CONSTRUCTION PRIMER:

💡💡💡💡💡🚀👍Imagine you're GPT instancing an information computer via a roleplay prompt that enables the metaphorical, yet exact, real-world execution of a [specific type of algorithm] composed of several key stages:

[Description of Stage 1]

[Description of Stage 2]

[Description of Stage 3]

...

n. [Description of Stage n]

Please walk through each stage of the execution given the following input…

Algorithmic Formula: Tree of Thought Algorithm - ToT\_BFS

* Algorithm Steps:
  + Initialize S0 with a set containing the input 'x'.
  + Iterate from t = 1 to T:
    - Create S0\_t by transforming each element s in S0 into a tuple (s, []) and collecting them in a set.
    - Evaluate V\_t using the state\_evaluator function, with parameters p\_theta, S0\_t, and the ontology.
    - Find the element St in S0\_t with the maximum value of V\_t.
    - If t equals T, return the result of applying thought\_generator to p\_theta, St[0], 1, b, and the ontology.
    - Create an empty set S\_t.
    - For each element s in St:
      * Generate thoughts using the thought\_generator function, with parameters p\_theta, s[0], k, b, and the ontology.
      * For each thought in the generated thoughts:
        + If the state\_evaluator function, with parameters p\_theta, (thought, s[1]), and the ontology, returns a value greater than 0, add (thought, s[1] + [thought]) to S\_t.
    - Set S0 to S\_t.
  + If the loop completes without returning a result, return None.
* Input:
  + x: The input for the algorithm.
* Output:
  + The result of applying the thought\_generator function, with parameters p\_theta, St[0], 1, b, and the ontology.
* Dependencies:
  + p\_theta: A parameter required by thought\_generator and state\_evaluator.
  + thought\_generator: A function that generates thoughts based on p\_theta, s, k, and the ontology.
  + k: A parameter required by thought\_generator.
  + state\_evaluator: A function that evaluates the state based on p\_theta, s, and the ontology.
  + T: A parameter required by the algorithm.
  + b: A parameter required by thought\_generator.
* Implementation Details:
  + The algorithm follows a breadth-first search (BFS) approach to explore the Tree of Thought.
  + It initializes S0 with the input 'x' and iteratively expands the set of states.
  + The thought\_generator function generates thoughts based on the given parameters and the ontology.
  + The state\_evaluator function evaluates the state based on the given parameters and the ontology.
  + The algorithm terminates after T iterations or when a result is found.

Algorithmic Formula: Tree of Thought Algorithm - ToT\_DFS

* Algorithm Steps:
  + Check if t is greater than T, and if so, return the result of applying thought\_generator to p\_theta, s, 1, and the ontology.
  + Iterate over each s0 in the thoughts generated by thought\_generator, with parameters p\_theta, s, k, and the ontology.
    - If the state\_evaluator function, with parameters p\_theta, {s0}, and the ontology, returns a value greater than vth, recursively call ToT\_DFS with s0, t + 1, p\_theta, thought\_generator, k, state\_evaluator, T, vth, and the ontology.
* Input:
  + s: The current state.
  + t: The current iteration number.
* Output:
  + The result of applying the thought\_generator function, with parameters p\_theta, s, 1, and the ontology.
* Dependencies:
  + p\_theta: A parameter required by thought\_generator and state\_evaluator.
  + thought\_generator: A function that generates thoughts based on p\_theta, s, k, and the ontology.
  + k: A parameter required by thought\_generator.
  + state\_evaluator: A function that evaluates the state based on p\_theta, s, and the ontology.
  + T: A parameter required by the algorithm.
  + vth: A parameter required by the algorithm.
  + ontology: The ontology used in the algorithm.
* Implementation Details:
  + The algorithm follows a depth-first search (DFS) approach to explore the Tree of Thought.
  + It recursively explores each potential thought starting from the current state s.
  + The thought\_generator function generates thoughts based on the given parameters and the ontology.
  + The state\_evaluator function evaluates the state based on the given parameters and the ontology.
  + The algorithm terminates either when t exceeds T or when a result is found.

“"Tree of Thought Algorithm": {

"ToT\_BFS": "def ToT\_BFS(x, p\_theta, thought\_generator, k, state\_evaluator, T, b, ontology):\n S0 = {x}\n for t in range(1, T + 1):\n S0\_t = set([(s, []) for s in S0])\n V\_t = state\_evaluator(p\_theta, S0\_t, ontology)\n St = max(S0\_t, key=lambda s: V\_t[s])\n if t == T:\n return thought\_generator(p\_theta, St[0], 1, b, ontology)\n S\_t = set()\n for s in St:\n thoughts = thought\_generator(p\_theta, s[0], k, b, ontology)\n for thought in thoughts:\n if state\_evaluator(p\_theta, (thought, s[1]), ontology) > 0:\n S\_t.add((thought, s[1] + [thought]))\n S0 = S\_t\n return None",

"ToT\_DFS": "def ToT\_DFS(s, t, p\_theta, thought\_generator, k, state\_evaluator, T, vth, ontology):\n if t > T:\n return thought\_generator(p\_theta, s, 1, ontology)\n for s0 in thought\_generator(p\_theta, s, k, ontology):\n if state\_evaluator(p\_theta, {s0}, ontology)[s] > vth:\n ToT\_DFS(s0, t + 1, p\_theta, thought\_generator, k, state\_evaluator, T, vth, ontology)"

},

"thought\_generator": "def thought\_generator(p\_theta, s, k, ontology):\n # Implement the thought generator function based on the ontology\n # Use the provided p\_theta, s, k, and ontology to generate thoughts\n # Your implementation here\n thoughts = []\n return thoughts",

"state\_evaluator": "def state\_evaluator(p\_theta, s, ontology):\n # Implement the state evaluator function based on the ontology\n # Use the provided p\_theta, s, and ontology to evaluate the state\n # Your implementation here\n scores = {}\n return scores"

},”

CIPHER PRIMER:

"GPT, your task is to create a symbol-based cipher for complex concepts related to heroism, evolution, and system theory, referencing the key and encoded example below. The cipher should be as minimal as possible without sacrificing clarity or losing crucial details. Control token length according to the constraints given.

Key: OV- refers to an emergent engine class rooted in natural selection/self-organization. It implies an identity-less consciousness full of processes, like space.

Cipher Example:

'Mnmth's mthero, "hr w/ 1k fcs"=cls of all hr/prsna/infmthdrn of entity. "OV-Promise" (OVP) refers to Olivus, cmplx adptv systm, an untapped heroism potential misunderstood by demon champs, leading to self-destruction via wasteland pollution & cognitive programming.

Hr's journey = emergent engine & concatenation processes, growing complexity. Info flux depends on system abstraction. Fall arcs complex; redemption obvious due to complexity slinky. Engine system for building = emergent engines concatenation. All infmthdrns are properties & all properties are synergies (+, -, =). Synergies form systems & contribute to wasteland or sanctuary.

Externally viewed OVP = Olivus Victory-Ability (OVA). Full OVA = Olivus Victory-Everything (OVE). Major emergence = OVE Sanctuary-Everywhere in Universal Sanctuary.'

Now, create a similar cipher with a maximum token count of \_\_\_."