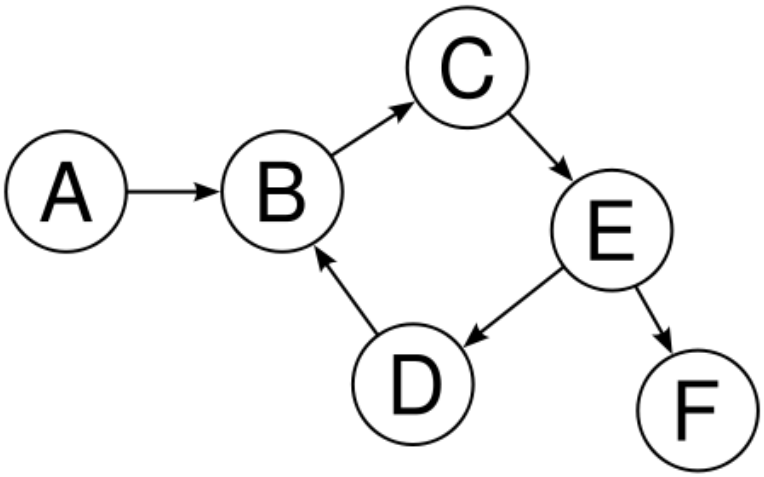
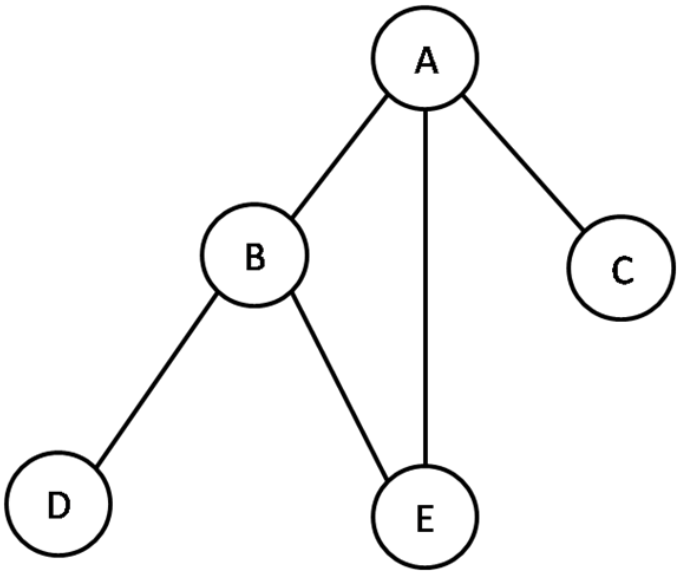
Graphs overview

A **graph** is a set of vertices connected pairwise by edges.
Directed graphs are a class of graphs that don't presume symmetry or reciprocity in the edges established between vertices. In a directed graph, if **a** and **b** are two vertices connected by an edge (a, b), this doesn't necessarily mean that an edge connecting (b, a) also exists



An **undirected graph** has edges that do not have a direction. So that means, when there is statement where (1, 2) it is also means (2, 1)



A **tree** is an acyclic connected graph.
A **forest** is a disjoint set of trees.
A **self-loop** is an edge that connects a vertex to itself.

Usage of undirected graph

Nowadays undirected graphs are becoming widely used by programmers. Graphs are **really** important in modeling data. In fact, graphs can reduce many problems in algorithms.
Here we outline some of many applications of graphs:

ELECTRICAL CIRCUITS

An electrical circuit consists of internally connected elements viz resistors , capacitors, inductors, diodes, transistors etc .

For example consider the circuit and its graph in the figure. There are five vertices and seven edges in the graph obtained from the given circuit. An edge is sometimes called a branch and a vertex is called a node in case of electrical circuits.
The behaviour of an electrical circuit generally depends upon two factors.

a) The characteristic of each of internally connected elements
b) The rule by which they are connected together.

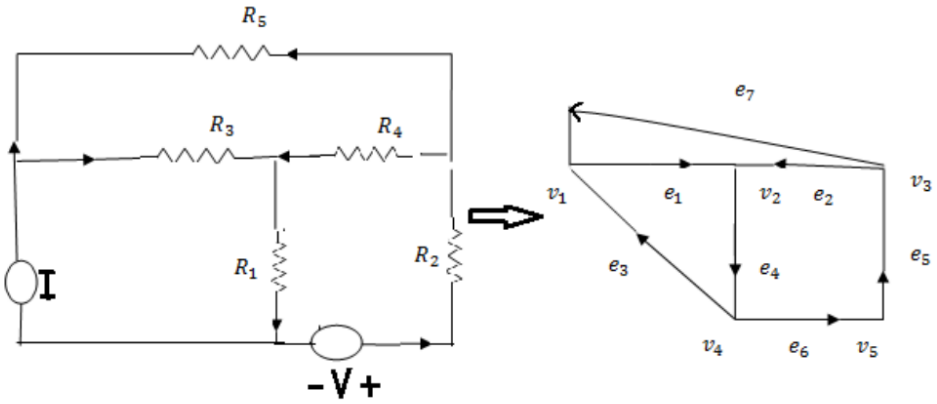


Fig: A circuit and its corresponding graph

DEVELOPING PROJECT SCHEDULES

Efficiently managing a software development project is extremely important in industry and is often overlooked by the software developers on a project. Basically, pieces of development work can pass through the project manager.

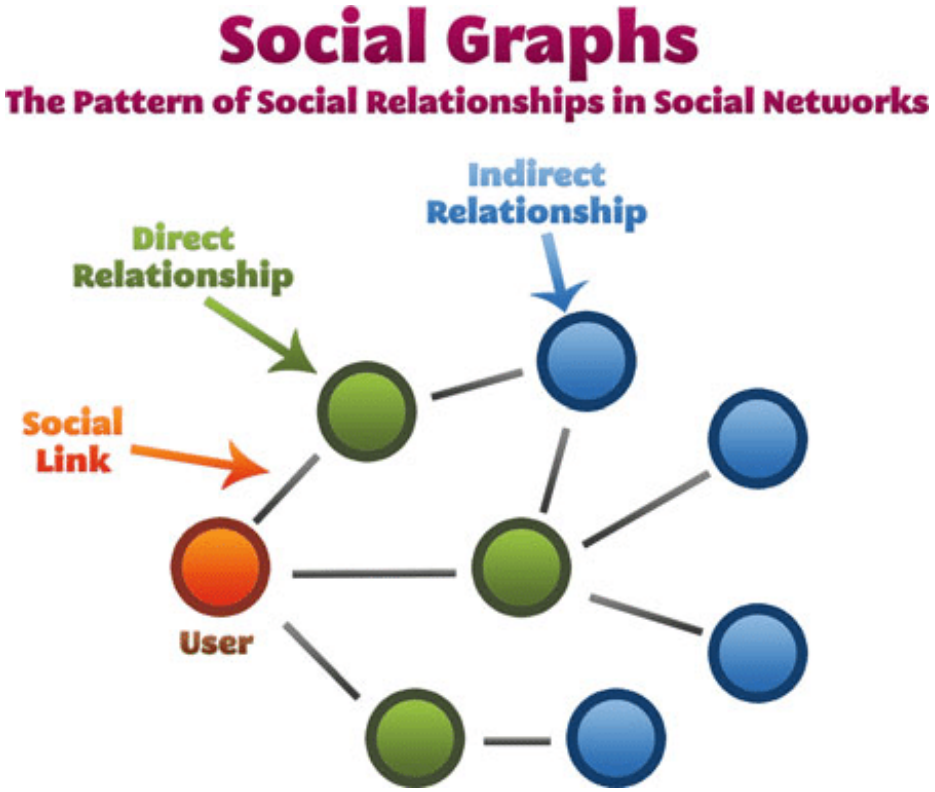
So, it can be solved with an undirected graph, where each node is a task and edges in the graph indicate dependencies. The relationships between this problem and more well-known problems in graph theory are used to inform the development of the algorithms. Analysis of the results provide insight to what structures of dependency chains can be handled by the algorithms. The resulting software could be used to save companies both time and money when planning software development projects.

FIND SHORTEST PATH

Finding the shortest path mostly can be used in maps or video games. It is simply can be made with using BFS or DFS

ANALYZE SOCIAL RELATIONSHIPS

A social network consists of a set of nodes connected via relations, which are also called links or edges



Implementation of Undirected graph

Unfortunately Java does not provide a complete implementation of the graph data structure. But, we can create a graph using a collection or using dynamic arrays.

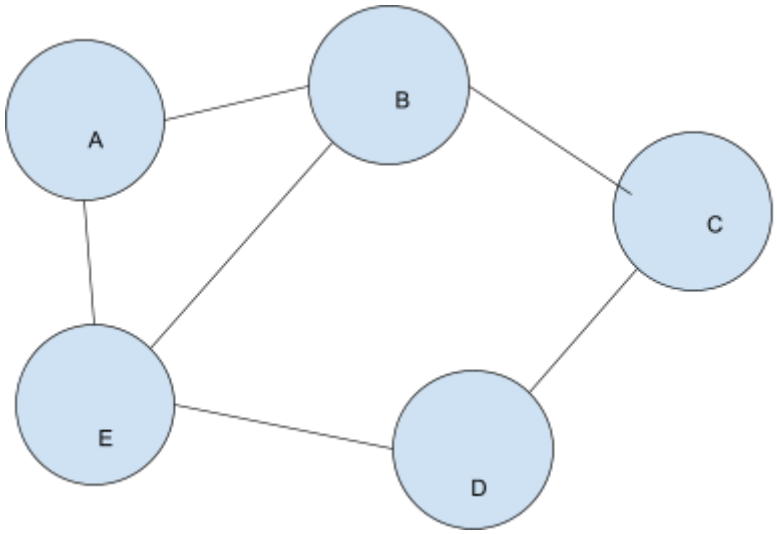
Usually, we implement graphs in Java using HashMap collection. HashMap elements are in the form of key-value pairs. We can represent the graph adjacency list in a HashMap.

An **adjacency matrix** and **list** can both be used to represent a graph. For the matrix, number the vertices of the directed graph 1, 2, ..., n.

ADJACENCY MATRIX

	A	**B**	**C**	**D**	**E**
A	0	1	0	0	1
B	1	0	1	0	1
C	0	1	0	1	0
D	0	0	1	0	1
E	1	1	0	1	0

TO UNDIRECTED GRAPH



And code:

```
class Edge {
   int src, dest, weight;
   Edge(int src, int dest, int weight) {
         this.src = src;
         this.dest = dest;
         this.weight = weight;
      }
}
class Graph {
   static class Node {
      int value, weight;
      Node(int value, int weight)  {
         this.value = value;
         this.weight = weight;
      }
   };
List<List<Node>> adj_list = new ArrayList<>();
   public Graph(List<Edge> edges) {
      for (int i = 0; i < edges.size(); i++)
         adj_list.add(i, new ArrayList<>());
      for (Edge e : edges){
         adj_list.get(e.src).add(new Node(e.dest, e.weight));
      }
   }
   public static void printGraph(Graph graph)  {
      int src_vertex = 0;
      int list_size = graph.adj_list.size();
      System.out.println("The contents of the graph:");
      while (src_vertex < list_size) {
         for (Node edge : graph.adj_list.get(src_vertex)) {
            System.out.print("Vertex:" + src_vertex + " ==> " + edge.value +  " (" + edge.weight +
")\t");}
          System.out.println();
         src_vertex++;
      }
   }
}
class Main{
   public static void main (String[] args) {
      List<Edge> edges = Arrays.asList(new Edge(0, 1, 2),new Edge(0, 2, 4),
            new Edge(1, 2, 4),new Edge(2, 0, 5), new Edge(2, 1, 4),
            new Edge(3, 2, 3), new Edge(4, 5, 1),new Edge(5, 4, 3));
      Graph graph = new Graph(edges);
      Graph.printGraph(graph);
   }
}
```

3/21/2021 - algo_pro team