Open in Colab

Math 210

Aaron Graybill

Problem Set 6

4/8/21

In [ ]:
```
import numpy as np
from scipy.optimize import linprog
import pandas as pd
import itertools
```

In [44]:
```
!git clone https://github.com/aarongraybill/Math210 ProblemSets
```

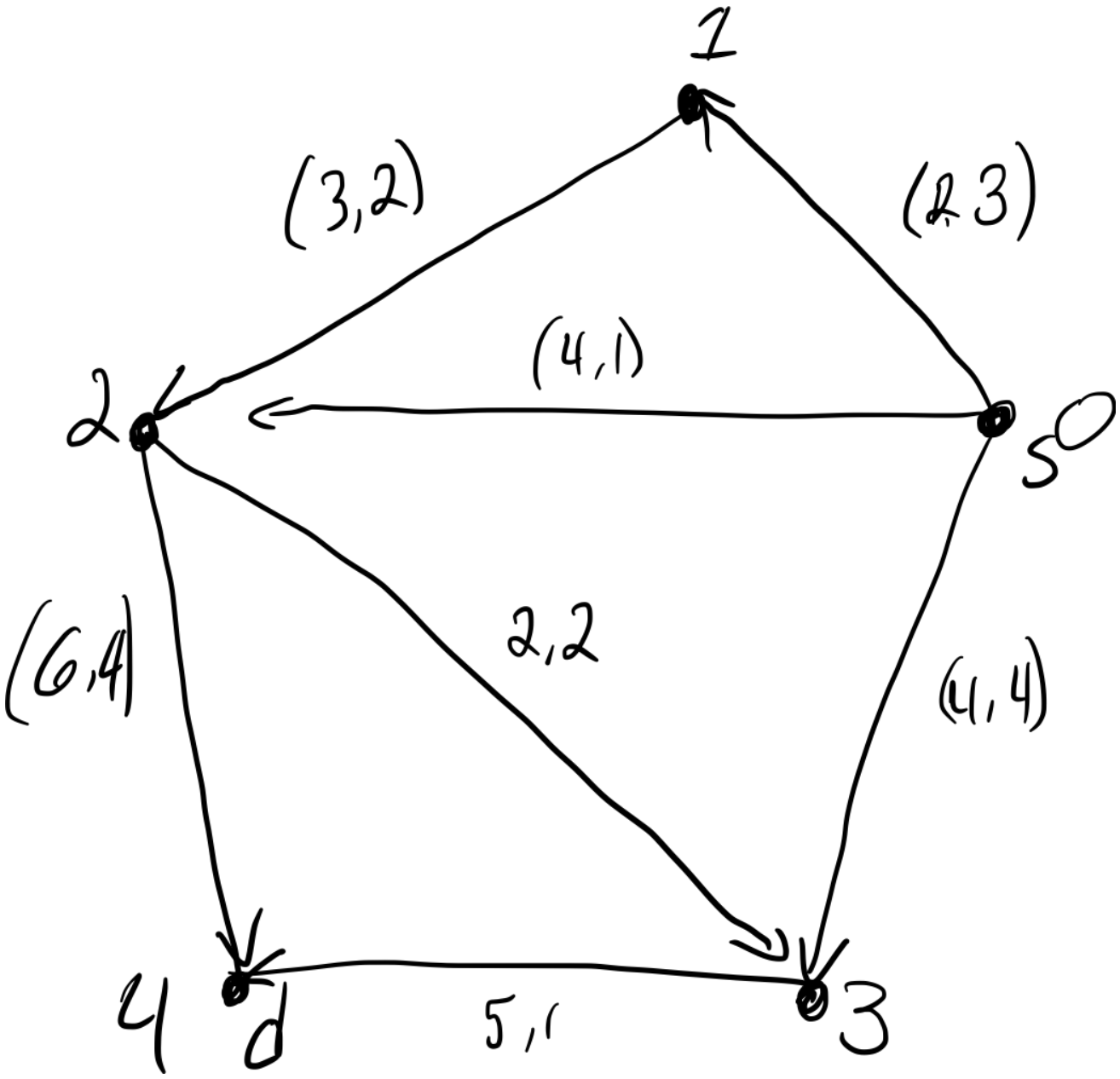
```
Cloning into 'ProblemSets'...
remote: Enumerating objects: 327, done.
remote: Counting objects: 100% (327/327), done.
remote: Compressing objects: 100% (293/293), done.
remote: Total 327 (delta 171), reused 90 (delta 31), pack-reused 0
Receiving objects: 100% (327/327), 7.73 MiB | 14.42 MiB/s, done.
Resolving deltas: 100% (171/171), done.
```

In [43]:
```
!rm -rf ProblemSets
```

Problem 1.

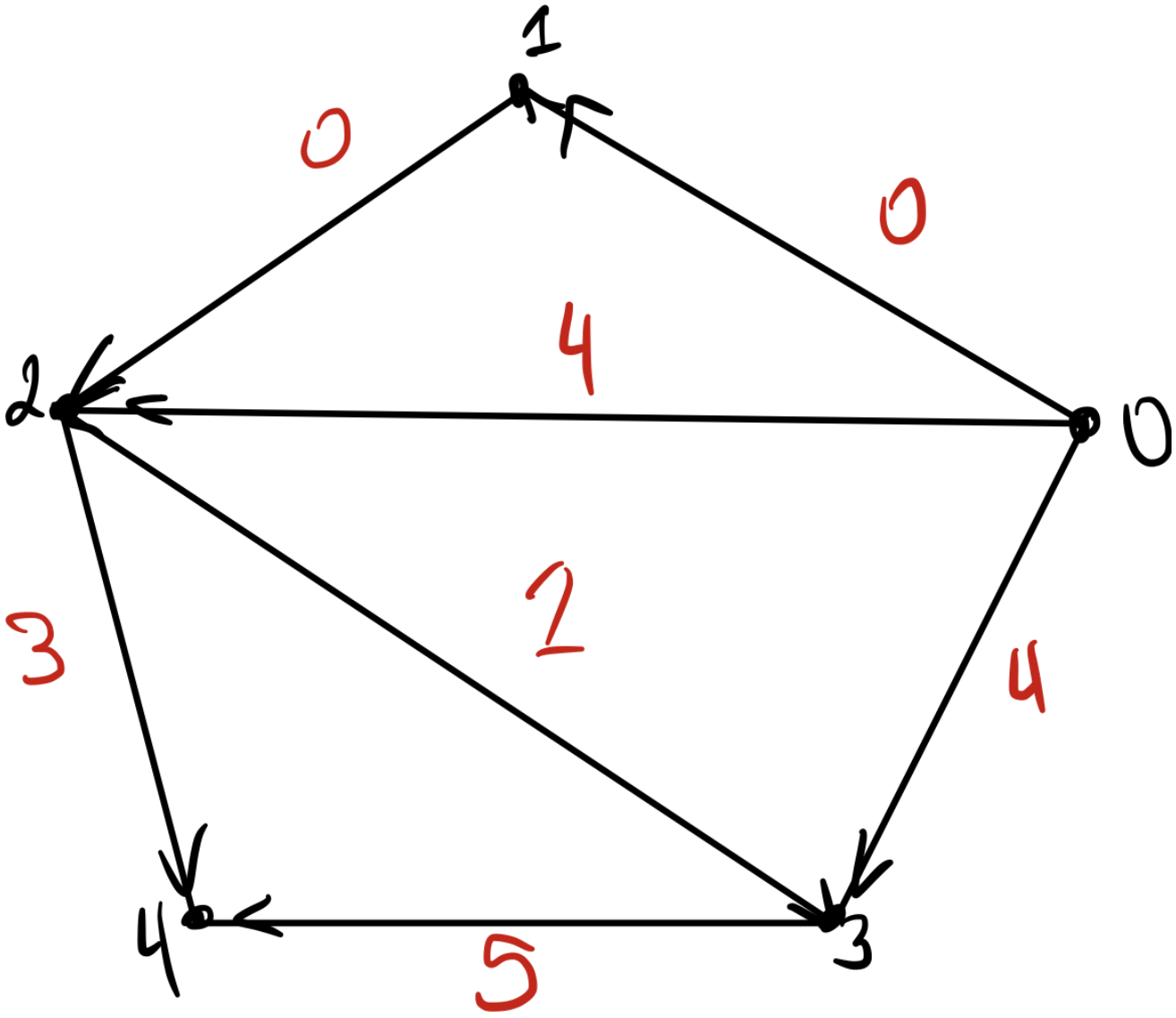
In [ ]:
```
# Gabe and I converted **his** maxflow code, but i had a substantial part
# in this min cost implementation

def maxflow(nvert,edgemat,reqflow) :
  nrows = edgemat.shape[0]
  A = np.identity(nrows)
  b = np.zeros((1,nrows))
  for i in range(nrows) :
    b[0,i] = edgemat[i][2]
  b2 = np.zeros((1,nvert-2))
  A3 = np.zeros((1,nrows))
  for i in range(nrows) :
    if edgemat[i][1] == nvert-1:
      A3[0,i] = 1
  A2 = np.zeros([nvert-2,nrows])
  for count, value in enumerate(edgemat[:,0:2]):
    colnum = count
    count = count+1
    if (value[0]==0) :
      A2[value[1]-1,colnum] = 1
    elif (value[1]==(nvert-1)) :
      A2[value[0]-1,colnum] = -1
    else :
      A2[value[1]-1,colnum] = 1
      A2[value[0]-1,colnum] = -1
  c = edgemat[:,3]
  #print(A)
  #print(c)
  reqflow = np.array([[reqflow]])
  print(linprog(c,A_ub=A,b_ub=b,A_eq=np.append(A2,A3,axis=0),b_eq=np.append(b2,reqflow,axis=1),method='simplex')
```



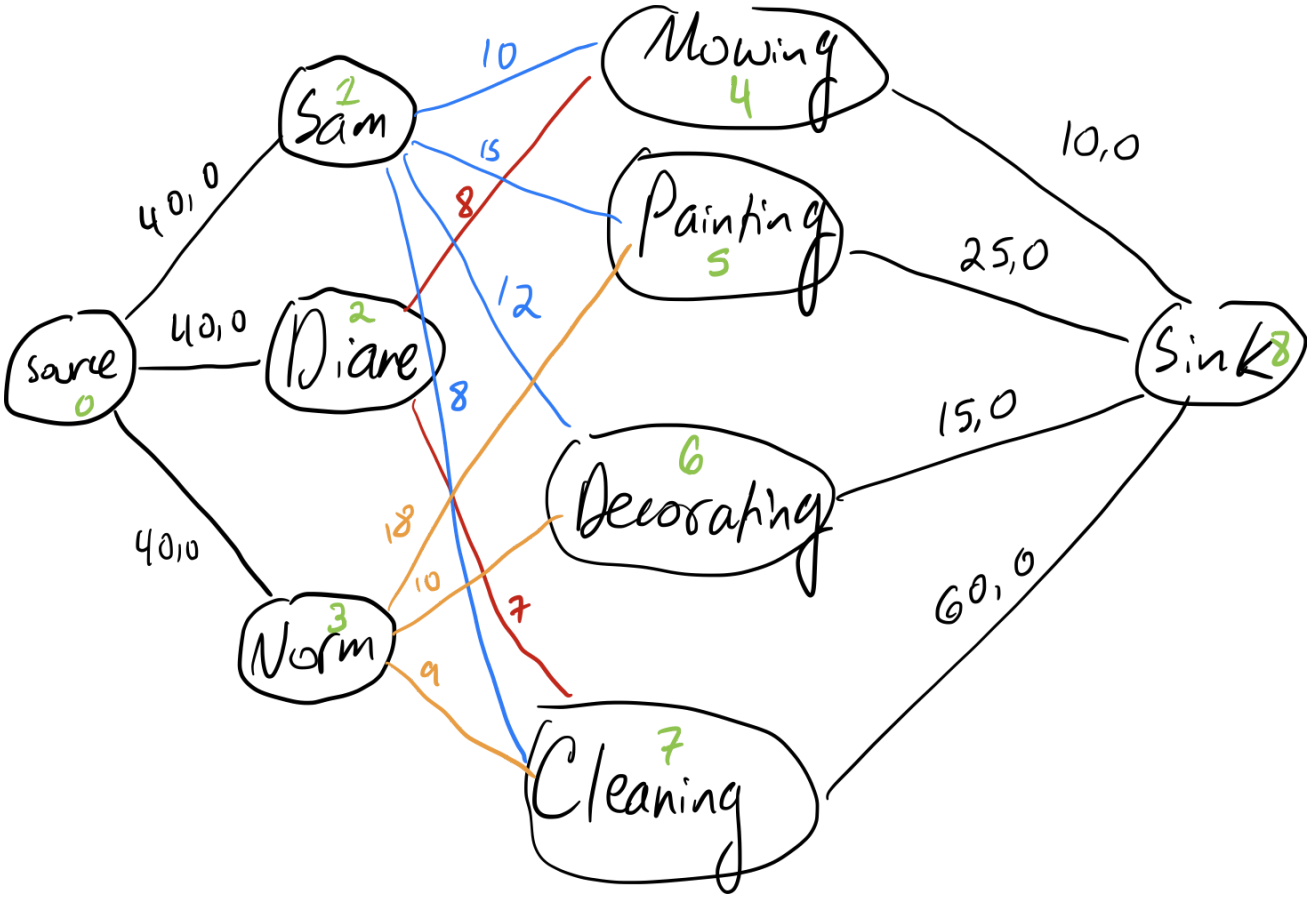
In [ ]:
```
a=np.array([[0,1,2,3],
            [0,2,4,1],
            [0,3,4,4],
            [1,2,3,2],
            [2,3,2,2],
            [2,4,6,4],
            [3,4,5,1]])
maxflow(5,a,8)
```

```
     con: array([0., 0., 0., 0.])
     fun: 39.0
 message: 'Optimization terminated successfully.'
     nit: 13
   slack: array([2., 0., 0., 3., 1., 3., 0.])
  status: 0
 success: True
       x: array([0., 4., 4., 0., 1., 3., 5.])
```



Problem 2.

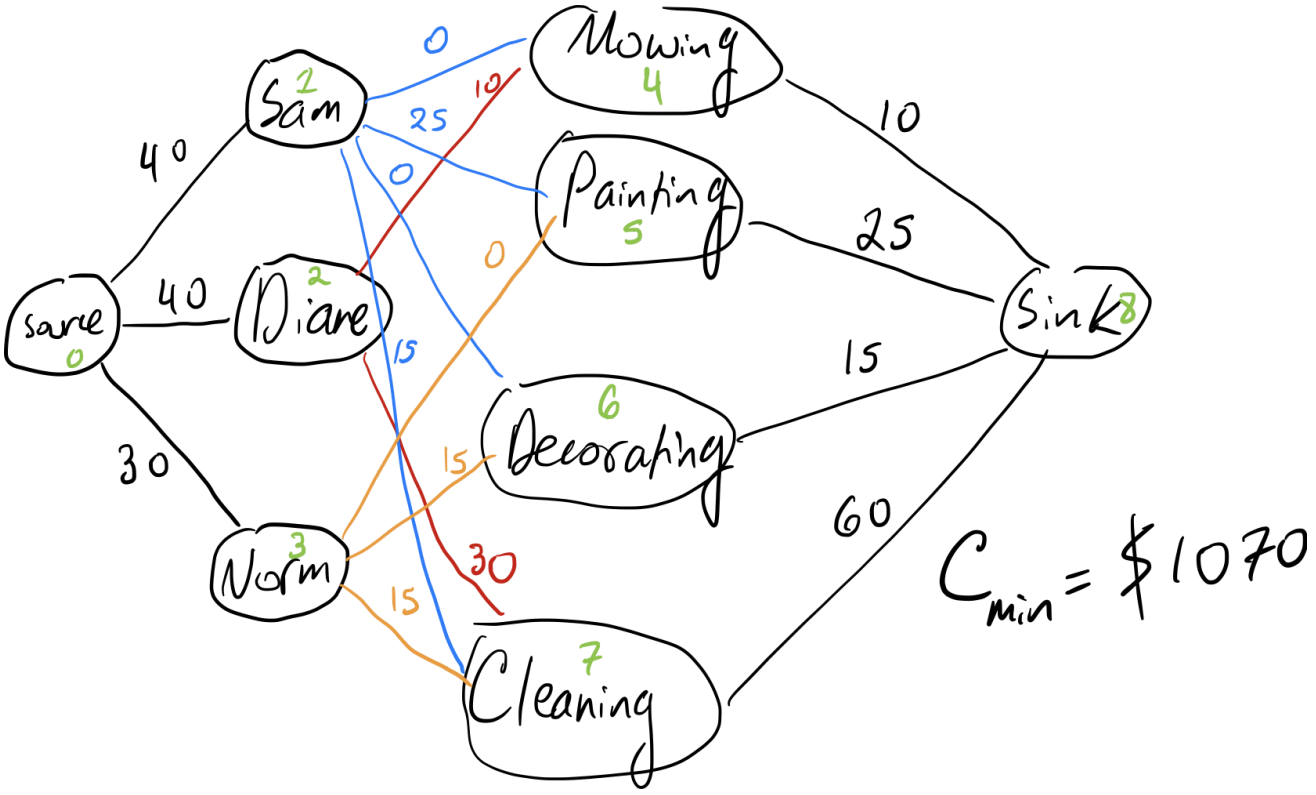
I set up the problem as follows:



In [ ]:
```
a=np.array([[0,1,40,0],
            [0,2,40,0],
            [0,3,40,0],
            [1,4,40,10],
            [1,5,40,15],
            [1,6,40,12],
            [1,7,40,8],
            [2,4,40,8],
            [2,7,40,7],
            [3,5,40,18],
            [3,6,40,10],
            [3,7,40,9],
            [4,8,10,0],
            [5,8,25,0],
            [6,8,15,0],
            [7,8,60,0]
            ])
maxflow(9,a,110)
```

```
     con: array([0., 0., 0., 0., 0., 0., 0., 0.])
     fun: 1070.0
 message: 'Optimization terminated successfully.'
     nit: 35
   slack: array([ 0.,  0., 10., 40., 15., 40., 25., 30., 10., 40., 25., 25.,  0.,
         0.,  0.,  0.])
  status: 0
 success: True
       x: array([40., 40., 30.,  0., 25.,  0., 15., 10., 30.,  0., 15., 15., 10.,
        25., 15., 60.])
```

Meaning that the optimal flows can be written as:



Problem 3.

Run a bunch of code:

In [ ]:
```
def print_tableau(a,indep_names,dep_names,indep_names_dual,dep_names_dual):
#
# Given matrix "a" and lists of variables names "indep_names" and "dep_names",
#   and (for the dual) "indep_names_dual" and "dep_names_dual",
# this function prints the matrix and labels in standard tableau format
# (including adding the -1, the minus signs in the last column, and labeling the lower-right as obj)
#
# First, check the inputs: indep_names and dep_names_dual should be one shorter than the number of columns of A
#                          dep_names and indep_names_dual should be one shorter than the number of rows of A
#
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in A
    ncols = a.shape[1]
    nindep = len(indep_names)
    nindep_dual = len(indep_names_dual)
    ndep = len(dep_names)
    ndep_dual = len(dep_names_dual)
    if nindep != ncols-1:
        print("WARNING: # of indep vbles should be one fewer than # columns of matrix")
    if ndep != nrows-1:
        print("WARNING: # of dep vbles should be one fewer than # rows of matrix")
    if nindep_dual != nrows-1:
        print("WARNING: # of indep dual vbles should be one fewer than # rows of matrix")
    if ndep_dual != ncols-1:
        print("WARNING: # of dep dual vbles should be one fewer than # columns of matrix")
# Now do the printing (uses a variety of formatting techniques in Python)
    print("          ",end="")       # On first line, leave blank space so we can fit in dual labels lower down
    for j in range(ncols-1):                    # Print the independent variables in the first row
        print(indep_names[j].rjust(10),end="")  # rjust(10) makes fields 10 wide and right-justifies;
                                                #    the end command prevents newline)
    print("        -1")                         # Tack on the -1 at the end of the first row
    for i in range(nrows-1):
        print(indep_names_dual[i].rjust(10),end="")
        for j in range(ncols):                  # Print all but the last row of the matrix
                print("%10.3f" % a[i][j],end="") # The syntax prints in a field 10 wide, showing 3 decimal point
        lab = "= -" + dep_names[i]
        print(lab.rjust(10))
    print("        -1",end="")
    for j in range(ncols):
        print("%10.3f" % a[nrows-1][j],end="")  # Print the last row of the matrix, with label "obj" at end
    lab = "= obj"
    print(lab.rjust(10))
```

```
    print("          ",end="")
    for j in range(ncols-1):
        lab = "=" + dep_names_dual[j]
        print(lab.rjust(10),end="")
    print("  =dualobj")
    print(" ")    # Put blank line at bottom
```

In [ ]:
```
def pivot(a,pivrow,pivcol,indep_names,dep_names,indep_names_dual,dep_names_dual) :
#
# Given matrix "a", a row number "pivrow" and column number "pivcol",
#  and lists of variable names "indep_names" and "dep_names", this
#  function does three things:
#    (1) outputs the new version of the matrix after a pivot,
#    (2) updates the lists of variable names post-pivot
#    (3) prints the new matrix, including labels showing the variable names
#
# First, check the inputs: indep_names should be one shorter than the number of columns of A
#                          dep_names should be one shorter than the number of rows of A
#                          you should not be pivoting on the last row or last column
#
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in A
    ncols = a.shape[1]
    nindep = len(indep_names)
    nindep_dual = len(indep_names_dual)
    ndep = len(dep_names)
    ndep_dual = len(dep_names_dual)
    if nindep != ncols-1:
        print("WARNING: # of indep vbles should be one fewer than # columns of matrix")
    if ndep != nrows-1:
        print("WARNING: # of dep vbles should be one fewer than # rows of matrix")
    if nindep_dual != nrows-1:
        print("WARNING: # of indep dual vbles should be one fewer than # rows of matrix")
    if ndep_dual != ncols-1:
        print("WARNING: # of dep dual vbles should be one fewer than # columns of matrix")
    if pivrow > nrows-1 or pivcol > ncols-1:
        print("WARNING: should not pivot on last row or column")
    newa = a.copy()       # make a copy of A, to be filled in below with result of pivot
    p = a[pivrow-1][pivcol-1]   # identify pivot element
    newa[pivrow-1][pivcol-1] = 1/p   # set new value of pivot element
    # Set entries in p's row
    for j in range(ncols):
        if j != pivcol-1:
            newa[pivrow-1][j]=a[pivrow-1][j]/p;
    # Set entries in p's column
    for i in range(nrows):
        if i != pivrow-1:
            newa[i][pivcol-1]=-a[i][pivcol-1]/p;
    # Set all other entries
    for i in range(nrows):
        for j in range(ncols):
            if i != pivrow-1 and j != pivcol-1:
                r = a[i][pivcol-1]
                q = a[pivrow-1][j]
                s = a[i][j]
                newa[i][j]=(p*s-q*r)/p
    # Now transfer the new tableau into a
    for i in range(nrows) :
        for j in range(ncols) :
            a[i][j] = newa[i][j]
    # Now swap the variable names
    temp = indep_names[pivcol-1]
    indep_names[pivcol-1]=dep_names[pivrow-1]
    dep_names[pivrow-1]=temp
    temp = indep_names_dual[pivrow-1]
    indep_names_dual[pivrow-1]=dep_names_dual[pivcol-1]
    dep_names_dual[pivcol-1]=temp
    print_tableau(newa,indep_names,dep_names,indep_names_dual,dep_names_dual) # Print the matrix with updated la
    return 0;
```

In [ ]:
```
def column_delete(a,col_to_remove,indep_names,dep_names,indep_names_dual,dep_names_dual) :
    import numpy as np
    anew = np.delete(a,col_to_remove-1,axis=1)
    del indep_names[col_to_remove-1]
    del dep_names_dual[col_to_remove-1]
    print_tableau(anew,indep_names,dep_names,indep_names_dual,dep_names_dual)
    return anew
```

In [ ]:
```
def row_delete(a,row_to_remove,indep_names,dep_names,indep_names_dual,dep_names_dual) :
    import numpy as np
    anew = np.delete(a,row_to_remove-1,axis=0)
```

```
    del dep_names[row_to_remove-1]
    del indep_names_dual[row_to_remove-1]
    print_tableau(anew,indep_names,dep_names,indep_names_dual,dep_names_dual)
    return anew
```

In [ ]:
```
def target(a) :
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in "a"
    ncols = a.shape[1]
    import numpy as np
    v = np.empty(ncols-1)
    for i in range(ncols-1):
        v[i]=a[nrows-1,i]
    biggest_c = np.max(v)
    where_is_biggest_c = np.argmax(v)+1
    if biggest_c > 0 :
        return where_is_biggest_c
    else :
        return -1
```

In [ ]:
```
def select(a,pivcolnum) :
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in A
    ncols = a.shape[1]
# First task: work down the column and record the b/a ratios in a vector v
#     except record -1 if a is negative or zero
    import numpy as np
    v = np.zeros(nrows-1)
    for i in range(nrows-1):
        if a[i,pivcolnum-1]>0 :
            v[i] = a[i,ncols-1]/a[i,pivcolnum-1]
        else :
            v[i] = -1
# Second task: if max b/a > -1, find min b/a by hand (ignoring zero entries in v)
    if np.max(v) > -1 :
        min_so_far = np.max(v)+1  # Initialize min to be for-sure bigger than the min
        for i in range(nrows-1):
            if v[i] > -1 and v[i] < min_so_far :
                min_so_far = v[i]
                where_is_min = i+1   # Add 1 to use human numbering
        return where_is_min      # Once we've scanned v for min, we can return result

    else :          # Otherwise, we find the m
        return -1
```

In [ ]:
```
def simplexbf(a,indep_names,dep_names,dual_indep_names,dual_dep_names):
# Run the simplexbf algorithm
# Inputs: np.array "a" (assumed to be basic feasible)
#         lists of variable names indep_names and dep_names (pivot will catch if they're wrong size)
# Output: -1 if we stop because problem is unbounded, 0 if we continue to a solution
#         -9 if we take too many steps
    nrows = a.shape[0]    # use shape to find # of rows and cols in A
    ncols = a.shape[1]
    print("Starting SimplexBF (will do nothing if solution can already be determined)")
    pivcol = target(a)
    nsteps = 0
    while pivcol > -1 and nsteps < 50:  # Repeat until either solution found or 50 pivots completed
        pivrow = select(a,pivcol)
        if pivrow == -1 :
            return -1    # If select reports -1, problem is unbounded, so exit this function
        else :
            pivot(a,pivrow,pivcol,indep_names,dep_names,dual_indep_names,dual_dep_names)
            nsteps=nsteps+1
            pivcol = target(a)
    if nsteps >= 50:
        return -9    # we took too many pivots
    else:
        return 0
```

In [ ]:
```
def targetnbf(a):
    nrows = a.shape[0]
    ncols = a.shape[1]
    import numpy as np
    checkrow = nrows-2
    while a[checkrow,ncols-1] >= -0.00000001 :
        if checkrow == 0 :   # if still in the "while" and at the top,
            return -1        # all the b's were >= 0, so return -1
        else :
            checkrow = checkrow-1
    return checkrow+1    # if we exit the "while", we found a negative
#                            b, so return current row # (in human numbering)
```

In [ ]:
```
def selectnbf(a,targetrow) :
#  Given inputs "a" (tableau as an np.array, numbers only, no labels)
#       and "targetrow" (a row that has a negative b; start-at-1 numbering assumed),
#  computes a pivot that could be chosen by SimplexNBF and
#  outputs "pivrow" and "pivcol", the row and column (start-at-1 numbering) of that pivot
#  If the targeted row has no negative aij, returns -2 for both pivrow and pivcol
    nrows = a.shape[0]
    ncols = a.shape[1]
    import numpy as np
    targetrow = targetrow-1   # convert to start-at-0
    pivcol = ncols-2     # column index of last aij
    while a[targetrow,pivcol] >= 0 :
        if pivcol == 0 :   # if pivcol makes it to zero, all aij
            return [-2,-2]   # in this row were >= 0, so problem infeasible
        else :
            pivcol = pivcol-1
    minsofar = a[targetrow,ncols-1]/a[targetrow,pivcol] # we found a negative aij
    pivrow = targetrow
    for i in range(targetrow+1,nrows-2):   # now check below it for a smaller bi/aij with aij>0
        if a[i,pivcol]>0 and a[i,ncols-1]/a[i,pivcol] < minsofar :
            minsofar = a[i,ncols-1]/a[i,pivcol]
            pivrow = i
    return [pivrow+1,pivcol+1]   # Return result (shifted to start-at-1 numbering)
```

In [ ]:
```
def simplexnbf(a,indep_names,dep_names,dual_indep_names,dual_dep_names):
# Run the simplexnbf algorithm
# Inputs: np.array "a"
#         lists of variable names indep_names and dep_names (pivot will catch if they're wrong size)
# Output: -2 if we stop because problem is infeasible, 0 if we stop at a basic feasible tableau
#.        -9 if we take too many pivots
#    (Also, the tableau "a" and variable-lists are updated with each pivot)
    nrows = a.shape[0]
    ncols = a.shape[1]
    print("Starting SimplexNBF (will do nothing if already basic feasible)")
    nsteps = 0
    targetrow = targetnbf(a)
    while targetrow > -1 and nsteps < 50:  # Repeat until either basic feasible tableau produced or 50 pivots co
        [pivrow,pivcol] = selectnbf(a,targetrow)
        if pivrow == -2 :
            return -2    # If selectnbf reports -2, problem is infeasible, so exit this function
        else :
            pivot(a,pivrow,pivcol,indep_names,dep_names,dual_indep_names,dual_dep_names)
            nsteps=nsteps+1
            targetrow = targetnbf(a)
    if nsteps >= 50:
        return -9    # took too many pivots
    else:
        return 0
```

In [ ]:
```
def simplex(a,indep_names,dep_names,dual_indep_names,dual_dep_names) :
# Runs the simplex algorithm (doing NBF if needed, then BF)
# Inputs: np.array "a"
#         lists of variable names indep_names and dep_names (pivot will catch if they're wrong size)
# Output: -2 if problem is infeasible
#         -1 if problem is unbounded
#         0 if problem has a solution
#    (Also, the tableau "a" and variable-lists are updated with each pivot)
    nrows = a.shape[0]
    ncols = a.shape[1]
    print("Initial tableau")
    print_tableau(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
    code = simplexnbf(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
    if code == -2 :
        print("Problem is infeasible")
        return -2
    elif code == -9 :
        print("SimplexNBF took too many pivots")
    else :
        code = simplexbf(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
        if code == -1 :
            print("Problem is unbounded")
            return -1
        elif code == -9 :
            print("SimplexBF took too many pivots")
        else :
            print("Problem has solution, final tableau is shown above")
            return 0
```

In [ ]:
```
def simplexeq(a,k,indep_names,dep_names,dual_indep_names,dual_dep_names) :
# Specialized function to do the "pre-simplex" step to handle tableaus where the
```

```
#   first k rows correspond to equality constraints.
    nrows = a.shape[0]
    ncols = a.shape[1]
    print_tableau(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
    for i in range(k) :
        j=0
        pivcol=-1
        for j in range(ncols-1) :
            if abs(a[i,j]) > 0.000001 :
                pivrow=i+1
                pivcol=j+1
                break
        if pivcol == -1 :
            return -3
        else :
            pivot(a,pivrow,pivcol,indep_names,dep_names,dual_indep_names,dual_dep_names)
            a=column_delete(a,pivcol,indep_names,dep_names,dual_indep_names,dual_dep_names)
            ncols=ncols-1
    code = simplex(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
    return code
```

In [ ]:

```
# In this one, the first constraint is an equality constraint, so we use simplexeq
#  (with second argument = 1 to indicate there is one equality constraint)
a = np.array([[0,-1,-1,-1,-1],
              [-1,0,-2,1,0],
              [-1,2,0,-1,0],
              [-1,-1,1,0,0],
              [-1,0,0,0,0]])
print(a)
a = a.astype(float)
indep_names = ["ucirc","q1","q2","q3"]
dep_names = ["0","t1","t2","t3"]
dual_indep_names = ["vcirc","p1","p2","p3"]
dual_dep_names = ["0","s1","s2","s3"]
print_tableau(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
pivot(a,1,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
a = column_delete(a,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
pivot(a,2,1,indep_names,dep_names,dual_indep_names,dual_dep_names)
a = row_delete(a,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
simplex(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
```

```
[[ 0 -1 -1 -1 -1]
 [-1  0 -2  1  0]
 [-1  2  0 -1  0]
 [-1 -1  1  0  0]
 [-1  0  0  0  0]]
              ucirc        q1        q2        q3        -1
    vcirc     0.000    -1.000    -1.000    -1.000    -1.000      = -0
       p1    -1.000     0.000    -2.000     1.000     0.000      = -t1
       p2    -1.000     2.000     0.000    -1.000     0.000      = -t2
       p3    -1.000    -1.000     1.000     0.000     0.000      = -t3
       -1    -1.000     0.000     0.000     0.000     0.000      = obj
                 =0       =s1       =s2       =s3  =dualobj

             ucirc         0        q2        q3        -1
       s1    -0.000    -1.000     1.000     1.000     1.000      = -q1
       p1    -1.000     0.000    -2.000     1.000    -0.000      = -t1
       p2    -1.000     2.000    -2.000    -3.000    -2.000      = -t2
       p3    -1.000    -1.000     2.000     1.000     1.000      = -t3
       -1    -1.000     0.000    -0.000    -0.000    -0.000      = obj
                 =0    =vcirc       =s2       =s3  =dualobj

             ucirc        q2        q3        -1
       s1    -0.000     1.000     1.000     1.000      = -q1
       p1    -1.000    -2.000     1.000    -0.000      = -t1
       p2    -1.000    -2.000    -3.000    -2.000      = -t2
       p3    -1.000     2.000     1.000     1.000      = -t3
       -1    -1.000    -0.000    -0.000    -0.000      = obj
                 =0       =s2       =s3  =dualobj

                t1        q2        q3        -1
       s1    -0.000     1.000     1.000     1.000      = -q1
        0    -1.000     2.000    -1.000     0.000  = -ucirc
       p2    -1.000    -0.000    -4.000    -2.000      = -t2
       p3    -1.000     4.000    -0.000     1.000      = -t3
       -1    -1.000     2.000    -1.000    -0.000      = obj
                =p1       =s2       =s3  =dualobj

                t1        q2        q3        -1
       s1    -0.000     1.000     1.000     1.000      = -q1
       p2    -1.000    -0.000    -4.000    -2.000      = -t2
       p3    -1.000     4.000    -0.000     1.000      = -t3
       -1    -1.000     2.000    -1.000    -0.000      = obj
                =p1       =s2       =s3  =dualobj
```

```
Initial tableau
                 t1        q2        q3        -1
        s1    -0.000     1.000     1.000     1.000     = -q1
        p2    -1.000    -0.000    -4.000    -2.000     = -t2
        p3    -1.000     4.000    -0.000     1.000     = -t3
        -1    -1.000     2.000    -1.000    -0.000     = obj
                 =p1       =s2       =s3  =dualobj

Starting SimplexNBF (will do nothing if already basic feasible)
                 t1        q2        t2        -1
        s1    -0.250     1.000     0.250     0.500     = -q1
        s3     0.250     0.000    -0.250     0.500     = -q3
        p3    -1.000     4.000    -0.000     1.000     = -t3
        -1    -0.750     2.000    -0.250     0.500     = obj
                 =p1       =s2       =p2  =dualobj

Starting SimplexBF (will do nothing if solution can already be determined)
                 t1        t3        t2        -1
        s1     0.000    -0.250     0.250     0.250     = -q1
        s3     0.250    -0.000    -0.250     0.500     = -q3
        s2    -0.250     0.250    -0.000     0.250     = -q2
        -1    -0.250    -0.500    -0.250     0.000     = obj
                 =p1       =p3       =p2  =dualobj

Problem has solution, final tableau is shown above
```

Out[ ]: 0

The output above shows that optimal play consists of each players playing scissors 50% of the time and the other two 25% of the time. The expected average payoff is zero.

Problem 4.

a.

In [ ]:
```
# In this one, the first constraint is an equality constraint, so we use simplexeq
#  (with second argument = 1 to indicate there is one equality constraint)
a = np.array([[0,-1,-1,-1,-1,-1],
              [-1,-1,1,-1,2,0],
              [-1,-1,-1,1,1,0],
              [-1,0,1,1,-1,0],
              [-1,0,0,0,0,0]])
print(a)
a = a.astype(float)
indep_names = ["ucirc","q1","q2","q3","q4"]
dep_names = ["0","t1","t2","t3"]
dual_indep_names = ["vcirc","p1","p2","p3"]
dual_dep_names = ["0","s1","s2","s3","s4"]
print_tableau(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
pivot(a,1,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
a = column_delete(a,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
pivot(a,2,1,indep_names,dep_names,dual_indep_names,dual_dep_names)
a = row_delete(a,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
simplex(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
```

```
[[ 0 -1 -1 -1 -1 -1]
 [-1 -1  1 -1  2  0]
 [-1 -1 -1  1  1  0]
 [-1  0  1  1 -1  0]
 [-1  0  0  0  0  0]]
                ucirc        q1        q2        q3        q4        -1
     vcirc      0.000    -1.000    -1.000    -1.000    -1.000    -1.000      = -0
        p1     -1.000    -1.000     1.000    -1.000     2.000     0.000     = -t1
        p2     -1.000    -1.000    -1.000     1.000     1.000     0.000     = -t2
        p3     -1.000     0.000     1.000     1.000    -1.000     0.000     = -t3
        -1     -1.000     0.000     0.000     0.000     0.000     0.000     = obj
                  =0       =s1       =s2       =s3       =s4  =dualobj

                ucirc         0        q2        q3        q4        -1
        s1     -0.000    -1.000     1.000     1.000     1.000     1.000     = -q1
        p1     -1.000    -1.000     2.000    -0.000     3.000     1.000     = -t1
        p2     -1.000    -1.000    -0.000     2.000     2.000     1.000     = -t2
        p3     -1.000     0.000     1.000     1.000    -1.000    -0.000     = -t3
        -1     -1.000     0.000    -0.000    -0.000    -0.000    -0.000     = obj
                  =0    =vcirc       =s2       =s3       =s4  =dualobj

                ucirc        q2        q3        q4        -1
        s1     -0.000     1.000     1.000     1.000     1.000     = -q1
        p1     -1.000     2.000    -0.000     3.000     1.000     = -t1
        p2     -1.000    -0.000     2.000     2.000     1.000     = -t2
        p3     -1.000     1.000     1.000    -1.000    -0.000     = -t3
        -1     -1.000    -0.000    -0.000    -0.000    -0.000     = obj
```

```
                =0       =s2       =s3       =s4  =dualobj

                t1        q2        q3        q4        -1
        s1   -0.000     1.000     1.000     1.000     1.000     = -q1
         0   -1.000    -2.000     0.000    -3.000    -1.000  = -ucirc
        p2   -1.000    -2.000     2.000    -1.000    -0.000     = -t2
        p3   -1.000    -1.000     1.000    -4.000    -1.000     = -t3
        -1   -1.000    -2.000    -0.000    -3.000    -1.000     = obj
                =p1       =s2       =s3       =s4  =dualobj

                t1        q2        q3        q4        -1
        s1   -0.000     1.000     1.000     1.000     1.000     = -q1
        p2   -1.000    -2.000     2.000    -1.000    -0.000     = -t2
        p3   -1.000    -1.000     1.000    -4.000    -1.000     = -t3
        -1   -1.000    -2.000    -0.000    -3.000    -1.000     = obj
                =p1       =s2       =s3       =s4  =dualobj

Initial tableau
                t1        q2        q3        q4        -1
        s1   -0.000     1.000     1.000     1.000     1.000     = -q1
        p2   -1.000    -2.000     2.000    -1.000    -0.000     = -t2
        p3   -1.000    -1.000     1.000    -4.000    -1.000     = -t3
        -1   -1.000    -2.000    -0.000    -3.000    -1.000     = obj
                =p1       =s2       =s3       =s4  =dualobj

Starting SimplexNBF (will do nothing if already basic feasible)
                t1        q2        q3        t3        -1
        s1   -0.250     0.750     1.250     0.250     0.750     = -q1
        p2   -0.750    -1.750     1.750    -0.250     0.250     = -t2
        s4    0.250     0.250    -0.250    -0.250     0.250     = -q4
        -1   -0.250    -1.250    -0.750    -0.750    -0.250     = obj
                =p1       =s2       =s3       =p3  =dualobj

Starting SimplexBF (will do nothing if solution can already be determined)
Problem has solution, final tableau is shown above
```

Out[ ]: 0

The output above indicates that the row player should play strategy one with probability .25, and strategy three with probability .75. Player R never plays strategy 2.

The column player plays strategies 1 and 4 with probabilties .75 and .25 respectively. Player C never plays strategy two or four.

b.

Note that the column player recieves a higher payoff playing strategy one in every state than strategies two and three. These two strategies are dominated. However, strategy four is not dominated by strategy one as it pays more in when the opponent plays three.

There are no dominant strategies for the row player.

Reducing the tableau to the following and solving gives:

c.

In [ ]:
```
# In this one, the first constraint is an equality constraint, so we use simplexeq
#  (with second argument = 1 to indicate there is one equality constraint)
a = np.array([[0,-1,-1,-1],
              [-1,-1,2,0],
              [-1,-1,1,0],
              [-1,0,-1,0],
              [-1,0,0,0]])
print(a)
a = a.astype(float)
indep_names = ["ucirc","q1","q4"]
dep_names = ["0","t1","t2","t3"]
dual_indep_names = ["vcirc","p1","p2","p3"]
dual_dep_names = ["0","s1","s4"]
print_tableau(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
pivot(a,1,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
a = column_delete(a,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
pivot(a,2,1,indep_names,dep_names,dual_indep_names,dual_dep_names)
a = row_delete(a,2,indep_names,dep_names,dual_indep_names,dual_dep_names)
simplex(a,indep_names,dep_names,dual_indep_names,dual_dep_names)
```

```
[[ 0 -1 -1 -1]
 [-1 -1  2  0]
 [-1 -1  1  0]
 [-1  0 -1  0]
 [-1  0  0  0]]
              ucirc        q1        q4        -1
     vcirc    0.000    -1.000    -1.000    -1.000      = -0
        p1    -1.000    -1.000     2.000     0.000     = -t1
```

```
        p2    -1.000    -1.000     1.000     0.000     = -t2
        p3    -1.000     0.000    -1.000     0.000     = -t3
        -1    -1.000     0.000     0.000     0.000     = obj
                 =0       =s1       =s4  =dualobj

               ucirc         0        q4        -1
        s1    -0.000    -1.000     1.000     1.000     = -q1
        p1    -1.000    -1.000     3.000     1.000     = -t1
        p2    -1.000    -1.000     2.000     1.000     = -t2
        p3    -1.000     0.000    -1.000    -0.000     = -t3
        -1    -1.000     0.000    -0.000    -0.000     = obj
                 =0    =vcirc       =s4  =dualobj

               ucirc        q4        -1
        s1    -0.000     1.000     1.000     = -q1
        p1    -1.000     3.000     1.000     = -t1
        p2    -1.000     2.000     1.000     = -t2
        p3    -1.000    -1.000    -0.000     = -t3
        -1    -1.000    -0.000    -0.000     = obj
                 =0       =s4  =dualobj

                  t1        q4        -1
        s1    -0.000     1.000     1.000     = -q1
         0    -1.000    -3.000    -1.000  = -ucirc
        p2    -1.000    -1.000    -0.000     = -t2
        p3    -1.000    -4.000    -1.000     = -t3
        -1    -1.000    -3.000    -1.000     = obj
                  =p1       =s4  =dualobj

                  t1        q4        -1
        s1    -0.000     1.000     1.000     = -q1
        p2    -1.000    -1.000    -0.000     = -t2
        p3    -1.000    -4.000    -1.000     = -t3
        -1    -1.000    -3.000    -1.000     = obj
                  =p1       =s4  =dualobj

Initial tableau
                  t1        q4        -1
        s1    -0.000     1.000     1.000     = -q1
        p2    -1.000    -1.000    -0.000     = -t2
        p3    -1.000    -4.000    -1.000     = -t3
        -1    -1.000    -3.000    -1.000     = obj
                  =p1       =s4  =dualobj

Starting SimplexNBF (will do nothing if already basic feasible)
                  t1        t3        -1
        s1    -0.250     0.250     0.750     = -q1
        p2    -0.750    -0.250     0.250     = -t2
        s4     0.250    -0.250     0.250     = -q4
        -1    -0.250    -0.750    -0.250     = obj
                  =p1       =p3  =dualobj

Starting SimplexBF (will do nothing if solution can already be determined)
Problem has solution, final tableau is shown above
```

Out[ ]: 0

Reading off of the above output shows that the solution is identical. There is this pesky $p_2 = 0$ even though it's not dominated which is interesting.