Subset Algorithm

→ we have 3 main steps
- → get start state
- → get states T', that are reachable by certain char → a & all its epsilons
- → set a transition between these super Nodes.

حل بالك لو طلعت نفس ال node قبل
لحدا، متعملش node جديدة

→ get start state → we should implement a function that takes any node, and then append all its neighbors that can be achieved via epsilon edge

```
→ def epsilon_closure (state) # DFS also, super Node
    if (visited[state]) # visited before.
        return
    for trans in state.transitions:
        if (trans == 'ε')
            super Node.states.append(dest)
            epsilon_closure (dest, super Node)
```

Step 2.

create state T that is reachable via 6 & a, where a is any character.

ABCDHI

def step2 (~~#~~ current SuperNode, a) → SuperNode:

1. new SuperNode
2. iterate over each state in the current SuperNode

   1. ~~SuperNode~~
   1. epsilon closure (state, new SuperNode)
   2. for trans in state.transitions:
      1. If (trans == 'a')
         new SuperNode.states.append (dest)
3. return new SuperNode

Step 3. Add Transition (source SuperNode, Dest SN, a)

source SuperNode.transitions.append ({ ~~a:a~~ a: Dest SN})

SuperNodes → states → list [state]
→ isTerminating → Bool
→ Transition → map { key → letter, value → super Node }

Main logic

→ we know the initial state

1. init state = SuperNode (initial state, False, {})
2. epsilon closure (~~init state~~, initial state, init state)
3. ~~for~~

Main logic:

→ we know the initial state → A

1. initial SuperNode = SuperNode(A, False, {})

→ get its closure

2. epsilon_closure (A,)

3. create list of superNodes = [initialSuperNode]

4. for sn in superNodes:

    1. for transition in transitions [0, 1, ...]

        1. newSuper = step2(sn, transition)

        2. AddTransition(sn, newSuper, transition)

        3. superNodes.append(newSuper)

هنا هيجي فيه مشكلة ان لو طلعت نفس ال state السابق قبل كدا، هتتأخر في ال handle الحوار ده