```
class Graph(object):

    def __init__(self, graph_dict=None):
        if graph_dict == None:
            graph_dict = {}
        self._graph_dict = graph_dict

    def edges(self, vertice):
        return self._graph_dict[vertice]

    def all_vertices(self):
        return set(self._graph_dict.keys())

    def all_edges(self):
        return self.generate_edges()

    def add_vertex(self, vertex):
        if vertex not in self._graph_dict:
            self._graph_dict[vertex] = []

    def add_edge(self, edge):
        edge = set(edge)
        vertex1, vertex2 = tuple(edge)
        for x, y in [(vertex1, vertex2), (vertex2, vertex1)]:
            if x in self._graph_dict:
                self._graph_dict[x].add(y)
            else:
                self._graph_dict[x] = [y]

    def generate_edges(self):
        edges = []
        for vertex in self._graph_dict:
            for neighbour in self._graph_dict[vertex]:
                if {neighbour, vertex} not in edges:
                    edges.append({vertex, neighbour})
        return edges

    def iter (self):
        self._iter_obj = iter(self._graph_dict)
        return self._iter_obj

    def next (self):
        return next(self._iter_obj)

    def str (self):
        res = "vertices: "
        for k in self._graph_dict:
            res += str(k) + " "
        res += "\nedges: "
        for edge in self.__generate_edges():
            res += str(edge) + " "
        return res

    def find_path(self, start_vertex, end_vertex, path=None):

        if path == None:
            path = []
        graph = self._graph_dict
        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return path
        if start_vertex not in graph:
            return None
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_path = self.find_path(vertex,
                                               end_vertex,
                                               path)
                if extended_path:
                    return extended_path
        return None

    def check_graph(self):
        list_path = []
        for vertex in self._graph_dict:
            for next_vertex in self._graph_dict:
                if vertex != next_vertex:
                    path = self.find_path(vertex, next_vertex)
                    if path == None:
                        list_path.append(path)
                    elif path != None:
                        list_path.extend(path)
        if None in list_path:
           print("Yes")
        elif None not in list_path:
            print("No")

n = { "a" : {"b"},
      "b" : {"a", "c", "d"},
      "c" : {"b", "d", "e"},
      "d" : {"b", "c", "f"},
      "e" : {"c"},
      "f" : {"d"},
      "g" : {"h"},
      "h" : {"g", "i", "j"},
      "i" : {"h", "j", "k"},
      "j" : {"h", "i", "l"},
      "k" : {"i"},
      "l" : {"j"}
    }

graph = Graph(n)

for vertice in graph._graph_dict:
    print(f"Edges of vertice {vertice}: ", graph.edges(vertice))

print("\nUnconnected Graph")
graph.check_graph()
```

```
Edges of vertice a:  {'b'}
Edges of vertice b:  {'c', 'a', 'd'}
Edges of vertice c:  {'e', 'b', 'd'}
Edges of vertice d:  {'b', 'f', 'c'}
Edges of vertice e:  {'c'}
Edges of vertice f:  {'d'}
Edges of vertice g:  {'h'}
Edges of vertice h:  {'g', 'j', 'i'}
Edges of vertice i:  {'j', 'h', 'k'}
Edges of vertice j:  {'i', 'l', 'h'}
Edges of vertice k:  {'i'}
Edges of vertice l:  {'j'}

Unconnected Graph
Yes
```