, ruchbah);
cassiopeia.addEdge(ruchbah, segin);
```

You got it!

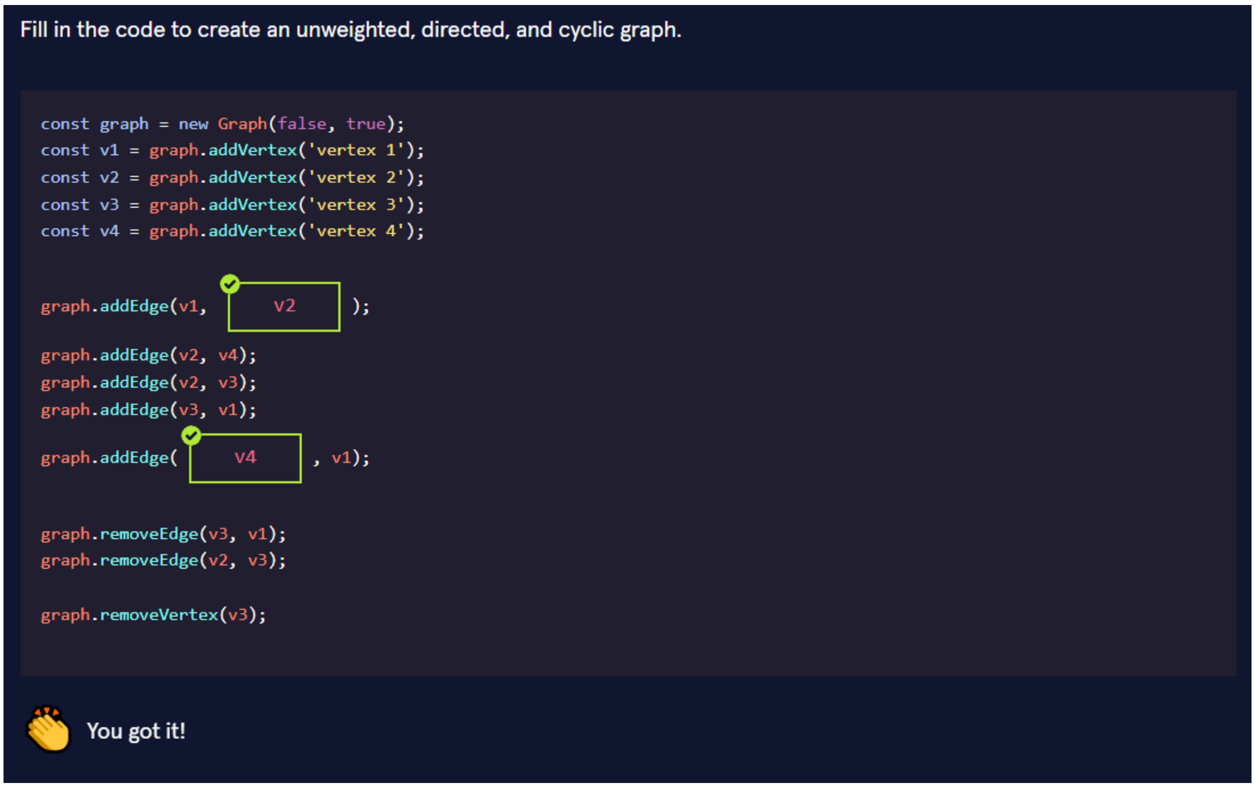


Fill in the code so that the graph is disconnected.

Remember a graph is disconnected if vertex exists that does not have any edges leading to or from it.

```
const graph = new Graph(false, false);

const v1 = graph.addVertex('v1');
const v2 = graph.addVertex('v2');
const v3 = graph.addVertex('v3');
const v4 = graph.addVertex('v4');

graph.addEdge(v1, v2);
graph.addEdge(v2, v3);
graph.addEdge(v1, v4);
graph.addEdge(v4, v2);

graph.removeEdge(v2, v3);
```

You got it!


Fill in the code to create an unweighted, directed, and cyclic graph.

```
const graph = new Graph(false, true);
const v1 = graph.addVertex('vertex 1');
const v2 = graph.addVertex('vertex 2');
const v3 = graph.addVertex('vertex 3');
const v4 = graph.addVertex('vertex 4');

graph.addEdge(v1, v2 );
graph.addEdge(v2, v4);
graph.addEdge(v2, v3);
graph.addEdge(v3, v1);
graph.addEdge( v4 , v1);

graph.removeEdge(v3, v1);
graph.removeEdge(v2, v3);

graph.removeVertex(v3);
```

You got it!