Questions:

1. *Run ./fork.py -s 10 and see which actions are taken. Can you predict what the process tree looks like at each step? Use the -c flag to check your answers. Try some different random seeds (-s) or add more actions (-a) to get the hang of it.*

First a forks b:

```
a
|___b
```

Then a forks c:

```
a
|___b
|___c
```

Then c EXITS:

```
a
|___b
```

Then a forks d:

```
a
|___b
|___d
```

Finally a forks e:

```
a
|___b
|___d
|___e
```

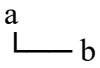
2. *One control the simulator gives you is the fork percentage, controlled by the -f flag. The higher it is, the more likely the next action is a fork; the lower it is, the more likely the action is an exit. Run the simulator with a large number of actions (e.g., -a 100) and vary the fork percentage from 0.1 to 0.9. What do you think the resulting final process trees will look like as the percentage changes? Check your answer with -c.*

The tree will look very shallow when the fork percentage is very low. Because each forked process will likely exit and won't have much chance to fork. Conversely, when the fork percentage is very high, the tree's depth is intense because each forked process has an increased opportunity to fork again before it exists.

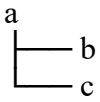
3. Now, switch the output by using the -t flag (e.g., run ./fork.py -t). Given a set of process trees, can you tell which actions were taken?

Process Tree:

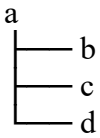
Action is a fork b



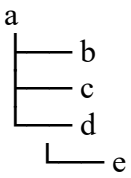
Action is a fork c



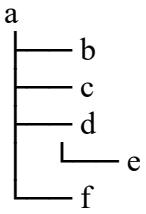
Action is a fork d



Action d fork e



Action a fork f



4. *One interesting thing to note is what happens when a child exits; what happens to its children in the process tree? To study this, let's create a specific example: ./fork.py -A a+b,b+c,c+d,c+e,c-. This example has process' a' create' b', which in turn creates' c', which then creates' d' and' e'. However, then,' c' exits. What do you think the process tree should like after the exit? What if you use the -R flag? Learn more about what happens to orphaned processes on your own to add more context.*

First, I thought when the parent process exits, its children's process will become the grandparent process. But, in fact, when process c exits, its children process d and e become process a's children. Finally, when using the -R flag, process d, and e become b's children.

5. One last flag to explore is the -F flag, which skips intermediate steps and only asks to fill in the final process tree. Run ./fork.py -F and see if you can write down the final tree by looking at the series of actions generated. Use different random seeds to try this a few times.

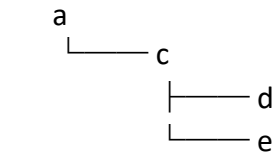
```
Action: a forks b
a
|___b

Action: a forks c
a
|___b
|___c

Action: c EXITS
a
|___b

Action: a forks d
a
|___b
|___d

Action: d forks e
a
|___b
|___d
    |___e
```

6. Finally, use both -t and -F together. This shows the final process tree, but then asks you to fill in the actions that took place. By looking at the tree, can you determine the exact actions that took place? In which cases can you tell? In which can't you tell? Try some different random seeds to delve into this question.

Process Tree:
a

Action?
Action?
Action?
Action?
Action?

Final Process Tree:



Based on the tree (4 processes) and there total 5 actions. So then there must be 4 forks, and one exits. Moreover, we didn't use the -R flag then:
a fork c
c fork d
c fork e
We can tell which processes fork which processes. There is an exit process that we can't tell whether it forked from a, b, c, d, or e.