Hanoi Algorithm Design & Complexity Analysis
by Carlos Flores

Implement an algorithm and program it to solve a Tower of Hanoi puzzle for the following graph $\mathcal{G}$ =($\mathcal{V}$,$\mathcal{E}$) with the Vertices, $\mathcal{V}$ = {Start, Aux 1, Aux 2, Aux 3, and Dest}, and Directed Edges $\mathcal{E}$ = {(Start, Aux1), (Aux1, Aux2), (Aux2, Aux3), (Aux3, Aux1), (Aux3, Dest)}.

The last edge $(Aux3, Dest)$ will be referred to as the Final Edge, the first edge $(Start, Aux1)$, will be referred to as the Begin Edge, while the rest of the edges in $\mathcal{G}$ will be considered the Next Edges. The motivation for this distinction is to identify the edges used to liberate the largest disc (the Next edges), to identify the specific edge that will be used to move discs into the cycle made by the Next Edges (the Begin Edge), and to identify the specific edge that will be used to park the largest disc not on the Dest vertex (the Final Edge).

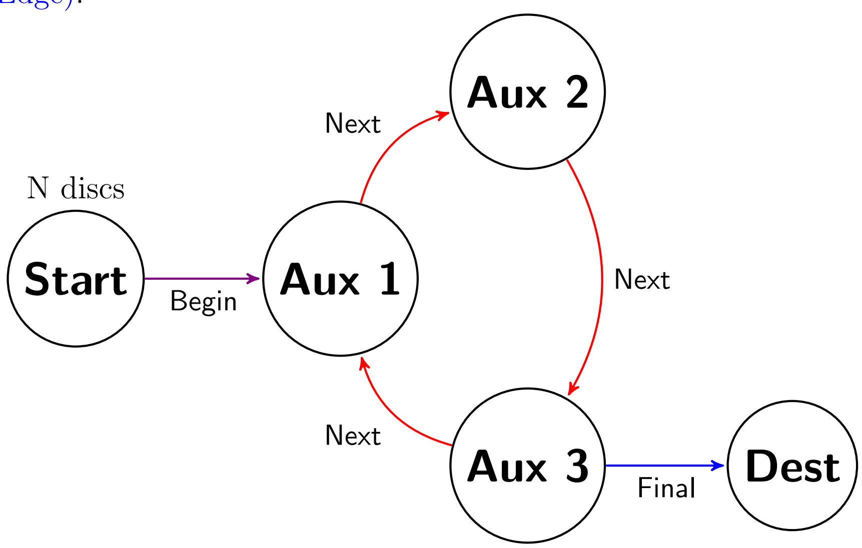
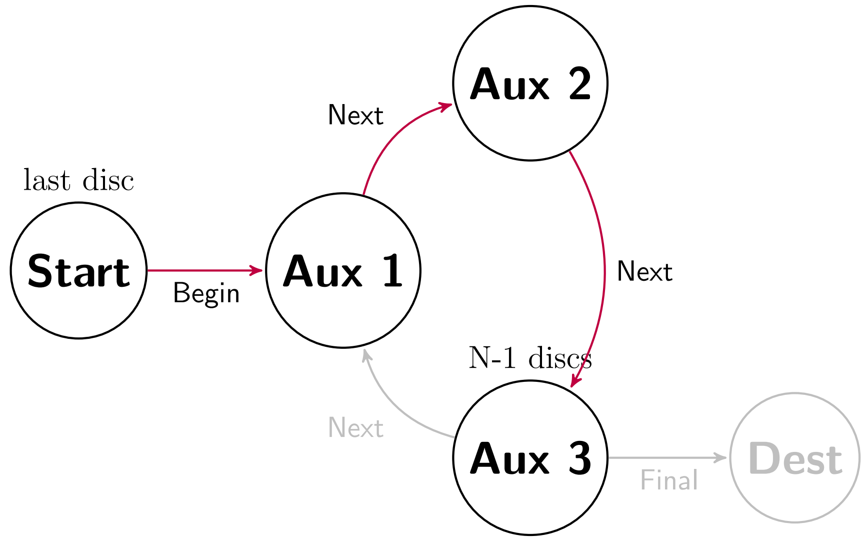
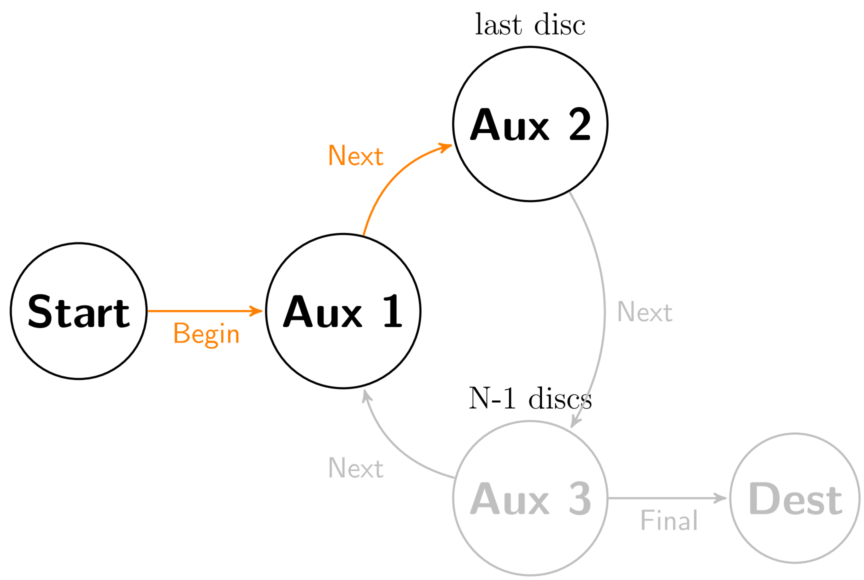


Figure 1.a Tower of Hanoi puzzle for $\mathcal{G}$ =($\mathcal{V}$,$\mathcal{E}$).

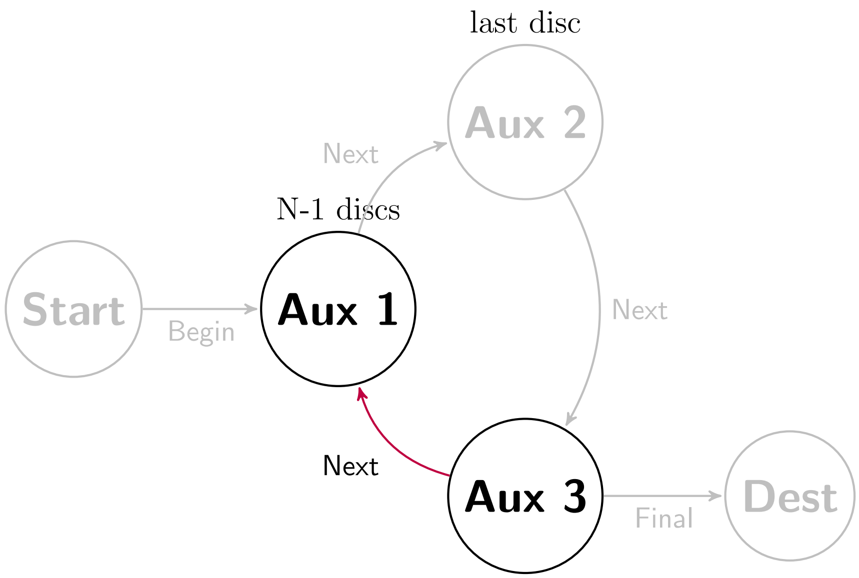
We can clearly see that the cycle made by the Next edges can be used to move discs around the graph in order to solve the puzzle.
Our goal is to move a stack of N number of discs from Start to Dest. In order to move N number of discs from Start to Dest we must move (N-1) discs to access the last disc. This can be achieved by the following recursive pattern:



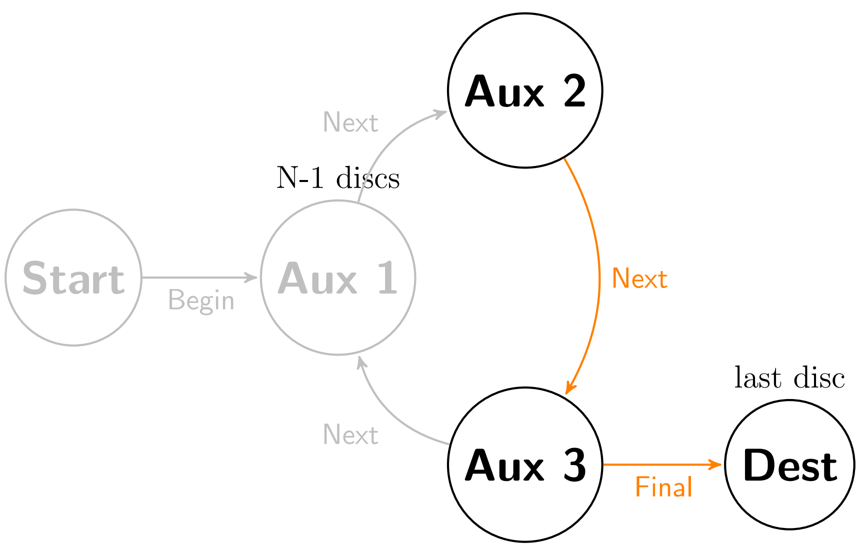
(1) *Figure 1.b* Move (N-1) discs from Start to Aux 3.
This will take T(N-1) operations.



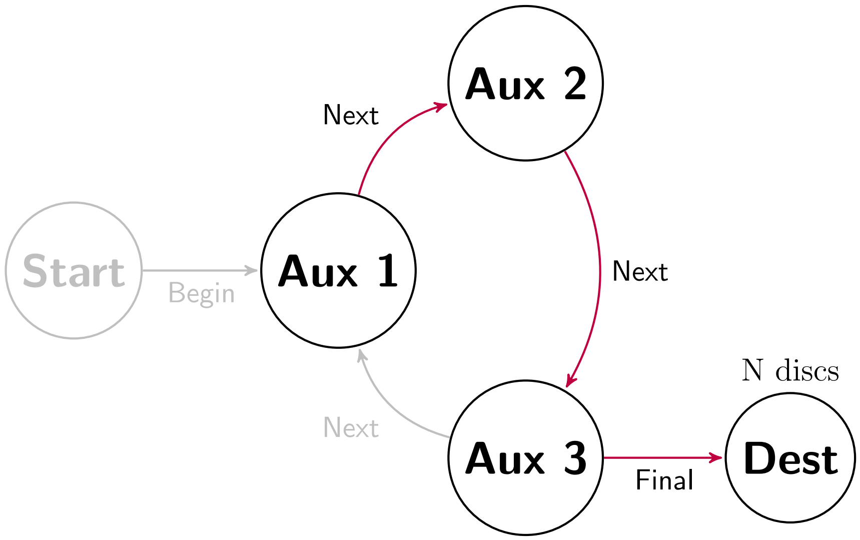
(2) *Figure 1.c* Move the last disc from Start to Aux 2.
This will take a T(1) operation.



(3) *Figure 1.d* Move (N-1) discs from Aux 3 to Aux 1.
This will take T(N-1) operations.



(4) *Figure 1.e* Move 1 disc from Aux 2 to Dest.
This will take a T(1) operation.



(5) *Figure 1.f* Move (N-1) discs from Aux 1 to Dest.
This will take T(N-1) operations.

Now all of the discs have been moved from Start to Dest. We moved the stack of (N-1) discs three times and the original size of the input was decremented by one. The sum of all operations give us the recurrence relation:

$$\{T(N) = 3T(N-1) + 2(1) | R = 3, D = 1\}$$

Therefore, according to the Master Theorem of Reduction by Subtraction:

$$\text{T(N)} = \theta(3^{(N/1)}) \text{ which simplifies to:}$$

$$\boxed{T_{time}(N) = \theta(3^N)} \tag{1}$$

Additionally, the space required to hold the data within memory will remain constant since all recursive invocations free up memory that was used when they terminate. Therefore the space complexity is:

$$\boxed{T_{space}(N) = \theta(N)} \tag{2}$$