Homework— fork.py (Simulation)

# Question 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 different random seeds (-s) or add more actions (-a) to get the hang of it.

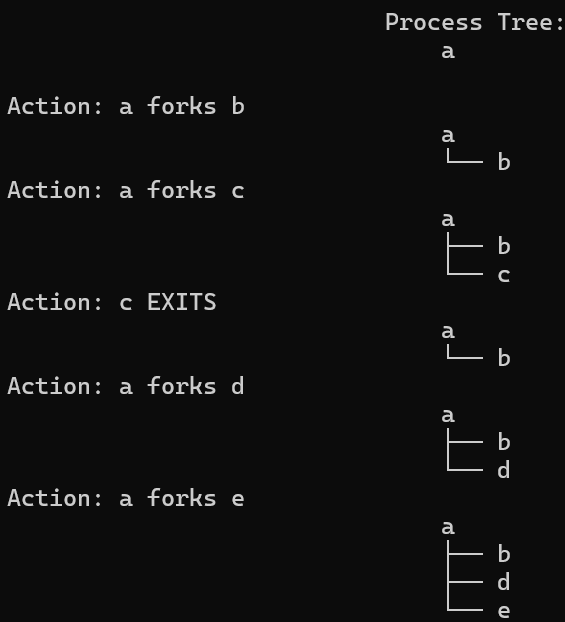
## Answer

Run with seed 10 (deterministic). The tree evolves via forks (adding children) and exits (removing nodes). Predict each step by tracking which process acts.

## Explanation

The simulator selects a process and either forks (creates a new child) or exits it based on the random seed. With -c the simulator prints the process tree at each action; by observing forks (nodes added) and exits (nodes removed) you can follow the steps.

## Output



# Question 2

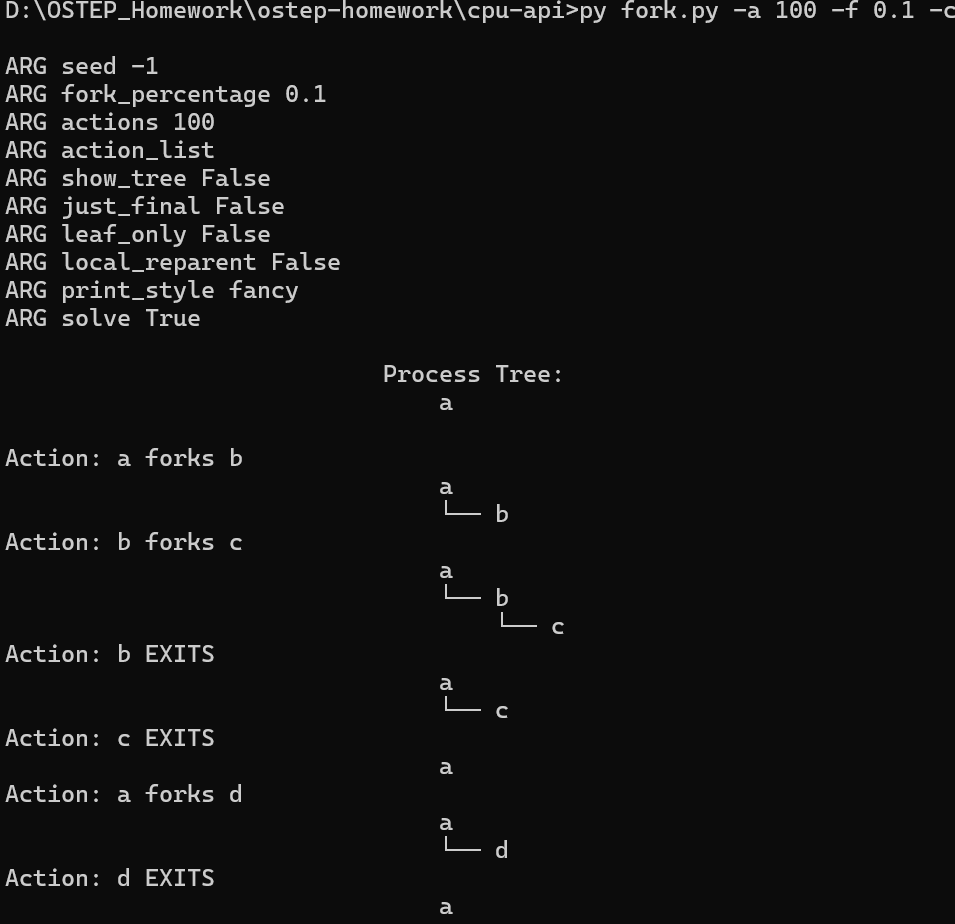
Run with many actions (-a 100) and vary fork percentage -f from 0.1 to 0.9. What do the final trees look like as percentage changes?

## Answer

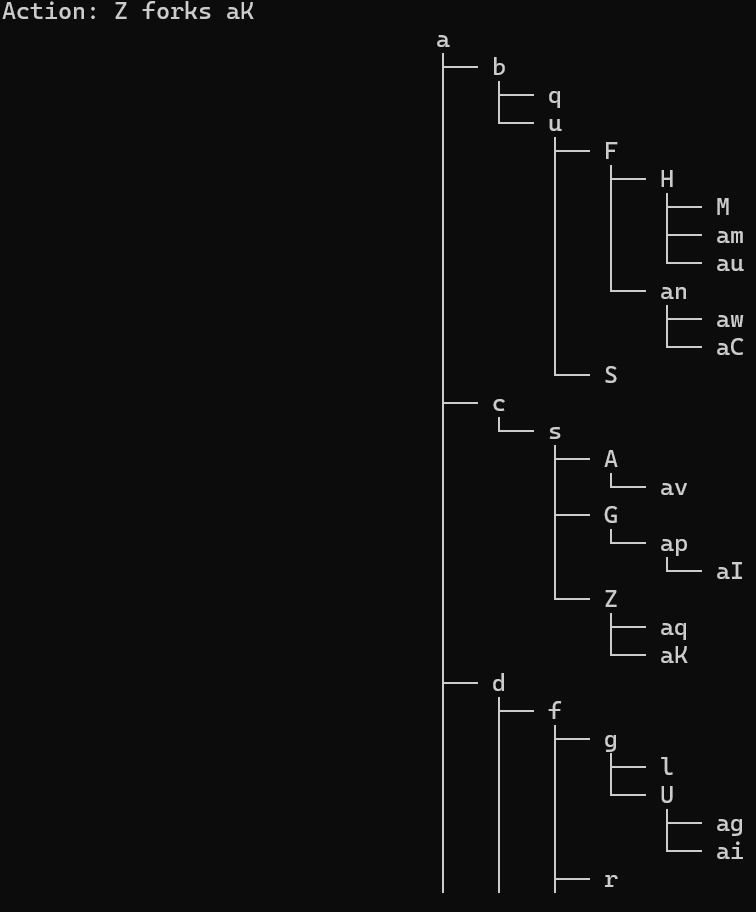
Lower fork percentage yields small/shallow trees; higher percentages produce larger, bushier trees. At ~0.5 the tree is moderate.

## Explanation

Fork probability controls growth vs shrinkage. With 0.1, exits dominate so the tree remains small. With 0.9, forks dominate and the tree quickly multiplies.



**At -f 0.9:**



# Question 3

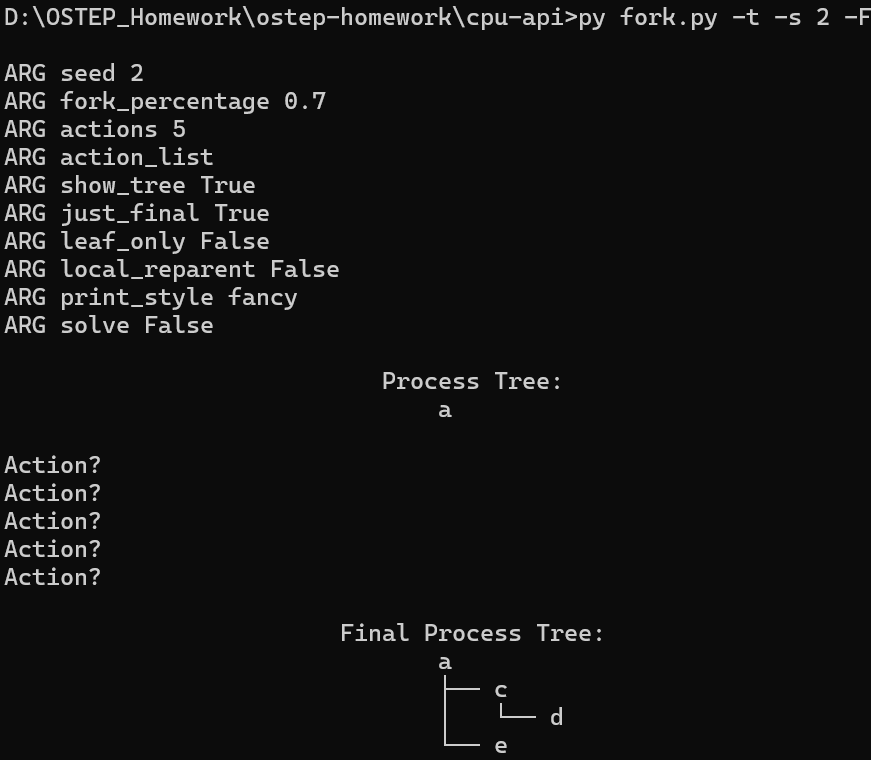
Switch output with -t (show final tree). Given a set of trees, can you tell which actions were taken?

## Answer

Not always. Some final trees are compatible with many action sequences; some (like linear chains) map to a small number of sequences.

## Explanation

The -t mode hides the step-by-step actions. You can sometimes deduce required forks (e.g., number of nodes implies at least that many forks), but the order of forks/exits is often ambiguous because exits may remove intermediate nodes that make histories equivalent.



# Question 4

Investigate what happens when a child exits. Run: ./fork.py -A a+b,b+c,c+d,c+e,c-

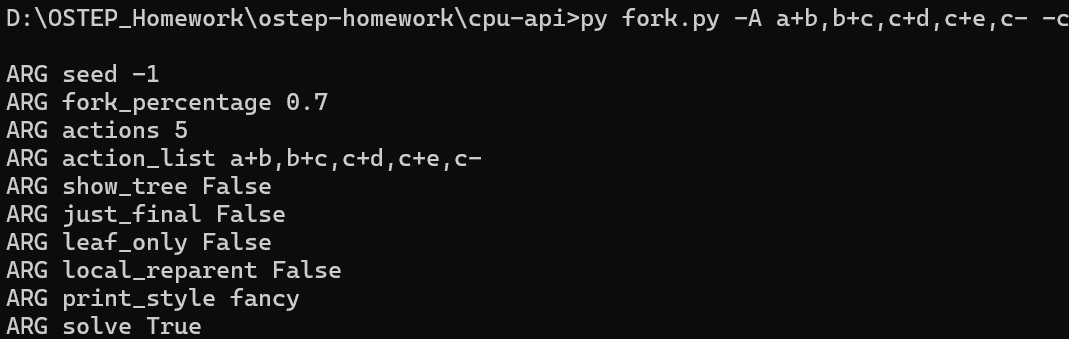
## Answer

When 'c' exits, its children 'd' and 'e' are orphaned and re-parented to the root (or to 'init' in real OS).

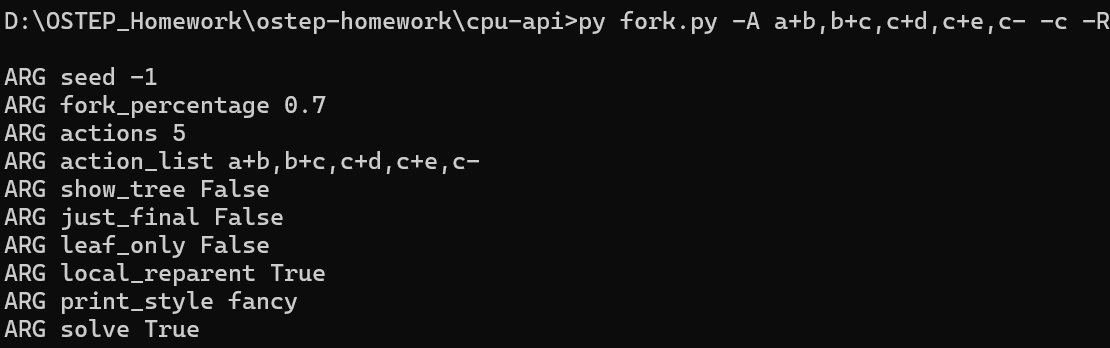
## Explanation

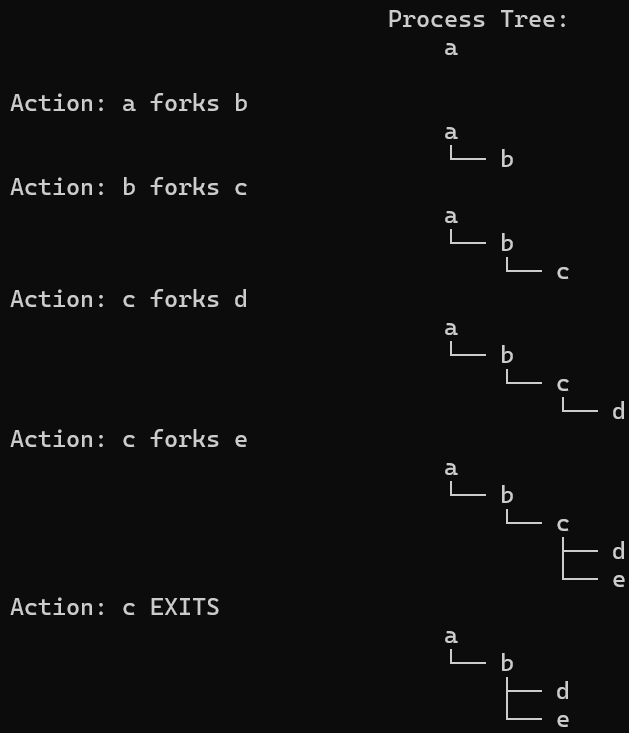
The -A sequence runs exact actions: a forks b; b forks c; c forks d; c forks e; c exits. After c exits, d and e have no parent 'c', so in the simulator they are reconnected to the root (a) or to an 'init' node depending on options. The -R flag changes the behavior to optionally remove or re-parent differently (see README).

## Commands to run









# Question 5

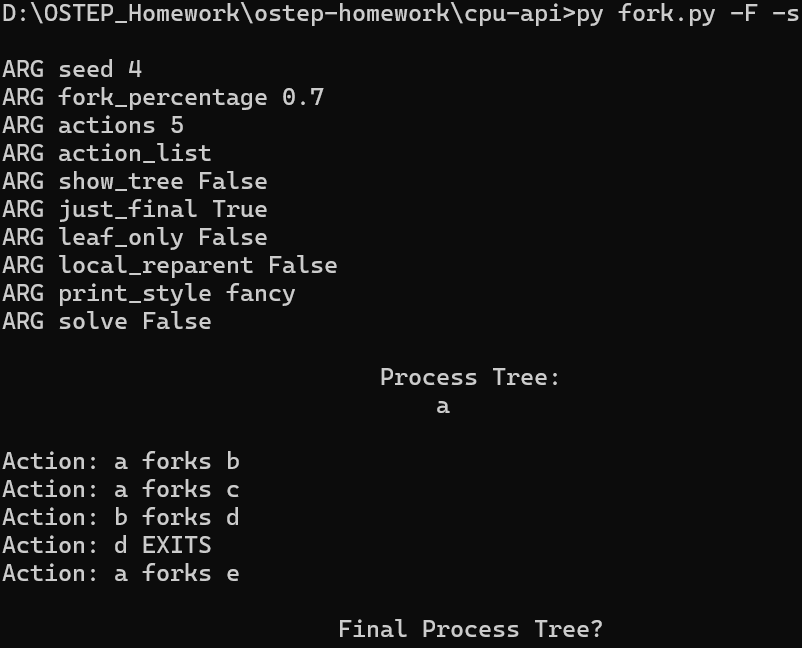
Use -F to skip intermediate steps and only show the final tree. Can you deduce the actions that led to it?

## Answer

Sometimes yes, sometimes no. -F forces you to infer a plausible action sequence; multiple sequences may be valid.

## Explanation

From the final tree you can usually infer the minimum number of forks required (equal to nodes-1), but not the exact order or which nodes exited earlier. The exercise trains backward reasoning about possible histories.



# Question 6

Use -t and -F together to show the final tree, then reconstruct actions. In which cases is the sequence uniquely determinable?

## Answer

Unique reconstruction possible for simple trees (e.g., linear chains). For more complex trees, many sequences can produce same final tree.

## Explanation

When every node (except root) has exactly one child in a straight chain, the only way to get that final chain is successive forks. But if branching exists, exits can reorder parent-child relationships via re-parenting, creating ambiguity.

## 