|  |
| --- |
|  |
|  |
| |  | | --- | | Operations Research  21-393  Fall 2010  Kevin Jang Heather Lin Ting Ou-Yang | |
| |  |  |  | | --- | --- | --- | | November 28, 2010 | Course | **scheduling** | |

|  |
| --- |
|  |
|  |

# Table of Contents

# Motive………………………………………………………………………...3

# Visualization………………………………………………………………....4

# First Attempt………………………………………………………………....7

# Integer Linear Programming…………………………………………….....8

# Examples..…...……………………………………………………………..10

# Conclusion…...……………………………………………………………..16

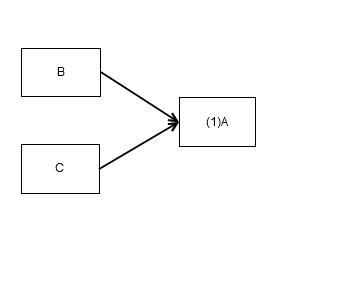
# Motive

# One of the biggest changes between high school and college for a lot of people is the sudden freedom to pick from a lot more classes than before. You don’t have to take an English course, a Mathematics course and a History course every semester. But this leaves a lot of questions about what you should take. Some people spend time taking classes they find fun. Others try to finish as fast as possible. And many consider the possibility of picking up majors or minors in other fields.     Once you start looking for an additional major or minor, you begin to wonder how many extra classes are needed. If you are getting two similar degrees such as a Computer Science and a Mathematics degree, there are a lot of overlaps and there are ways to double count many of your courses. For example, if you take Great Theoretical Ideas of Computer Science, that is one of the required courses for the first degree and counts as a Discrete Math course for the other. Our goal in this project is to develop a method that finds the most efficient way to take the fewest number of classes required for a given set of degrees. The set of degrees can be any combination of majors or minors, so as an example, we could find the fewest number of classes required to complete a degree in Operations Research with an additional major in Economics and a minor in Computer Science.

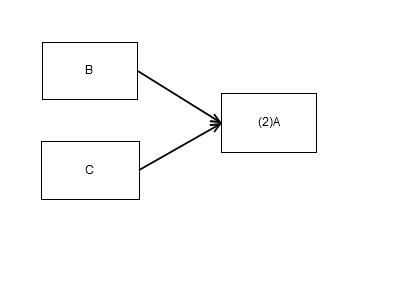
# Visualization of Problem

# The first step we took was to find an intuitive way to visualize the problem. At first we thought the best way to solve this problem was using a dependency tree, so we started visualizing it as a graph. The graph has rectangular nodes representing classes. Each class is connected to its prerequisites. If a class has prerequisites, the class name is preceded by a (x) with x being a number indicated the number of prerequisites needed for this class (examples to follow.

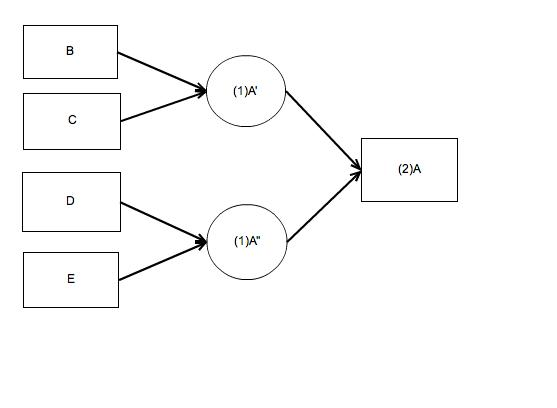
# If a class A requires one of two classes B or C as a prerequisite, it will be connected to both classes, but the number preceding the name will be (1), indicating that we only need to have taken one of B or C to proceed to the class A.



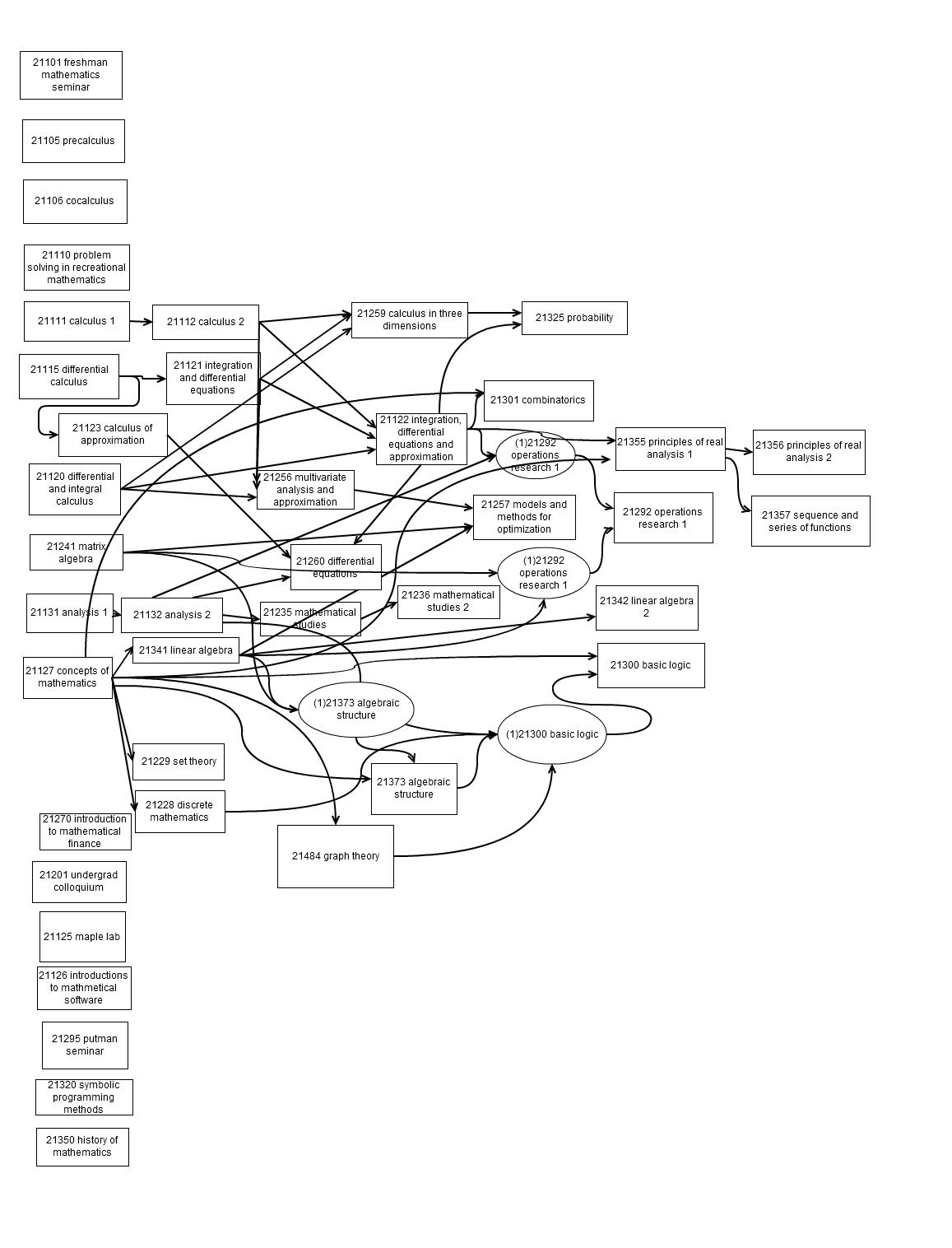
If a class A needs two classes B and C, both classes will be connected to the class as prerequisites and the preceding number will be (2), indicating both prerequisites are needed to continue.



# If a class A has prerequisites B or C and D or E, we set up dummy nodes which are the exact same as a regular node: node A’ is connected to B and C, and has prerequisite class count of 1, node A’’ is connected to D and E, and has prerequisite count of 1. Both dummy nodes A’ and A’’ have cost 0. A is connected to both A’ and A’’ and has a prerequisite class count.



# Our problem involves creating a graph that includes every single class that the university has to offer. Of course, this would be extremely tedious so the furthest we went was graphing about 90% of the courses available in the mathematics department. This should give a good idea of how complicated the whole graph could be.



# First Attempt

Our first attempt at solving the problem was to create a dependency tree. We thought this was the most intuitive way to solve the problem. Using the graph above we added a few variables to each node.

1) cost (C) - the cost of taking the class, for example we could measure in units  
2) additional\_cost (A) - minimum (cost + additional\_cost) of prerequisites of node  
3) prerequisite\_class\_count (P) - the number of prerequisites required to take the class

Note: we do not give variable A a value until step 4 of the algorithm

With these variables we implemented an algorithm. Given a set of required classes, this algorithm would find the fewest number of classes needed in order to fulfill the courses in the set. Our attempted algorithm is shown below.

Let T be total\_cost = 0

1. Underline required classes
2. T = T + Ci for all underlined classes i
3. Change C of underlined nodes to 0
4. Each node is now assigned A
   1. If there are no prerequisites, then A = 0
   2. Otherwise A = min{ A1 + C1, A2 + C2 +… + An + Cn} for prerequisites 1…n
5. Iterate through each underlined node in order of smallest A to largest A
   1. Underline prerequisite of Ni with lowest Ai
   2. For new underlined node, repeat steps 2 through 5
6. Once every underlined node has gone through step 5, T should include the cost of all underlined courses and the underlined courses should highlight the shortest paths to every class.

This algorithm is effectively a dynamic programming solution. Each time we underline a new node, the calculations for its costs are saved into the solution itself. After a few tests however, this algorithm proved to be inefficient and in some cases just plain wrong. Once we found this out, we realized that the most intuitive solution was not necessarily the best solution, nor was it the simplest. We proceeded to create a Linear Integer Program.

# Integer Linear Programming

We can model our problem with integer programming and it is in fact quite straightforward. Again, our question focuses on the minimum courses required, whether it is the minimum courses required to obtain a minor or double major. Therefore, our objective is minimizing the number of courses taken given a set of constraints. Here we describe the basic features of our system.

* For convenience we use variable Xi where i is the course number.
* Each course is a variable with possible values 0 or 1. Xk = 1 indicates that class k is being taken, and Xk = 0 indicates that the class is not being taken.
* If any course k is a requirement then we have Xk = 1 as a constraint. Ex. Course C is a requirement. We will have constraint XC = 1.
* Our objective will then be to minimize , where  are courses that are offered by the university.
* We categorized pre-requisites into five different types of pre-requisites below.

***1. Simple pre-requisite***

Suppose class A is a pre-requisite for class B. In other words, class B requires only class A. Then we have

.

This will ensure that when course B is taken, course A must be taken as well.

***2. either or pre-requisites***

Suppose course C requires course A or B. Then we have



This will ensure that when C is taken, either A or B must be taken.

***3. A and B pre-requisites***

Suppose course C requires course A and B.

Then we have two constraints:



This will ensure when C is taken, both A and B must be taken as well.

***4. C and either A or B***

Suppose course D requires either course A or B, at the same time must require C.

Then we have the constraints:



This ensures that when D is taken, C and either A or B must be taken.

***5. Pool of Courses***

Suppose we have major related requirements that require N number of courses from a pool of courses. To demonstrate we use the H&SS requirements as an example. Suppose it requires 4 non-technical courses from the pool of classes 

We have:



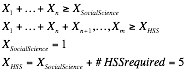
Where N= 4 means at least 4 courses in the pool of 20 courses must be taken to fulfill the requirement. We believe these cases should cover all the types of pre-requisites by using them in variations.

Note that this is also applicable to depth elective sequences.

***6. Overlap of pools of courses***

Suppose we are required to take 1 social science courses from a pool of social sciences courses that are also H&SS courses. On top of that, we are also to take 4 H&SS courses in general, without overlapping with the social science course.

We would model this by having:



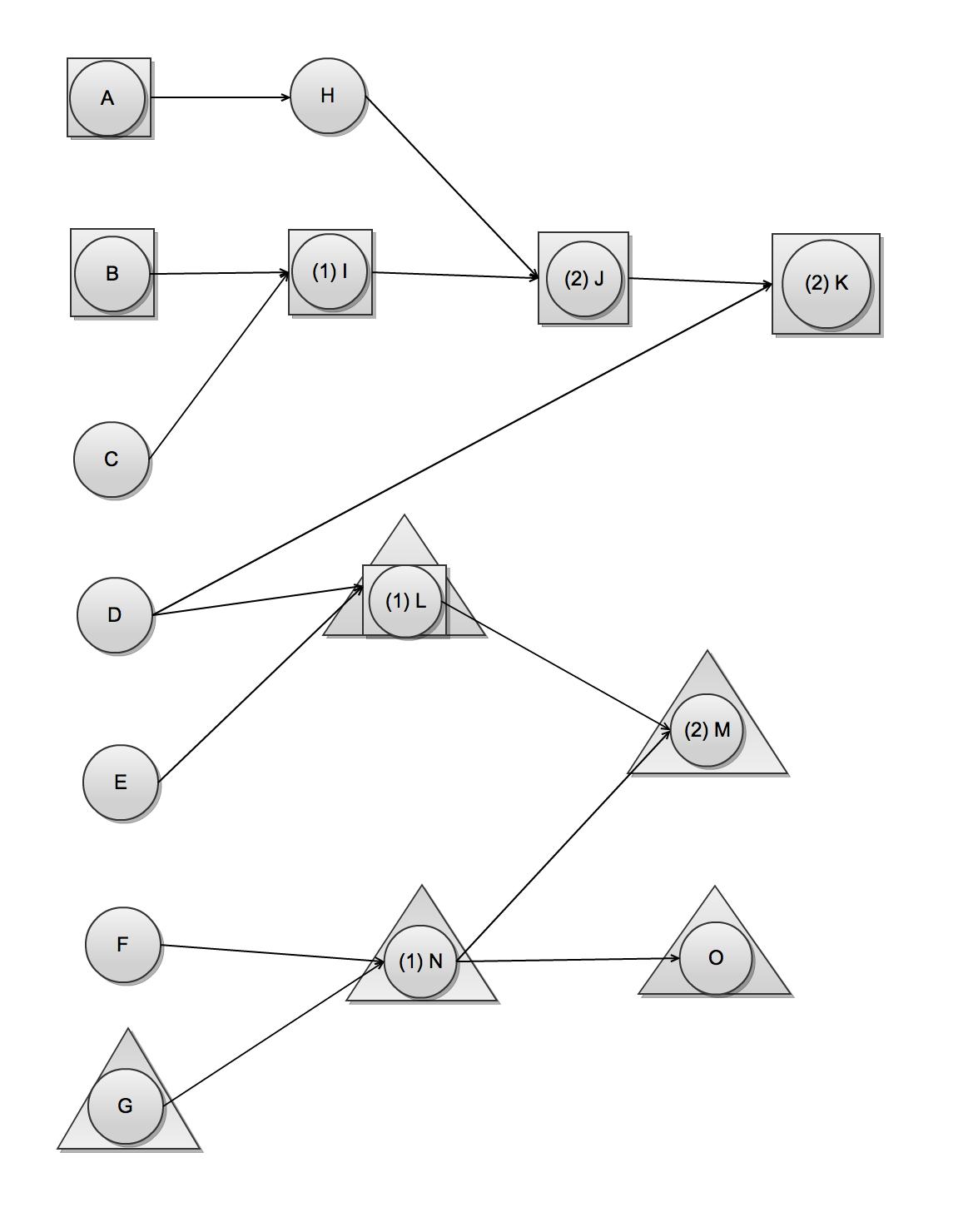
where  are courses that are in the pool of social sciences courses and are the courses that are only in the pool of H&SS courses but not in the social science pool.

# Examples

**First Example:**

*A small hypothetical model to examine the validity of our setup*

In this model we have set up course pre-requisites so that we can examine several of the different types of pre-requisites mentioned above. This is to demonstrate how integer programming can find the minimum courses required to obtain a double major in degree square and degree triangle. The details are listed below:

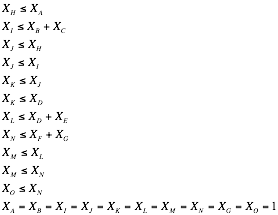


Degree I (Square) requires : A, B, I, J, K, L

Degree II (Triangle) requires: L, M, G, N, O

Variables: Xa~Xo

Minimize 

Subject to: 

min: a+b+c+d+e+f+g+h+i+j+k+l+m+n+o;

c1: a>=h;

c2:b+c>=i;

c3:j<=h;

c4:j<=i;

c5:k<=j;

c6:k<=d;

c7:l<=d+e;

c8:n<=f+g;

c9:m<=l;

c10:m<=n;

c11:o<=n;

a=1;

b=1;

i=1;

j=1;

k=1;

l=1;

m=1;

g=1;

o=1;

Solving this by simplex method we get

a = 1.0;

b = 1.0;

c = 0.0;

d = 1.0;

e = 0.0;

f = 0.0;

g = 1.0;

h = 1.0;

i = 1.0;

j = 1.0;

k = 1.0;

l = 1.0;

m = 1.0;

n = 1.0;

o = 1.0;

This demonstrates:

* Even though D and E are options for L. The program will pick D over E since D is a pre-requisite for K

**Second Example:**

*Real Application for A major in Math OR and minor in CS.*

We will demonstrate that this model will find the least number of courses required for this combination of major and minor. In particular, we want to show that it actually finds the minimum courses required. Suppose path A is the shortest path for obtaining a major in Math OR alone and path B is the shortest path for obtaining a minor in CS alone. We want to show that the shortest path to obtain both degree is not merely the addition of the two paths. If it was only the addition of two paths then there will be no point in solving this problem at all.

min: X21111 + X21112 + X21115+ X21120 + X21121 + X21122 + X21123 + X21127 + X21131 + X21132 + X21201 + X21228 +X21301 +X21484 + X21241 + X21257 + X21259 + X21260 +X21292 +X21341+X21369 + X15110 +X21393 +X06262 +X18202 + X21256 +X06221 +X33106 +X06100 + X36225+ X21325 + X36226 + X15359 + X36217 + X15211 + X15251 + X15121 +X36626 + X36625 + X36401 +X36410 + X15212 + X21270 + X21355 + X21365 + X21366 + X21370 + X21373 + X21420 + X21484 + X36461 + X36462 + X36463 + X36464 + X70371 +X70460 +X70471 + X21420 + X70207 + X36220 + X36247 + X36202 + X36207 +X70122 + X73100 + X73150 + X73200;

X21112 + X21120 + X21121 >= X21122;

X21111 >= X21112;

X21115 >= X21121;

X21228 + X21301 + X21484 >= Xdm

X21127 >= X21228;

X21122 >= X21301;

X21127 >= X21301;

X21127 >= X21484;

X21112 + X21120 + X21121 >= X21259;

X21122 + X21123 + X21132 >= X21260;

X21121 >= X21123;

X21131 >= X21132;

X21122 + X21132 >= X21292;

X21241 + X21341 >= X21292;

X21127 >= X21341;

X15110 >= X21369;

X21259 >= X21369;

X21257 + X21292 >= X21393;

X06262 +X18202 + X21241 + X21256 + X21341 >= X21257;

X06221 >= X06262;

X21122 >= X06262;

X06100 >= X06221;

X33106 >= X06221;

X21120 >= X33106;

X21122 +X21123 >= X18202;

X21112 + X21120 + X21121 >= X21256;

X21256 +X21259 >= X36225;

X36225 + X21325 >= Xprb;

X21122 + X21259 >= X21325;

X15359 +X21325 + X36217 + X36225 >= X36226;

X15211 >= X15359;

X15251 >= X15359;

X21241 >= X15359;

X21259 >= X15359;

X15121 >= X15211;

X21127 >= X15211;

X15110 >= X15121;

X21127 >= X15251;

X15110 + X15121 >= X15251;

X21112 + X21122 + X21123 + X21256 + 21259 >= X36217;

X36226 + X36310 + X36626 >= X36401;

X36625 >= X36626;

X21122 + X21123 + X21256 >= X36625;

X36401 >= X36402;

X21325 + X36217 + X36225 + X36625 >= X36410;

X15211 + X15212 + X21270 + X21355 + X21365 + X21366 + X21370 + X21373 + X21420 + X21484

+X36461 + X36462 + X36463 + X36464 + X70371 +X70460 +X70471 >= Xdepth;

X15251 >= X15212;

X21122 + X21127 >= X21355;

X21241 >= X21366;

X21270 >= X21370;

X21256 + X21259 >= X21370;

X21241 + X21341 >= X21373;

X21127 >= X21373;

X21260 + X18202 >= X21420;

X21370 >= X21420;

X36225 + X21325 + X36217 >= X21420;

X21127 >= X21484;

X36226 + X36626 >= X36461;

X36226 + X36626 >= X36462;

X21257 + X21292 >= X70371;

X70207 + X36202 + X36220 + X36247 >= X70371;

X21120 + X21121 >= X70207;

X36201 + X36207 + X36220 + X36247 + X70207 >= X36202;

X21112 + X21120 + X21121 >= X36207;

X21120 +X21121 + X21112 >= X36220;

X21120 + X21121 + X21112 >= X36247;

X21257 + X21292 >= X70460;

X70371 >= X70471;

X73100 + X21120 >= X73150;

X21120 >= X73200;

X73150 >= X73200;

X21256 + X21259 >= X73200;

*Minimum courses obtaining Math OR major = 27*

21120, 21122, 21127, 21201(x2), 21484, 21241, 21259, 21260, 21292, 21369, 15110, 21393, 36225, 36226, 36401,36410, 21366, 21373, 36461, 36462, 70460, 70122, 73100, 73150, 73200, 36402

*Minimum courses obtaining Math OR major and CS minor = 31*

15211,15251,15121,15212,15295,15354, 21120,21122,21127, 21201(x2), 21484, 21241, 21260,21292,21369,15110,21393,36225,36226,36401,36410,21355, 21373, 70460,70122,73100,73150,73200

We see that the flexibility to double count lies within the depth requirement of math OR major. Thus when we set CS minor to be one of the goals, the program will automatically find the shortest path to double count courses and minimize courses needed. We see highlighted in yellow the two sets of depth elective chosen are different. In particular the CS depth electives are chosen when considering the minor since their pre-requisites are already required by the minor or could be double counted.

# Conclusion

Our problem here was to find a way to minimize the number of classes required to finish up a set of degrees. We first wanted to come up with a way to visualize the problem. We visualized the problem in the form of a graph, and by creating a graph we thought the most intuitive way to solve this problem was by using that graph.

Again, this project shows that the most intuitive way to solve a problem is not necessarily the most efficient. Our dynamic programming solution proved to be overly complicated and problematic for us. We redefined our problem in the form of a integer linear program. Originally, we thought this would be overly complicated and tedious as well, but it proved to be much simpler than we anticipated.

Our solutions with this method are not guaranteed to be what a student wants to take if he or she were trying to obtain the majors specified. Class difficulty and unit count are not accounted for in this system. The students’ preferences for certain classes over others are not accounted for. This simply finds the fewest number of courses required to complete the given degrees. However, this proof of concept can applied to universities other than our own. Therefore, we have created a method for any student in any university to try and find the “fastest” way to finish a set of degrees.