Luke Davidson
CS 5180
Ex6

1.) a.) A multi-step bootstrapping method will definitely perform better than the one step method (n=0) in Figure 8.3, although I don't believe a multi-step bootstrapping method would outperform that of Dyna-Q for a couple main reasons. First, Dyna-Q involves the implementation of both learning and planning, allowing for the benefits of both to be exploited. Dyna-Q will hold this advantage over any of the algorithms introduced in Chapter 7. Second, Dyna-Q allows an extra degree of updates to occur to the state-action value function compared to the methods in Chapter 7 (assuming $n > 0$). More updates is due to more exploration which ultimately leads to the calculation of more optimal functions.

b.) We could also do n-step returns for the planning phase if we made a modification such as the one the footnote suggests (allowing all actions to be randomly selected, and identifying values for R, S' if that state-action paid doesn't exist yet in the model). If we do not make that modification, some S' states wouldn't have any actions that exist yet in the model and the n-step returns wouldn't be able to continue. A major advantage of using n-step returns in the planning phase is that we would be able to perform multiple steps during the same time step (wouldn't be limited to individual timesteps), allowing for greater flexibility and faster learning, although a disadvantage is that some of these *n* future steps would not be as accurate during early episodes due to initial bias of what we identify them to be if they do not previously exist (footnote difference).

2.) a.) Pseudocode for Dyna-Q+ with footnote suggestions is shown below:

1. Initialize Q(S, A) and Model(S, A) for all s in S and a in A(s)
2. Reset environment, current_timestep = 0
3. Loop:
    a. S → current non-terminal state
    b. A → e-greedy(S, Q)
    c. Take action A, observe reward R, next state S', and termination status
    d. Update Q:

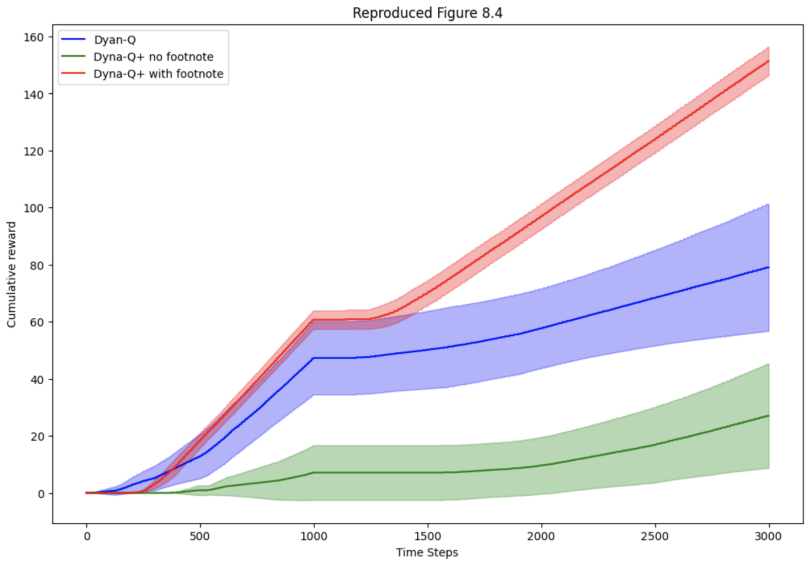
$$\text{(d) } Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

    e. Update Model: Model(S, A) → (R, S', current_timestep)
    f. Loop for range(n):
        i. S → randomly previously observed state
        ii. A → random action from action set ["up", "down", "left", "right"]
        iii. Check if selected A in Model(S, a)
            1. If True: R, S', last_timestep = Model(S, A)
            2. If False: R, S', last_timestep = 0, S, 0
        iv. R += kappa*(sqrt(current_timestep - last_timestep))
        v. Update Q:

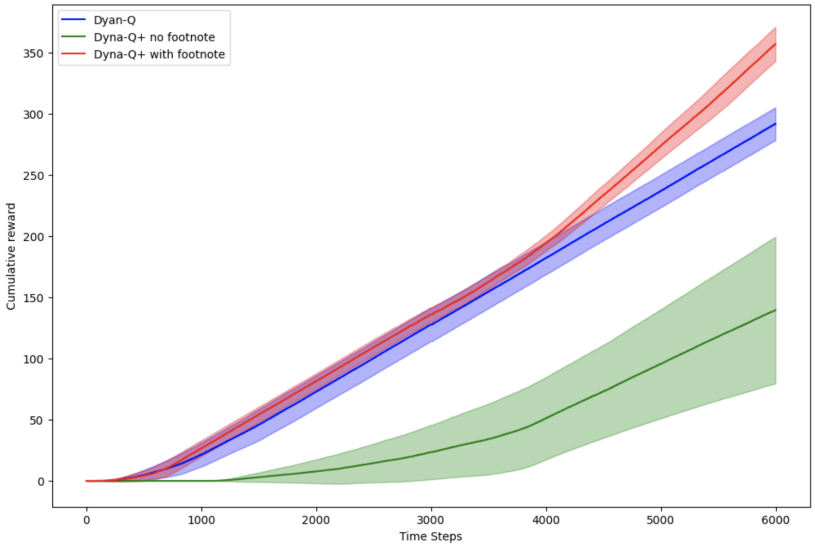
$$\text{(d) } Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

    g. current_timestep += 1
    h. If terminated: reset environment and continue; else: S = S'

2.) b.) Reproduced Figure 8.4 for the BlockingMaze environment for Dyna-Q and Dyna-Q+ with and without the footnote suggestion:



Reproduced Figure 8.5 for the ShortcutMaze environment for Dyna-Q and Dyna-Q+ with and without the footnote suggestion:



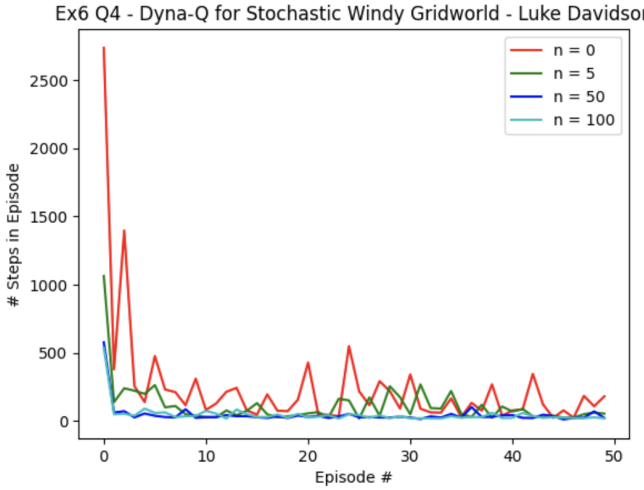
2.) c.) As seen in the above plots, the footnote suggestion had a large effect on the results (red versus green plots). We see much greater cumulative rewards when implementing the footnote suggestion due to the allowance of much more exploration in early episodes. Without the footnote suggestion, we are limited to the random selection of actions that have already been selected in that given state during the planning phase. With the footnote suggestion, all actions are available for selection. When all actions are available for selection, exploration greatly increases, especially in early episodes when maybe only one or two actions had been previously selected in a given state, and an accurate state-value function can be found much more quickly, allowing for faster overall learning.

3.) The main difference between exploration bonuses in UCB action selection and Dyna-Q+ is that in Dyna-Q+, the reward bonus directly affects/changes the state-action value function, while in UCB action selection the extra bonus does not have an effect on the state-action value function. The difference in the training between using these two methods due to this difference is mainly seen in the planning phase. In UCB, this exploration bonus only encourages exploration in the action selection, which occurs outside of the planning phase. When adding the bonus directly to the reward, and therefore directly changing Q(S, A), the exploration bonus can also be seen during the planning phase. This means that way more exploration is possible when adding the bonus directly to the reward, as the exploration can be exploited in the planning phase. I would assume that adding the bonus directly to the reward (Dyna-Q+) would allow the agent to learn much more quickly when changes in the environment are made, like the switch between grid worlds, due to this added exploration in the planning phase. On the other hand, an advantage to adding the bonus during action selection is that it does not affect the state-value function directly, and therefore there will be less variability in the state-action value function compared to the optimal state-action value function.

4.) a.) To modify Dyna-Q for a stochastic environment, I essentially added another dimension to my model, which was the stochastic dimension of wind. This wind dimension had four possible outcomes based on the environment and state: wind-1, wind, wind+1, or no wind. I modified the *step* method in the environment to output the wind type that was randomly chosen (if any), then added that observed reward and state prime to the model based on the wind type. Then, in the planning step, I added another step to select a wind type that had occurred in the state action pair, and used that to update Q. My modified pseudocode for the stochastic environment is below:

1. Initialize Q(S, A) and Model(s, a, w) for all s in states, a in A(s), and w in (w+1, w, w-1, none)
2. Loop:
    a. S → current non-terminal state
    b. A → e-greedy(S, Q)
    c. Take action A, observe reward R, next state S', termination status, and the wind type W that was randomly selected (if any)
    d. Update Q: $(\mathrm{d})\ Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    e. Update Model: Model(S, A, W) → (R, S)
    f. Loop for range(n):
        i. S → randomly previously observed state
        ii. A → random action in model[S]
        iii. W → random wind type in model[S][A]
        iv. R, S' = Model(S, A, W)
        v. Update Q:
        $(\mathrm{d})\ Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    g. If terminated: reset environment and continue; else: S = S'

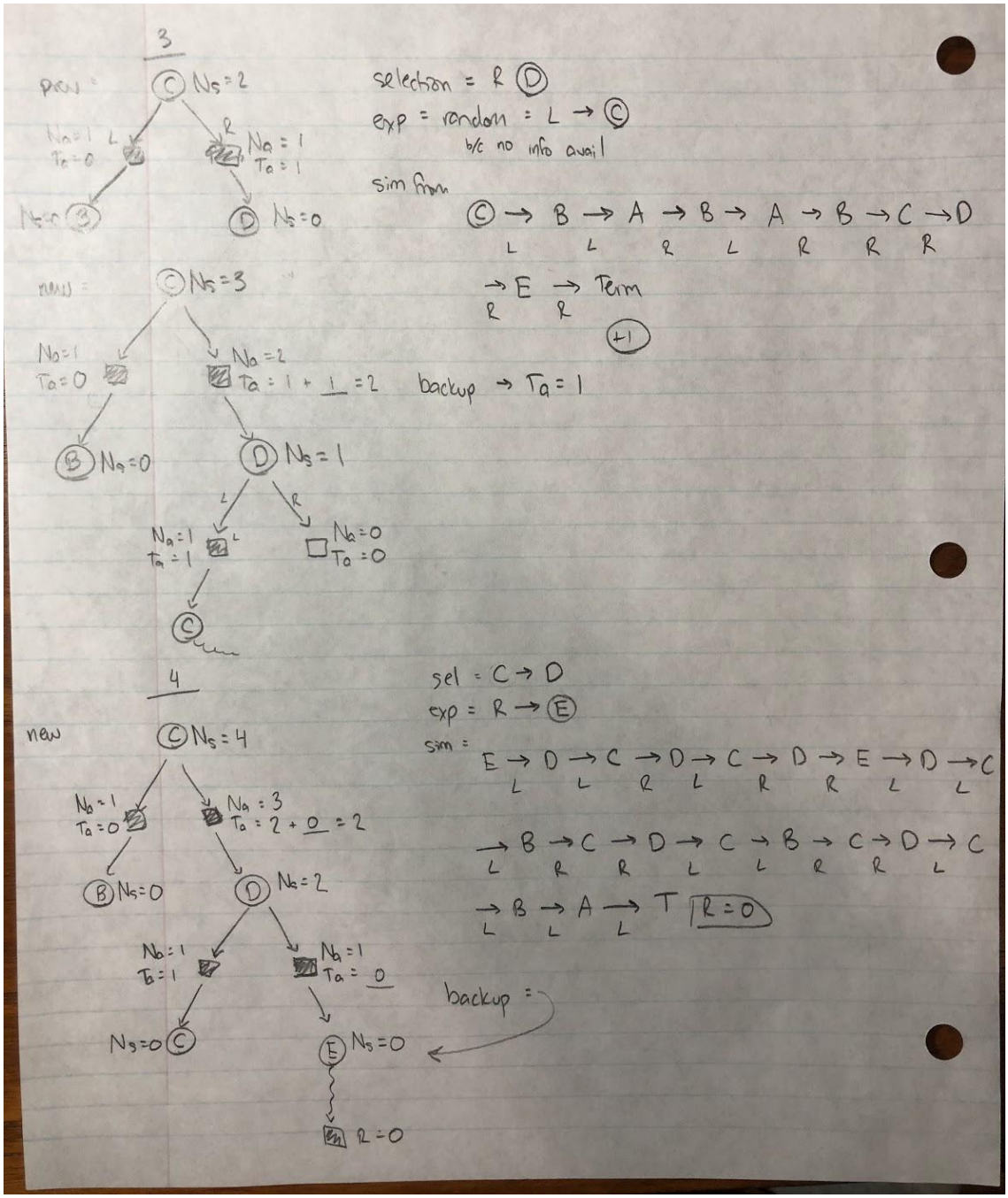
b.) Plot similar to Figure 8.2 for my modified Dyna-Q for stochastic Windy Grid world:

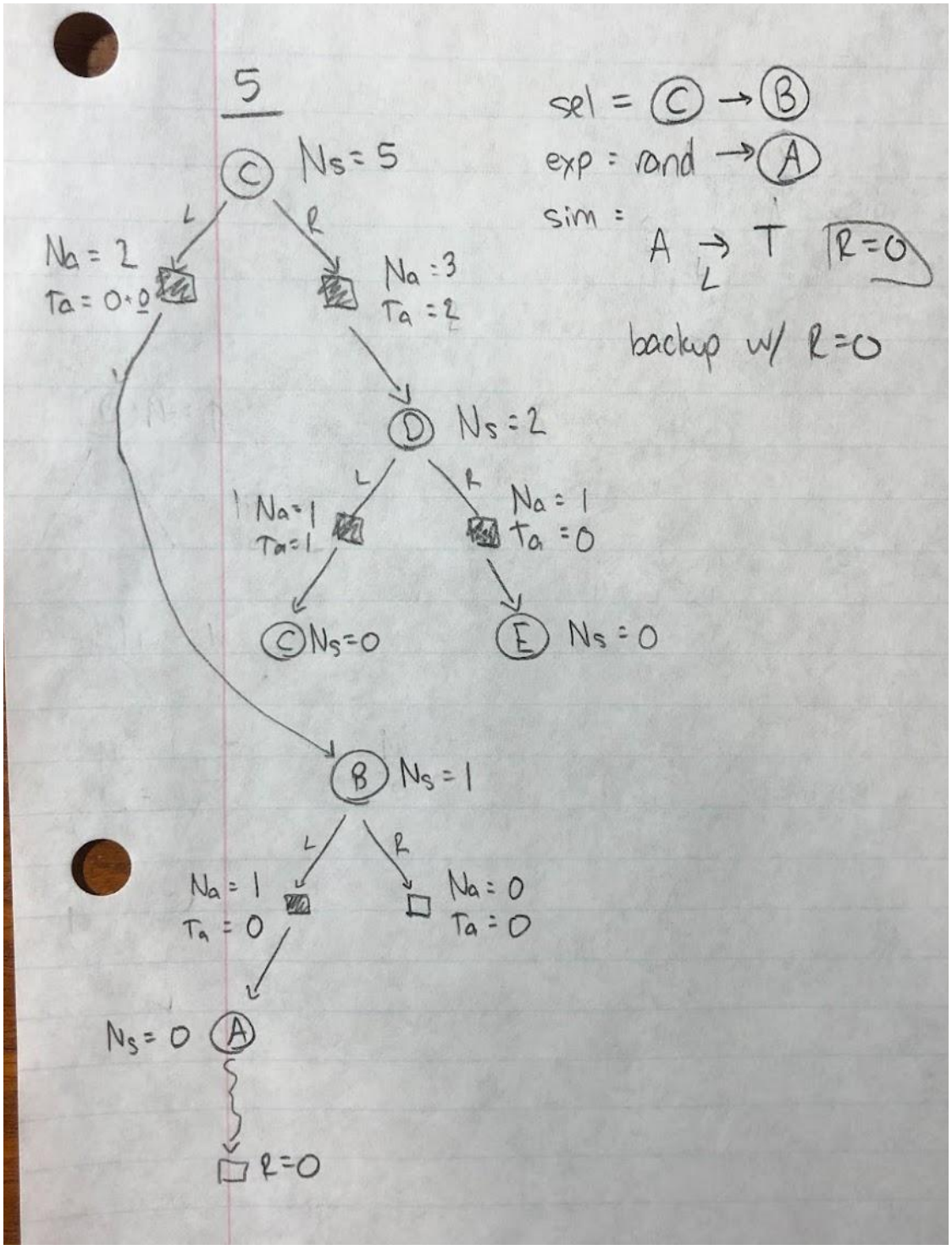


5.) Final 5 iterations (3rd iteration included):

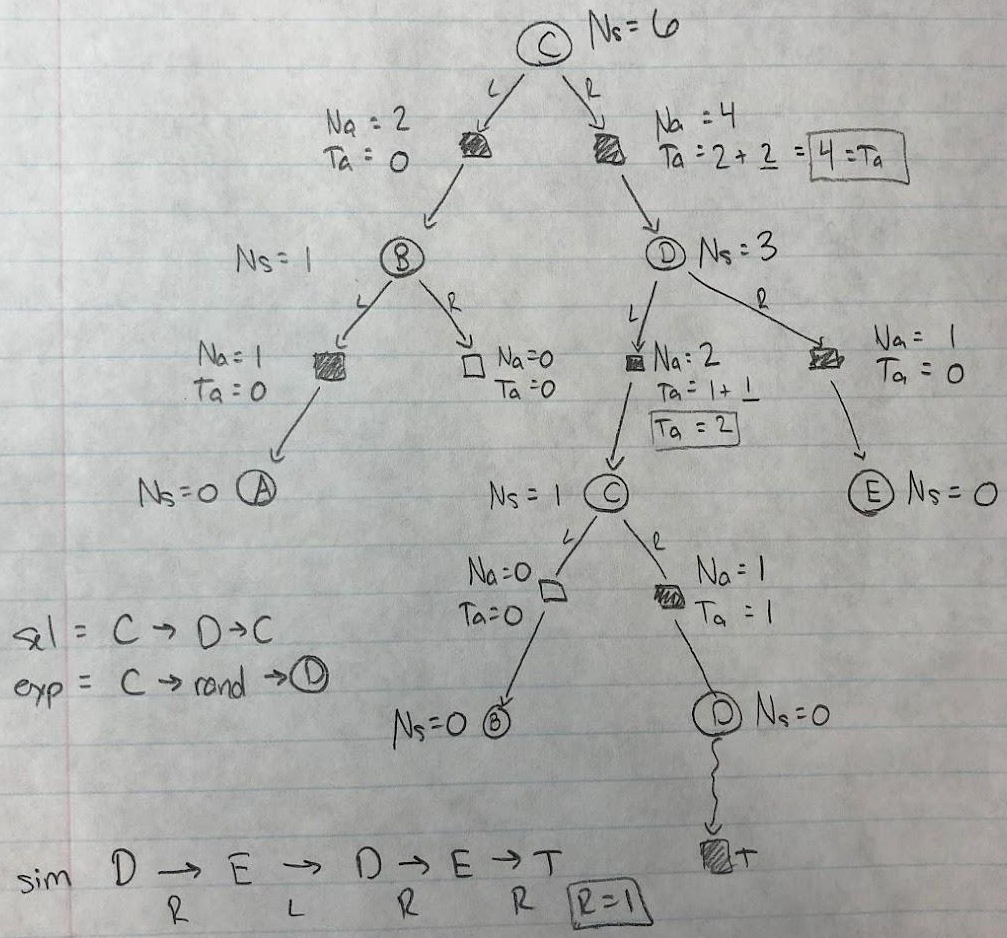
Code I used to determine argmax of UCB:

```
def calc_UCB(Ta, Na, Ns):
   return Ta/Na + math.sqrt(2*math.log(Ns)/Na)

Ns          = 4
Na_left     = 3
Ta_left     = 2
Na_right    = 1
Ta_right    = 0

UCB_left = calc_UCB(Ta_left, Na_left, Ns)
UCB_right = calc_UCB(Ta_right, Na_right, Ns)

UCBs = [UCB_left, UCB_right]
choice = np.argmax(UCBs)

if UCB_left == UCB_right:
   print("Tie! Choose randomly.")
else:
   if choice == 0:
       print("Left")
   else:
       print("Right")
```


3
C Ns=2
Na=1 Ta=0
Na=1 Ta=1
D Ns=0
C Ns=3
Na=1 Ta=0
Na=2 Ta=1+1=2
backup → Ta=1
B Na=0
D Ns=1
Na=1 Ta=1
Na=0 Ta=0
4
new
C Ns=4
Na=1 Ta=0
Na=3 Ta=2+0=2
B Ns=0
D Ns=2
Na=1 Ta=1
Na=1 Ta=0
Ns=0 C
E Ns=0
R=0
selection = R D
exp = random = L → C
b/c no info avail
sim from
C → B → A → B → A → B → C → D
L L R L R R R
→ E → Term
R R
+1
sel = C → D
exp = R → E
sim =
E → D → C → D → C → D → E → D → C
L L R L R R L L
→ B → C → D → C → B → C → D → C
L R R L L R R L
→ B → A → T R=0
L L L
backup =


5
sel = C → B
exp = rand → A
sim =
A → T R=0
L
backup w/ R=0
C Ns=5
L
R
Na=2
Ta=0+0
Na=3
Ta=2
D Ns=2
L
R
Na=1
Ta=1
Na=1
Ta=0
C Ns=0
E Ns=0
B Ns=1
L
R
Na=1
Ta=0
Na=0
Ta=0
Ns=0 A
R=0

(6) sel : C → D → C



sel = C → D → C

exp = C → rand → D

sim D → E → D → E → T
R L R R R=1

backup shown above

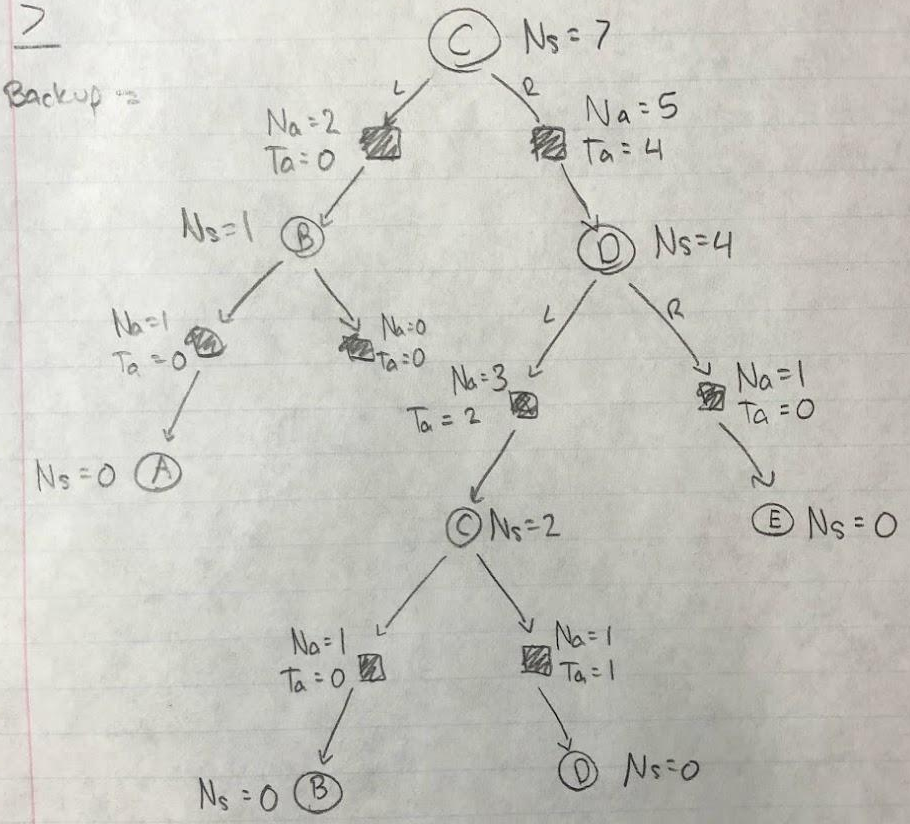
sel = C → D → C B

exp = C → B

sim : B → A → B → C → B → E → B → A → T R:0
L R R L R L L L

≥

Backup =



8

selection: C → D → E

exp: E → rand → T $\boxed{R+1}$

sim: not needed

backup: