HW: Magic Spells

# Overview

The Professor for the Defense Against the Dark Arts (DADA) class is teaching students about the basic wand operations that form the basis for one of the most useful and powerful spell-casting disciplines.

As one knows, every witch and wizard has a wand. The wand has a length and orientation (or direction). The manipulation of these two properties channels and magnifies the user’s magic to create the various magical spell effects. Thus, the wand can actually be seen as a vector in the space (yes, the kind of vector we learn about in Calculus class) and the spells as simple linear algebra operations on these vectors.

In this homework assignment, you will implement operations and perform some of the more powerful dark art defense spells. These spells are commonly used by scientists working in research laboratories addressing challenges such as weather forecasting, producing animated movies, designing more efficient batteries, or scheduling airline pilots in order to minimize flight departure delays.

The spells you must implement are:

1. amax := Index of the first element with maximum absolute value.
2. asum := Sum of the absolute values of a vector’s elements.
3. axpy := Add a scalar multiple of a vector, “a x plus y” (y = a\*x + y).
4. copy := Copy vector x into vector y.
5. dot := Compute the dot product of two vectors.
6. norm2 := Compute the Euclidean norm (i.e. magnitude, or length) of a vector.
7. scale := Multiply a vector by a scalar (x = a\*x).
8. swap := Swap vectors x and y.

# 

# Requirements

When developing your solution to this problem, ensure that your submission conforms to the following requirements and assumptions:

* Name the source file containing the function definitions dada.cpp.
  + A valid but incomplete version of this file is included in the starter code.
  + You should start with the code provided and incrementally add your own code to implement the functions.
* Refer to dada.h for function declarations and descriptions.
  + A valid and complete version of this file is included in the starter code.
  + You can modify the file, but you do not need to, and therefore **should not**.
* You will submit exactly two files:
  + dada.cpp
  + dada.h
* You may use only the following header files
  + iostream (not needed, but helpful when debugging locally)
  + cmath
  + dada.h
* Comparing a signed value to an unsigned value (e.g., a loop variable, int i, and a length variable, unsigned int len), is generally unsafe. If either operand is unsigned, an unsigned comparison is used. This can cause problems (e.g. comparing a negative number to an unsigned number). Because of this, the compiler (when so instructed) will complain about the comparison. The solution is to only do a comparison between the same types of data (e.g. declare the loop variable as unsigned int i, instead). Further, whenever a value should never be negative, it is good practice to declare the variable which holds it as unsigned (the exception being floating-point numbers, which do not have a primitive unsigned type).
* You should unit test your code locally before submitting it to Mimir.
  + You may not find Mimir to be as helpful in this assignment.
* Suggested compilation: g++ -std=c++17 -Wall -Werror -Weffc++ -pedantic dada.cpp

# 

# Getting Started

1. Start early.
2. Download the starter code.
3. Read the header file.
4. Compile and run the program.
   1. It won’t do anything, but it also won’t crash
5. Submit it to Mimir.
6. Pick a test and a function to implement first and implement it just enough to pass the test
   1. Plan your program on paper (digital or analog) before mashing the keyboard. Think about how your approach would handle the same input that we have in this prompt.
7. Recompile and rerun.
   1. Check for errors.
   2. If no errors, move on
   3. Else, start debugging
8. Resubmit to Mimir.
   1. If the target test is passing, move on
   2. Else, start debugging.  
      Consider running your local machine by creating a main function that invokes the function that you are working on.
9. Continue by pickling a new test and writing just enough code to pass it (step 4)

# Submission

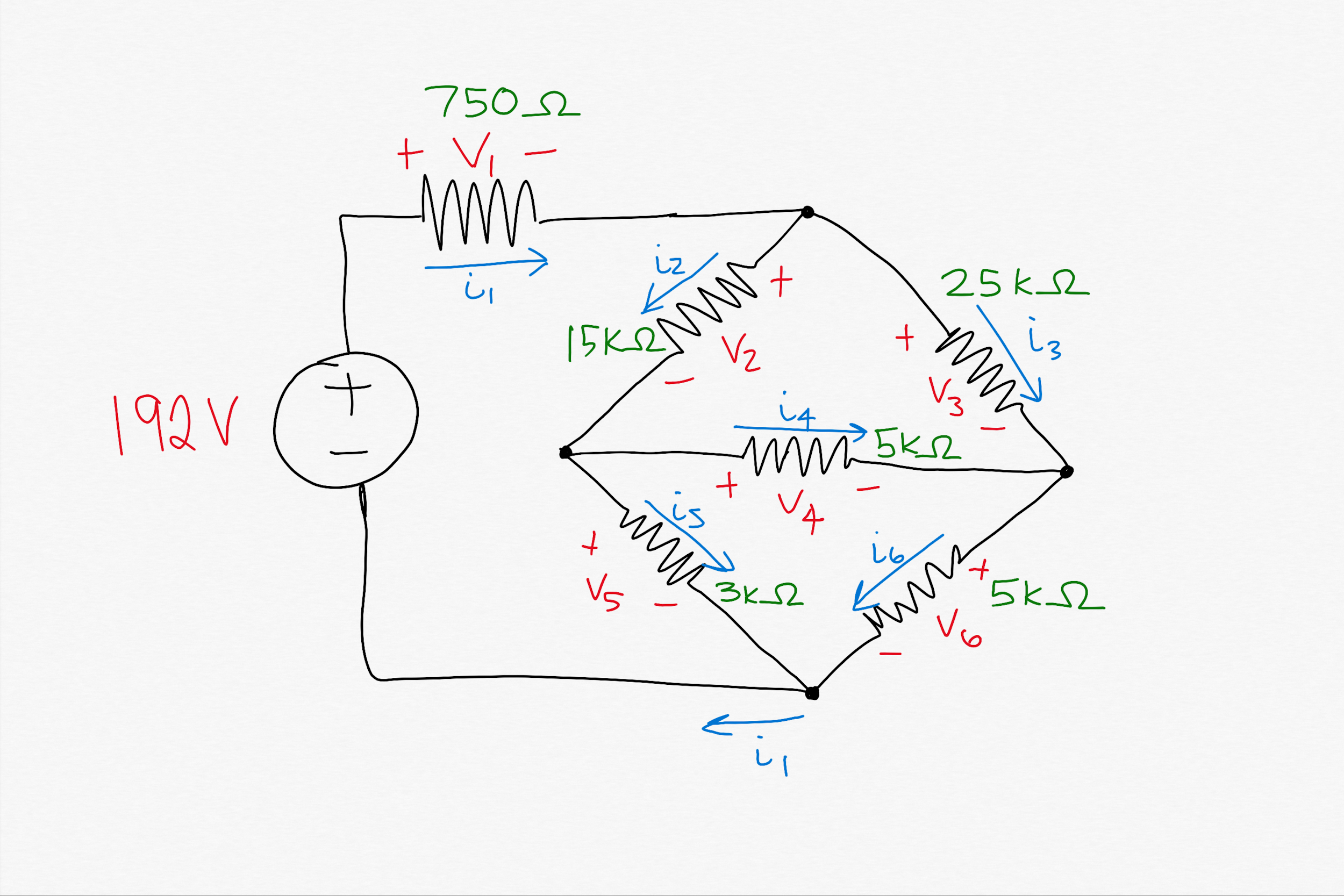
The source files to submit are named

1. dada.cpp
2. dada.h

# Have fun!

# Advanced Defense Against the Dark Arts

## Solving Resistive Circuits



P 3.51 (Electric Circuits, Nilsson & Reidel) Find the power dissipated in the 3kΩ resistor.

Solution:

1. Solve for i5.
2. Compute P = IR = i5 \* 3 kΩ

Solve for i5:

KCL equations:

i1 - i2 - i3 = 0

i2 - i4 - i5 = 0

i3 + i4 - i6 = 0

KVL equations: Ohm’s law: V = I \* R

V1 + V2 + V5 = 192 ⇒ 750i1 + 15000i2 + 3000i5 = 192

V1 + V2 + V4 + V6 = 192 ⇒ 750i1 + 15000i2 + 5000i4 + 5000i6= 192

V1 + V3 + V6 = 192 ⇒ 750i1 + 25000i3 + 5000i6 = 192

6 equations, 6 unknowns: hit it with ye olde *eliminatus* spell (Gaussian Elimination)

i5 = 10 mA ⇒ P = 10 mA \* 3 kΩ = 30 W

# Fitting Polynomials to Points

Given a set of *n* points, find the (*n*+1)-degree polynomial that intersects all *n* points.

For example, suppose the points were:

P1 = (-8, 840)

P2 = (-6, 120)

P3 = (7, 7920)

P4 = (5, 3024)

P5 = (3, 840)

These 5 points can be connected by a (unique) polynomial of degree 4:

ax4 + bx3 + cx2 + dx + e = 0

Plug in given (x,y) values:

a(-8)4 + b(-8)3 + c(-8)2 + d(-8) + e = 840

a(-6)4 + b(-6)3 + c(-6)2 + d(-6) + e = 120

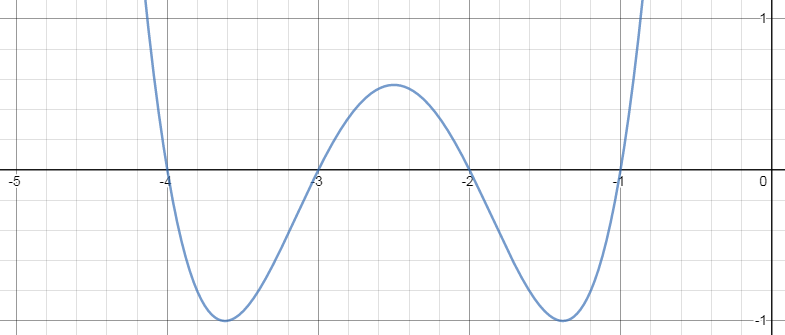
a(7)4 + b(7)3 + c(7)2 + d(7) + e = 7920

a(5)4 + b(5)3 + c(5)2 + d(5) + e = 3024

a(3)4 + b(3)3 + c(3)2 + d(3) + e = 840

5 equations, 5 unknowns: hit it with ye olde *eliminatus* spell (Gaussian Elimination)

Answer: x4 + 10x3 + 35x2 + 50x + 24 = 0



# Document Similarity

Given a set of documents, find the pair of documents that are most similar. Alternatively, given a single document and set of documents, find the document in the set which is most similar to the single document.

Solution

1. Represent documents as term vectors (frequency of each of a set of words).
2. Compute *cosine similarity* between documents D1 and D2 as the dot product of D1 and D2 divided by the product of the magnitude of the vectors: D1 . D2 / (|D1|\*|D2|)
   1. In DADA, use dot(D1, D2) / (norm2(D1) \* norm2(D2))
3. Find the maximum similarity between all pairs of documents.