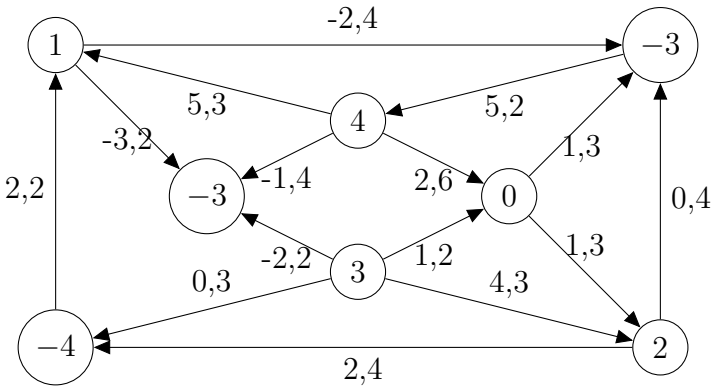
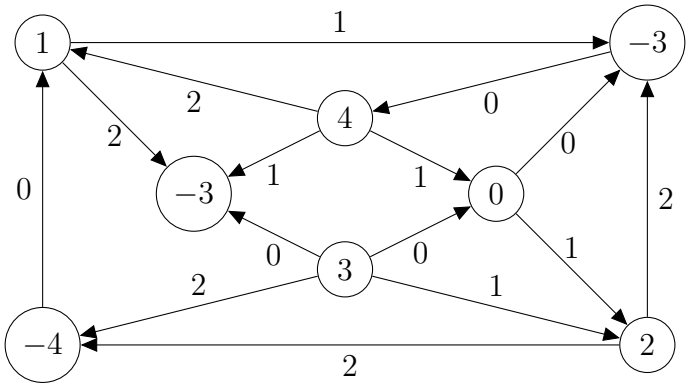
MATH-566 HW 08

Due **Nov 9** before class (regularly). Just bring it before the class and it will be collected there.

Consider the following network M with costs and capacities depicted on edges and boundary in vertices.



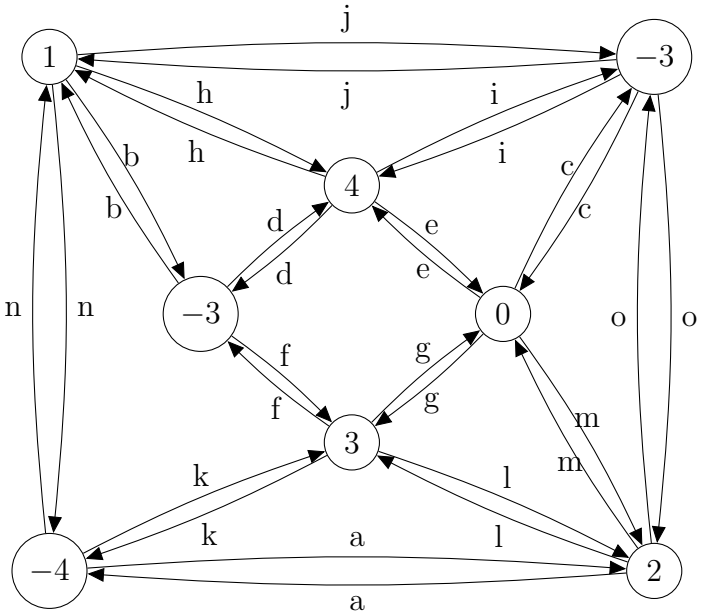
1: (*Try Min Cost Flow algorithm*)
Consider the following b-flow f in M.



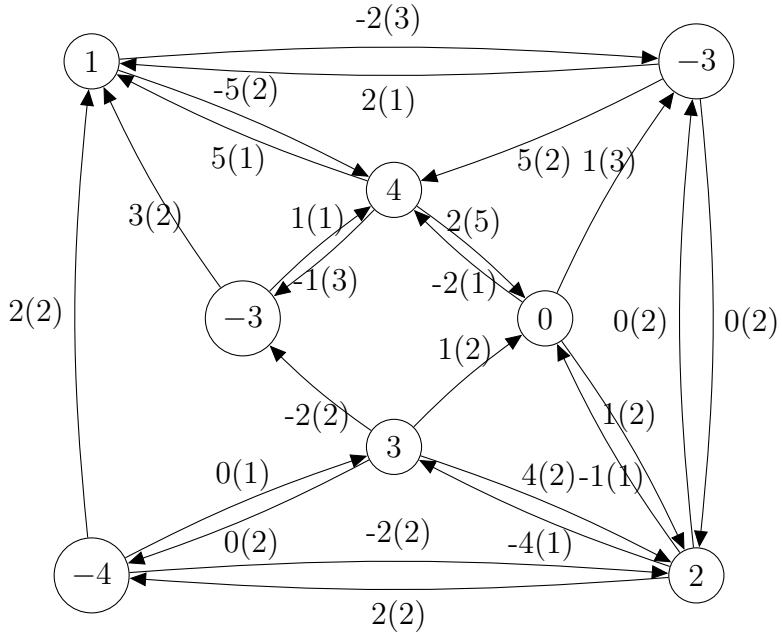
Compute the cost of f.

Start computing the minimum cost b-flow by finding a sequence of augmenting cycles starting from f. (No need to use minimum mean cycles, do two augmentations. No need to solve it to optimality.)

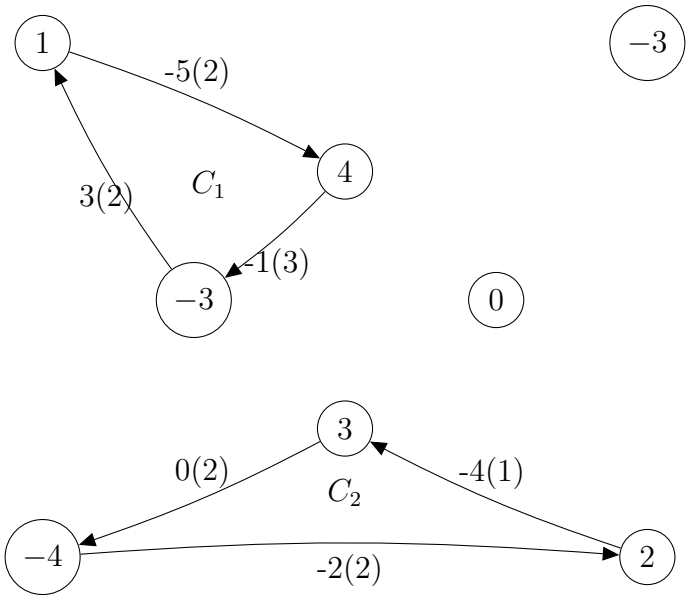
You may use the following template to create residual graphs for finding the cycle.



Solution:
First we create a reduced graph. We include cost first and then capacity capacity second. We are trying a cycle with negative cost (and then push as much as we can on it, which is the smallest capacity of edges of the cycle).

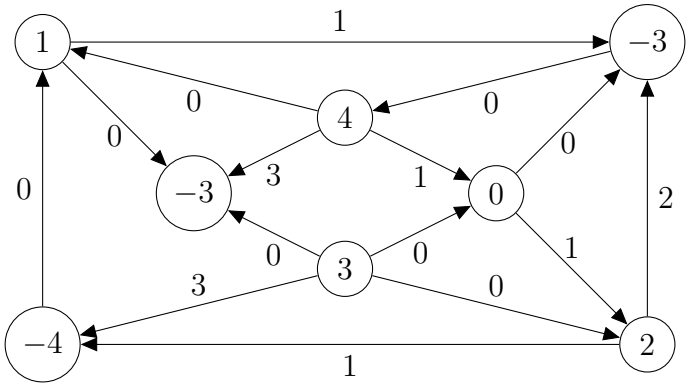


We can find two disjoint negative cycles (we should be doing one by one).



The flow on C_1 can be augmented by 2 and flow on C_2 can be augmented by 1.

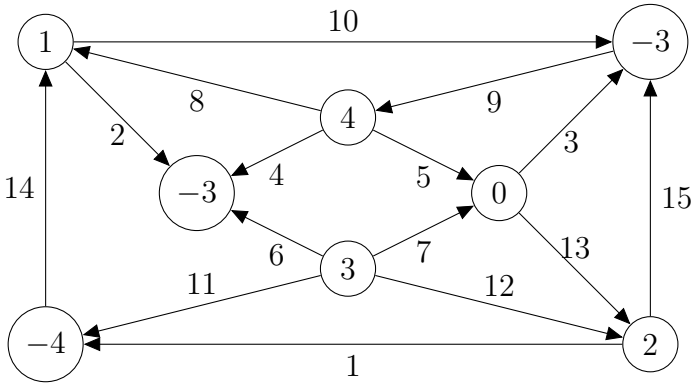
The resulting flow is



Total cost of the flow is zero.

2: (*Min Cost Flow as Linear Program*)
Solve minimum cost b-flow for M using linear programming. That is, formulate the problem using linear programming and solve it using Sage or APMonitor. Then draw the resulting network.

Solution:
We do the following labeling of the edges



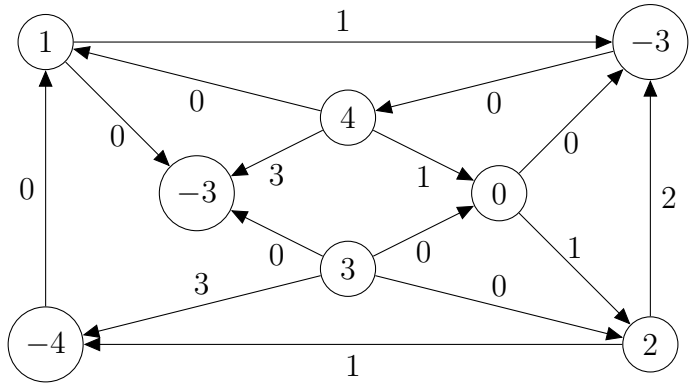
Then we use this Sage code

```
p = MixedIntegerLinearProgram(maximization=False)
e = p.new_variable(integer=False, nonnegative = True)
p.add_constraint( e[14]-e[11]-e[1]      == -4)
p.add_constraint( e[1]+e[15]-e[12]-e[13] == 2)
p.add_constraint( e[6]+e[7]+e[11]+e[12]  == 3)
p.add_constraint( -e[2]-e[4]-e[6]  == -3)
p.add_constraint( e[3]+e[13] - e[5]-e[7]  == 0)
p.add_constraint( e[4]+e[5]+e[8]-e[9]  == 4)
p.add_constraint( e[2]+e[10]-e[14]-e[8]  == 1)
p.add_constraint( e[9]-e[10]-e[3]-e[15]  == -3)
p.set_objective(2*e[1]-3*e[2]+e[3]-e[4]+2*e[5]-2*e[6]+e[7]+5*e[8]+5*e[9]-2*e[10]
                        +0*e[11]+4*e[12]+e[13]+2*e[14]+0*e[15])
p.show()
opt = p.solve()
print opt
p.get_values(e)
```

Solution from Sage is

$$\{1:1, 2:0, 3:0, 4:3, 5:1, 6:0, 7:0, 8:0, 9:0, 10:1, 11:3, 12:0, 13:1, 14:0, 15:2\}$$

which corresponds to



The cost is 0.

3: (*Max Flow* $\subset$ *Min Cost Flow*)
Show that the Maximum Flow Problem can be regarded as a special case of the Minimum Cost Flow problem. That is, for an instance of Maximum Flow Problem find a reformulation to Minimum Cost Flow problem whose solution can be interpreted as a solution to Maximum Flow Problem. That is, find *simple* algorithm that is solving Maximum Flow Problem and using Minimum Cost Flow as a black box subroutine once.

Solution:
Add an extra edge from t to s and put capacity infinity on the edge and cost -1. Cost on all other edges 0, boundary function at all vertices also 0. The minimum cost flow will try to push as much as possible through the ts edge, which in turn tries to push as much as possible through the network from s to t and it gives maximum flow.