

# Graph theoretic approach for automating electric circuit

Leanne Dong

Gina Cody School of Engineering and Computer Science  
Concordia University Montréal

July 7, 2020

# Overview

- 1 Graph theoretic foundation
  - Basic concepts
  - Applications to electric circuits
- 2 Network topology for circuit
- 3 Setting up Kirchhoff's equation
  - Find spanning trees and fundamental set of circles
  - Setting up the Kirchhoff's law equations
- 4 Solving the Kirchhoff's law equations
  - Numerical methods
- 5 Exercises and Projects
  - Exercise
  - Student projects

# Graph vocabulary

- **Graph**  $G = (V, E)$  is a finite set of points ( $V$ : vertices or nodes) and lines joining some of these points ( $E$  edges, arcs, links); Graphs can be directed and **undirected**;
- Cycle/Circuit/Loop: a path such that the final node of the last link coincides with the initial node of the first link;
- Tree: a graph that is connected but has no cycles;
- Spanning tree  $T$ : For a connected graph  $G$ , this is just a subgraph that contains all nodes of  $G$ ;
- Branches: The edges of a spanning tree  $T$ ;
- Incidence matrix  $A_{inc} = \{a_{ij}\}$  is

$$a_{ij} = \begin{cases} 1 & \text{if } j\text{th edge is incident on the } i\text{th node} \\ 0 & \text{if } j\text{th edge is not incident on the } i\text{th node} \end{cases}$$

# Graph vocabulary

- Complete graph: this is a graph where each node is joined by an arc to every other node.
- Partial graph: a graph that contains all the nodes of the original graph but only some edges.
- subgraph: a graph that contains only a subset of the nodes of the original and all of the edges that join them.
- connected graph: a graph is connected if there is at least one chain from any nodes to every other node.

# Why graph theory for electric circuits?

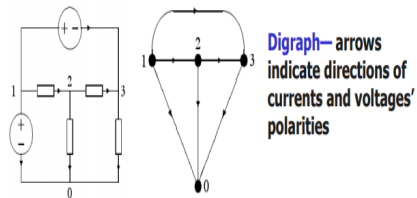
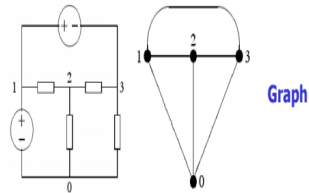
- Powerful, scalable methods to model complex networks arising many applications
- A Simple method which allows us to derive the differential equations for even very complicated circuits

# Graph for electric circuit: The idea

- Algebraic graph theory for electronic network simulation first due to Poincaré and Veblen
- ▶ Gotlieb and Corneil (1967)'s idea of generating spanning trees by finding fundamental set of cycles (cycle base)
- The above was further improved by ▶ Paton (1969). The DFS search was used to construct cycle base: Computational very efficient!
- Further improved by ▶ Tiernan (1970). Efficient but complicated. Fortunately this has been implemented in the Boost C++ libraries. See in particular the graph library . Boost::graph provide a comprehensive range of headers including ▶ Tiernan's efficient search algorithm in finding the fundamental cycles of the graph.

# Type of graphs in electrical theory

- Intuitively, circuits are graphs that describe the interconnection of electrical elements
  - passive elements such as resistances, capacitances, inductances
  - active elements
  - sources (or excitation)
- Two variables: voltage ( $V$ ) and currents ( $I$ ).



# Loop

- A loop is a set of branches of graph forming a closed path.
- Example: branches  $\{a, c, d\}$ ,  $\{b, d, e\}$  and  $\{a, b, e, c\}$ .

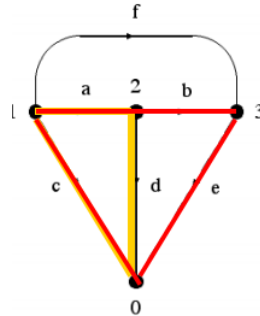


Figure: Branches and loops



# Tree and co-tree

- A **tree** is a set of branches of a graph which contains no loop.
- Thus, a tree is a maximal set of branches can be connected without forming a loop.
- After a tree is chosen, the remaining branches form a **co-tree**

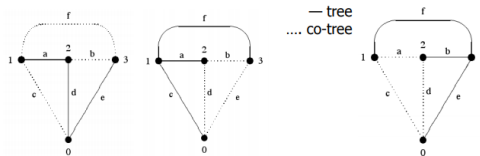


Figure: Tree and co-tree

# Graph visualization

We are currently developing an undirected graph extension of `Simple-graph-tool`, which is a free tool visualizing graph theory and supporting weighted directed graph. Our **Main technology** are Qt (a framework for making GUI for applications in C++ language) and CMake (native build environment across platform)

# A very simple example

Table: PSPICE netlist.

I1	0	1	1Amp
R2	1	0	1Ohm
R3	1	2	1Ohm
R4	2	0	1Ohm

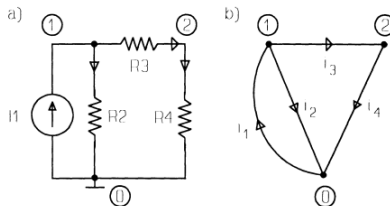


Figure: A simple circuit and its topology

# Kirchhoff's law again

**KVL:** At each loop or mesh,

$$\sum_{loop} V_k = 0$$

**KCL:** At each node,

$$\sum I_{inflow} = \sum I_{outflow}$$

## A very simple example: KCL

The KCL equations w.r.t. node 1, 2 and 0 are

$$-I_1 + I_2 + I_3 = 0$$

$$-I_3 + I_4 = 0$$

$$I_1 - I_2 - I_4 = 0$$

Linear dependency!  $\implies$  , Reference node at which the KCL equations may be omitted.

## A very simple example: KCL

Suppose we take node 0 as the reference node, so eliminate the 3rd equation, hence

$$\begin{pmatrix} -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{pmatrix} = 0$$

or simply

$$AI = 0$$

where  $I$  is the vector of branch currents and  $A$  is known as the **incidence matrix** containing only coefficients 0 or 1,  $-1$ .

## A very simple example: KVL

- Certain assembly of branches form loops (i.e. closed paths)
- In our last example, the loops are  $\{I_1, R_2\}$ ,  $\{I_1, R_3, R_4\}$  and  $\{R_2, R_3, R_4\}$
- Conservation of energy: work done around a loop should be 0. Hence, the sum of voltages of branches constituting the loop must be 0. In the similar vein, the KVL equations can be written as a compact matrix form as

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = 0$$

or simply

$$BV = 0$$

## A very simple example: summary

By the superposition principle, the topological matrices  $A$  and  $B$  both contain the same full information on the assembly of branches.



# Analysis circuit networks

- **Mesh Current Analysis:** See tutorial
- Nodal Voltage Analysis

We choose Mesh current method over Nodal approach as we can exploit the independence of the loops. In this way, we obtain minimum set of equations needed to solve the circuit, without redundancies.

## Assumption of Mesh current method (verso.c):

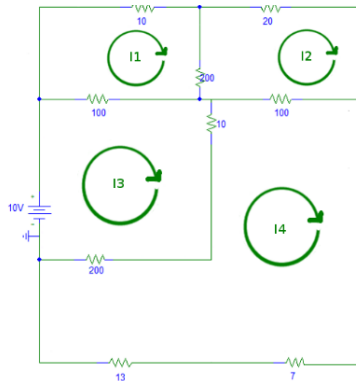
- Circuit is built up of several meshes with branches which contain resistance;
- By the superposition principle, one can split the analysis of the full circuit as the analysis of independent meshes;
- Set specific direction of the current in each mesh and so the original circuit is splitted into many meshes with given current direction (directed meshes).
- Superpose the solution to find the global solution of the circuit equations.

# The formal procedure

- 1 Describe the circuit in a computer-readable format (how data was stored and read);
- 2 Extract a set of independent loops;
- 3 For each loop, we associate a loop current;
- 4 Use KVL to set up a system of  $n$  equations, one for each independent loop;
- 5 Solve the system of equations and thus find the values of the loop currents ( $I$ );
- 6 Use the superposition principle to find the total current in each circuit branch.

## A concrete example

- We show a set of independent loops with associated loop currents
- Resistance are conventionally assumed to be ohms.
- Voltage values are in volt.



## A concrete example: setting up the Kirchhoff equations

At each branch  $b$ , the total voltage drop  $V^{(b)}$  across the branch is the sum of the voltage drop due to the loop currents  $I_{l'}$  that traverse that branch:

$$V^{(b)} = \sum_{b \in l} \sum_{l' \in L_b} R^{(b)} I_{l'}$$

So basically the inflow outflow relations,

$$(10 + 200 + 100)I_1 - 200I_2 - 100I_3 + 0I_4 = 0 \quad (1)$$

$$-200I_1 + (20 + 100 + 200)I_2 + 0I_3 - 100I_4 = 0$$

$$-100I_1 + 0I_2 + (100 + 10 + 200)I_3 - (200 + 10)I_4 = 10$$

$$0I_1 - 100I_2 - (10 + 200)I_3 + (100 + 7 + 13 + 200 + 10)I_4 = 0$$

# Why find spanning trees

The construction of structures as spanning trees and efficient traversal methods to access the information is crucial to solve the electrical or hydraulic network problem.

# Traverse information in a Graph: DFS vs BFS

Importance	Key	BFS	DFS
1	Definition	Breadth First Search	Depth First Search
2	Data structure	Use <b>queue</b> to find shortest path	Use <b>stack</b> to find shortest path
3	Source	BFS is better when target is closer	DFS is better when target is far
4	Speed	Slower	Faster
5	Time complexity	$O(V + E)$	$O(V + E)$

- Check [DFS graph traversal C++ implementation](#)
- Check our [simple-graph-tool](#) for demonstration of DFS and BFS

# Methods of finding cycle bases/Loop detection: Constructing spanning tree

- The central idea of construct cycle bases is to generate a spanning tree from the undirected graph.
- After the spanning tree is built, we look for all edges which are present in the graph but not in the tree.
- Adding one of the missing edges to the tree will form a cycle (the so-called fundamental cycles).
- A graph can have many different spanning trees depending on the chosen root node and the way the tree was built.
- Hence, each spanning tree constructs its own fundamental cycle.



- Every fundamental cycle form a cycle basis. The set of all fundamental cycles forms a cycle bases. See [▶ Tutorial](#)
- Every cycle basis is complete, it does not matter which spanning tree was used to generate the cycle basis, each basis is equivalently appropriate to construct all possible cycles of the graph.
- The typical algorithms for constructing spanning trees are DFS or BFS.
- DFS is preferred for efficiency: each node differs by only one edge from the main branch. Check tutorial [▶ here](#).

# Methods of finding cycle bases/Loop detection: Literature

- [▶ Gotlieb and Corneil \(1967\)](#): Implemented by a Physicist Prof. Milotti's group, see [▶ CircuitMath](#).
- The above was further improved by [▶ Paton \(1969\)](#). The DFS search was used to construct cycle base: Computational very efficient!
- Further improved by [▶ Tiernan \(1970\)](#).

In term of storage and speed, Paton and Tiernan are better. Moreover, Tiernan's algorithm has been written into the modern C++ numerics libraries like [▶ Boost](#)

# Method of finding cycle bases/Loop detection: Gotlieb and Corneil (1967)

## Assumptions

- Undirected graph is represented by its adjacency matrix  $A = \{a_{ij}\}$  is

$$a_{ij} = \begin{cases} 1 & \text{if } j\text{th node is connected with the } i\text{th node} \\ 0 & \text{if } j\text{th node is not connected with the } i\text{th node} \end{cases}$$

- Graph is connected
- Each connected component can be separated

# Method: Gotlieb and Corneil (1967)

## Key steps

- Construct a set of disjoint trees by choosing some edges.
  - ① The set of  $n$  node is partitioned w.r.t. some chosen edges to form a set of disjoint trees with adjacency matrix  $B$ .
  - ② For each row  $i$  of  $A$ , set  $b_{ji} = b_{ij} = 1$  if  $a_{ij}$  is the first **superdiagonal** element of the  $i$ th row of  $A$  to equal 1; otherwise set  $b_{ji} = b_{ij} = 0$ .
    - ① For  $i = 3$ , the **superdiagonal** elements are  $j = 4, 5, 6, 7$ .
    - ② The first **superdiagonal** with  $i = 3$  is  $j = 4$ , hence  $B(3, 4) = B(4, 3) = 1$ . The first **superdiagonal** with  $i = 4$  is  $j = 7$ , hence  $B(4, 7) = B(7, 4) = 1$ .

	1	2	3	4	5	6	7
1	0	0	1	1	0	0	1
2	0	0	0	1	1	1	0
3	1	0	0	1	0	0	0
4	1	1	1	0	0	0	1
5	0	1	0	0	0	0	0
6	0	1	0	0	0	0	0
7	1	0	0	1	0	0	0

	1	2	3	4	5	6	7
1	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0
3	1	0	0	1	0	0	0
4	0	1	1	0	0	0	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0

## Method: Gotlieb and Corneil (1967)

- Divide the set of nodes of the graph into connected components w.r.t. the edges chosen in Matrix B.
- Join the components by adding appropriate edges to yield a spanning tree.
- Produce a set of fundamental cycles.

## Method: Paton (1969)

This has been implemented by a computer scientist Philipp Sch. He demonstrated how in principle one can enumerate all cycles of a graph. However the method does not scale up well.

## Method: Tiernan (1970)

A theoretically very eddifficient search algorithm that uses an exhaustive search to find all elementary circuits of a directed graph. For more detail, one shall consult with

- The official Boost Graph Library site : [▶ boost::Graph](#)
- Textbook: [▶ The Boost Graph library](#)
- Uni-Resource: [▶ Brown](#)
- Tutorial: [▶ technical-recipes](#)

# Circuit equations

Having found the independent loops, we can set up the system of equations like the one we show before and solve them. To fix idea, we consider mainly simple linear circuits. The solutions can be trivially attained using the LU decomposition method via some scientific software. One choice is the routines **ludcmp** and **lubksb** in the Numerical Recipes C library



# Branch currents and voltages

- The solution of linear equation yields all the independent loop currents;
- From them, we can find the branch currents by summing over all the independent loop currents that traverse a given branch.
- By looping over all branches, one can construct a new matrix  $I = \{i_{ik}\}$  where the element  $i_{jk}$  stores the current in the branch that links the  $i$ -th node with the  $k$ -th node.
- Similarly, we calculate a voltage matrix  $V = \{v_{jk}\}$ , where the matrix element  $v_{jk} = r_{jk} i_{jk}$  stores the voltage drop in the branch that links the  $j$ th node with the  $k$ th node.
- Voltage are set by the power supplies w.r.t. a ground terminal: one can find the voltage of any node w.r.t. ground by following any path along the circuit that joins the node with the ground node and summing all the voltage drops.
- Finally, save the results on a file and save for further analysis

## Potential improvements

- A comparison of graph-based method with the nodal method used in established programs such as SPICE
- Optimizing circuit representation in the description file
- Deal with sparse matrix case, the program efficiency could be boosted with the use of sparse matrix code
- code could be cleaned up with C++, performance could be boosted via better data structures, such as linked lists (represent loops) and other graph structures
- Can be extended to handle time-dependent voltage and currents, instead of the linear system we have seen, we can manipulate the corresponding DEs and use a DE solver
- Could be extended to include non-linear elements like semiconductor diodes, transistors, etc, this would require the intro of a nonlinear equation solver

## Potential improvements

- Can be implemented in Object-Oriented sense, utilize the efficient C++ library like Boost.
- Could work on a graphical interface both for input - by placing components in a special window - and for output - by automatically plotting output functions
- Could be used in the analysis of fluid flow - using the analogy between hydraulic and circuit networks.

**Rule of thumb for High performance code:** Efficiency with algorithms; Performance through data structure.

## Proposal: object oriented algorithms in C++

- OOP features allow handling data more efficiently.
- C++ is the most ideal language for building either an academic or for a live simulation system given **Performance** is the key.
- In case any security issue arise from accepting data over a network, we can consider switching to Rust. From C++ to Rust is not a long learning curve as they are relative.
- Java, C# are relatively memory safe. But low in performance compared to C++.
- We would encounter some PDEs in the CFD component later on, C++ allows us to integrate GPU in obtaining massively-parallel PDE solutions. Specifically, for solving large, sparse linear or nonlinear system.

# Numerical Algorithms

- Power system is represented by thousands of differential or algebraic equations (DAEs).
- DAEs are discretized using an integration algorithm.
- These equations are usually solved through Newton iteration.
- The core of the solution of the nonlinear system of equations is the solution of the linear system of equations;



# Algorithms for equations solver

- Gaussian elimination (Only for linear equations)
- Iterative methods: **Newton-type** algorithms;
- Hardy-cross method: suitable for large network

## Solving the previous concrete circuit problem

Simplify (1), write in matrix form, we have

$$\left( \begin{array}{cccc|c} 310 & -200 & -100 & 0 & 0 \\ -200 & 320 & 0 & -100 & 0 \\ -100 & 0 & 310 & -210 & 10 \\ 0 & -100 & -210 & 330 & 0 \end{array} \right)$$

### **Solution:**

We invoke C++ to solve the problem for us.

[https://godbolt.org/z/eTu\\_xC](https://godbolt.org/z/eTu_xC)

**Exercise:** Solve the above using nodal analysis.



# Solution: implementation in C++ with Eigen

```
#include <iostream>
#include </usr/include/eigen3/Eigen/Dense>

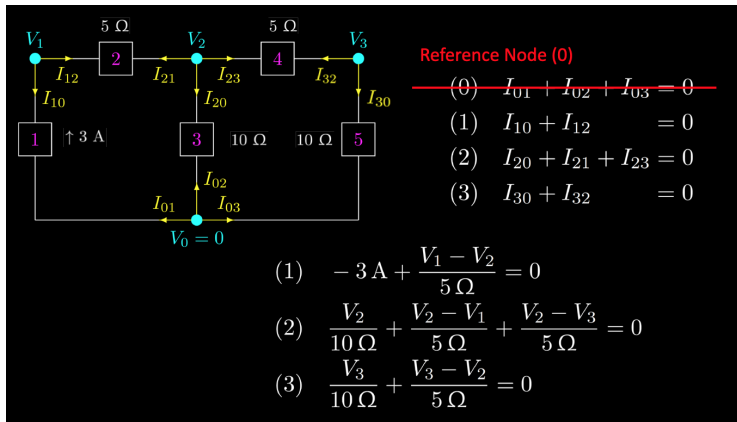
using namespace std;
using namespace Eigen;

int main()
{
    Matrix4f A;
    Vector4f b;
    A << 310,-200,-100,0, -200,320,0,-100, -100,0,310,-210, 0,-100,-210,330;
    b << 0, 0, 10,0;
    cout << "Here is the matrix A:\n" << A << endl;
    cout << "Here is the vector b:\n" << b << endl;
    Vector4f x = A.colPivHouseholderQr().solve(b);
    cout << "The solution is:\n" << x << endl;
}
```

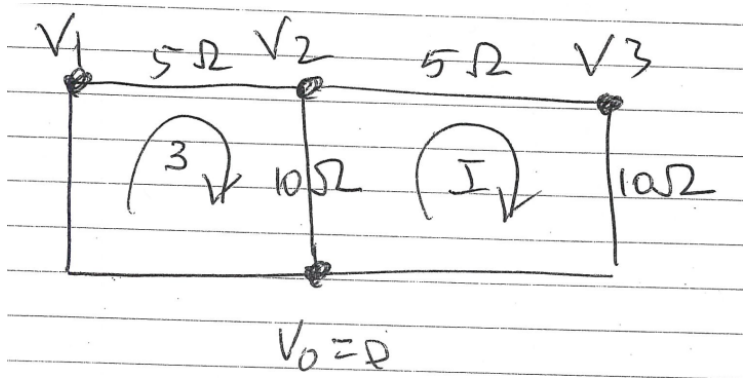
The solution is found to be 0.199872, 0.190431, 0.238743, 0.209634.



# Generic Example of a Network model (Linear case, Juergen, 2019)



# Network graph



$$\text{KVL: } \sum_{\text{loop}} V_k = 0$$

# Solution

Assume the positrons approach. That is

$$V_a \xrightarrow[R]{I} V_b \quad I > 0 \text{ implies } V_a > V_b \implies IR = V_a - V_b$$

**Circuit A:**

$$V_1 = 3 \times 5 + 3 \times 10 - 10I = 0$$

**Circuit B:**

$$-5I - 10I - 10I + 3 \times 10 = 0$$

implies  $I = \frac{30}{25} = \frac{6}{5}$ , then

$$V_1 = 45 - 10I = 33.$$

We now find  $V_2$ ,  $V_3$  by

$$V_1 - V_2 = 3 \times 5 \rightarrow V_2 = V_1 - 15 = 18$$

$$V_2 - V_3 = 5I = 6 \rightarrow V_3 = 12$$

## Example: Linear case

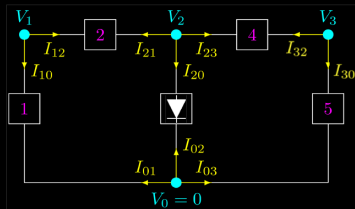
```
#include <iostream>
#include </usr/include/eigen3/Eigen/Dense>

using namespace std;
using namespace Eigen;

int main()
{
    Matrix3f A;
    Vector3f b;
    A << 0.2,-0.2,0,  -0.2,0.5,-0.2,  0,-0.2,0.3;
    b << 3, 0, 0;
    cout << "Here is the matrix A:\n" << A << endl;
    cout << "Here is the vector b:\n" << b << endl;
    Vector3f x = A.colPivHouseholderQr().solve(b);
    cout << "The solution is:\n" << x << endl;
}
```

# Generic Example of a Network model (Nonlinear case, Juergen, 2019)

## A Generic Example of a „Network Model“ Applied to a DC Network with a Diode



$$(1) \quad -3 \text{ A} + \frac{V_1 - V_2}{5 \Omega} = 0$$

$$(2) \quad I_s \left( \exp \left( \frac{qV_2}{nkT} \right) - 1 \right) + \frac{V_2 - V_1}{5 \Omega} + \frac{V_2 - V_3}{5 \Omega} = 0$$

$$(3) \quad \frac{V_3}{10 \Omega} + \frac{V_3 - V_2}{5 \Omega} = 0$$

# The Shockley Equation

$$I = I_s \left( \frac{qV}{n\kappa T} - 1 \right) = \left( \frac{V}{nV_T} - 1 \right)$$

where

- ❶  $I_s$  - **saturation current**.  $I_s \sim 10^{-14}$  for small signal diodes at 300 Kelvin: .
- ❷  $n$  - emission coefficient:  $n = 1, 2$  for small signal diodes.
- ❸  $\kappa = 1.38 \times 10^{-23} \text{ J/K}$  - the Boltzman's constant.
- ❹  $q = 1.60 \times 10^{-19} \text{ C}$  - the charge of the electron;
- ❺  $V_T$ -thermal voltage-  $V_T = \kappa T / q$

We will now implement this in C++. Assume  $n = 2$ ,  $T = 300$ ..  
Along with other constants, these together yields  $\frac{q}{n\kappa T}$  The  
complete example can be easily implemented in C++ with Eigen  
at [https://godbolt.org/z/p\\_LVgt](https://godbolt.org/z/p_LVgt).



# Exercise

Implement the Juergen's Diode example using mesh current method by hand. Then solve in your favorite programming languages. Solution can be found at [▶ my git repository](#).

# Creating adjacency matrix

```
#include <iostream>
#include <vector>
#include <utility>
void print_graph(const std::vector<std::vector<int> > &adj);
void addEdge(std::vector<std::vector<int> > &adj, int u, int v);
//double vector is bad as it triggers non continuous allocation and
hinder cache access; Try to improve it! */

int main()
{
    // Initialise array to hold adjacency matrix, vec<>s is already dynamic
    std::vector<std::vector<int> > adj(3, std::vector<int>(3));
    addEdge(adj,0,1);    // edge from node 0 to node 1
    addEdge(adj,0,2);
    addEdge(adj,1,2);
    print_graph(adj);
}

void print_graph(const std::vector<std::vector<int> > &adj)
{
    for(std::size_t i = 0; i < adj.size(); i++ )
    {
        for(std::size_t j = 0 ; j < adj[i].size(); j++ )
        {
            std::cout << adj[i][j]<< " ";
        }
        std::cout << std::endl;
    }
}

void addEdge(std::vector<std::vector<int> > &adj, int u , int v)
{
    adj[u][v]=1;
    adj[v][u]=1;
}
```

# Creating incidence matrix

```
#include <iostream>
#include <vector>
#include <utility>
void displayMatrix(const std::vector<std::vector<int>> & inc);
void addEdge(std::vector<std::vector<int>> & inc, int u, int v);
int edNo=0;

int main()
{
    // Initialise array to hold incidence matrix, vec<>s is already dynamic
    std::vector<std::vector<int>> inc(3, std::vector<int>(4));
    //there are 3 vertices and 4 edges in the graph pila has
    addEdge(inc, 0, 1);
    addEdge(inc, 0, 2);
    addEdge(inc, 1, 2);

    displayMatrix(inc);
}

void displayMatrix(const std::vector<std::vector<int>> & inc)
{
    for(std::size_t i = 0; i < inc.size(); i++)
    {
        for(std::size_t j = 0; j < inc[i].size(); j++)
        {
            std::cout << inc[i][j] << " ";
        }
        std::cout << std::endl;
    }
}

void addEdge(std::vector<std::vector<int>> & inc, int u, int v)
{
    //function to add edge into the matrix with edge number
    inc[u][edNo]=1;
    inc[v][edNo]=1;
    edNo++; //increase the edge number
}
```

# Project 1

- Learn thoroughly Milloti's source codes, re-implement Milotti's approach in C++ with a more efficient graph algorithms. (Hint: Try DFS to find spanning tree)
- Develop a simulator (Either Qt or OpenGL) to visualize the Electrical Distribution Network and displays results in a more intuitive manner for any specific configuration of the power flow problem. This visualizer also allows for easy interaction with the program.

## Project 2

- Implement a Diode example using Milotti's graph theoretic approach. (C or C++)
- **Hint:** Extend the current C project to nonlinear element and the construction of systems of differential equations to manipulate time-dependent signals, much like in the well-known program SPICE.

# Resources for learning C++

First, why? Check [▶ Why CS student must learn C++ as their main language](#). C++ was invented by computer scientist Bjarne Stroustrup. It is a languages that was created to fit for humans for all domains. Check Bjarne's short interview [▶ here](#).

## Books

- Effective Modern C++ by Scott Myers
- Discovering Modern C++: An intensive Course for Scientists, Engineers and Programmers
- C++17 in detail

**Video Lectures:** These are useful if you are unsure where to start and how to set things up.

- Cherno
- Modern C++ for Computational Scientists
- C++ Fundamentals Including C++17 by Kate Gregory