6.01 Final Exam Spring 2017

Name: **Section Number:**

Kerberos (Athena) name:

Section	Design Lab Time
1:	Wednesday 9:30am
2:	Wednesday 2:00pm
3:	Thursday 2:00pm

Please WAIT until we tell you to begin.

During the exam, you may refer to any written or printed paper material.
You may NOT use any electronic devices (including calculators, phones, etc).

If you have questions, please **come to us at the front** to ask them.

Enter all answers in the boxes provided.

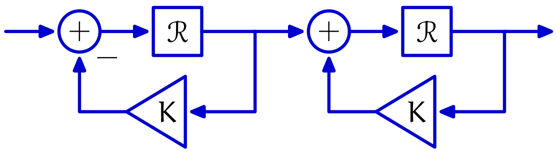
Extra work may be taken into account when assigning partial credit,
but **only work shown on pages with QR codes will be considered.**

Question 1:	20 Points
Question 2:	16 Points
Question 3:	24 Points
Question 4:	30 Points
Question 5:	20 Points
Question 6:	24 Points
Question 7:	24 Points
Question 8:	20 Points
Total:	178 Points

1 Response, S'il Vous Plaît (20 Points)

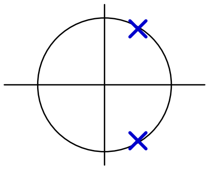
Determine which (if any) of the system functionals and pole diagrams shown on the right corresponds to each system shown on the left. Notice that some of the adders' inputs have minus signs, which indicate that the associated input should be subtracted (as though it were preceded by a gain of -1). In all cases, $K = 0.95$.

A

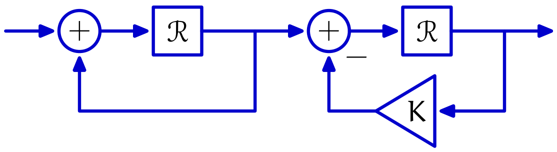


1

$$\frac{\mathcal{R}^2}{1-\mathcal{R}+K\mathcal{R}^2}$$

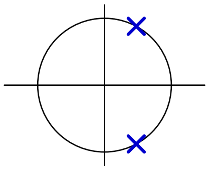


B

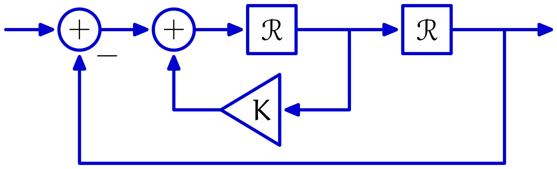


2

$$\frac{\mathcal{R}^2}{1-K\mathcal{R}+\mathcal{R}^2}$$

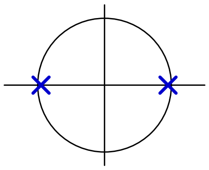


C

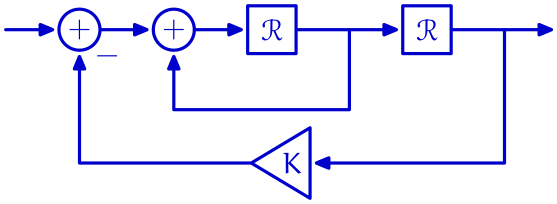


3

$$\frac{\mathcal{R}^2}{1-K^2\mathcal{R}^2}$$

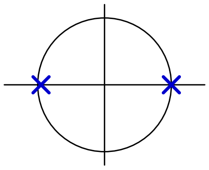


D



4

$$\frac{\mathcal{R}^2}{1-\mathcal{R}+K\mathcal{R}-K\mathcal{R}^2}$$



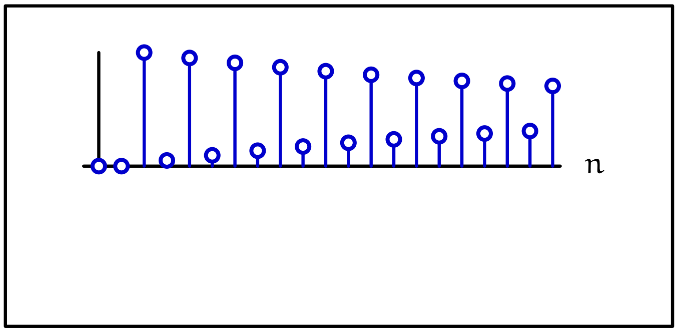
System Functional and poles for **A** (1, 2, 3, 4, or none):

System Functional and poles for **B** (1, 2, 3, 4, or none):

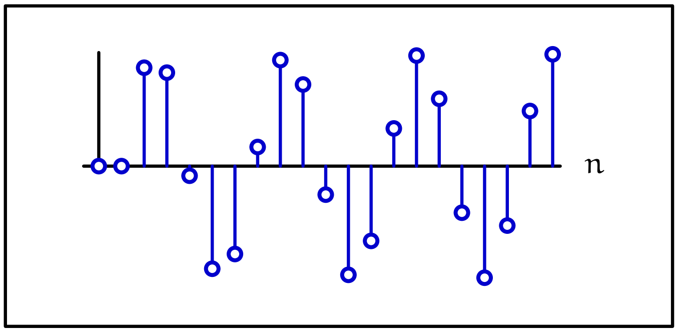
System Functional and poles for **C** (1, 2, 3, 4, or none):

System Functional and poles for **D** (1, 2, 3, 4, or none):

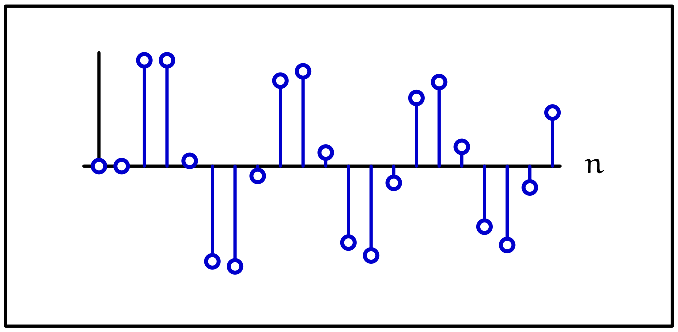
Determine if any of the systems on the previous page (A, B, C, or D) **OR** system functionals and poles (1, 2, 3, or 4) corresponds to each of the following unit-sample responses. Although multiple answers may be correct, you only need a single right answer to receive full credit.



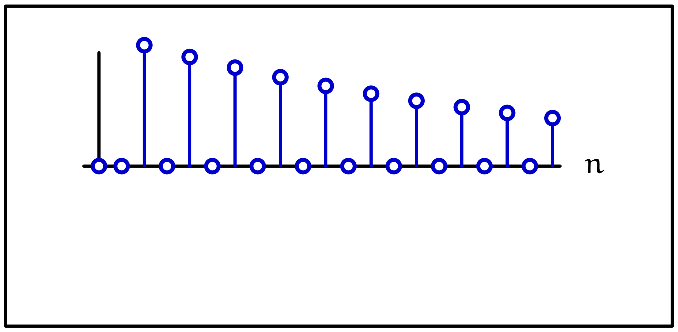
A, B, C, D, 1, 2, 3, 4, or none: □



A, B, C, D, 1, 2, 3, 4, or none: □



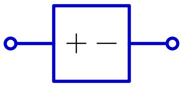
A, B, C, D, 1, 2, 3, 4, or none: □



A, B, C, D, 1, 2, 3, 4, or none: □

2 Boxing Day (16 Points)

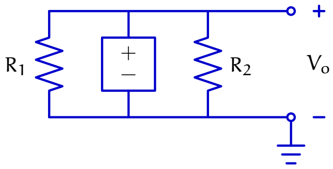
Solve for the voltage V_o in each of the following circuits. Assume that the following:



represents a circuit whose Thévenin equivalent voltage and resistance are V_{TH} and R_{TH}, respectively, where V_{TH} is open-circuit voltage measured from the + terminal to the − terminal.

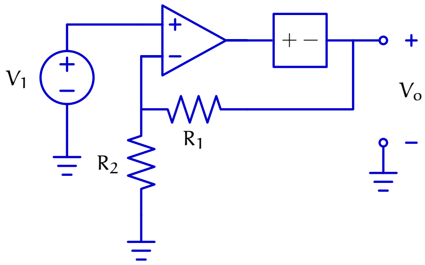
Make the ideal op-amp assumption, and enter your answers in terms of V_{TH}, R_{TH}, and any other voltages or resistances necessary from the circuit. You may use + and || to denote series and parallel combinations of resistances, respectively, rather than solving out completely.

2.1 Circuit 1



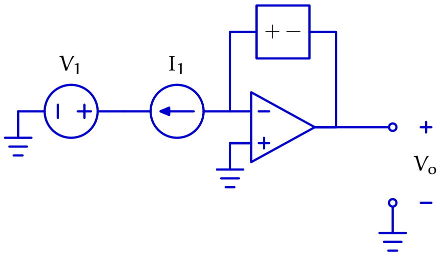
$V_o =$

2.2 Circuit 2



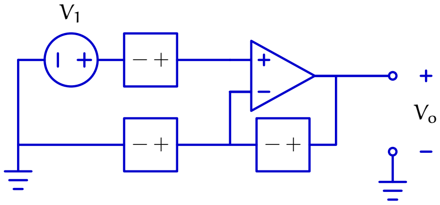
$V_o =$

2.3 Circuit 3



$V_o =$

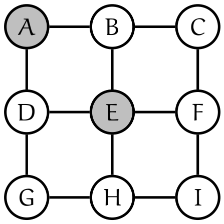
2.4 Circuit 5



$V_o =$

3 I Want To Ride My Bicycle (24 Points)

Here is a map of part of a weird city:



You have to walk along the roads; walking along any single segment takes 10 minutes.

Shaded nodes are hubway stations, where you can pick up or return a bike. You do not start with a bike and you must not have a bike when you arrive at your goal. It costs 2 minutes to pick up a bike and 2 minutes to drop off a bike.

If you are biking, you have to follow the same road segments, but biking takes 5 minutes to get from one node to an adjacent node.

3.1 I Want To Ride My Bike

Let's start by considering the case where bicycles are not available (we can only walk). In that case, what path is found from A to H by a Uniform Cost Search? What is the cost associated with that path? How many states are expanded during the search? Feel free to break ties in any order.

Path Found:

Cost:

States expanded:

Now consider the case where bicycles are available (according to the rules above). In this case, what path is found from A to H by a Uniform Cost Search? What is the cost associated with that path? How many states are expanded during the search? Feel free to break ties in any order.

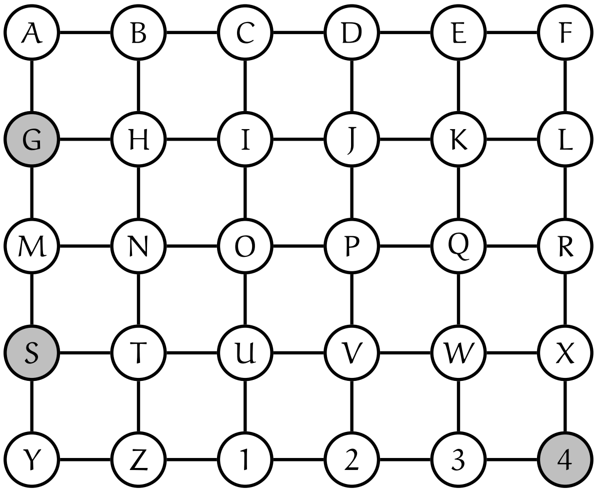
Path Found:

Cost:

States expanded:

3.2 I Want To Ride My Bicycle

Here is a map of a different city:



Assuming only walking is possible, which nodes are *guaranteed* not to be expanded by a uniform cost search from M to 3 in this map where costs are assigned as discussed previously, regardless of how ties are broken?

Nodes not expanded: □

If getting on or off a bicycle took n minutes (instead of 2 minutes), what is the smallest nonnegative integer value of n such that a Uniform Cost Search from M to 3 in the map above is guaranteed to result in a path that does not involve riding a bicycle, regardless of how ties are broken?

$n =$ □

3.3 I Want To Ride It Where I Like

For this question, assume the following:

- We are operating in a city where each link is 1 mile, and it still takes 10 minutes to traverse a link when walking (you're a fast walker!).
- As before, there are some locations where bicycles can be picked up or dropped off.
- As in the original formulation, it takes 2 minutes to get on or off of a bicycle, and it takes 5 minutes to traverse a link on a bicycle.

Consider the following statements:

A. The cost of the shortest walking-only path is an admissible heuristic for the walking-and-biking case.

B. The Euclidean distance between nodes is an admissible heuristic for the walking-only case.

C. The cost of the shortest path in the walking-and-biking case is an admissible heuristic for the walking-only case.

D. The Euclidean distance between nodes, times 10, is an admissible heuristic for the walking-only case.

E. The Euclidean distance between nodes, times 10, is an admissible heuristic for the walking-and-biking case.

F. The Euclidean distance between nodes, times 5, is an admissible heuristic for the walking-and-biking case.

G. The Euclidean distance between nodes, times 5, plus 4, is an admissible heuristic for the walking-and-biking case.

Enter the letters associated with all the true statements in the box below:

True statements:

4 Pets (30 Points)

4.1 Cat Me If You Can

Adam has lost his cat in his apartment. He thinks the cat is either in the kitchen (with probability 0.4) or in the bedroom (with probability 0.6).

On any given day, if the cat is in the kitchen and Adam spends a day looking for it in the kitchen, the probability that he will find the cat that day is 0.25. Similarly, if the cat is in the bedroom and Adam spends a day looking for it there, the probability that he will find the cat that day is 0.2. These probabilities hold for any given day, regardless of what happened on previous days.

Assume that the cat cannot move from one room to another. Adam can only search during the daytime, and he can travel from one room to the other only at night.

1. In which room should Adam look to maximize the probability he finds his cat on the first day of the search?

2. Assuming that Adam flips a fair coin to determine where to look on the first day (he looks in the kitchen with probability 0.5, and in the bedroom with probability 0.5), what is the probability that he finds the cat on the first day?

3. If Adam flips a fair coin to determine where to look on the first day, and he finds the cat on the first day, what is the probability that he looked in the kitchen?

4. Given that Adam looked in the kitchen on the first day but didn't find his cat, what is the probability that the cat is in the kitchen?

5. If, instead, Adam looked in the bedroom on the first day and did not find the cat, what is the probability that he finds the cat on the second day, assuming that he looks in the kitchen on the second day?

4.2 Hamster Dance

Now, Adam has lost his pet hamster instead. Unlike the cat, the hamster can move from room to room in between days. The hamster's movement from room to room is governed by the following set of distributions, where R_n represents the hamster's location on day n.

$$\Pr(R_{n+1} = r \mid R_n = \text{kitchen}) = \begin{cases} 0.6, & r = \text{kitchen} \\ 0.3, & r = \text{bedroom} \\ 0.1, & r = \text{closet} \\ 0 & \text{otherwise} \end{cases}$$

$$\Pr(R_{n+1} = r \mid R_n = \text{bedroom}) = \begin{cases} 0.5, & r = \text{bedroom} \\ 0.5, & r = \text{closet} \\ 0 & \text{otherwise} \end{cases}$$

$$\Pr(R_{n+1} = r \mid R_n = \text{closet}) = \begin{cases} 0.9, & r = \text{bedroom} \\ 0.1, & r = \text{closet} \\ 0 & \text{otherwise} \end{cases}$$

Answer the following questions about this scenario:

1. Assume that we had no idea of the hamster's location and so started with a uniform distribution over rooms on day 0. With what probability would the hamster be in each of these rooms on day 1 (one day later)?

Kitchen: □ Bedroom: □ Closet: □

2. Instead, assume that we knew that, with probability 1, the hamster was in the kitchen on day 0. With what probability would the hamster be in each of these rooms on day 2 (two days later)?

Kitchen: □ Bedroom: □ Closet: □

3. Assume that we knew that, with probability 1, the hamster was in the kitchen on day 0. With what probability would the hamster be in each of these rooms after infinitely-many days (assume that the hamster lives forever)?

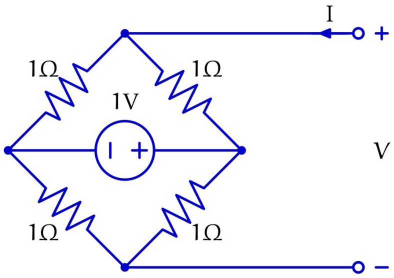
Kitchen:		Bedroom:		Closet:	

4. Assume that we had no idea of the hamster's location and so started with a uniform distribution over rooms on day 0. With what probability would the hamster be in each of these rooms after infinitely-many days (assume that the hamster lives forever)?

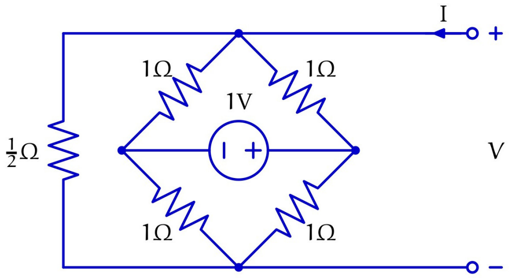
Kitchen:		Bedroom:		Closet:	

5 Building Bridges (20 Points)

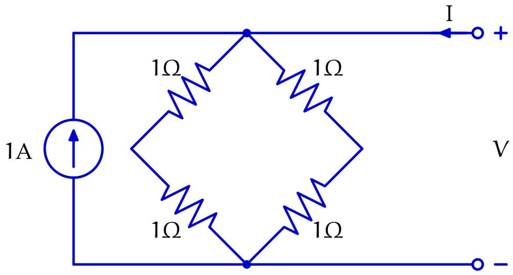
For each of the following circuits, determine which (if any) of plots A-H (on the following page) show the relationship between V and I.



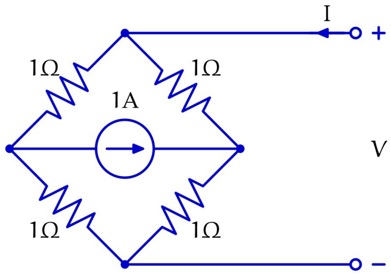
Plot (**A-H** or **None**):



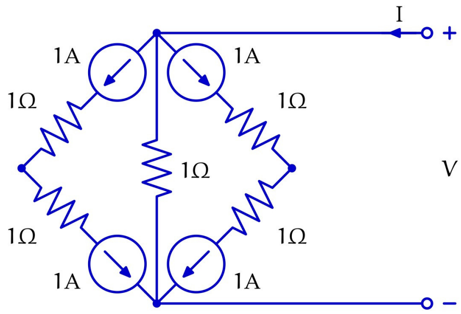
Plot (**A-H** or **None**):



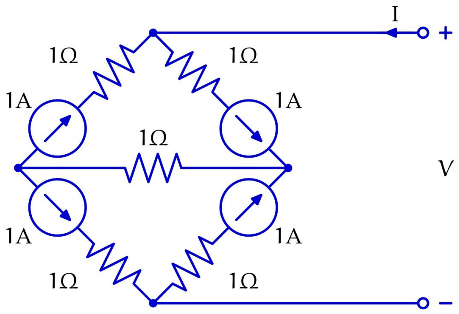
Plot (**A-H** or **None**):



Plot (**A-H** or **None**):

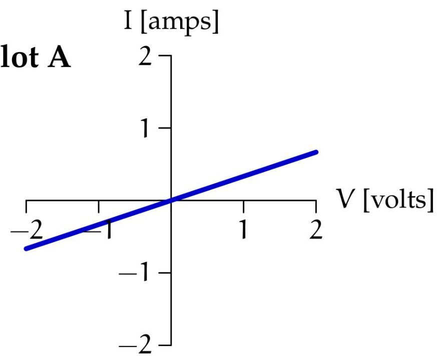


Plot (**A-H** or **None**):

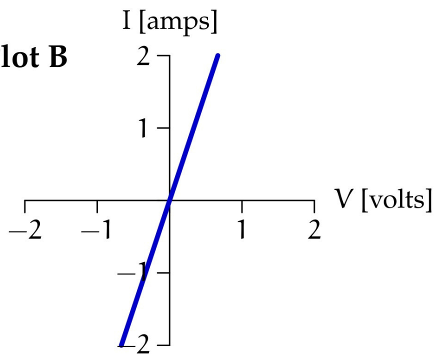


Plot (**A-H** or **None**):

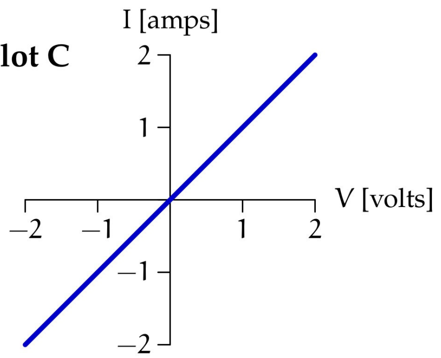
Plot A



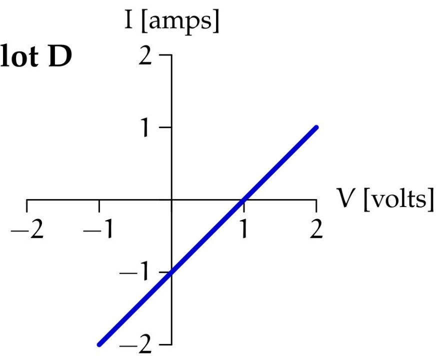
Plot B



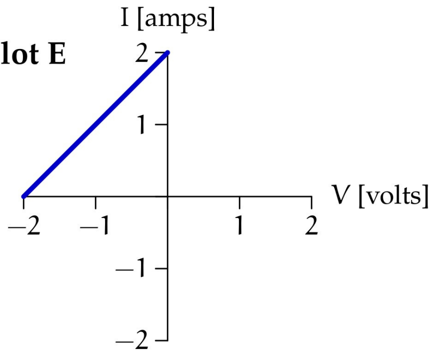
Plot C



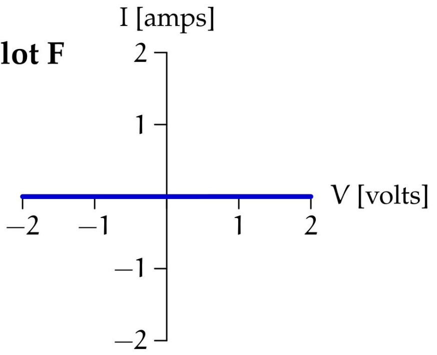
Plot D



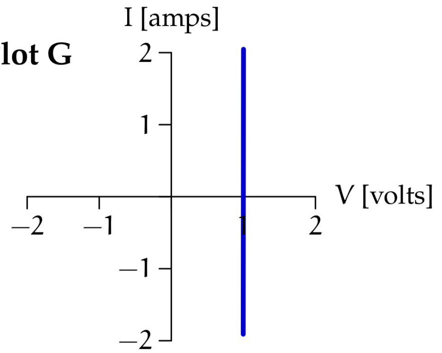
Plot E



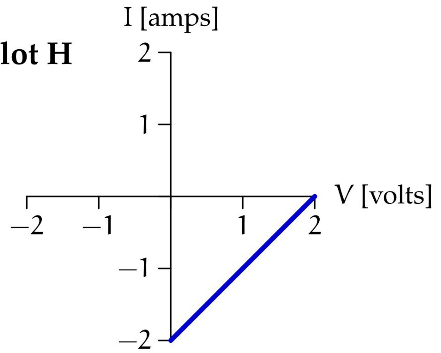
Plot F



Plot G



Plot H



6 Hop To It! (24 Points)

Imagine using graph search to write a computer program to play a video game where you control a small frog. The goal of the game is to get the frog to a goal location in a dynamic 2-dimensional grid.

Certain cells in the grid are dangerous at certain times (either because of an obstacle, or because an enemy is occupying that cell at a particular time) and, thus, the frog cannot go there without ending the game. Assume that there exists a function `is_safe((r, c), t)` that takes as input a tuple describing the frog's location in the 2-d world, as well as a discrete time. This function returns `True` if it is possible for the frog to occupy the cell `(r,c)` at time `t`. This function will also check to see if a cell is within the bounds of the game world.

Also, assume that the frog's goal location is stored in a variable called `goal_cell`.

The following code can be used to plan a path for the frog using breadth-first search:

```
start_state = ((0,0), 0) # ((row, column), time)

def goal_test(state):
    return state[0] == goalCell

def frog_successors(state):
    ((r, c), t) = state
    new_t = t+1
    children = []
    for (dr, dc) in [(-1,0), (0,0), (1,0), (0,-1), (0,1)]:
        new_r = r+dr
        new_c = c+dc
        if is_safe((new_r, new_c), new_t):
            children.append(((new_r,new_c), new_t))
    return children

path = search(frog_successors, start_state, goal_test)
```

6.1 High Scores

In three different game modes, the player's score in the frog game is computed in three different ways:

- In game mode **A**, the player's score is given by `max(0, 1000-time)`, where `time` is the number of timesteps it takes to reach the goal.
- In game mode **B**, the player's score is given by `max(0, 1000-num_moves)`, where `num_moves` is the number of timesteps on which the frog moved (took an action other than standing still).
- In game mode **C**, the player's score is given by `max(0, 1000-num_pos)`, where `num_pos` is the total number of distinct (r,c) locations the frog occupied during the game.

For which of the game modes, if any, is BFS (using the code above) guaranteed to result in a path that maximizes the player's score? Enter all your answers in the box below. Enter `None` if BFS is not guaranteed to return such a path in any of the game modes.

Some combination of A, B, and C; or None

6.2 Super Secret Bonus Level

In a bonus level, your virtual frog starts with 3 lives. Each time it enters an unsafe cell (as determined by the `is_safe` function), it loses one life. The frog must reach the goal with at least 1 life remaining, and so it can enter up to 2 cells that are not considered "safe" before finding the goal. In this level, the player's score is given by: `max(0, 1000-time + 500*lives_remaining)`, where `lives_remaining` represents the number of lives the frog had when finishing the level.

Write a new `start_state`, `goal_test`, and `frog_successors` that can be used in conjunction with `lib601`'s `uc_search` function to find a path to the goal that maximizes the player's score in this domain. Note that the source code for the lib601 search functions is included on the last page of this exam, which you may remove.

```
start_state =
```

```
def goal_test(state):
```

```
def frog_successors(state):
```

7 Turnip the Bayes (24 Points)

Ben Bitdiddle has given up on modern technology and is returning to his roots as a turnip farmer. He grows two different types of turnips: white turnips and yellow turnips.

He believes that:

- if the yellow turnips sprout, the white turnips will sprout with probability 0.2
- if the white turnips sprout, the yellow turnips will sprout with probability 0.4

Given this information, is it possible to determine which type of turnip is more likely to sprout overall?

Yes or No:

If yes, which type is more likely to sprout, and by how much?

If no, briefly describe what other information is needed and why.

In either case, **show your work in the box below.**

Not confident in his farming abilities, Ben also estimates that the probability that *neither* type of turnip will sprout is 0.8.

Taking into account all three of these pieces of information, what is the probability that the white turnips sprout but the yellow turnips do not?

Later, Ben begins experimenting with a new kind of purplish turnip. When he plants a new plant, he expects that it will have either 2 or 3 leaves, that it will either be tasty or not, and that it will either be purple (a successful experiment!) or not.

Over several decades of trials, he has observed that:

- Tasty turnips are half as likely to be purple as they are not to be purple.
- 12% of all turnips have three leaves.
- 25% of purple turnips have three leaves.
- 81% of turnips (regardless of color) are tasty.
- 9% of turnips are purple and have three leaves

Given this information, fill in the following table representing the joint distribution over color and tastiness of this new turnip:

	Purple	Not Purple
Tasty		
Not Tasty		

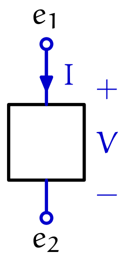
8 Circuit Solver (20 Points)

Ivana deBugyu is a minimalist programmer. Looking at 6.01's circuit solver, she finds the structure to be unnecessarily complicated. She observes that resistors, voltage sources, current sources, and any combinations thereof, can be described in a unified format, in terms of their I-V curves:

$$aV + bI + c = 0$$

Ivana throws out all the subclasses and provides this single class for components:

```
class OnePort:
    def __init__(self, a, b, c, e1, e2, i):
        self.e1 = e1
        self.e2 = e2
        self.i = i
        self.a = a
        self.b = b
        self.c = c
        self.equation = ([(a, self.e1),
                          (-a, self.e2),
                          (b, self.i),
                          (c, None)])
```



8.1 Primitives

Define a function `voltage_source(v0, e1, e2, i)` that returns an instance of the `OnePort` class defined above that is equivalent to a voltage source with voltage `v0` connected between nodes with names `e1` and `e2`, and with a current called `i` flowing through it, where e_1, e_2, and i are oriented as shown above.

```
def voltage_source(v0, e1, e2, i):
```

Similarly, define a function `current_source(i0, e1, e2, i)` to represent a current source with value `i0`.

```
def current_source(i0, e1, e2, i):
```

Finally, define a function `resistor(r, e1, e2, i)` to represent a resistor with resistance `r`.

```
def resistor(r, e1, e2, i):
```

8.2 Combinations

Define a function `parallel(c1, c2)`. This function should take two instances of `OnePort` as input, and it should return a new instance of `OnePort` representing the parallel combination of `c1` and `c2`. You should call the current flowing through this new component `'i'`.

You may assume that `c1.e1` and `c2.e1` are the same node; and that `c1.e2` and `c2.e2` are the same node.

For full credit, make sure your function properly deals with *all possible components `c1` and `c2`*. It should also return `None` in the case where `c1` and `c2` represent components that cannot be combined in parallel without violating any circuit laws.

```
def parallel(c1, c2):
```

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Source Code for lib601's Graph Search Functions

search (BFS/DFS)

```
def search(successors, start_state, goal_test, dfs = False):
    if goal_test(start_state):
        return [start_state]
    else:
        agenda = [SearchNode(start_state, None)]
        visited = {start_state}
        while len(agenda) > 0:
            if dfs:
                parent = agenda.pop(-1)
            else:
                parent = agenda.pop(0)
            for child_state in successors(parent.state):
                child = SearchNode(child_state, parent)
                if goal_test(child_state):
                    return child.path()
                if child_state not in visited:
                    agenda.append(child)
                    visited.add(child_state)
        return None
```

uc_search (UC/A*)

```
def uc_search(successors, start_state, goal_test, heuristic=lambda s: 0):
    if goal_test(start_state):
        return [start_state]
    agenda = [(heuristic(start_state), SearchNode(start_state, None, cost=0))]
    expanded = set()
    while len(agenda) > 0:
        agenda.sort()
        priority, parent = agenda.pop(0)
        if parent.state not in expanded:
            expanded.add(parent.state)
            if goal_test(parent.state):
                return parent.path()
            for child_state, cost in successors(parent.state):
                child = SearchNode(child_state, parent, parent.cost+cost)
                if child_state not in expanded:
                    agenda.append((child.cost+heuristic(child_state),child))
    return None
```

SearchNode class

```
class SearchNode:
    def __init__(self, state, parent, cost = 0.):
        self.state = state
        self.parent = parent
        self.cost = cost

    def path(self):
        p = []
        node = self
        while node:
            p.append(node.state)
            node = node.parent
        return p[::-1]
```

Worksheet (intentionally blank)