6.01 Final Exam: Spring 2010

Name:	**Section:**

Enter all answers in the boxes provided.

During the exam you may:

- read any paper that you want to
- use a calculator

You may not

- use a computer, phone or music player

For staff use:

1.	/12
2.	/10
3.	/18
4.	/6
5.	/12
6.	/4
7.	/20
8.	/18
total:	/100

1 A Library with Class (12 points)

Let's build a class to represent a library; let's call it `Library`. In this problem, we'll deal with some standard types of objects:

- A book is represented as a string – its title.
- A patron (person who uses the library) is represented as a string – his/her name.
- A date is represented by an integer – the number of days since the library opened.

The class should have an attribute called `dailyFine` that starts out as `0.25`. The class should have the following methods:

- `__init__`: takes a list of books and initializes the library.
- `checkOut`: is given a book, a patron and a date on which the book is being checked out and it records this. Each book can be kept for 7 days before it becomes overdue, i.e. if checked out on day x, it becomes due on day $x + 7$ and it will be considered overdue by one day on day $x + 8$. It returns `None`.
- `checkIn`: is given a book and a date on which the book is being returned and it updates the records. It returns a number representing the fine due if the book is overdue and 0.0 otherwise. The fine is the number of days overdue times the value of the `dailyFine` attribute.
- `overdueBooks`: is given a patron and a date and returns the list of books which that patron has checked out which are overdue at the given date.

Here is an example of the operation of the library:

```
>>> lib = Library(['a', 'b', 'c', 'd', 'e', 'f'])
>>> lib.checkOut('a', 'T', 1)
>>> lib.checkOut('c', 'T', 1)
>>> lib.checkOut('e', 'T', 10)
>>> lib.overdueBooks('T', 13)
['a', 'c']
>>> lib.checkIn('a', 13)
1.25
>>> lib.checkIn('c', 18)
2.50
>>> lib.checkIn('e', 18)
0.25
```

In the boxes below, define the `Library` class as described above. Above each answer box we repeat the specification for each of the attributes and methods given above. Make sure that you enter complete definitions in the boxes, including complete `class` and `def` statements.

Use a dictionary to store the contents of the library. Do not repeat code if at all possible. You can assume that all the operations are legal, for example, all books checked out are in the library and books checked in have been previously checked out.

1.1

Class definition:

Include both the start of the class definition and the method definition for `__init__` in this first answer box.

The class should have an attribute called `dailyFine` that starts out as `0.25`.

`__init__`: takes a list of books and initializes the library.

`checkOut`: is given a book, a patron and a date on which the book is being checked out and it records this. Each book can be kept for 7 days before it becomes overdue, i.e. if checked out on day x, it becomes due on day $x + 7$ and it will be considered overdue by one day on day $x + 8$. It returns `None`.

`checkIn`: is given a book and a date on which the book is being returned and it updates the records. It returns a number representing the fine due if the book is overdue and 0.0 otherwise. The fine is the number of days overdue times the value of the `dailyFine` attribute.

`overdueBooks`: is given a patron and a date and returns the list of books which that patron has checked out which are overdue at the given date.

1.2

Define a new class called `LibraryGrace` that behaves just like the `Library` class except that it provides a grace period (some number of days after the actual due date) before fines start being accumulated. The number of days in the grace period is specified when an instance is created. See the example below.

```
>>> lib = LibraryGrace(2, ['a', 'b', 'c', 'd', 'e', 'f'])
>>> lib.checkOut('a', 'T', 1)
>>> lib.checkIn('a', 13)
0.75
```

Write the complete class definition for `LibraryGrace`. To get full credit you should not repeat any code that is already in the implementation of `Library`, in particular, you should not need to repeat the computation of the fine.

Class definition:

2 Library State Machine (10 points)

We will now define a state machine class, called `LibrarySM`, to operate the library.

The state machine will be initialized with a list of books and will create an instance of the `Library` class and store it as an instance variable (assume that the `Library` class is defined in the same file as your definition of `LibrarySM`).

We will allow the state machine to modify this library instance, like we did the grid instances in design lab, for the sake of efficiency. However, your `getNextValues` method should not change any other instance variables.

Each input to the state machine will be a tuple of length 2; the first element is a string indicating an operation and the second element is the "argument" for that operation. The output of the machine should be `None` unless specified otherwise below.

The allowed types of inputs (and their outputs) are illustrated by example below:

- `('day', 3)` – advance the current date by 3 (or whatever integer is the argument); the date starts at 0. Output the new date in the format `('date', 5)`.
- `('start', 'T')` – start dealing with patron `'T'`.
- `('end', 'T')` – end dealing with patron `'T'`; the output should be the total accumulated fine for that patron since the most recent `start` (which may be 0.0), in the format `('total fine', 0.5)`.
- `('co', 'a')` – check out book `'a'` for the current patron.
- `('ci', 'a')` – check in book `'a'` for the current patron; the output should be the fine if this book is overdue or 0.0 if it's not, in the format `('fine', 0.5)`.

Here is an example of the operation of the machine.

```
>>> libsm = LibrarySM(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'])
>>> libsm.transduce([('day', 1),
                     ('start', 'T'),
                     ('co', 'a'), ('co', 'b'), ('co', 'c'),
                     ('co', 'd'), ('co', 'e'), ('co', 'f'),
                     ('end', 'T'),
                     ('start', 'X'),
                     ('co', 'g'), ('co', 'h'), ('co', 'i'),
                     ('end', 'X'),
                     ('day', 8),
                     ('start', 'T'),
                     ('ci', 'a'),
                     ('ci', 'b'),
                     ('end', 'T') ])

[('date', 1),
 None,
 None, None, None,
 None, None, None,
 ('total fine', 0.0),
 None,
 None, None, None,
 ('total fine', 0.0),
 ('date', 9),
 None,
 ('fine', 0.25),
 ('fine', 0.25),
 ('total fine', 0.5)]
```

1. Assuming that the inital date for the library is 0, what will the initial state of your machine be? Explain each component.

2. Write out the definition of `LibrarySM` class in Python. You can assume that all inputs will be legal (nobody will try to check out a book that is not in the library; there will not be a `start` for a new patron before the `end` of the previous patron; all the operations and arguments are legal, etc.).

Class definition:

Scratch paper

3 Machine Control (18 points)

Consider the following definition of a linear system, with gains `k1` and `k2`:

```
class Accumulator(sm.SM):
    startState = 0.0
    def getNextValues(self, state, inp):
        return(state+inp, state+inp)

def system(k1, k2):
    plant = Accumulator()
    sensor = sm.Delay()
    controller = sm.ParallelAdd(sm.Gain(k1),
                                sm.Cascade(Accumulator(), sm.Gain(k2)))
    return sm.FeedbackSubtract(sm.Cascade(controller, plant), sensor)
```

Note that `ParallelAdd` takes one input and provides it to two machines and outputs the **sum** of the outputs of the two machines.

Here's space to draw the block diagram, if you find it helpful (it won't be graded).

3.1

- Write a system function for the sensor.

- Write a system function for the plant.

- Write a system function for the controller, in terms of `k1` and `k2`.

- Write a system function for the cascade combination of the controller and plant, in terms of `k1` and `k2`.

- Write a system function for the whole system, in terms of `k1` and `k2`.

3.2

Imagine a different system, whose system function is given by

$$\frac{k_4\mathcal{R} - k_3 + k_3k_4}{(1-k_3)\mathcal{R}^2 + (k_3 + 3k_4 - 2)\mathcal{R} + 1}$$

If we pick $k_4 = 0$, give any non-zero value of k_3 for which the system will converge, or explain why there isn't one.

Scratch paper

4 Hot Bath (6 points)

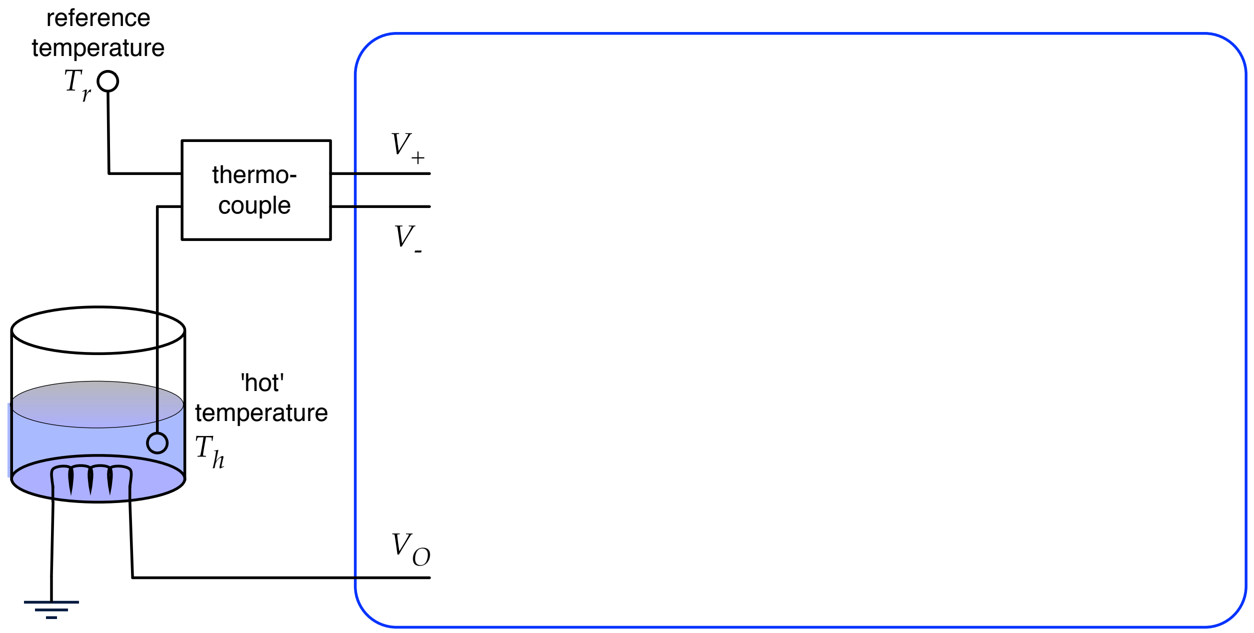
A *thermocouple* is a physical device with two temperature probes and two electronic terminals. If the probes are put in locations with different temperatures, there will be a voltage difference across the terminals. In particular,

$$V_+ - V_- = k(T_h - T_r)$$

where T_h is the temperature (°F) at the 'hot' probe, T_r is the temperature (°F) at the reference probe, and k is about 0.02.

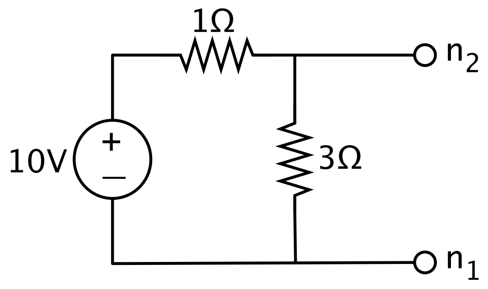
We have a vat of liquid that contains a heater (the coil at the bottom) and the 'hot' temperature sensor of the thermocouple. We would like to keep the liquid at the same temperature as the reference probe. The heater should be off if $T_r \leqslant T_h$ and be on, otherwise. When $T_r - T_h = 1°F$, then V_O should be approximately +5V.

Design a simple circuit (using one or two op-amps and some resistors of any values you want) that will achieve this; pick particular values for the resistors. Assume that we have a power supply of +10V available, and that the voltage difference $V_+ - V_-$ is in the range −10V to +10V.

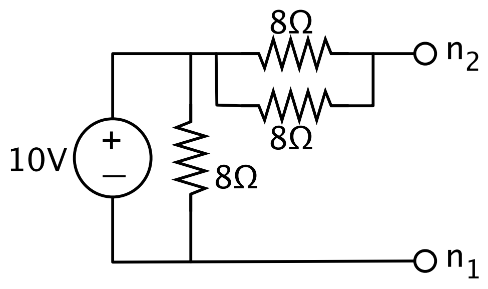


5 Equivalences (12 points)

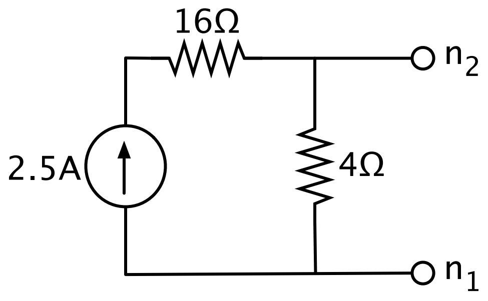
For each of the circuits below, provide the Thevenin equivalent resistance and voltage as seen from the $n_1 - n_2$ port. For the circuits with op-amps, treat them using ideal op-amp model.



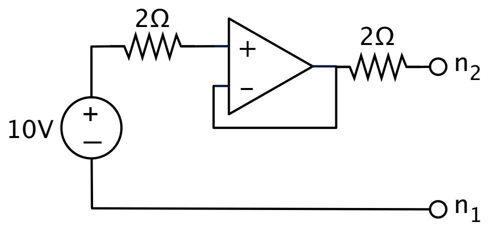
V_{th} = R_{th} =



V_{th} = R_{th} =



V_{th} = R_{th} =

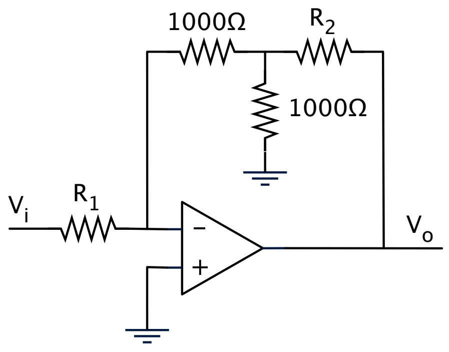


V_{th} = R_{th} =

Scratch paper

6 T circuit (4 points)

Provide values for resistors R_1 and R_2 that make $V_o = -3V_i$.



$R_1 =$

$R_2 =$

7 Coyote v. Roadrunner (20 points)

Consider a world with some population R of roadrunners and C of coyotes. Roadrunners eat insects and coyotes eat roadrunners (when they can catch them). The roadrunner population naturally increases, but when there are coyotes around, they eat the roadrunners and decrease the roadrunner population. The coyote population, in the absence of roadrunners, finds something else to eat, and stays the same, or declines a little.

7.1 Initial distribution

Let's assume that the roadrunner population (R) can be low, med or high, and that the coyote population (C) can be low, med, or high. So, there are **9 states**, each corresponding to some value of R and some value of C.

Here is the initial **belief state**, which is written as a joint distribution over C and R, $\Pr(C, R)$.

		Coyotes		
		low	med	high
Roadrunners	low	0.04	0.20	0.18
	med	0.08	0.16	0.02
	high	0.28	0.04	0.00

1. What is the marginal distribution $\Pr(C)$?

2. What is the distribution $\Pr(R|C = \mathrm{low})$?

3. What is the distribution $\Pr(R|C = \mathrm{high})$?

4. Are R and C independent? Explain why or why not.

7.2 Transitions

Let's start by studying how the roadrunner population evolves when there are no coyotes, represented by $C_t = \mathsf{low}$ (and the coyote population doesn't change).

$$\begin{aligned}
\Pr(R_{t+1} = \mathsf{low} \mid C_t = \mathsf{low}, R_t = \mathsf{low}) &= 0.1 \\
\Pr(R_{t+1} = \mathsf{med} \mid C_t = \mathsf{low}, R_t = \mathsf{low}) &= 0.9 \\
\Pr(R_{t+1} = \mathsf{high} \mid C_t = \mathsf{low}, R_t = \mathsf{low}) &= 0.0 \\
\Pr(R_{t+1} = \mathsf{low} \mid C_t = \mathsf{low}, R_t = \mathsf{med}) &= 0.0 \\
\Pr(R_{t+1} = \mathsf{med} \mid C_t = \mathsf{low}, R_t = \mathsf{med}) &= 0.3 \\
\Pr(R_{t+1} = \mathsf{high} \mid C_t = \mathsf{low}, R_t = \mathsf{med}) &= 0.7 \\
\Pr(R_{t+1} = \mathsf{low} \mid C_t = \mathsf{low}, R_t = \mathsf{high}) &= 0.0 \\
\Pr(R_{t+1} = \mathsf{med} \mid C_t = \mathsf{low}, R_t = \mathsf{high}) &= 0.0 \\
\Pr(R_{t+1} = \mathsf{high} \mid C_t = \mathsf{low}, R_t = \mathsf{high}) &= 1.0
\end{aligned}$$

1. Assume $C_t = \mathsf{low}$. For simplicity, also assume that we start out knowing with certainty that the roadrunner population is low. What is the distribution over the possible levels of the roadrunner population (low, med, high) after 1 time step?

2. Assume $C_t = \mathsf{low}$. For simplicity, also assume that we start out knowing with certainty that the roadrunner population is low. What is the distribution over the possible levels of the roadrunner population (low, med, high) after 2 time steps?

7.3 Observations

Imagine that you are starting with the initial belief state, the joint distribution from **problem 7.1**. If it helps, you can think of it as the following `DDist` over pairs of values (the first is the value of R, the second is the value of C):

```
DDist({('low', 'low') : 0.04, ('low', 'med') : 0.2, ('low', 'high') : 0.18,
       ('med', 'low') : 0.08, ('med', 'med') : 0.16, ('med', 'high') : 0.02,
       ('high', 'low') : 0.28, ('high', 'med') : 0.04, ('high', 'high') : 0.00})
```

You send an ecologist out into the field to sample the numbers of roadrunners and coyotes. The ecologist can't really figure out the absolute numbers of each species, but reports one of three observations:

- **moreC**: means that there are significantly more coyotes than roadrunners (that is, that the level of coyotes is **high** and the level of roadrunners is **med** or **low**, or that the level of coyotes is **med** and the level of roadrunners is **low**).
- **moreR**: means that there are significantly more roadrunners than coyotes (that is, that the level of roadrunners is **high** and the level of coyotes is **med** or **low**, or that the level of roadrunners is **med** and the level of coyotes is **low**).
- **same**: means that there are roughly the same number of coyotes as roadrunners (the populations have the same level).

1. If there is no noise in the ecologist's observations (that is, the observation is always true, given the state), and the observation is **moreR**, what is the resulting belief state $\Pr(C_0, R_0 \mid O_0 = \textbf{moreR})$ (the distribution over states given the observation)?

2. Now, we will assume that the ecologist's observations are fallible.

$$\begin{aligned}
\Pr(O_t = \textbf{moreC} \mid C_t > R_t) &= 0.9 \\
\Pr(O_t = \textbf{same} \mid C_t > R_t) &= 0.1 \\
\Pr(O_t = \textbf{moreR} \mid C_t > R_t) &= 0.0 \\
\Pr(O_t = \textbf{moreC} \mid C_t = R_t) &= 0.1 \\
\Pr(O_t = \textbf{same} \mid C_t = R_t) &= 0.8 \\
\Pr(O_t = \textbf{moreR} \mid C_t = R_t) &= 0.1 \\
\Pr(O_t = \textbf{moreC} \mid C_t < R_t) &= 0.0 \\
\Pr(O_t = \textbf{same} \mid C_t < R_t) &= 0.1 \\
\Pr(O_t = \textbf{moreR} \mid C_t < R_t) &= 0.9
\end{aligned}$$

Now, if the observation is **moreR**, what is the belief state $\Pr(C_0, R_0 \mid O_0 = \textbf{moreR})$ (the distribution over states given the observation)?

8 Ab und Aufzug (18 points)

Hans, Wilhelm, and Klaus are three business tycoons in a three-story skyscraper with one elevator. We know in advance that they will call the elevator simultaneously after their meetings tomorrow and we want to get them to their destinations as quickly as possible (time is money!). We also know that the elevator will be on the first floor when they call it. We're going to use search to find the best path for the elevator to take.

Hans	starts on the 2nd floor,	and wants to go to the 1st floor.
Wilhelm	starts on the 3rd floor,	and wants to go to the 1st floor.
Klaus	starts on the 3rd floor,	and wants to go to the 2nd floor.

State will be stored as a tuple whose first element is the location of the elevator and whose second element is a tuple with the locations of Hans, Wilhelm, and Klaus, in that order. A location of `None` means that the person in question is riding the elevator. So the state

```
  (2, (None, 1, 2))
```

means that Hans is on the elevator which is on the 2nd floor and Klaus is on the 2nd floor as well, but not on the elevator, whereas Wilhelm is on the 1st floor.

Our legal actions are:

```
legalActions =
  ['ElevatorDown', 'ElevatorUp', (0, 'GetsOn'), (0, 'GetsOff'),
   (1, 'GetsOn'), (1, 'GetsOff'), (2, 'GetsOn'), (2, 'GetsOff')]
```

where `"ElevatorUp"` causes the elevator to move one floor up, `"ElevatorDown"` causes it to move one floor down, and 0, 1, and 2 correspond to Hans, Wilhelm, and Klaus, respectively.

Let's say it takes one minute for the elevator to move one floor in either direction, and it takes one minute for anyone to get on or off the elevator unless Klaus is involved. It takes Klaus five minutes to get on or off the elevator (he's extremely slow); it also takes everyone else five minutes to get on or off the elevator if Klaus is already riding the elevator (he's extremely talkative).

8.1 Goal

Write the goal function.

```
    def goal(state):
```

Scratch paper

8.2 Search Strategies

In what follows: `ED` is `'ElevatorDown'`, `EU` is `'ElevatorUp'`, `0On` is `(0, 'GetsOn')`, etc. Also, **N** stands for `None`.

Suppose that we are in the middle of the search and the search agenda consists of the following nodes (**listed in the order that they were added to the agenda, earliest first**):

A: $(1, (2, 3, 3)) \xrightarrow{\texttt{EU}} (2, (2, 3, 3)) \xrightarrow{\texttt{EU}} (3, (2, 3, 3)) \xrightarrow{\texttt{2On}} (3, (2, 3, \mathbf{N})) \xrightarrow{\texttt{1On}} (3, (2, \mathbf{N}, \mathbf{N}))$

B: $(1, (2, 3, 3)) \xrightarrow{\texttt{EU}} (2, (2, 3, 3)) \xrightarrow{\texttt{0On}} (2, (\mathbf{N}, 3, 3)) \xrightarrow{\texttt{ED}} (1, (\mathbf{N}, 3, 3)) \xrightarrow{\texttt{0Off}} (1, (1, 3, 3))$

C: $(1, (2, 3, 3)) \xrightarrow{\texttt{EU}} (2, (2, 3, 3)) \xrightarrow{\texttt{EU}} (3, (2, 3, 3)) \xrightarrow{\texttt{1On}} (3, (2, \mathbf{N}, 3)) \xrightarrow{\texttt{2On}} (3, (2, \mathbf{N}, \mathbf{N})) \xrightarrow{\texttt{ED}} (2, (2, \mathbf{N}, \mathbf{N}))$

- Assume that no states other than the ones listed were visited in the search.
- An illegal action leaves you in the same state and Pruning Rule 1 applies (don't consider any path that visits the same state twice).
- Assume that the order of operations is as listed at the beginning of this problem:
  `ED`, `EU`, `0On`, `0Off`, `1On`, `1Off`, `2On`, `2Off`

Note that in general you may have to expand more than one node to find the next state that is visited.

8.2.1 If we are doing breadth-first search (BFS)

1. Starting from this agenda, which node (A, B, or C) gets expanded first (circle one)?
A **B** **C**

2. Which node gets added to the agenda first? Specify its parent node, action, and state.

3. What is the total path cost of the new node added to the agenda?

Agenda:

A: $(1, (2, 3, 3)) \xrightarrow{\text{EU}} (2, (2, 3, 3)) \xrightarrow{\text{EU}} (3, (2, 3, 3)) \xrightarrow{\text{2O}} (3, (2, 3, \mathbf{N})) \xrightarrow{\text{1On}} (3, (2, \mathbf{N}, \mathbf{N}))$

B: $(1, (2, 3, 3)) \xrightarrow{\text{EU}} (2, (2, 3, 3)) \xrightarrow{\text{0On}} (2, (\mathbf{N}, 3, 3)) \xrightarrow{\text{ED}} (1, (\mathbf{N}, 3, 3)) \xrightarrow{\text{0Off}} (1, (1, 3, 3))$

C: $(1, (2, 3, 3)) \xrightarrow{\text{EU}} (2, (2, 3, 3)) \xrightarrow{\text{EU}} (3, (2, 3, 3)) \xrightarrow{\text{1O}} (3, (2, \mathbf{N}, 3)) \xrightarrow{\text{2On}} (3, (2, \mathbf{N}, \mathbf{N})) \xrightarrow{\text{ED}} (2, (2, \mathbf{N}, \mathbf{N}))$

8.2.2 If we are doing depth-first search (DFS)

1. Starting from this agenda, which node (A, B, or C) gets expanded first (circle one)?
A **B** **C**

2. Which node gets added to the agenda first? Specify its parent node, action, and state.

3. What is the total path cost of the new node added to the agenda?

8.2.3 If we are doing breadth-first search with dynamic programming (BFS+DP)

1. Starting from this agenda, which node (A, B, or C) gets expanded first (circle one)?
A **B** **C**

2. Which node gets added to the agenda first? Specify its parent node, action, and state.

3. What is the total path cost of the new node added to the agenda?

Agenda:

A: $(1, (2, 3, 3)) \xrightarrow{\mathsf{EU}} (2, (2, 3, 3)) \xrightarrow{\mathsf{EU}} (3, (2, 3, 3)) \xrightarrow{\mathsf{2O}} (3, (2, 3, \mathbf{N})) \xrightarrow{\mathsf{1On}} (3, (2, \mathbf{N}, \mathbf{N}))$

B: $(1, (2, 3, 3)) \xrightarrow{\mathsf{EU}} (2, (2, 3, 3)) \xrightarrow{\mathsf{0On}} (2, (\mathbf{N}, 3, 3)) \xrightarrow{\mathsf{ED}} (1, (\mathbf{N}, 3, 3)) \xrightarrow{\mathsf{0Off}} (1, (1, 3, 3))$

C: $(1, (2, 3, 3)) \xrightarrow{\mathsf{EU}} (2, (2, 3, 3)) \xrightarrow{\mathsf{EU}} (3, (2, 3, 3)) \xrightarrow{\mathsf{1O}} (3, (2, \mathbf{N}, 3)) \xrightarrow{\mathsf{2On}} (3, (2, \mathbf{N}, \mathbf{N})) \xrightarrow{\mathsf{ED}} (2, (2, \mathbf{N}, \mathbf{N}))$

8.2.4 If we are doing uniform-cost search (ucSearch)

1. Starting from this agenda, which node (A, B, or C) gets expanded first (circle one)?
A **B** **C**

2. Which node gets added to the agenda first? Specify its parent node, action, and state.

3. What is the total path cost of the new node added to the agenda?

8.3 Heuristics

Frieda, Lola, Ulrike, and Xenia (four engineers) are asked to produce heuristics to speed up the search. In all the heuristics, distance is measured in number of floors.

- Frieda suggests that you use the maximum of the distances between each person and his destination plus 2 times the number of people who are not on the right floor.
- Lola suggests that you use the maximum of the distances between each person and his destination plus 10 if Klaus is not on the elevator and not on the right floor, plus 5 if Klaus is on the elevator.
- Ulrike suggests that you use the sum of the distances between each person and his destination.
- Xenia suggests that you use the maximum of the distances between each person and his destination.

Which of these heuristics are admissible? Circle Yes or No.

- **F**: Yes No
- **L**: Yes No
- **U**: Yes No
- **X**: Yes No