Name: Sen Wang
Student Id: 260923645

Question 1: Language Generation

We would like to design a simple language generation system which can generate sensible and grammatically correct sentences of English of up to 6 words long. The system is able to generate these 5 words: the, cat, sat, on, mat.

The sentences that we would want the system to generate are:
- the cat sat
- the cat sat on the mat

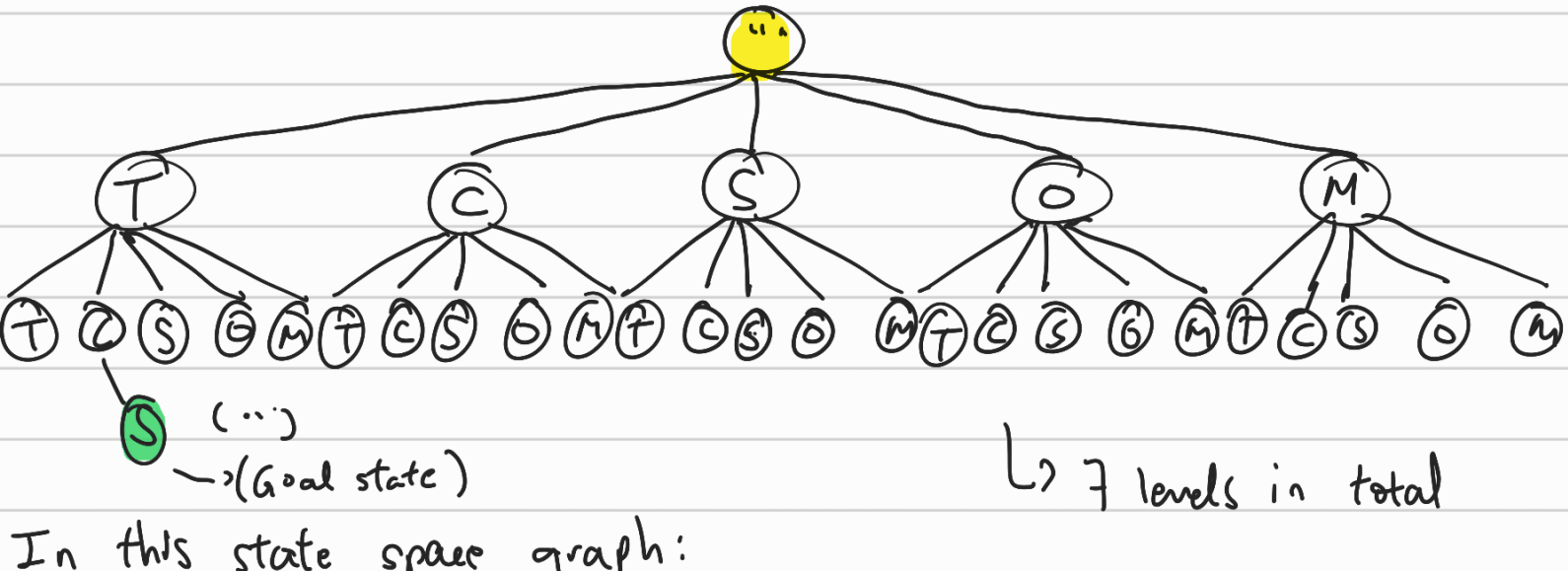
The system also incurs a cost for generating each word, equal to the number of letters the word contains.

a) Formulate the sentence generation process as a search problem, stating each of the parts of the search problem as shown in class.

- State space: All the possible configurations of sentences built using "the, cat, sat, on, mat" that are 6 words long or less.
- Initial state: Empty sentence
- Goal state: "The cat sat" and "the cat sat on the mat"
- Operators: Add a valid word to the current state if the length of the next sentence does not exceed 6.
- Path: A sequence of state and operators. In our case, a sentence represents the path.
- Path cost: the total number of letters in the sentence (path).
- Solution of search problem: Find a path from S_0 to $S_g \in G$
- Optimal solution: A solution that has the minimum number of letters.

b) Describe the state space graph of this problem. What does it look like? How many nodes are there in the search graph.

T = "the", C = "cat", S = "sat", O = "on", M = "mat"



↳ 7 levels in total

In this state space graph:
- Nodes correspond to states in S
- Edges correspond to operators (add a word)
- The initial state is represented by the yellow node.
- One possible goal state is represented by the green node ("the cat sat").

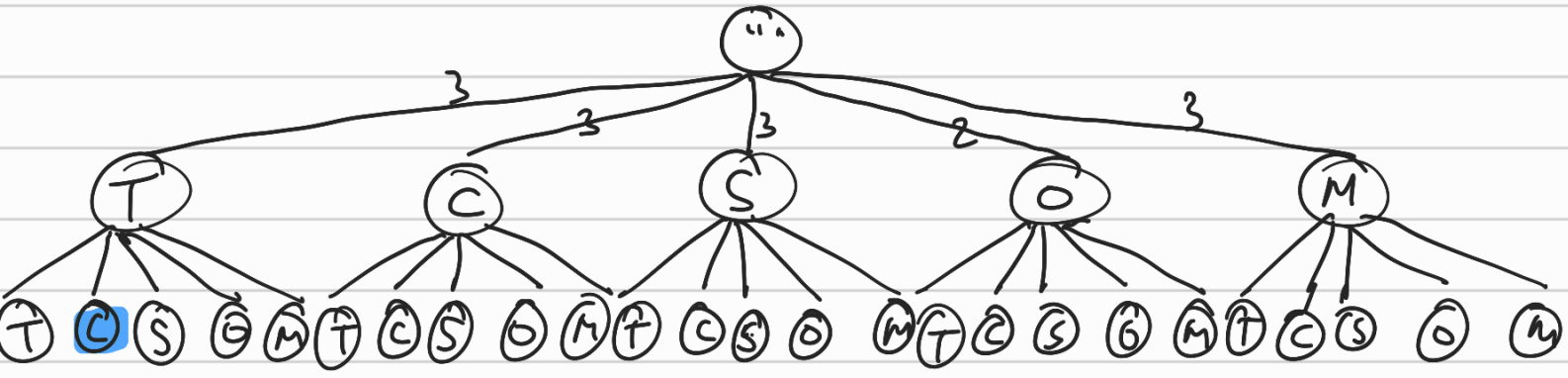
A search graph represents the exploration paths in the search procedure. We have a root node ("") which represent an empty sentence and there are 6 more levels in the tree (max of 6 words). Thus, there are $5^6 + 5^5 + 5^4 + 5^3 + 5^2 + 5^1 + 5^0$ nodes in total.

c) Draw the search tree down to a depth of 2 (ie, the tree has 3 levels in total, including the root.)

(see next page)

Search Tree

T = "the" , C = "cat" , S = "sat" , O = "on" , M = "mat"



- Every node represents partial solutions. For example, node C represents "the cat".
- Every edge corresponds to operators (append word) with a cost (length of the word)

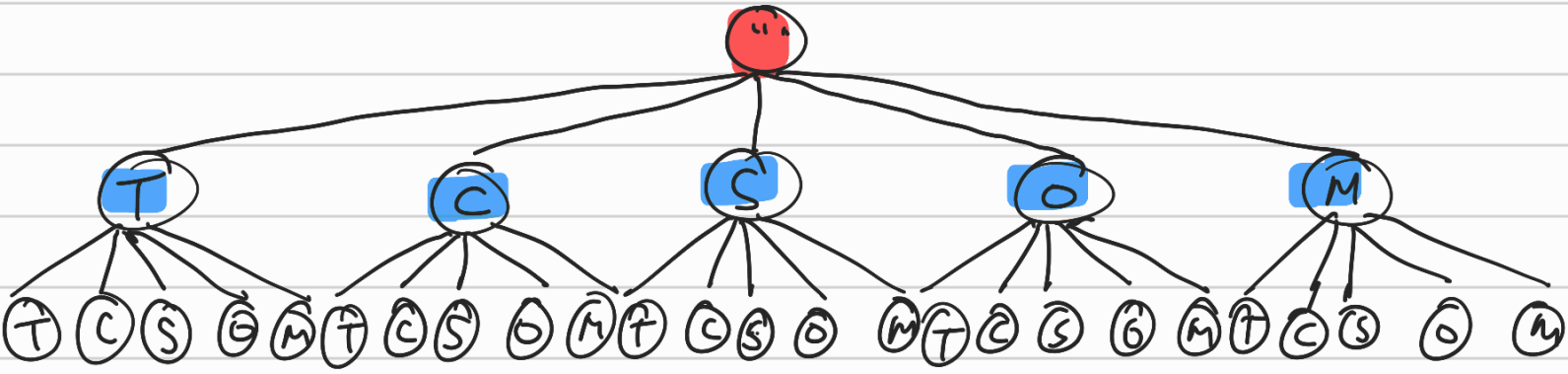
d) Trace the run of the search process using the following algorithms for up to 10 search steps. Given multiple states to explore that are otherwise equivalent in priority, the algorithm should prefer to generate the word that comes first alphabetically.

i) Breadth first search

: current ○ : unvisited

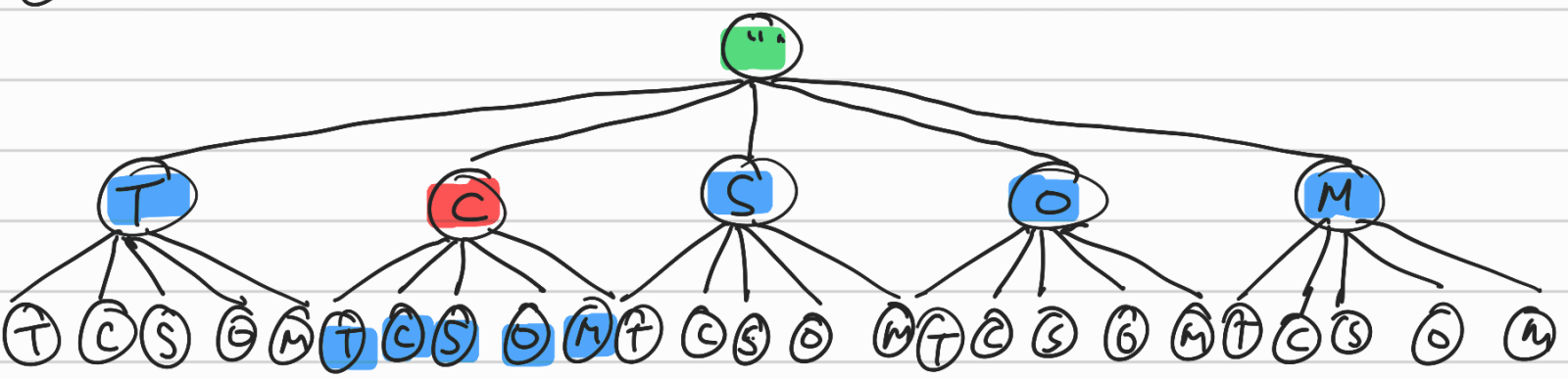
: visited : Frontier.

①

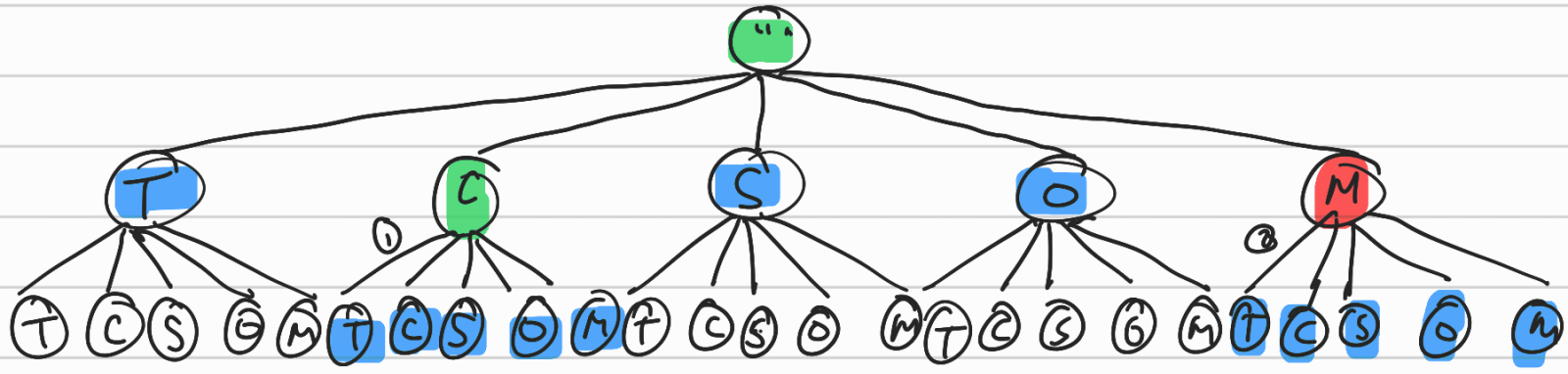


Priority Queue : [C, M, O, S, T]

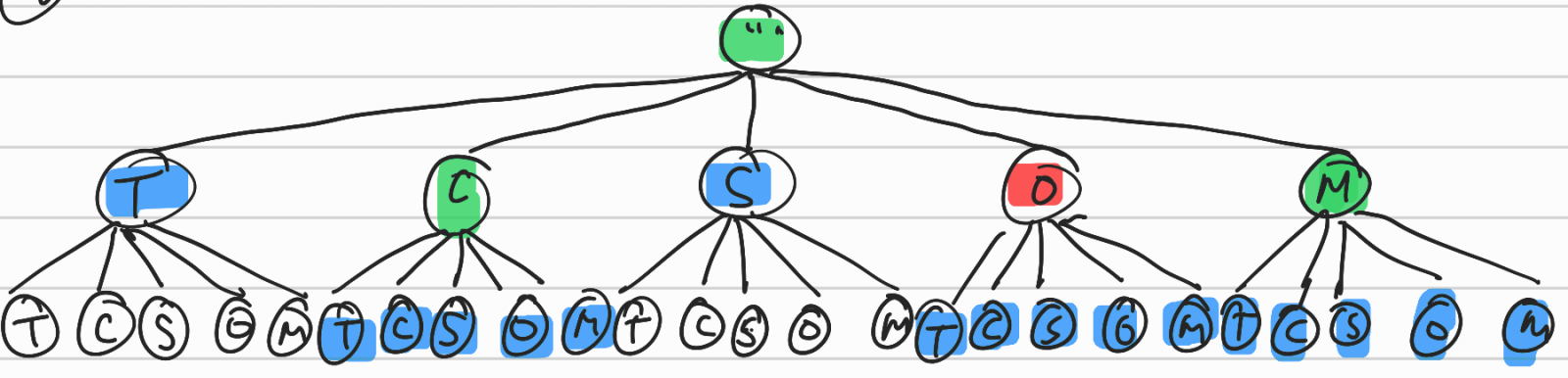
②



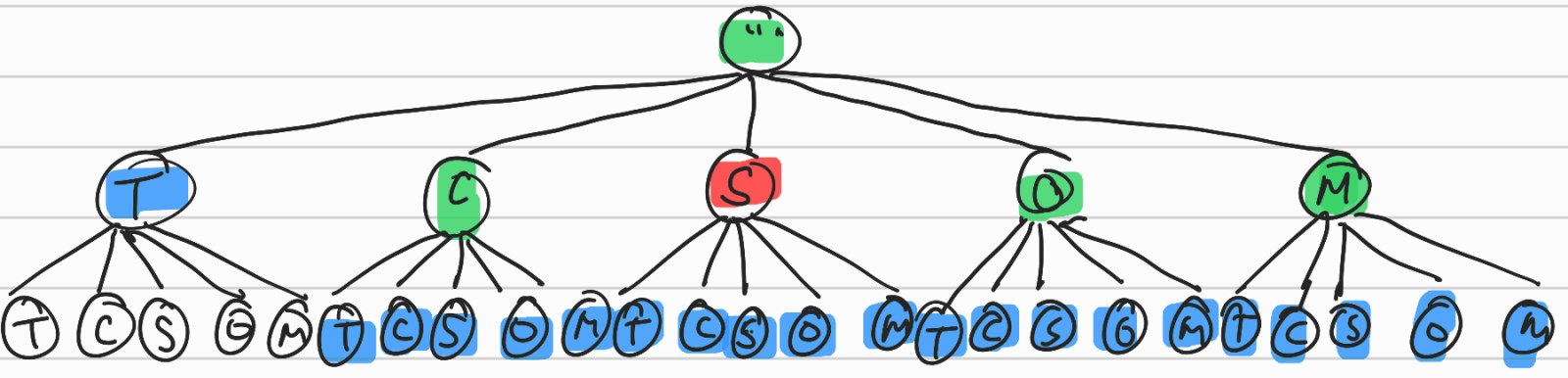
③



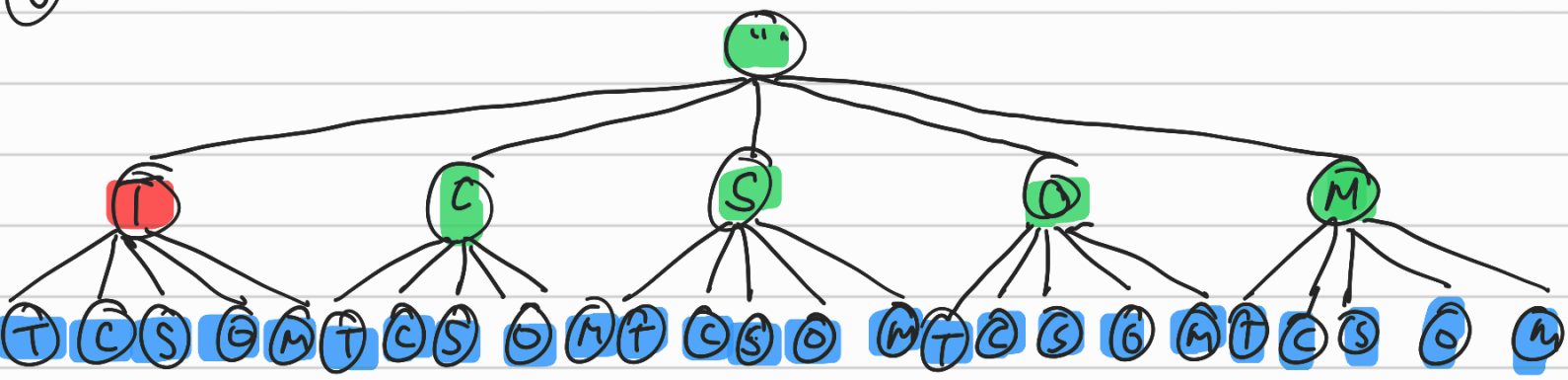
④



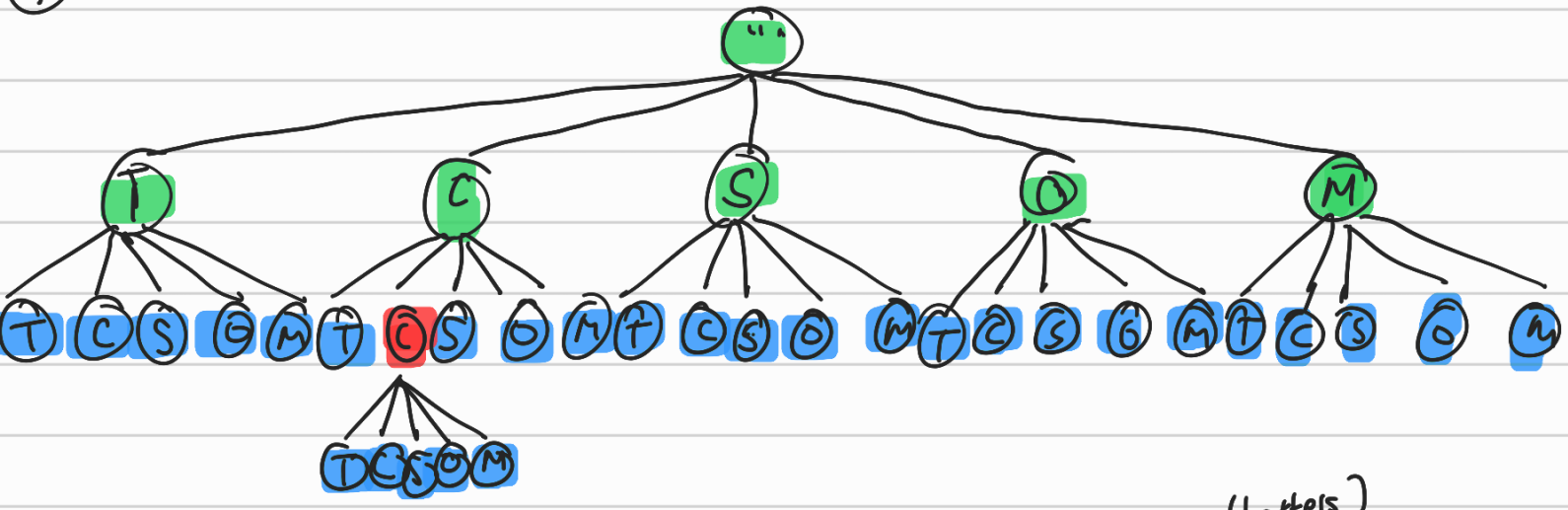
⑤



(6)



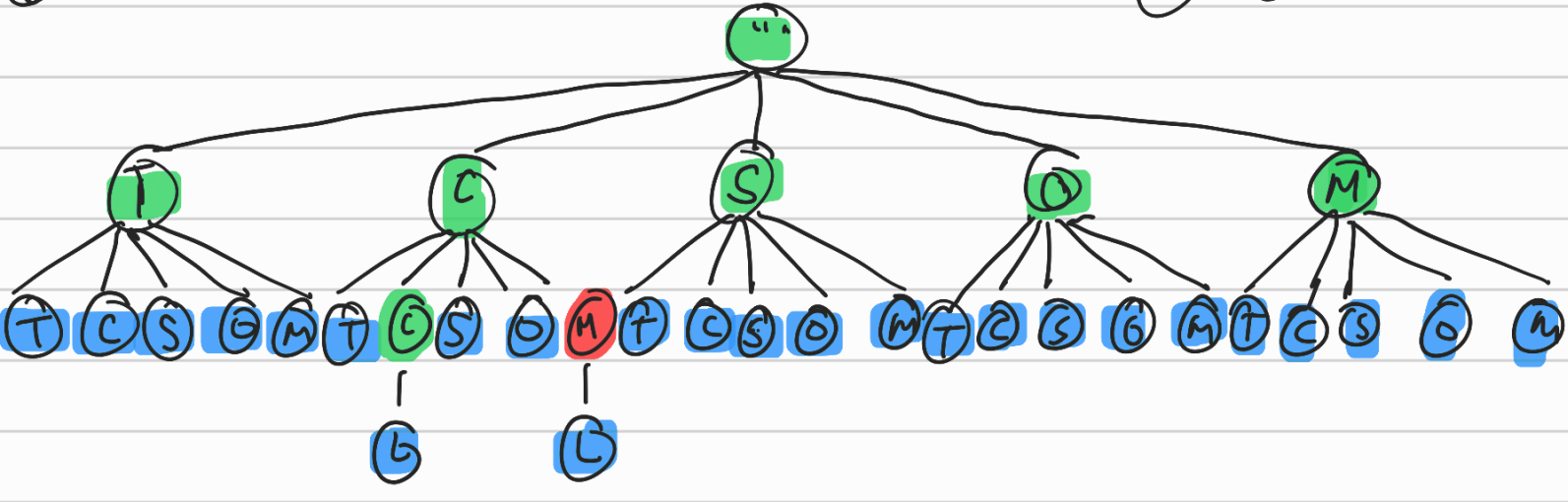
(7)



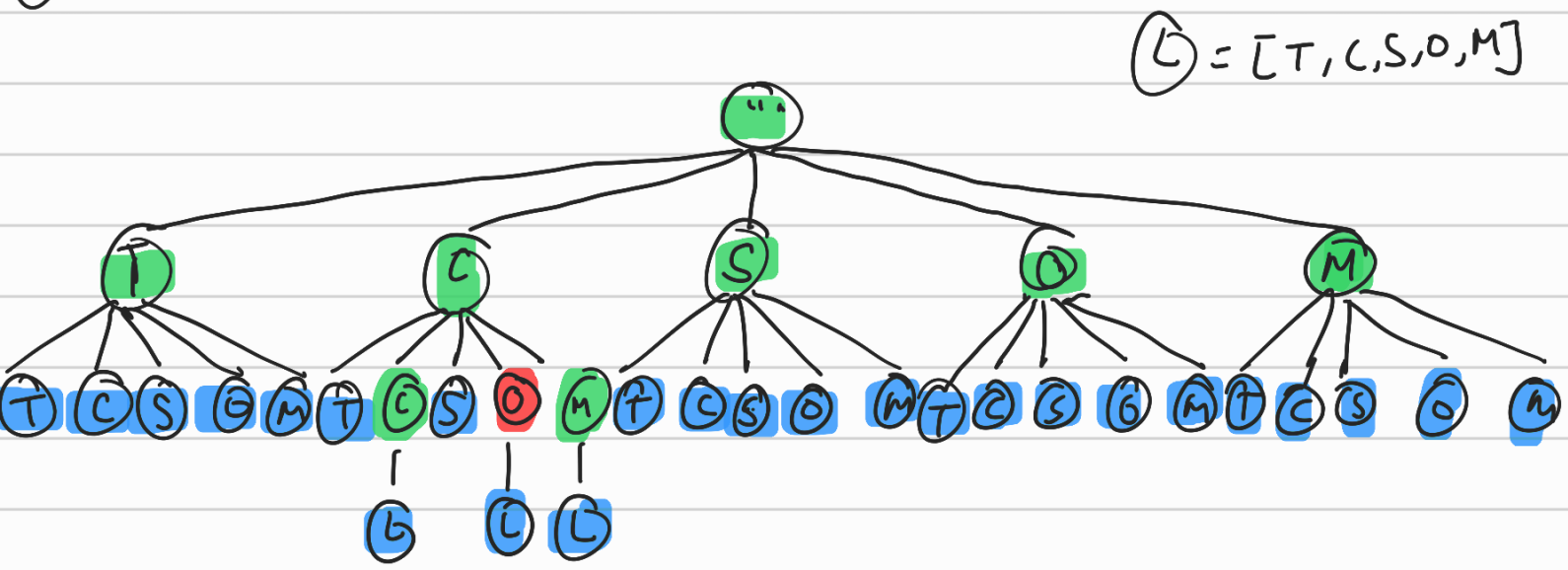
→ (letters)

(L) = [T, C, S, O, M]

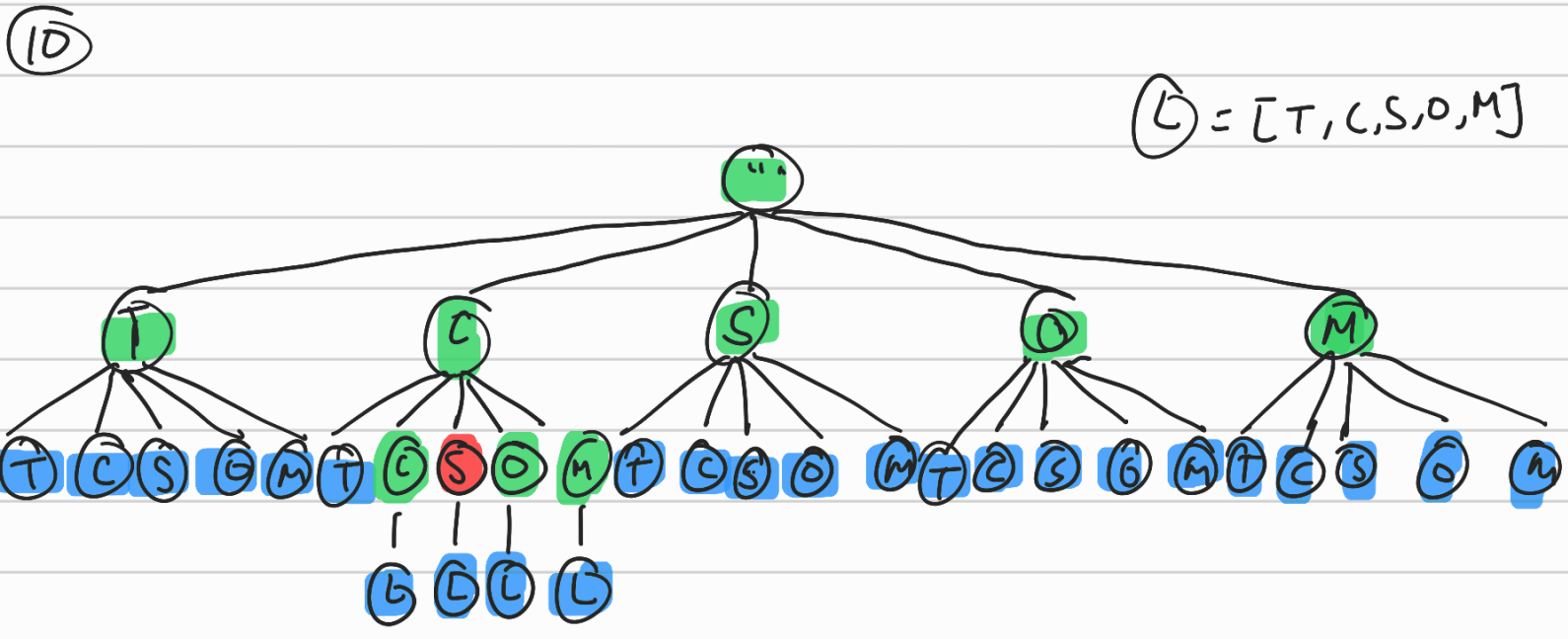
(8)



(c)

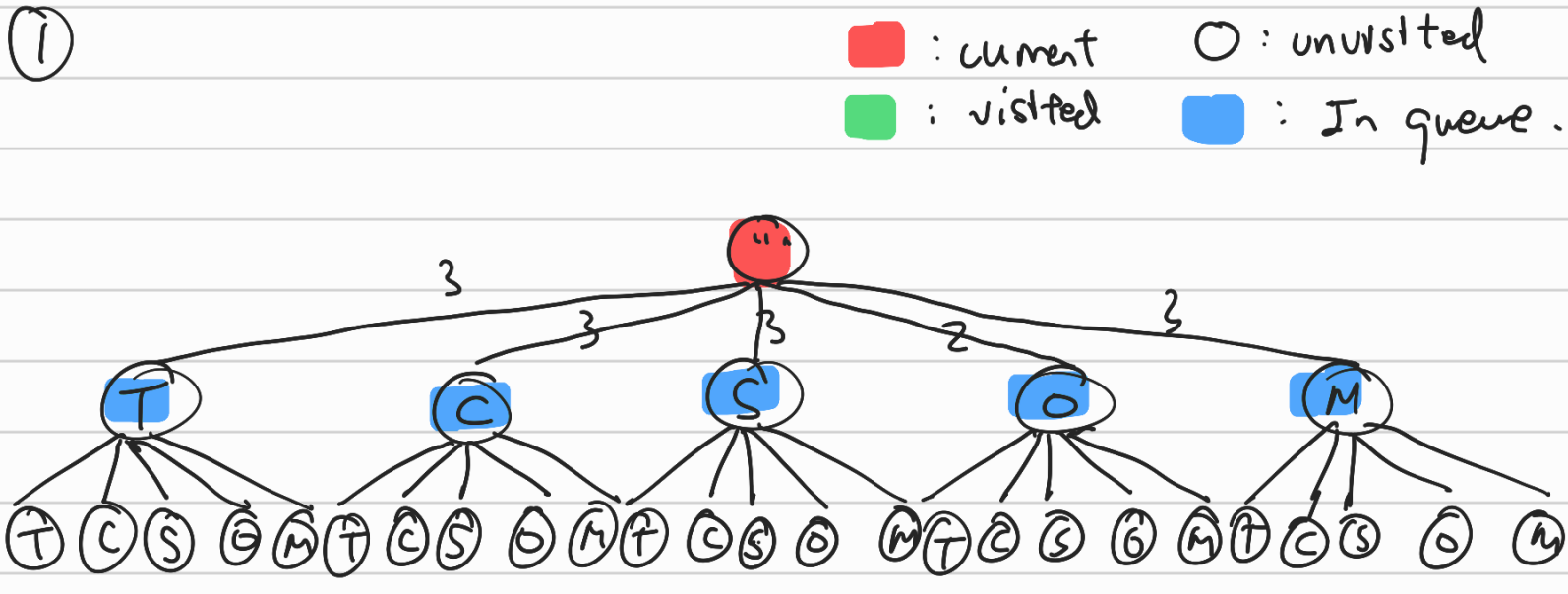


(d)

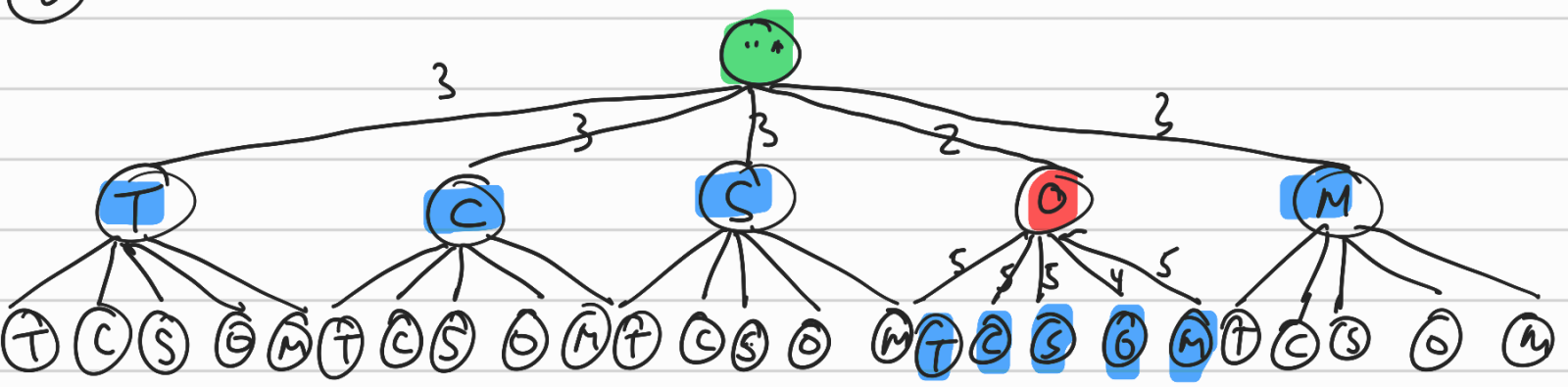


ii) Uniform Cost Search

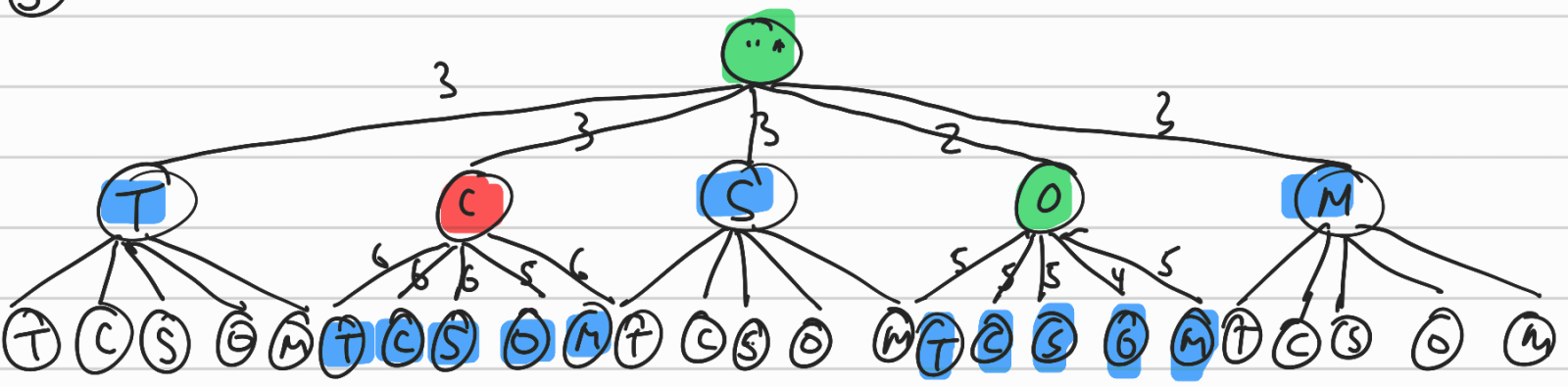
(1)



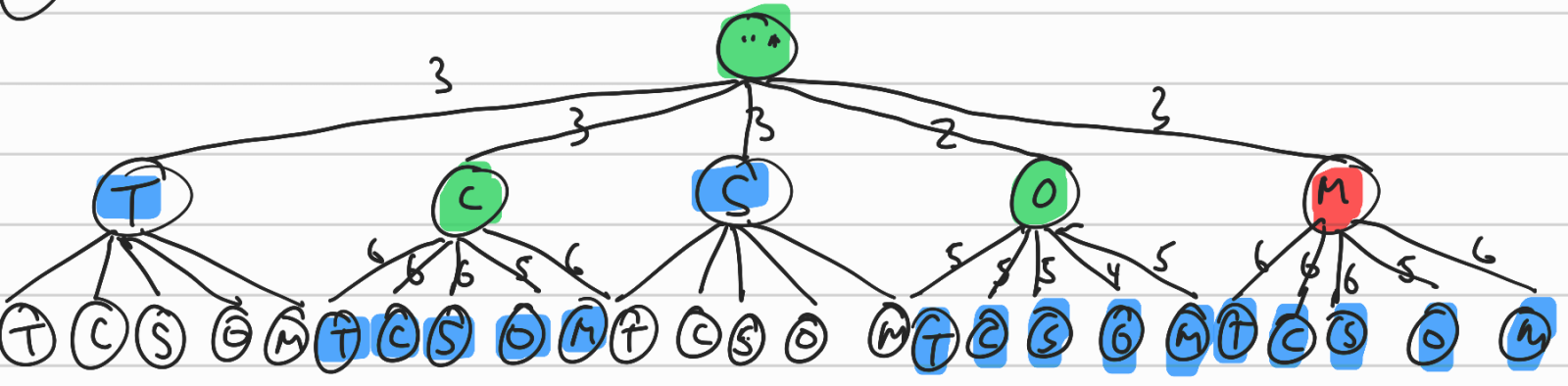
②



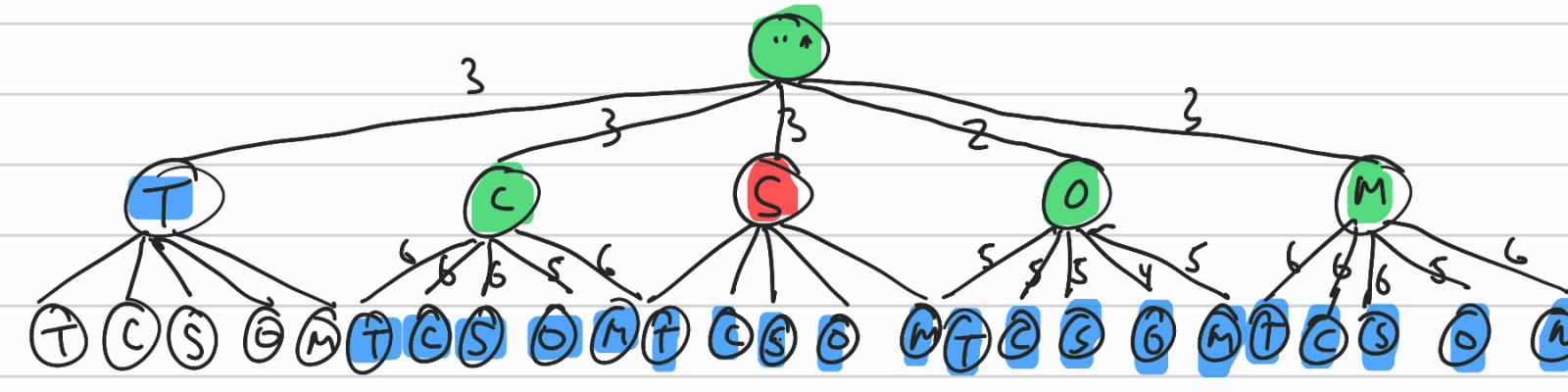
③



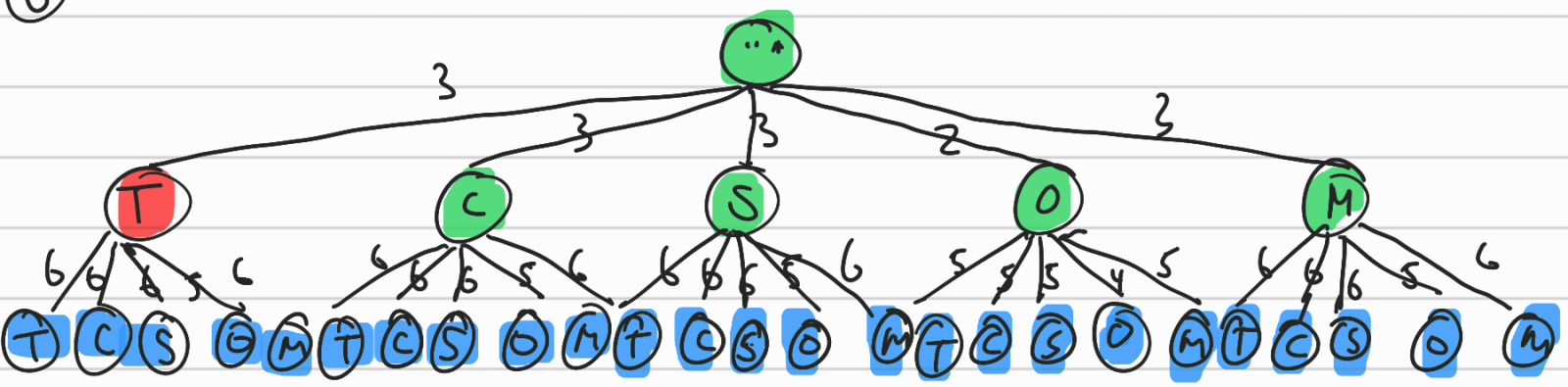
④



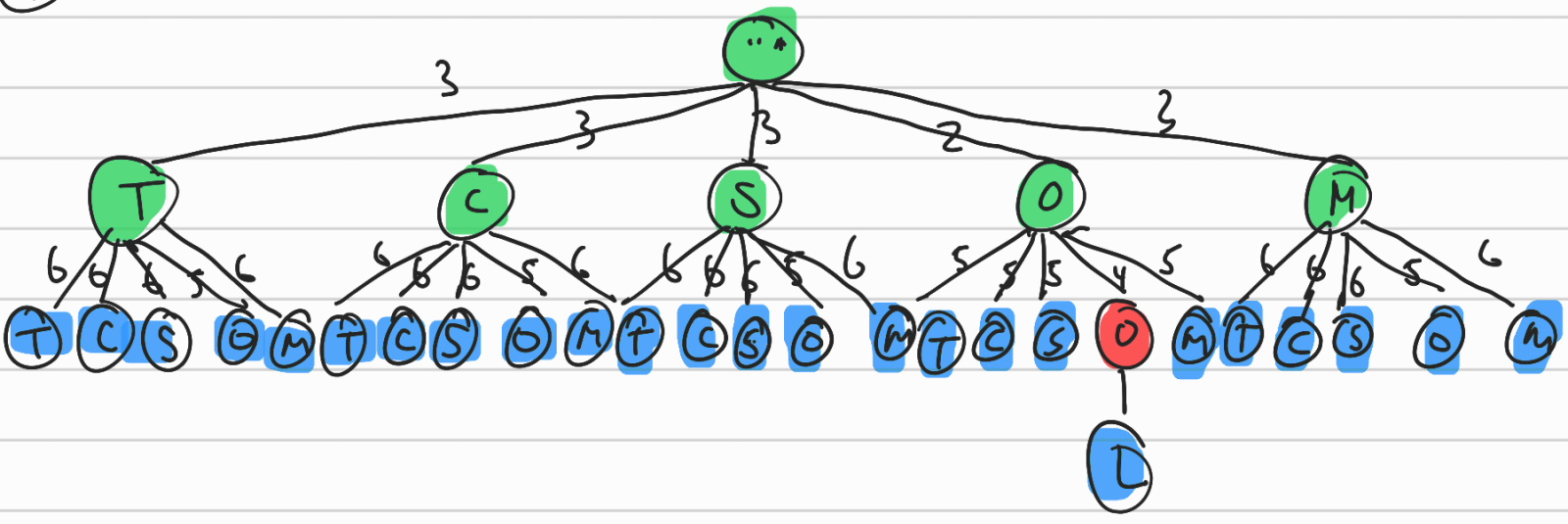
⑤



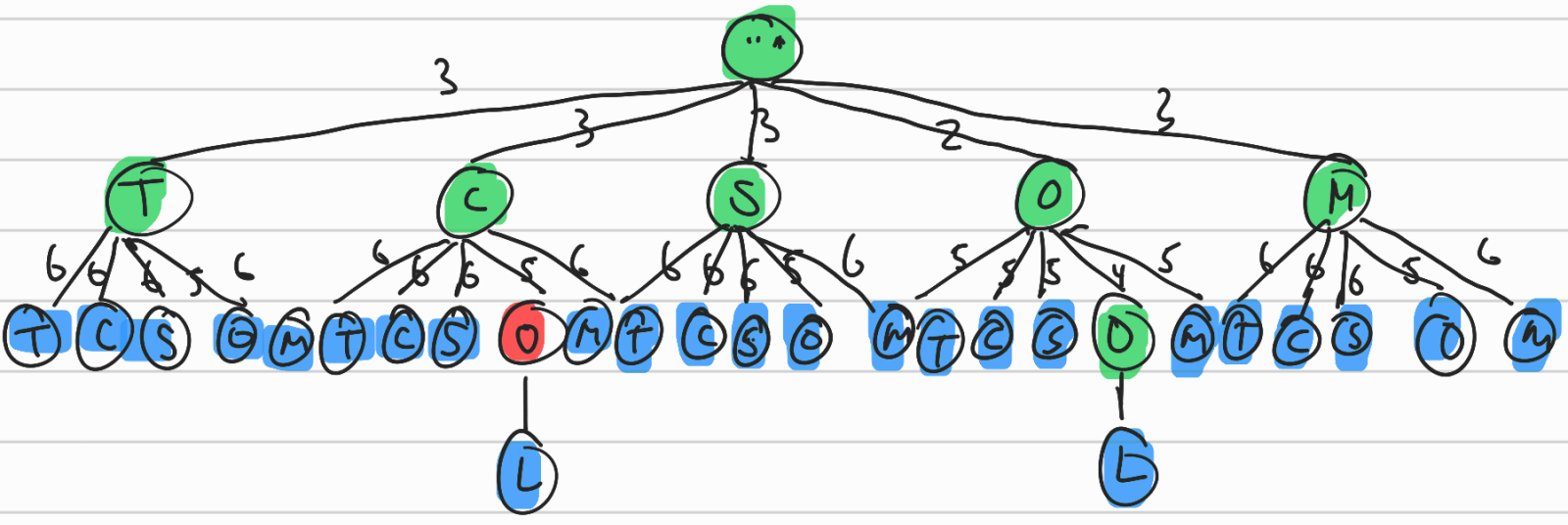
⑥



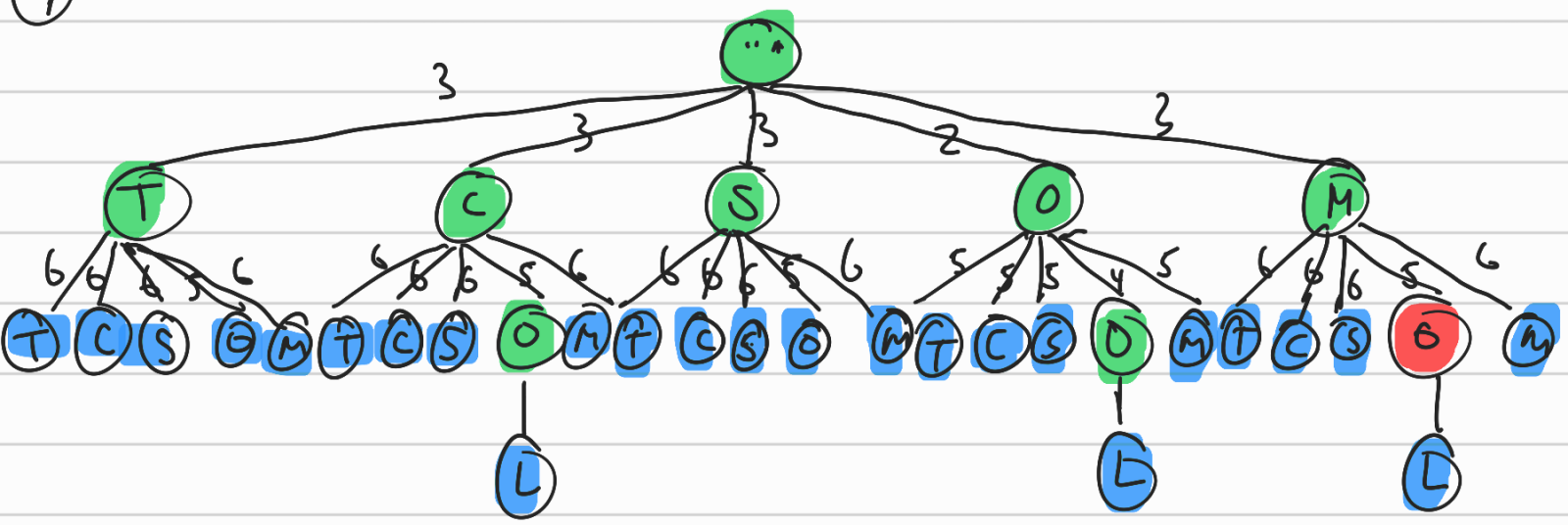
⑦



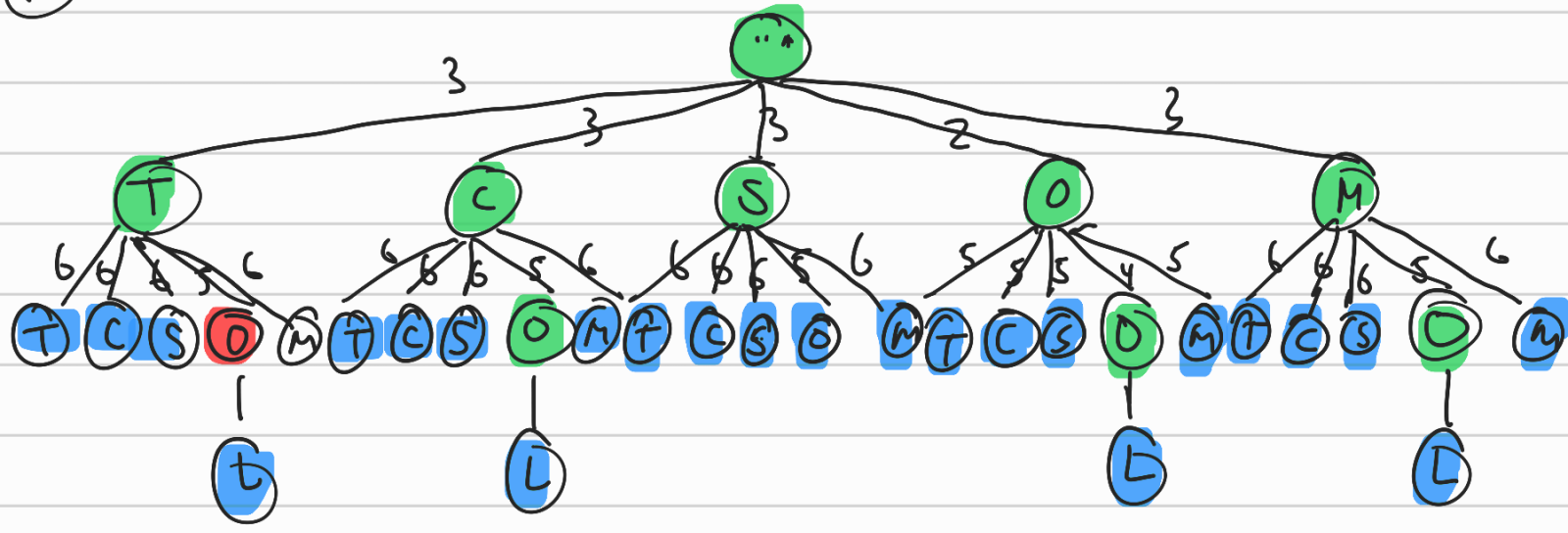
⑧



(9)



(10)



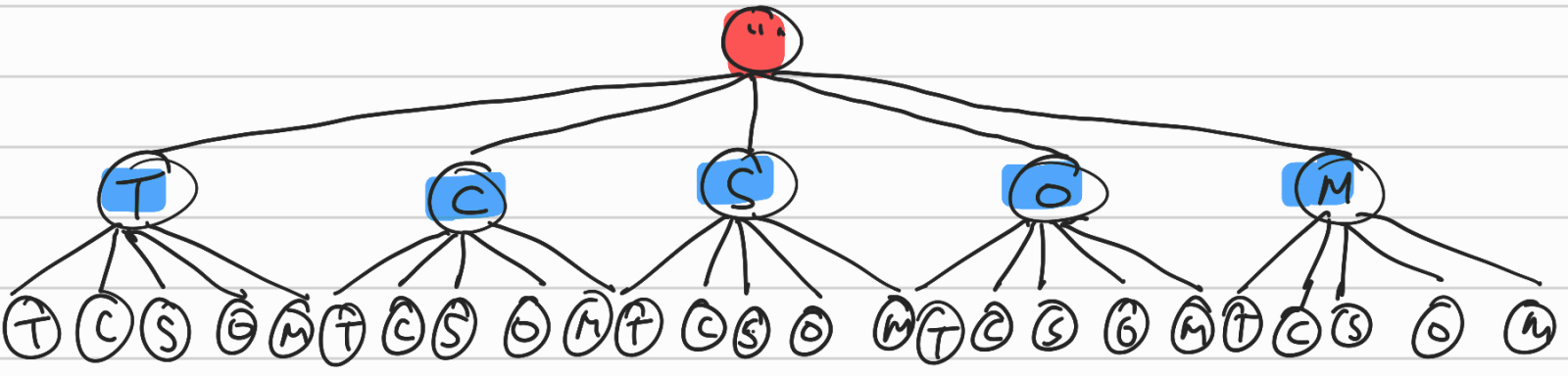
iii) Depth First Search

(Assuming a search tree of 7 levels)

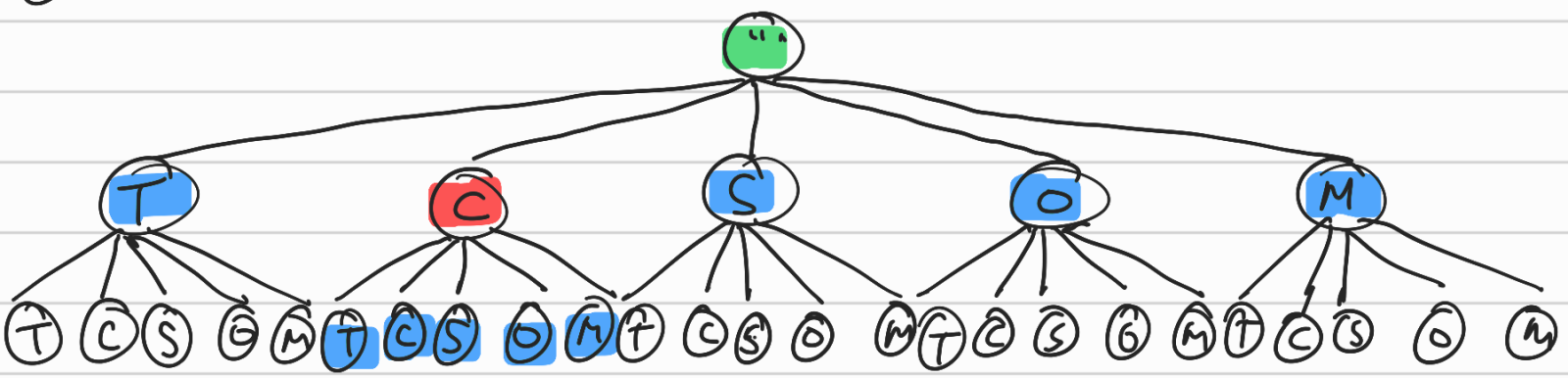
: current ○ : unvisited

: visited : In stack

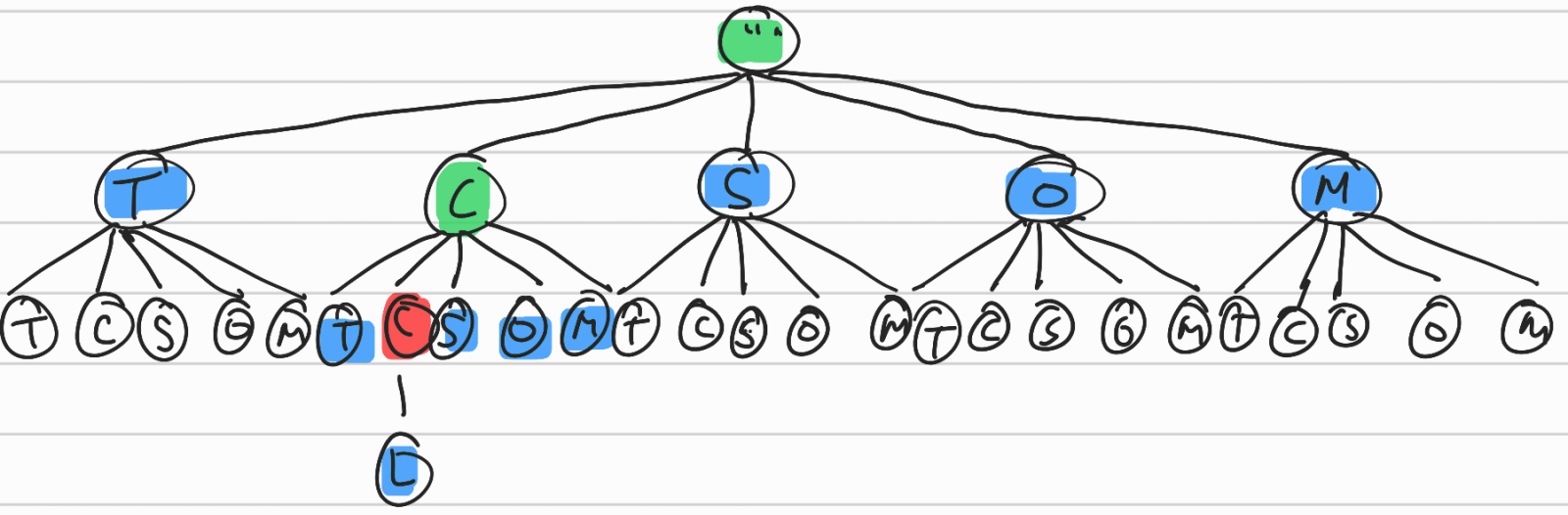
(1)



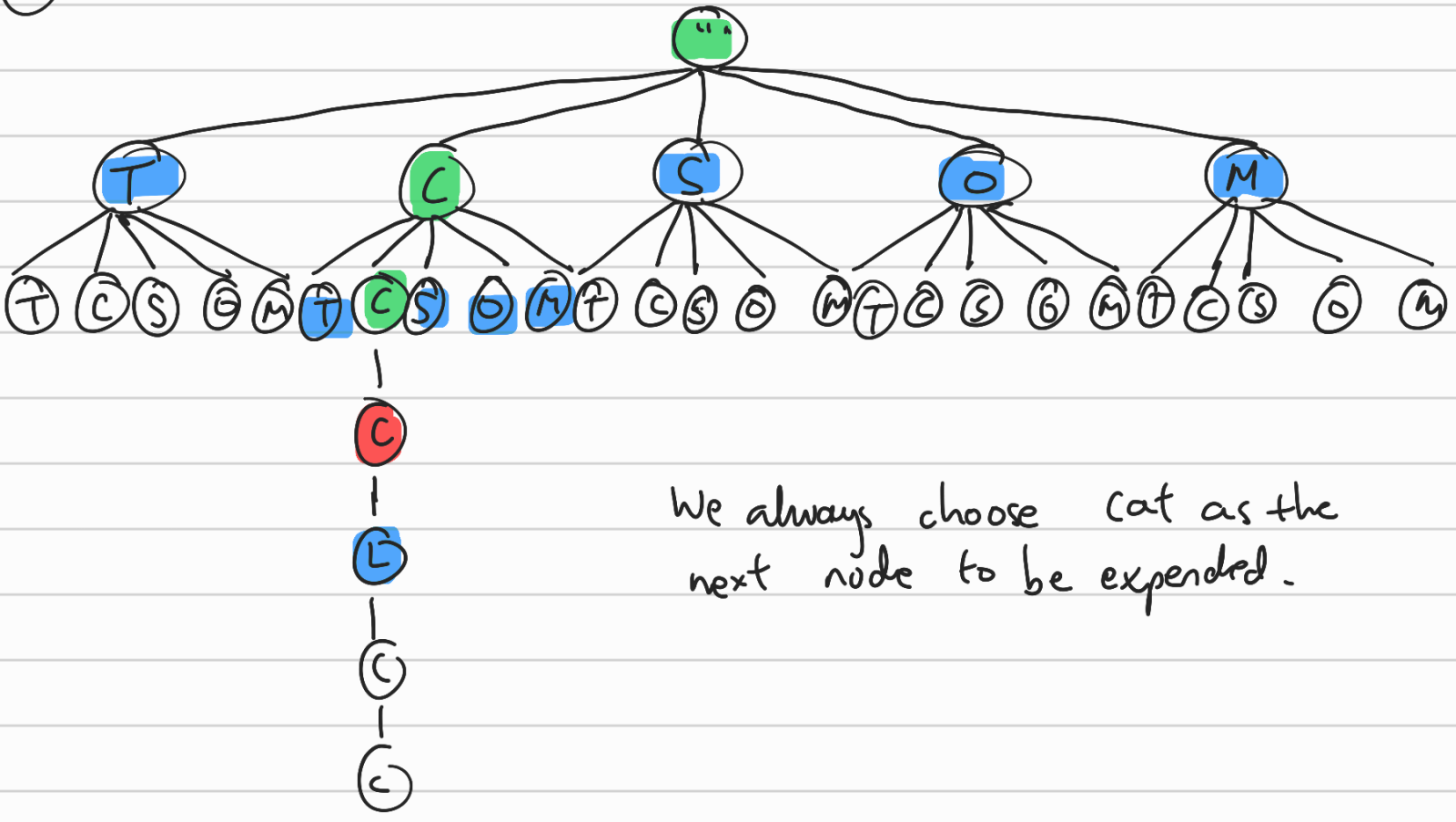
②



③

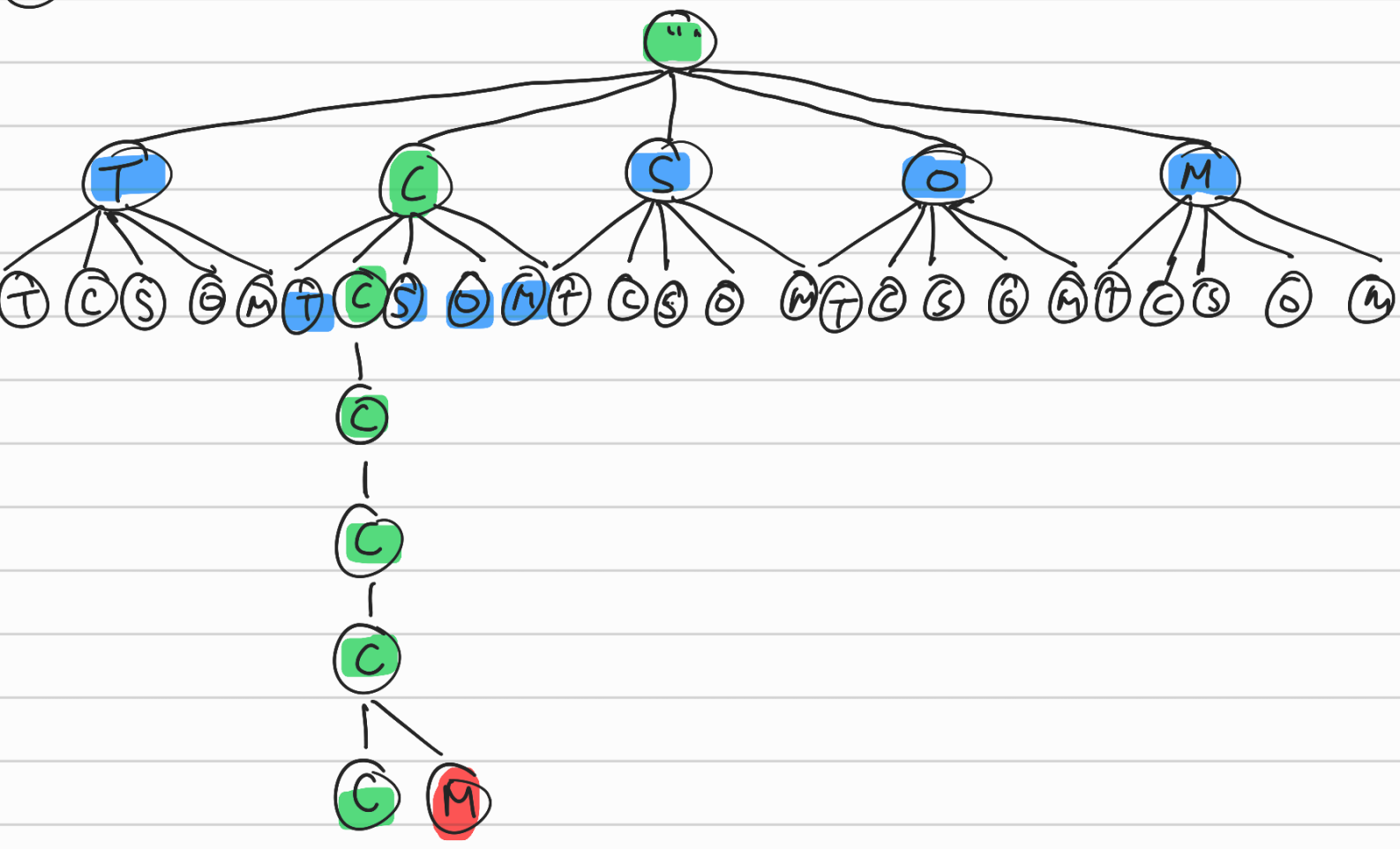


④

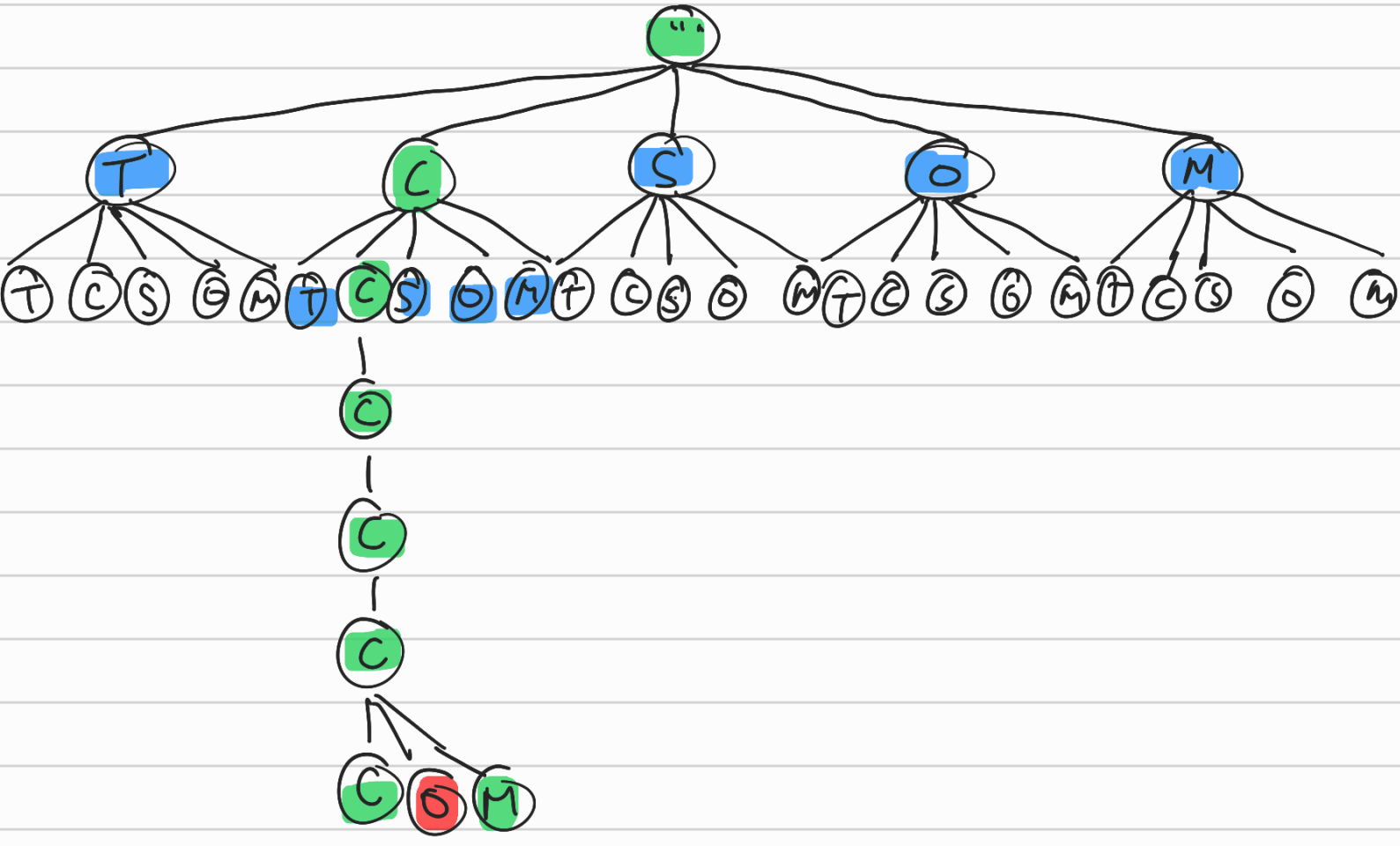


We always choose cat as the next node to be expended.

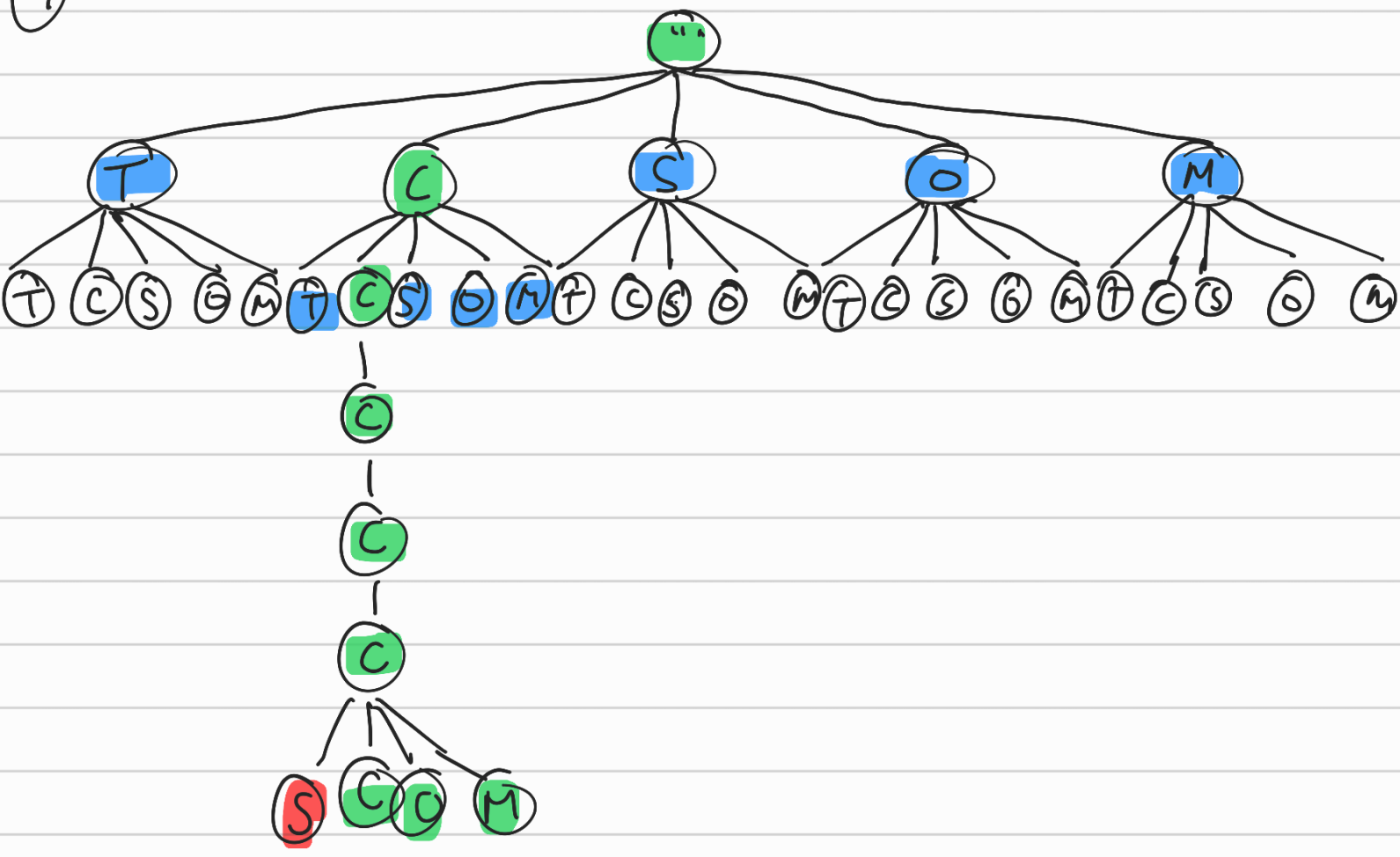
7



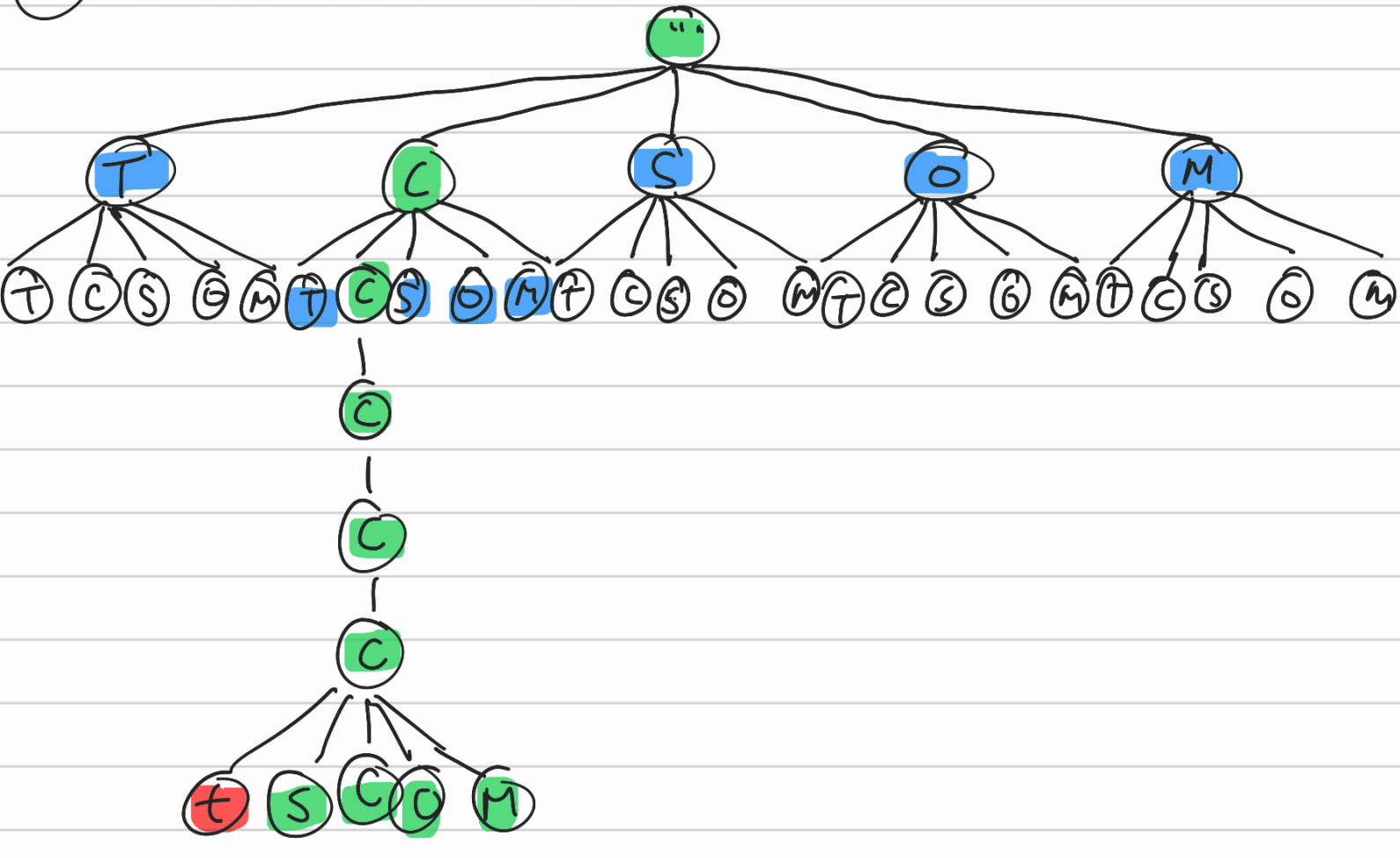
8



9



10



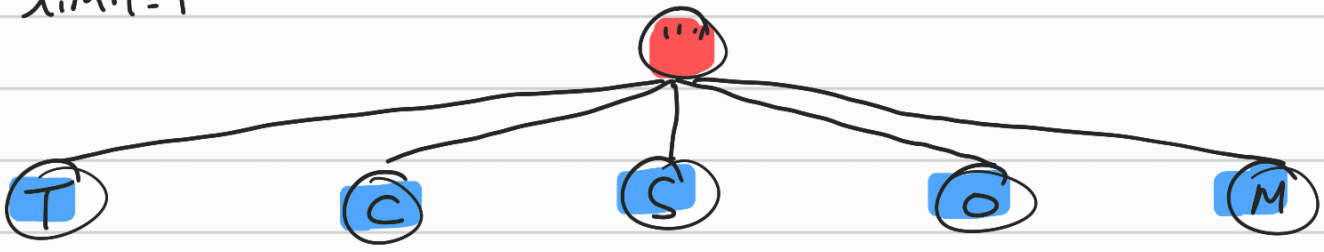
iv Iterative deepening

: current ○ : unvisited

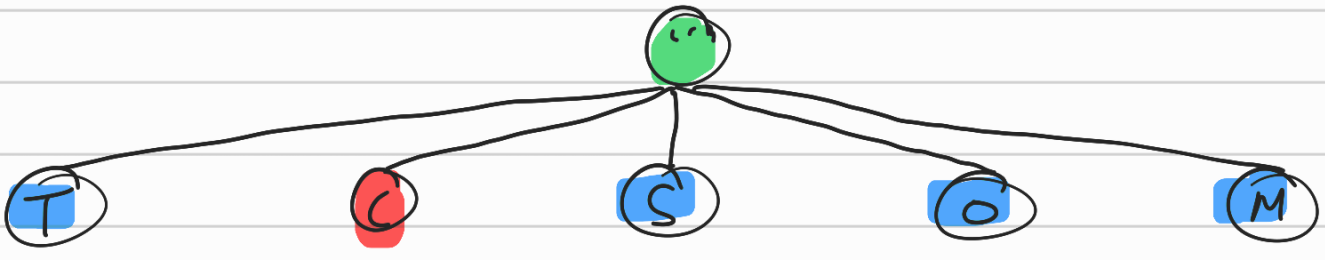
: visited : Frontier.

① limit=0

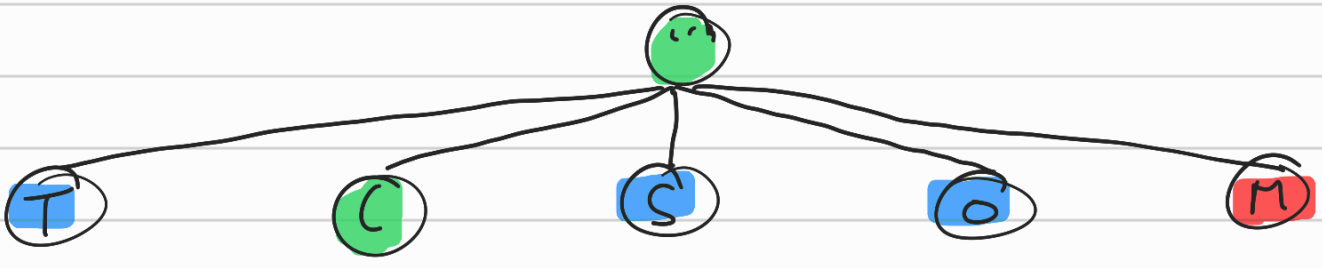
② limit=1



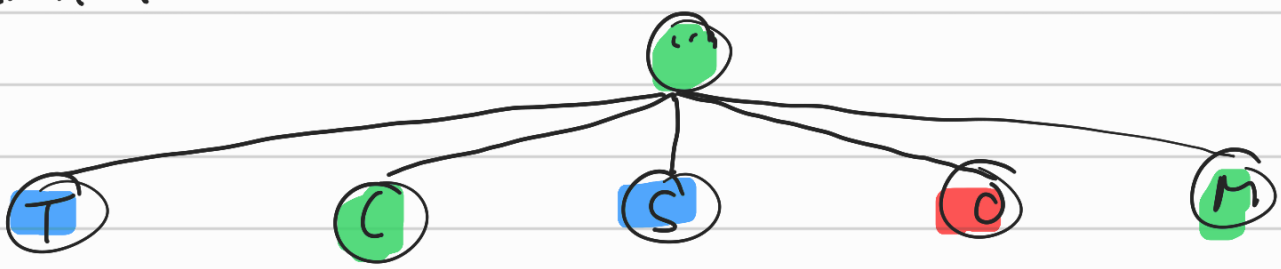
③ limit=1



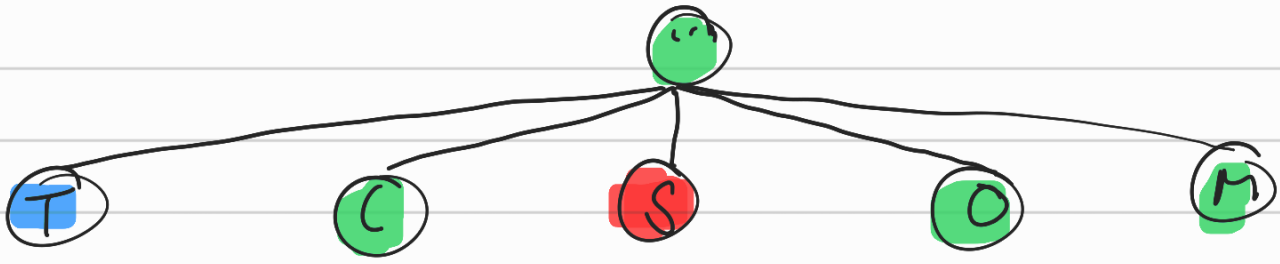
④ limit=1



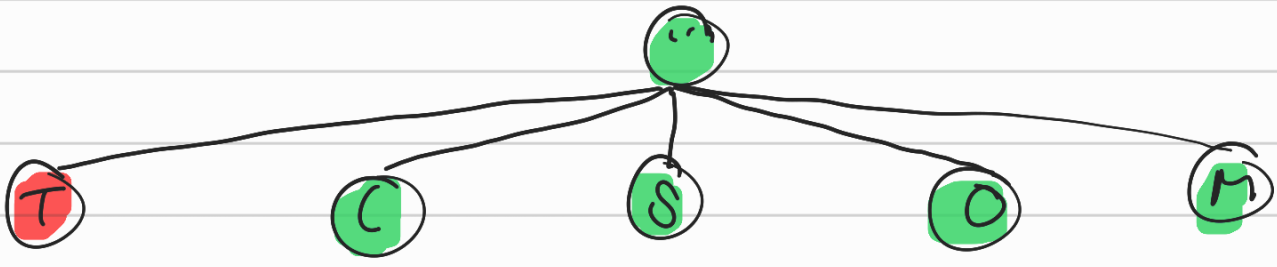
⑤ limit = 1



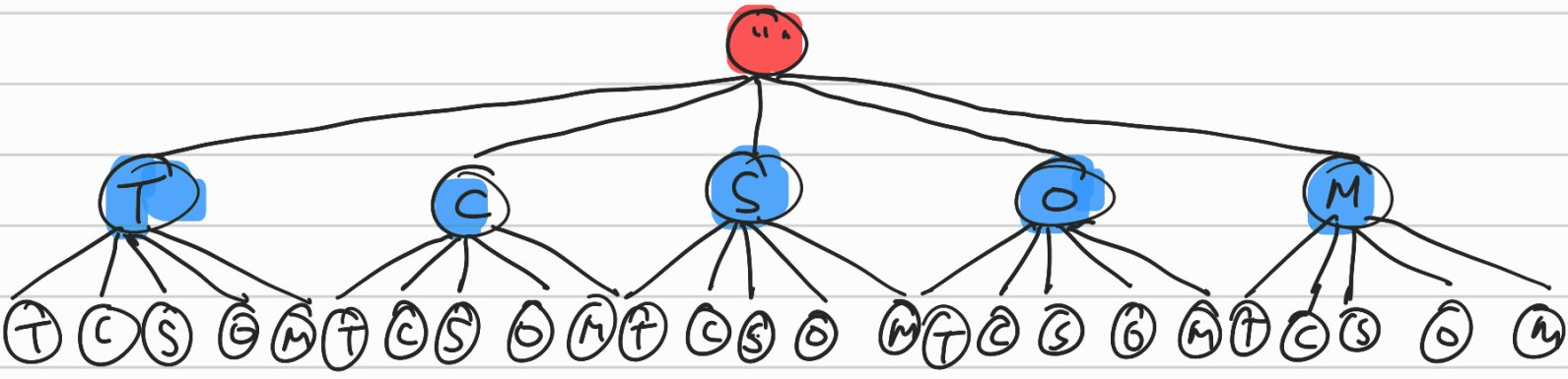
⑥ limit = 1



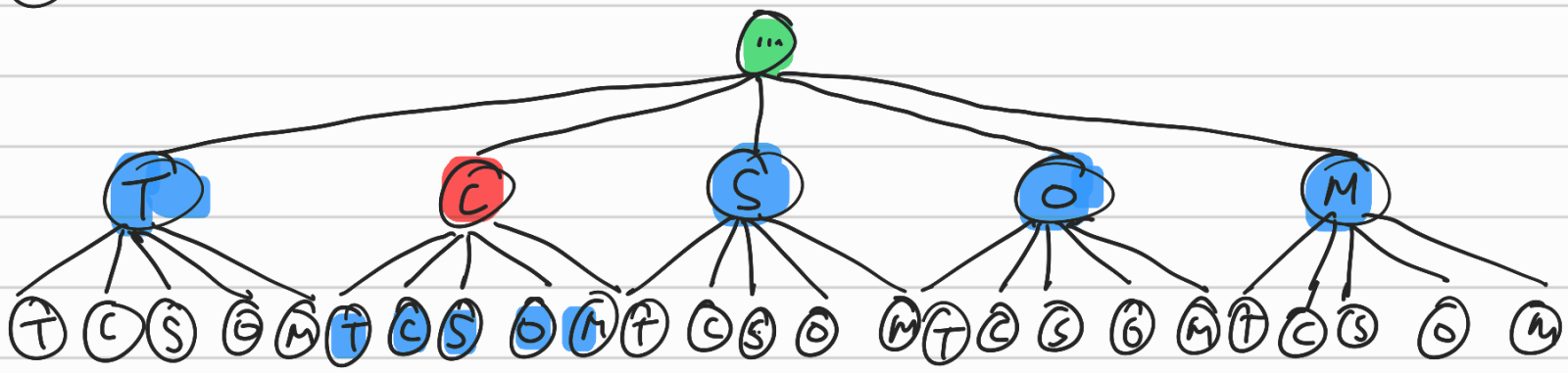
⑦ limit = 1



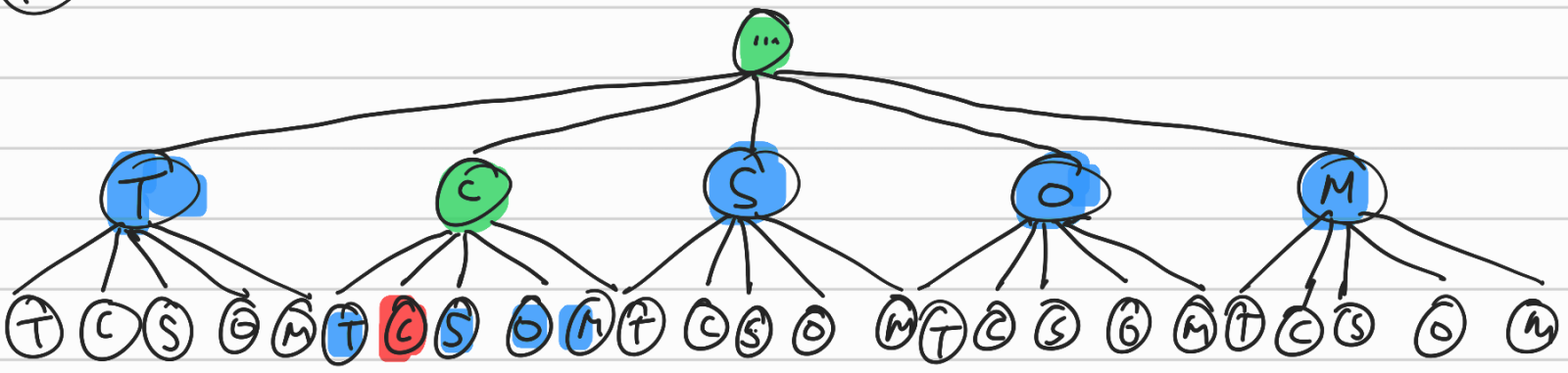
⑧ limit = 2



(9)



(10)



c) Reformulate this problem as a local search problem, stating each of the parts of a local search problem as shown in class.

For this problem the initial state is an empty string and the goal states are the two sentences mentioned in the problem statement. For every state, we can generate its neighbours by adding one of the words in [the, cat, sat, on, mat].

Since we know what the goal states are, the evaluation function would be the absolute difference between the two goal states vs the current state.

ex: state = "the cat"

eval(state) = abs(goal1, state) + abs(goal2, state)

= "the cat sat" - "the cat" + "the cat sat on the mat" - "the cat"

= 3 + 11 = 14

Question 2: Search Algorithm

a) Describe a state space in which iterative deepening search performs much worse than depth-first search (for example, $O(n^2)$ vs $O(n)$).

If we have a state space where every state has a single successor and the goal is located at the bottom of the search tree (the last state). In this case, DFS will find the goal state in $O(n)$ whereas IDS will find the goal state in $O(n^2)$ time.

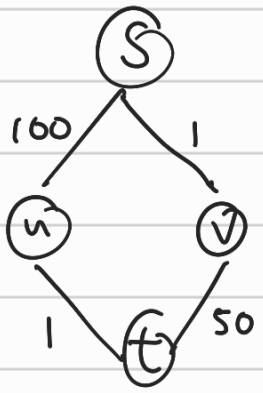
Prove each of the following statements, or give a counterexample.

b) Breadth-first search is a special case of uniform-cost search.

This is true because if all step costs are equivalent, uniform-cost search will behave just like BFS. More specifically, we will use a simple queue instead of a priority queue.

c) Best-first search is optimal in the case where we have a perfect heuristic (ie $h(n) = h^*(n)$, the true cost to the goal state).

This is not true, we can prove this by using a counter-example



where (S) = initial state
(t) = goal state

For this graph, the perfect heuristic will yield 1 for (u) and 50 for (v), which will make the algorithm use the (S) -> (u) -> (t) path with a total cost of 101 instead of the (S) -> (v) -> (t) path with a total cost of 51.

d) Suppose there is a unique optimal solution. Then A* search with a perfect heuristic will never expand nodes that are not in the path of the optimal solution.

Proof by contradiction: Suppose A* search expands nodes that are not in the path of the optimal solution.
However, we already know that A* search is optimal and complete given an admissible heuristic. We know that the heuristic is admissible, thus the current path must be the optimal path. If this is true, then we must have more than a single optimal solution since the algorithm would always yield the optimal solution

□

e) A* search with a heuristic which is admissible but not consistent is complete.

We know that we can call an admissible heuristic consistent if for every state s and successor s'

$$h(s) \leq c(s,s') + h(s')$$

This implies that f cannot decrease along any path:

$$f(s) = g(s) + h(s) \leq g(s) + c(s,s') + h(s') = f(s')$$

However, if the heuristic is not consistent, we can't hold this relation. This means that it is possible for f to decrease along any path, which can get the A* algorithm stuck in a cycle. If this happens, A* will not be able to find a solution which does not make the algorithm complete.

□

A1 Question 3:

a) The brute force function for solving tsp is located at line 30 of a1q3.py (tsp_solver_brute_force).
Here are the stats for the brute force solution (documented at the end of q3_logs.pdf):
{'mean': 2.4432349085582965, 'min': 1.397121188859637, 'max': 3.602886452745925, 'std': 0.3513613360988595}.

b) The random tour generation function is located at line 70 of a1q3.py (tsp_random_solver).
Here are the stats for the random solver solution (documented at the end of q3_logs.pdf):
{'mean': 3.602650495783112, 'min': 1.9934395671404106, 'max': 5.661186871758963, 'std': 0.7538414944553328}.

c) The hill climbing solution is located at line 107 of a1q3.py(tsp_hill_climbing).
Here are the stats for the hill climbing solution (documented at the end of q3_logs.pdf):
{'mean': 2.445095693715398, 'min': 1.397121188859637, 'max': 3.6028864527459254, 'std': 0.35127546933988923}

d) Random solver and hill climbing solution stats: