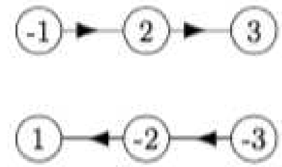
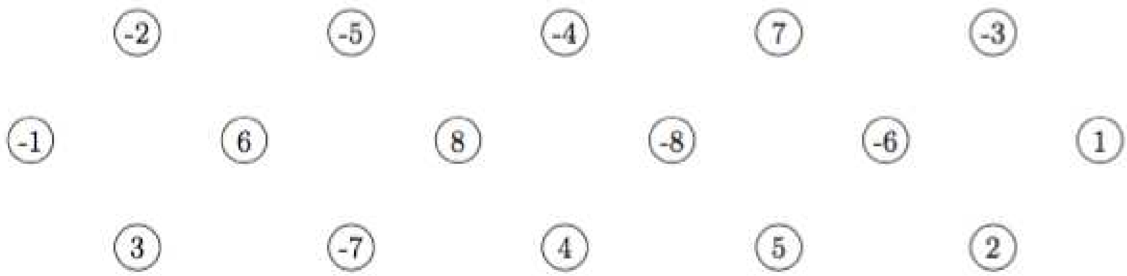
first name **last name** **id#**

1. At right is the implication digraph for boolean formula $f = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. In short form, we write $f = (1\ 2)(-2\ 3)$. Below, on the nodes, draw the implication digraph D for g =(-1 2)(1 3)(1 6)(2 -5)(-3 4)(4 -8)(-4 -8)(-5 -6)(5 8)(6 7)(-7 -8).





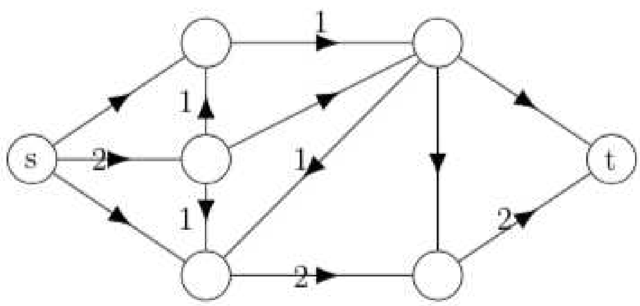
2. **On the digraph above**, circle each strongly connected component of D.
Below, draw the reduced digraph D' in which each scc of D is replaced with a single node.

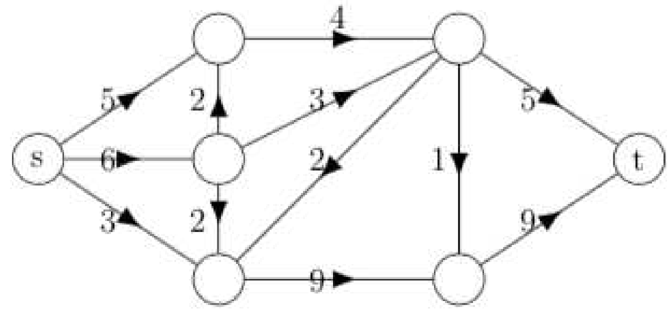
3. For g, either

below left give a satisfying assignment or **below right** explain why g is unsatisfiable.

```
     SATISFYING ASSIGNMENT
variable   x1 x2 x3 x4 x5 x6 x7 x8
0/1 value   _  _  _  _  _  _  _  _
```

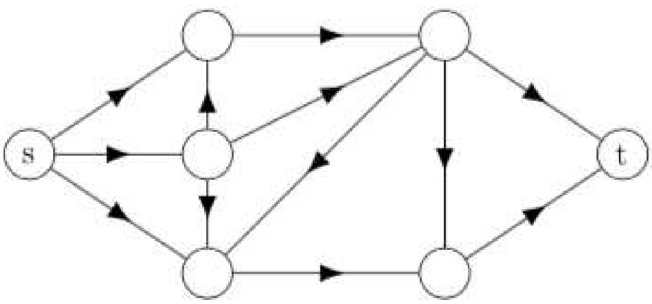
```
g is unsatisfiable because ...
```

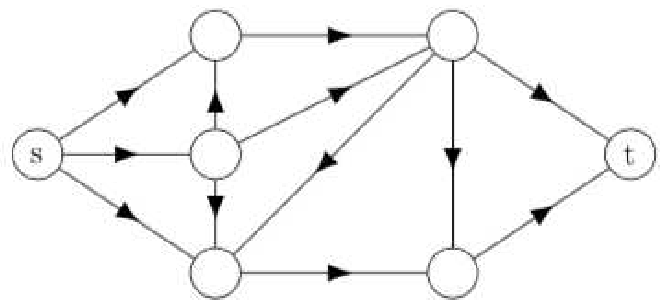




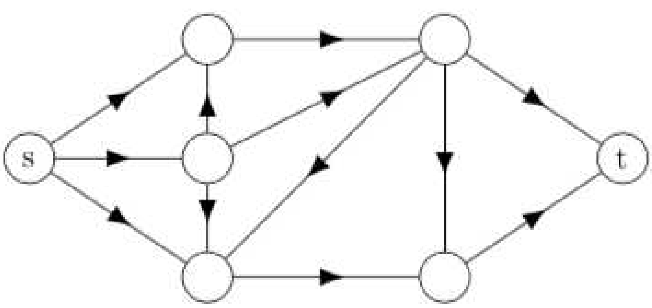
4.

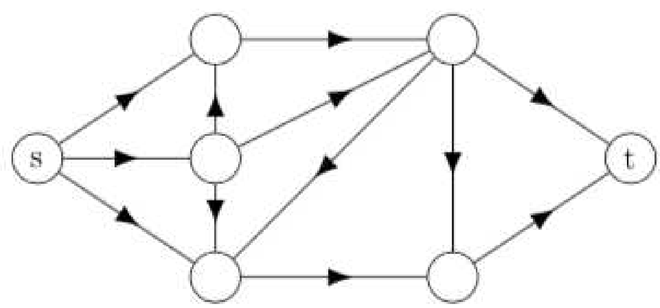
Above **right** is a network: arc labels show capacities. Above **left** is a flow in this network: arc labels show flow volume. On the digraph **below left**, show the residual digraph for the above flow. You might have to add extra arcs. On the digraph **below right**, show a maximum s-t flow for the above network **and** a min s-t cut (by circling the node set that forms one part of the cut).

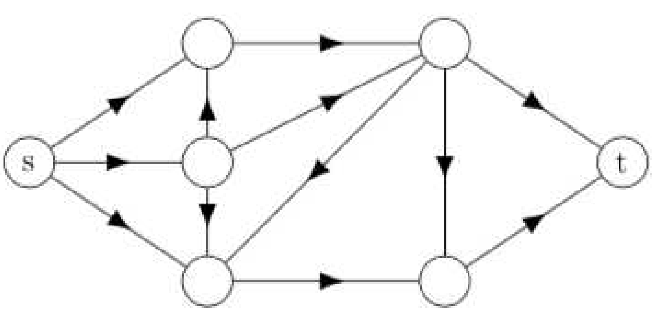


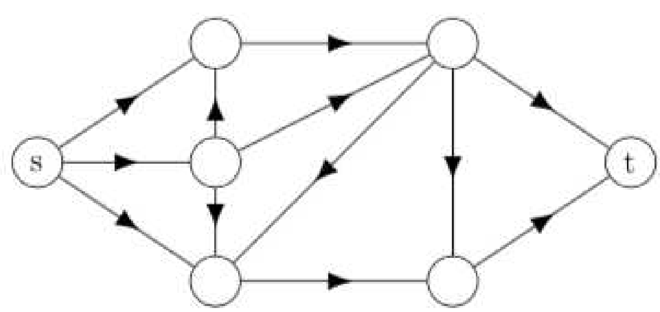


ROUGH WORK BELOW WILL NOT BE MARKED









5. Complete this definition: an algorithm is *polynomial time* if there is a nonnegative constant t such that, for

   every instance with ____________________ n, the runtime is in $O($ ________ $)$.

6. For weights `[5, 8, 6, 7]`, values `[6, 9, 8, 10]`, and capacity 13, give the missing entries of `K[w][j]` (rows 7, 11-13) from this execution of the dynamic-programming-by-weight (DPBW) knapsack algorithm.

...	...	...	...	...	...
4	0	0	0	0	0
5	0	6	6	6	6
6	0	6	6	8	8
7	0	_	_	_	_
8	0	6	9	9	10
9	0	6	9	9	10
10	0	6	9	9	10
11	0	_	_	_	_
12	0	_	_	_	_
13	0	_	_	_	_

7. Consider an instance to the knapsack problem with m items, each with weight and value in the range $[2^{m-1}, 2^m - 1]$, and with capacity $W = .75 \times m \times 2^m$. As a function of m, the number of bits needed to represent

   this instance is in $\Theta($ ________________ $)$. Explain briefly.

8. Let n be the number of bits from the previous question. DPBW knapsack runtime is in **(circle one)**

   $O(n \times 2^{\sqrt{n}})$ $O(n \times 2^{\log n})$ $O(n \times 2^n)$ $O(n^2 \times 2^{\sqrt{n}})$ $O(n^2 \times 2^{\log n})$ $O(n^2 \times 2^n)$. Explain briefly.

9. Hamiltonian cycle problem (HCP) asks whether an input graph has a cycle $(c_1, c_2, \ldots, c_n)$ that includes all n nodes. Travelling sales problem (TSP) asks whether an input weighted graph has a hamiltonian cycle whose weight is at most a given integer k. TSPT is TSP where weights satisfy the triangle inequality: for each graph triangle a, b, c, $w(a,b) \leq w(a,c) + w(b,c)$. Assume HCP is NP-complete. Assume TSPT is in NP. Prove TSPT is in NP-complete. **What is your overall plan in this proof? (circle a, b)** a) Give a polytime answer-preserving transformation from HCP instances to TSPT instances. b) Give a polytime answer-preserving transformation from TSPT instances to HCP instances.

HERE IN AT MOST 60 WORDS describe your transformation

HERE IN AT MOST 60 WORDS explain why your transformation is polytime

HERE IN AT MOST 60 WORDS explain why your transformation is answer-preserving

10.

```
    def solve(f, a):                           # satisfiability formula f, boolean assignment a
 1  if a == ’’ or len(f)==0:  return f, a                      # unsatisfiable or satisfied
 2  t = f.index(min(f,key=len))                               # clause with fewest literals
 3  if len(f[t]) == 0: return f, ’’                          # clause empty so unsatisfiable
 4  if len(f[t]) == 1:                                           # unit clause so no choice
 5   newf, newa = fix_and_propagate(f[t][0], f, a)
 6   return solve(newf, newa)
 7  literal = f[t][0]                                       # 1st literal in shortest clause
 8  newf, newa = fix_and_propagate(literal, f, a)                         # set literal TRUE
 9  if newa = ’’:                                               # this caused contradiction
10    newf, newa = fix_and_propagate(-literal, f, a)                     # set literal FALSE
11    return solve(newf, newa)
12  if len(newf)==0: return f, a            # if return, satisfied: if not, newa ok so far
13  return solve(newf, newa)
```

f is a list of clauses, each clause is a list of literals. Each character of string a is '0' or '1' or '?' (false, true, unassigned). f has n variables, at most $10 \times n$ clauses, each with at most 3 literals. Each `fix_and_propagate()` call takes $\Theta(n)$ time for each literal that it assigns. Each line 2 call takes $\Theta(n)$ time. `len()` takes time proportional to the length. Below, using big O notation, give a recurrence relation for $r(k)$, the worstcase runtime for `solve(f,a)` when `a` has exactly k unassigned literals.

Case A. $r(k) = O(k \times n)$ if `solve(f, a)` makes no recursive call to `solve( )`

because in the worst case `fix_and_propagate()` on line **(fill in blank)** ________________ assigns enough literals to satisfy f and execution returns on line ________________ .

Case B. $r(k) = O($ ________________ $) + r($ ________________ $)$ if `solve(f, a)` calls `solve( )` on line 6 after exactly j literals were assigned on line 5.

Case C. $r(k) = O($ ________________ $) + r(k - j)$ if `solve(f, a)` calls `solve( )` on line 11 after at most n literals were assigned on line 8 and exactly j literals were assigned on line 10.

Case D. $r(k) = O($ ________________ $) + r($ ________________ $)$ if `solve(f, a)` calls `solve( )` on line 13 after exactly j literals were assigned on line 8.

For each of B,C,D, the size of the parameter to r on the right-hand-side of the equation

(circle one) can be bigger than k can be as big as k is less than k .

This algorithm **(circle one)** is is not polytime. Justify **IN AT MOST 30 WORDS**.
Hint: this algorithm might not be correct.