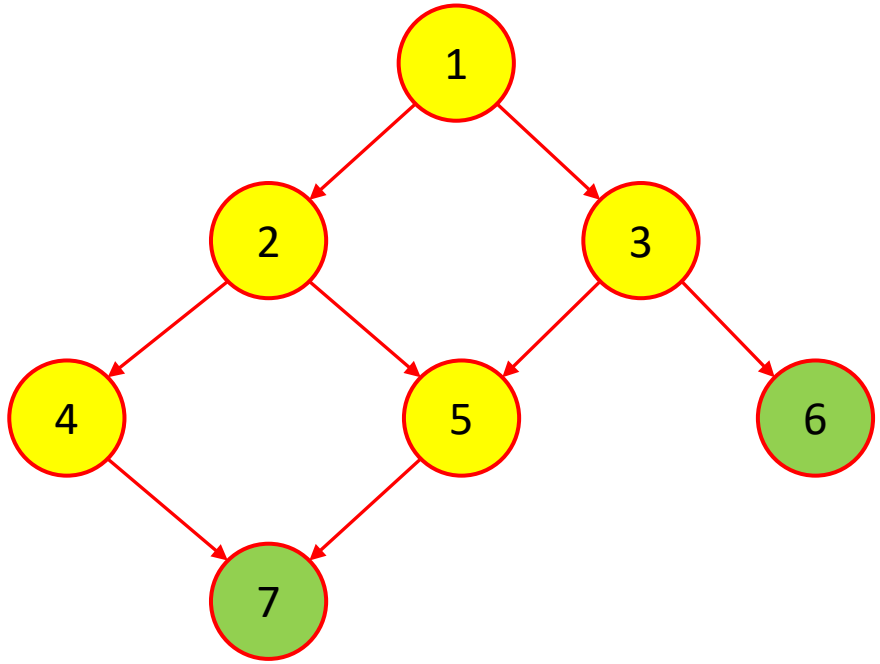
Question

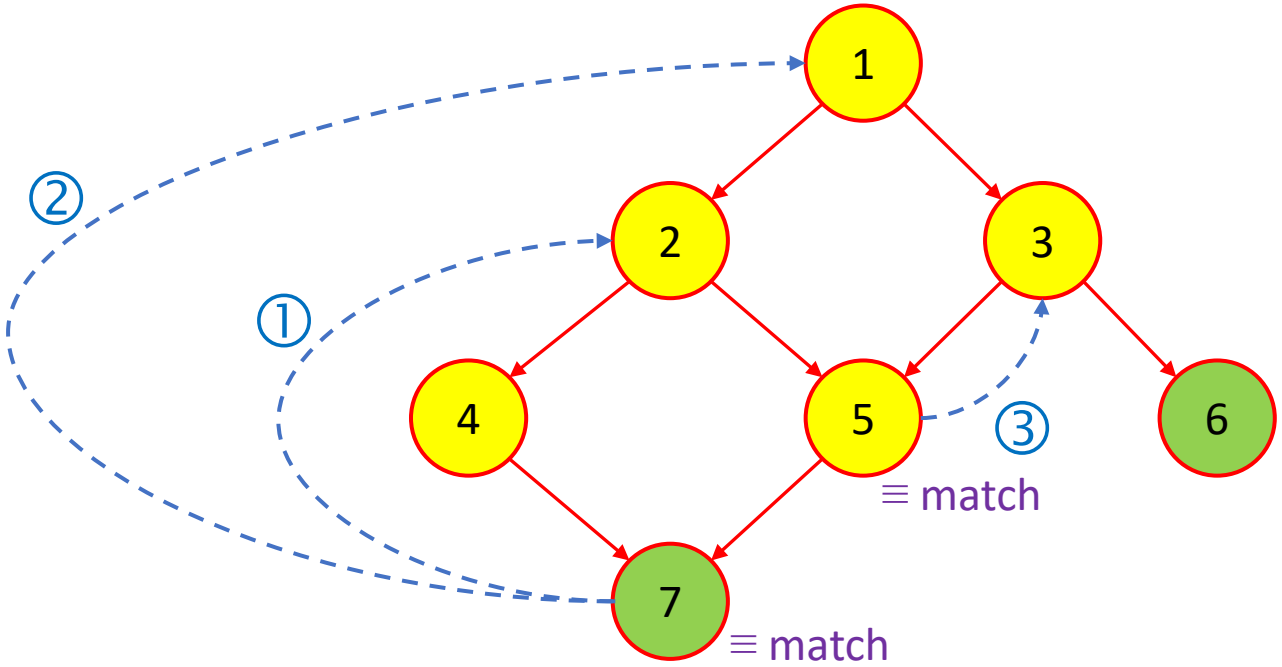
Question> Given the following state transition diagram, there are three backtracking arrows, when doing a depth-first-search (DFS) state space exploration. Fill in the blank start and end states in the table.



Backtracking Arrows	Start State	End State
First arrow		
Second arrow		
Third arrow		

Answer

Answer> States are visited in the order shown. When there is a match or termination, model checker backtracks to nearest transition pending exploration. In this way, each unique state is visited only once in the exploration.



Backtracking Arrows	Start State	End State
First arrow	7	2
Second arrow	7	1
Third arrow	5	3

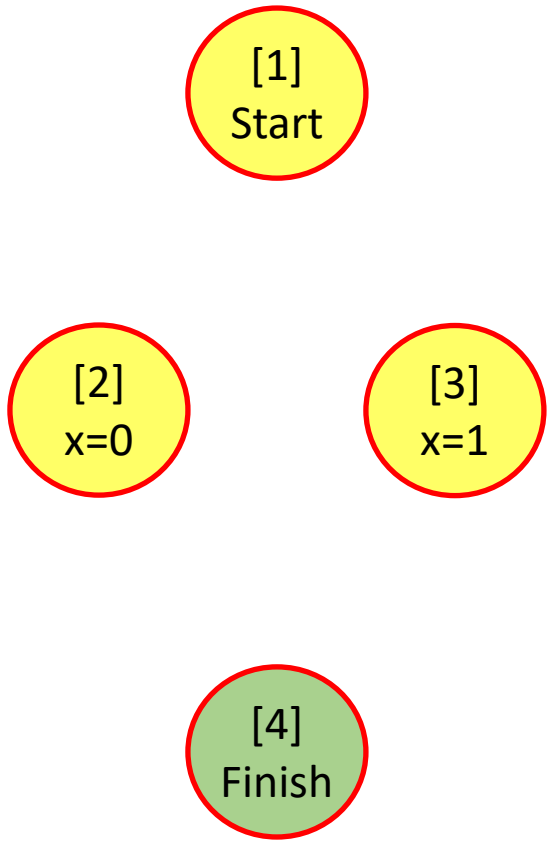
With matching & backtracking: 1 → 2 → 4 → 7 → 5 → 3 → 6

Without matching & backtracking: 1 → 2 → 4 → 7 → 1 → 2 → 5 → 7 → 1 → 3 → 5 → 7 → 1 → 3 → 6

Question

Question> Given the following code, there are four states in the transition diagram: 1, 2, 3, 4. Draw the transition edges yourself (including the backtracking edges) and then fill in the backtracking arrow table.

```
while(true) {
   x = rand.nextInt(2);
   if(x == 0) break;
}
```

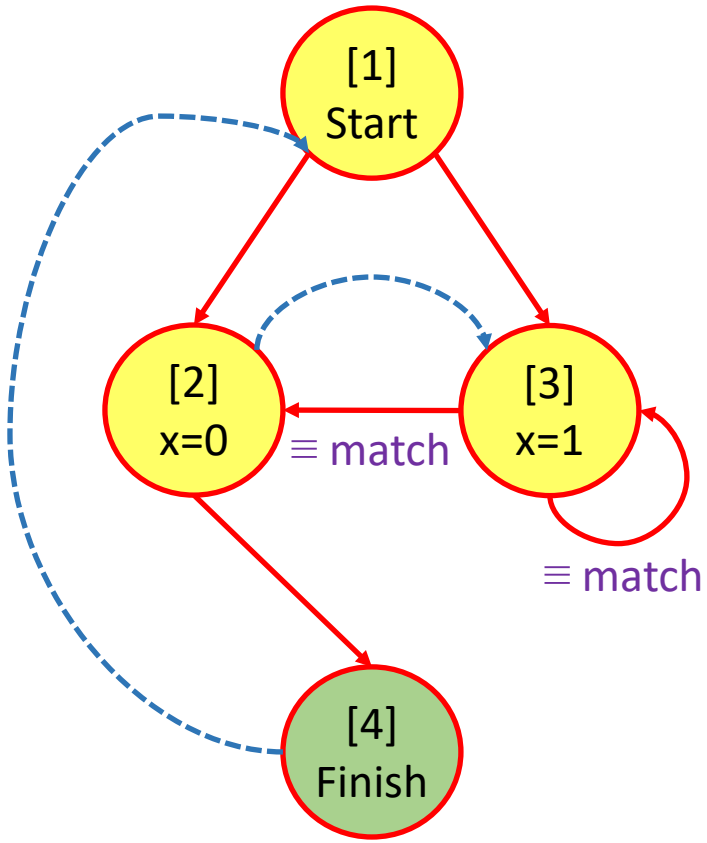


Backtracking Arrows	Start State	End State
First arrow		
Second arrow		
Third arrow		

Answer

Answer> Note that without matching & backtracking, the state space will expand infinitely due to the duplication of [2] and [3] states.

```
while(true) {
   x = rand.nextInt(2);
   if(x == 0) break;
}
```



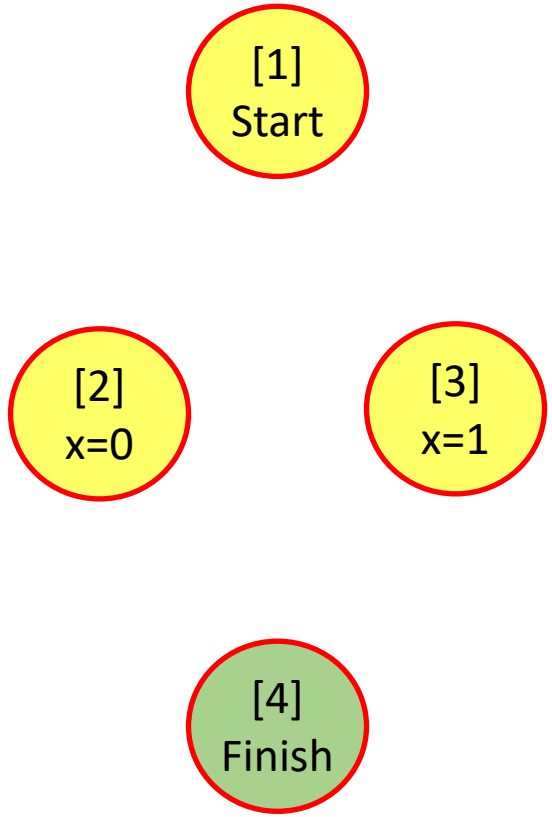
Backtracking Arrows	Start State	End State
First arrow	4	1
Second arrow	2	3
Third arrow	N/A	N/A

Order of visitation: 1 → 2 → 4 → 3

Question

Question> Given the following code, there are four states in the transition diagram: 1, 2, 3, 4. Draw the transition edges yourself (including the backtracking edges) and then fill in the backtracking arrow table.

```
while(true) {
   x = rand.nextInt(2);
   if(x == 1) break;
}
```

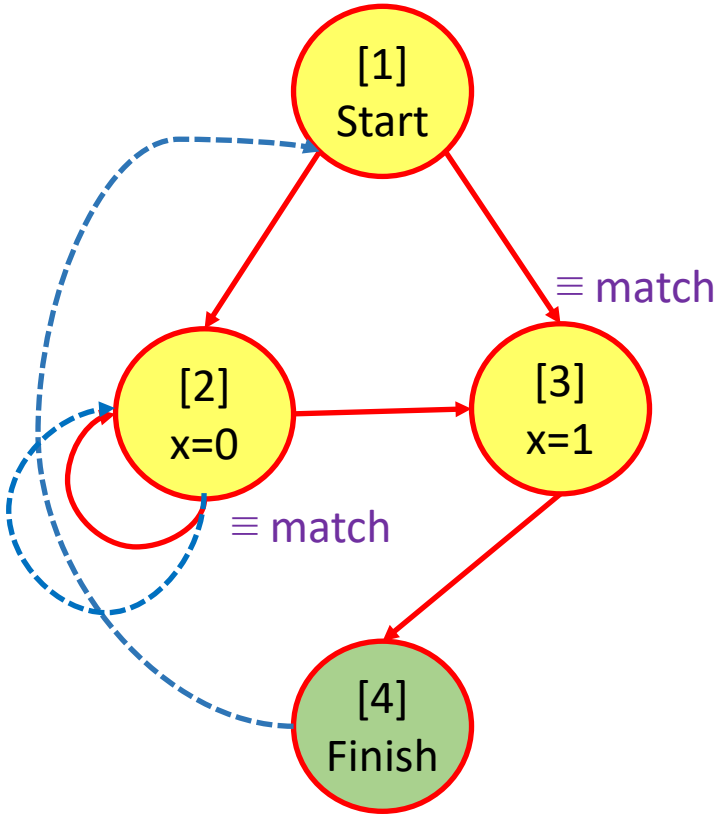


Backtracking Arrows	Start State	End State
First arrow		
Second arrow		
Third arrow		

Answer

Answer> Again, without matching & backtracking, the state space will expand infinitely due to duplication of [2] and [3] states.

```
while(true) {
   x = rand.nextInt(2);
   if(x == 1) break;
}
```



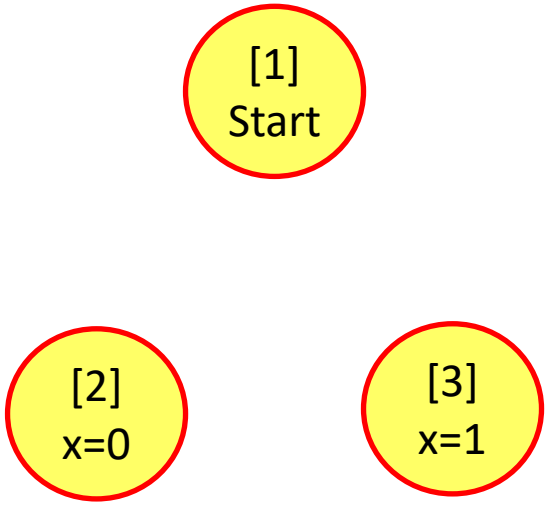
Backtracking Arrows	Start State	End State
First arrow	2	2
Second arrow	4	1
Third arrow	N/A	N/A

Order of visitation: 1 → 2 → 3 → 4

Question

Question> Given the following code, there are three states in the transition diagram: 1, 2, 3. Draw the transition edges yourself (including the backtracking edges) and then fill in the backtracking arrow table.

```
while(true) {
  x = rand.nextInt(2);
}
```

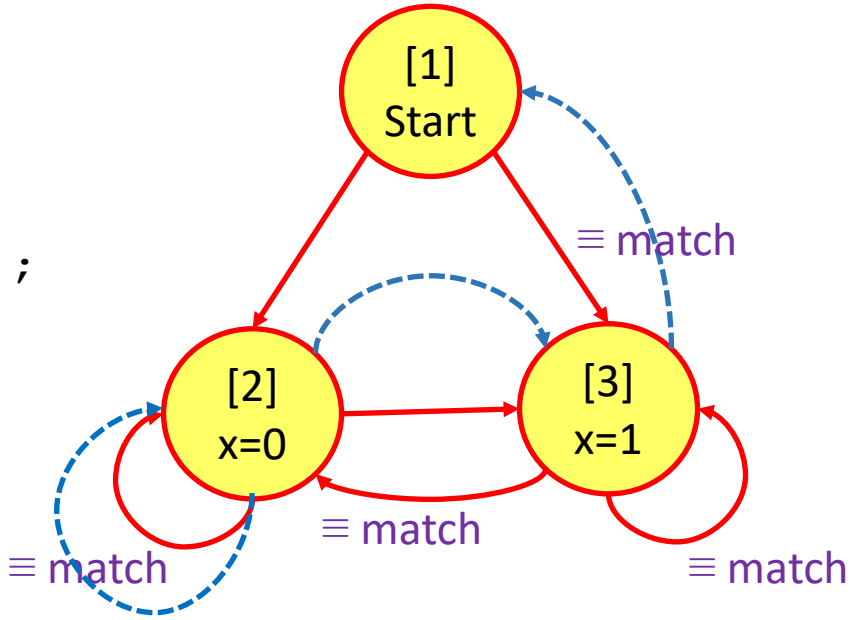


Backtracking Arrows	Start State	End State
First arrow		
Second arrow		
Third arrow		

Answer

Answer> Again, without matching & backtracking, the state space will expand infinitely due to duplication of [2] and [3] states.

```
while(true) {
   x = rand.nextInt(2);
}
```



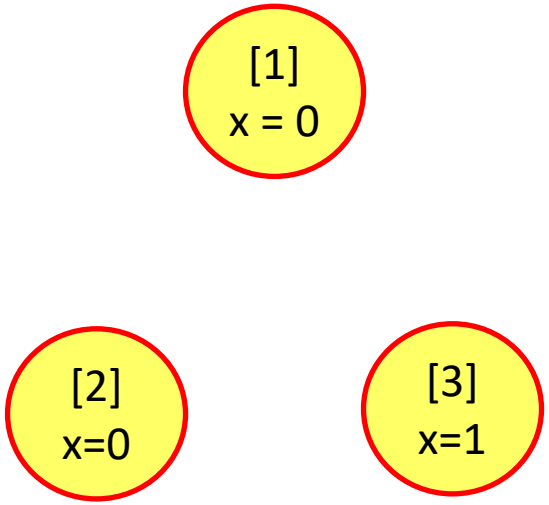
Backtracking Arrows	Start State	End State
First arrow	2	2
Second arrow	2	3
Third arrow	3	1

Order of visitation: 1 → 2 → 3

Question

Question> Given the following code, there are three states in the transition diagram: 1, 2, 3. Draw the transition edges yourself (including the backtracking edges) and then fill in the backtracking arrow table.

```
x = 0;
while(true) {
  x = (x + 1) % 2;
}
```

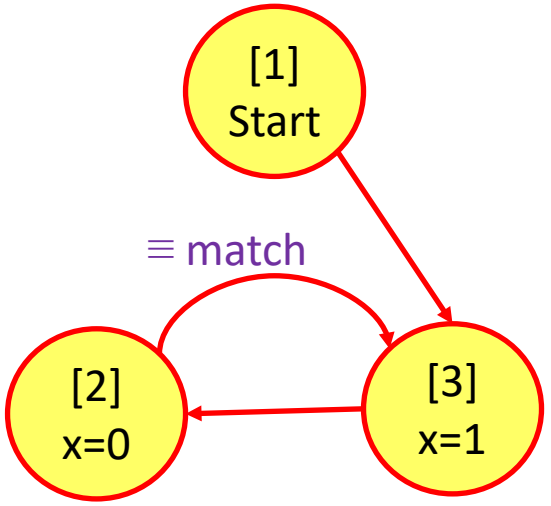


Backtracking Arrows	Start State	End State
First arrow		
Second arrow		
Third arrow		

Answer

Answer> Since this is a deterministic program, there can be no non-deterministic transitions, and hence no pending transitions to backtrack on.

```
x = 0;
while(true) {
  x = (x + 1) % 2;
}
```



Backtracking Arrows	Start State	End State
First arrow	N/A	N/A
Second arrow	N/A	N/A
Third arrow	N/A	N/A

Order of visitation: 1 → 3 → 2