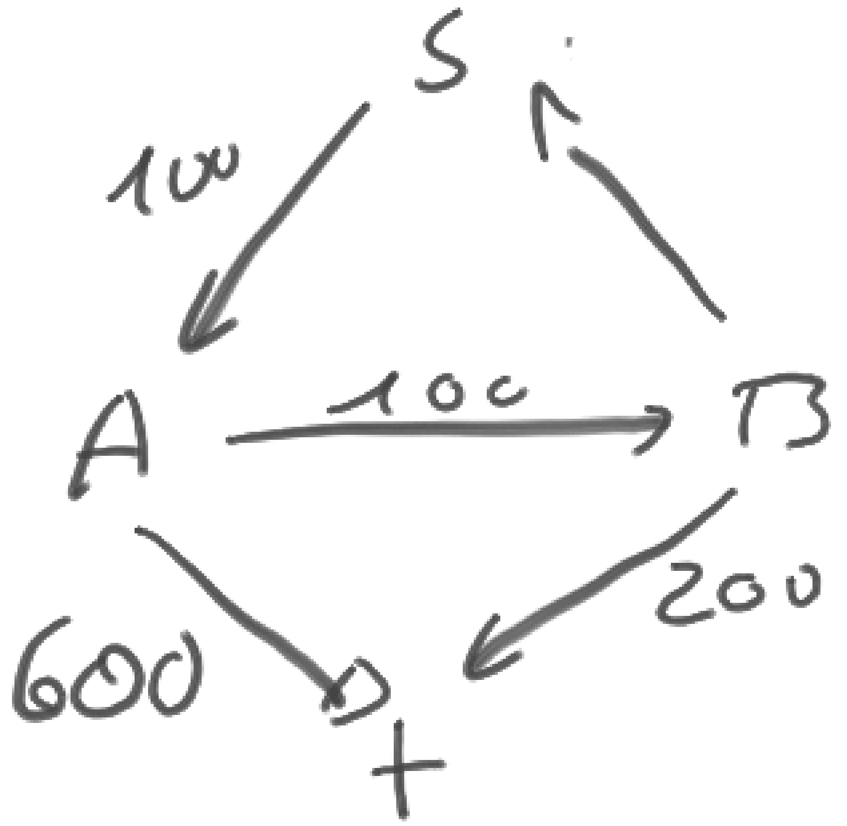
Tutorium #4

18/11/2022

Given a graph G (with positive edge weights), a source and target node and a nonnegative integer k, find the shortest path without using more than k nodes between the source and target.

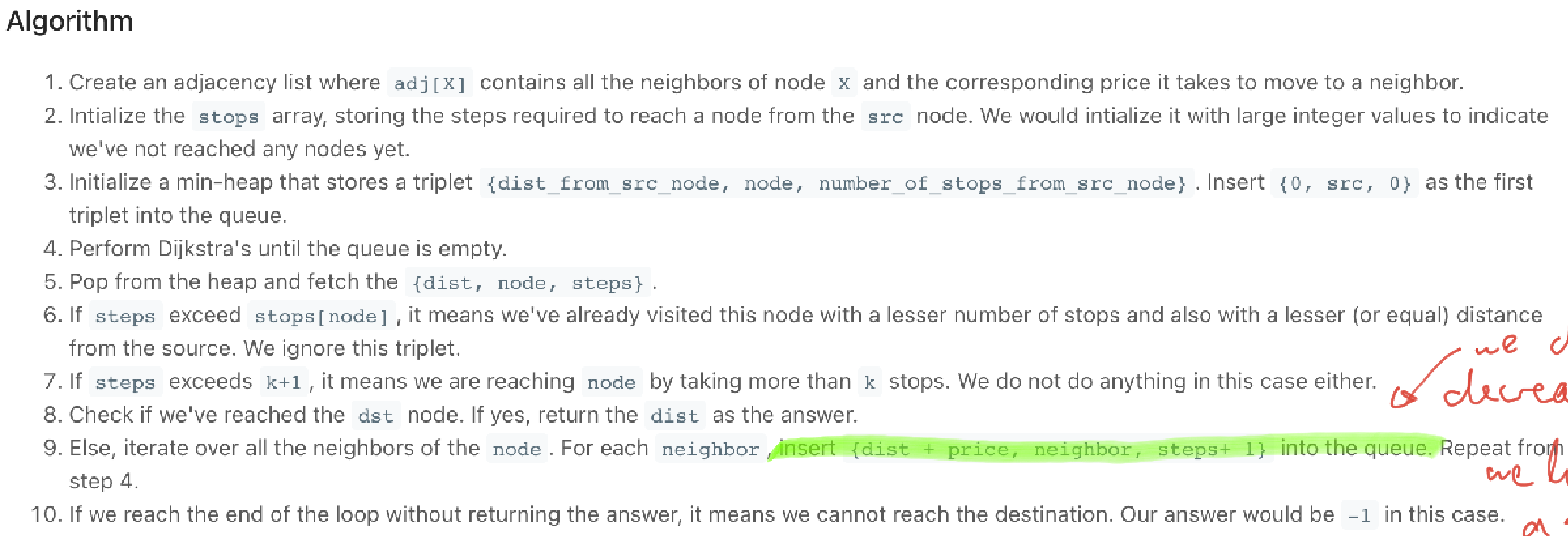
This could be a train network, where a passenger wants to get from A to B but maximally taking k intermediate stops.



try k=2

k=1

https://leetcode.com/problems/cheapest-flights-within-k-stops/solution/

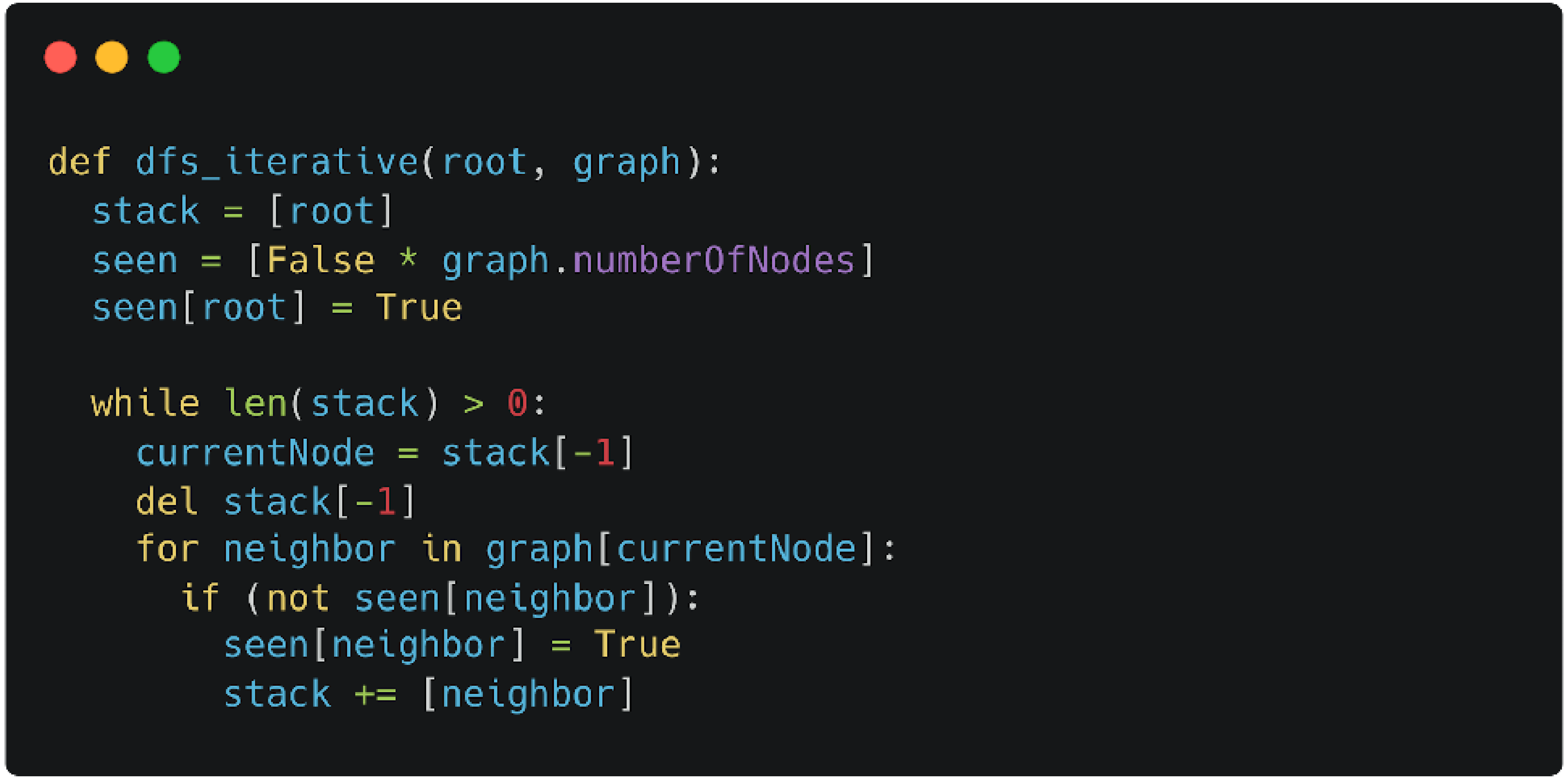


Algorithm

1. Create an adjacency list where `adj[X]` contains all the neighbors of node `X` and the corresponding price it takes to move to a neighbor.
2. Intialize the `stops` array, storing the steps required to reach a node from the `src` node. We would intialize it with large integer values to indicate we've not reached any nodes yet.
3. Initialize a min-heap that stores a triplet `{dist_from_src_node, node, number_of_stops_from_src_node}`. Insert `{0, src, 0}` as the first triplet into the queue.
4. Perform Dijkstra's until the queue is empty.
5. Pop from the heap and fetch the `{dist, node, steps}`.
6. If `steps` exceed `stops[node]`, it means we've already visited this node with a lesser number of stops and also with a lesser (or equal) distance from the source. We ignore this triplet.
7. If `steps` exceeds `k+1`, it means we are reaching `node` by taking more than `k` stops. We do not do anything in this case either.
8. Check if we've reached the `dst` node. If yes, return the `dist` as the answer.
9. Else, iterate over all the neighbors of the `node`. For each `neighbor`, insert `{dist + price, neighbor, steps+ 1}` into the queue. Repeat from step 4.
10. If we reach the end of the loop without returning the answer, it means we cannot reach the destination. Our answer would be `-1` in this case.

we don't decreasekey,
we have a PQ
without updating and decreasekey

Problem 1 (6 points)
Design a non-recursive depth first search starting from a node s and give **pseudocode!** The running time of $O(m+n)$ should not be exceeded.

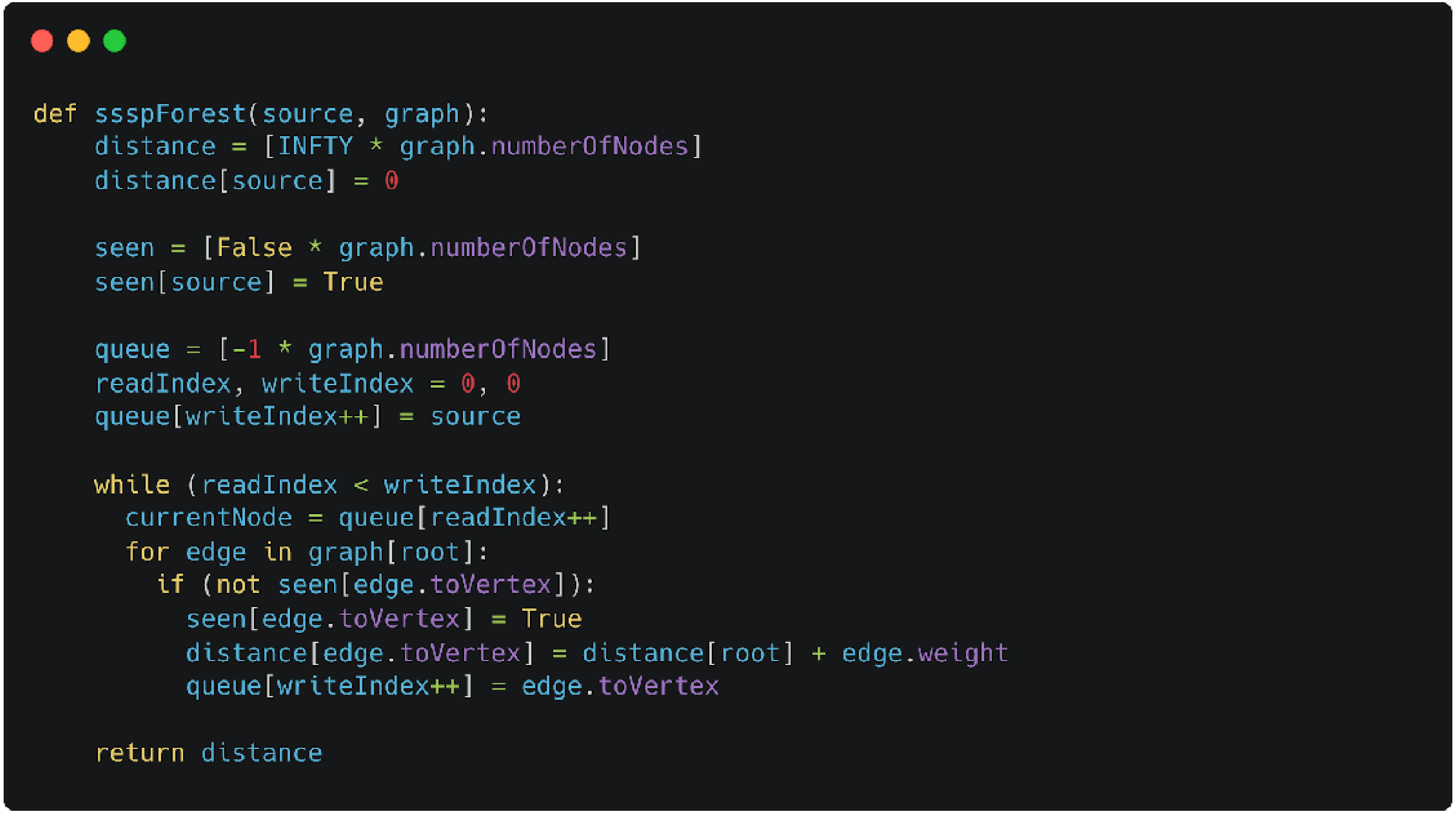


```
def dfs_iterative(root, graph):
  stack = [root]
  seen = [False * graph.numberOfNodes]
  seen[root] = True

  while len(stack) > 0:
    currentNode = stack[-1]
    del stack[-1]
    for neighbor in graph[currentNode]:
      if (not seen[neighbor]):
        seen[neighbor] = True
        stack += [neighbor]
```

Problem 2 (4 points) Outline an algorithm for single source shortest paths in undirected acyclic graphs with nonnegative edge weights that runs in time $O(m + n)$. Give a high level pseudocode of your algorthm.

Solution:

An undirected acyclic graph is a forest. Hence, all nodes reachable from the source node s are reachable only on a unique simple path. Hence, it suffices to perform BFS from s to find all these paths. Distance values can be computed by adding edge weights rather than by counding edges (as in BFS).



```
def ssspForest(source, graph):
    distance = [INFTY * graph.numberOfNodes]
    distance[source] = 0

    seen = [False * graph.numberOfNodes]
    seen[source] = True

    queue = [-1 * graph.numberOfNodes]
    readIndex, writeIndex = 0, 0
    queue[writeIndex++] = source

    while (readIndex < writeIndex):
      currentNode = queue[readIndex++]
      for edge in graph[root]:
        if (not seen[edge.toVertex]):
          seen[edge.toVertex] = True
          distance[edge.toVertex] = distance[root] + edge.weight
          queue[writeIndex++] = edge.toVertex

    return distance
```

Problem 3 (6 points) Give a description of an algorithm to compute the maximal *distance* between two nodes in an undirected, connected graph $G = (V, E)$. Your algorithm should run in time $O(nm)$ ($n := |V|, m := |E|$). Explain why your algorithm has this running time and why your algorithm is correct. How would you have to modify your algorithm if edges are weighted?

Remark: The *distance* between two nodes is the minimal number of edges of a path between those nodes in G. **More formally**, your algorithm should compute $D := \max_{u,v \in V} \min_{P \in \mathcal{P}(u,v)} |P|$, with $\mathcal{P}(u, v)$ being the set of all paths between u and v, and $|P|$ the number of edges in the path P.

Solution:

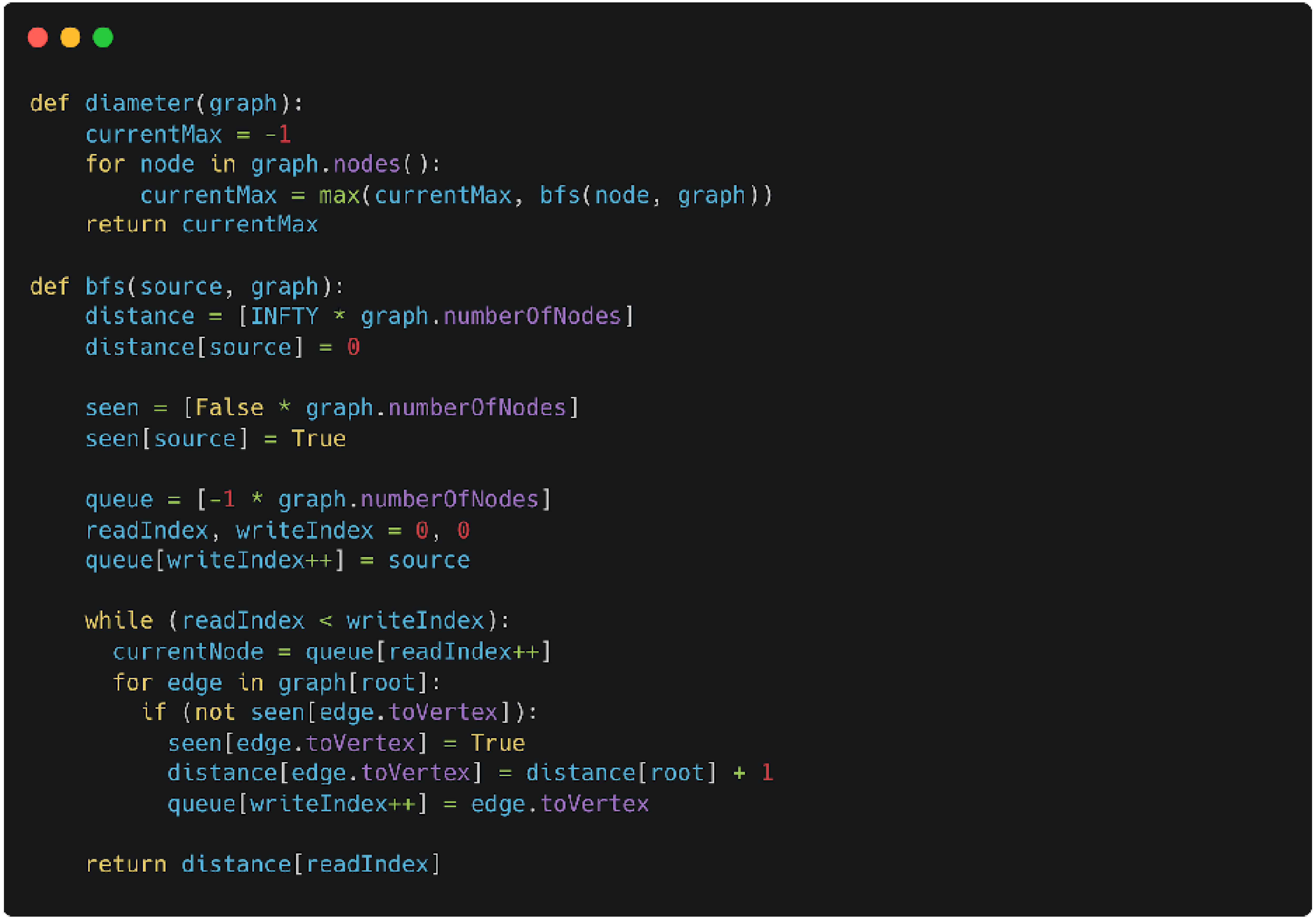
To solve the problem, we modify a breadth first search such that for a start node s it returns distances of the node with the maximum distance (this is the last node seen by the breadth first search). Hence, we obtain a function

$$bfs(s : NodeID) : \mathbb{N}_0 \ .$$

Then the following algorithm computes the diamter of the graph G:

```
function $durchmesser(V : Set$ of $NodeID, E : Set$ of $unordered\ Pair$ of $NodeID) : \mathbb{N}_0$
$res := h := 0 : \mathbb{N}_0$
for each $v \in V$ do
    $h := bfs(v)$
    if $h > res$ then $res := h$
end for                                                    // $n$ Aufrufe von $bfs$
return $res$
```

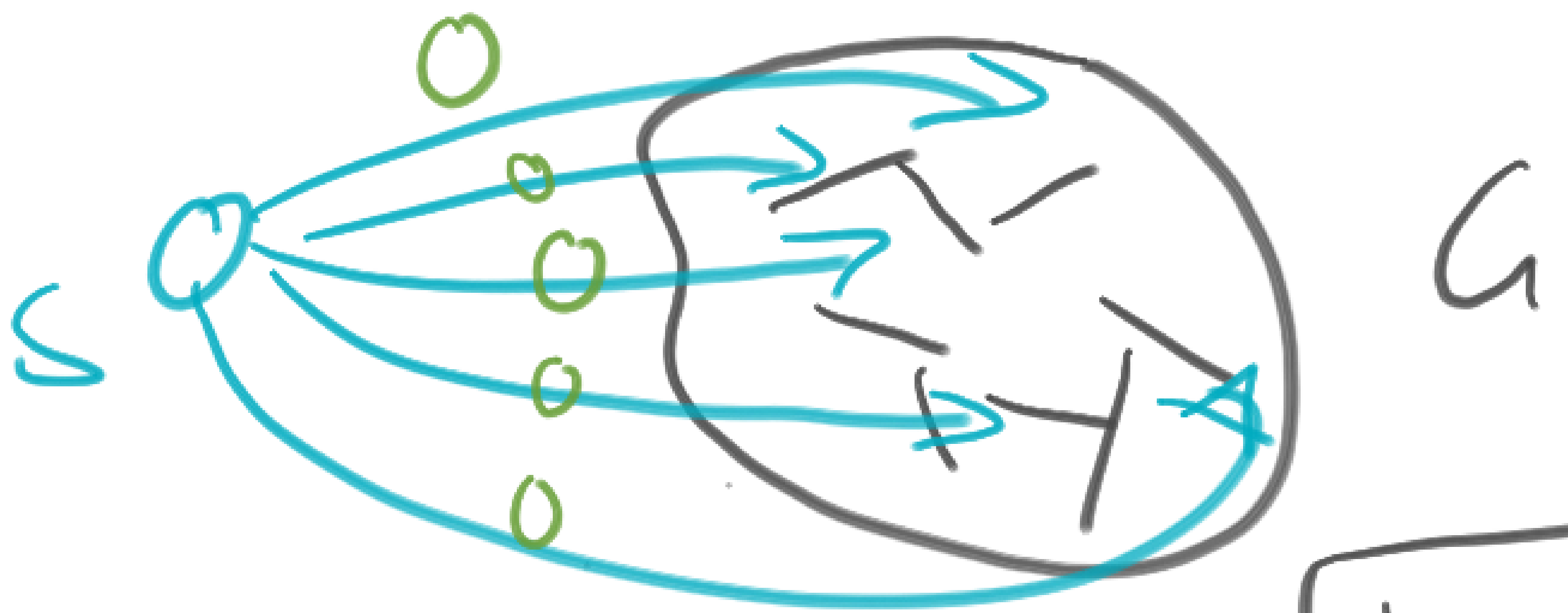
The function bfs takes $O(m + n)$ time, since a bfs touches each edge $O(1)$-times and we compute the distance to every other node. Since G is connected $n < m + 1$, we get bfs runs in time $O(m)$. Hence, overall we need $O(nm)$ time.



```
def diameter(graph):
    currentMax = -1
    for node in graph.nodes():
        currentMax = max(currentMax, bfs(node, graph))
    return currentMax

def bfs(source, graph):
    distance = [INFTY * graph.numberOfNodes]
    distance[source] = 0

    seen = [False * graph.numberOfNodes]
    seen[source] = True

    queue = [-1 * graph.numberOfNodes]
    readIndex, writeIndex = 0, 0
    queue[writeIndex++] = source

    while (readIndex < writeIndex):
      currentNode = queue[readIndex++]
      for edge in graph[root]:
        if (not seen[edge.toVertex]):
          seen[edge.toVertex] = True
          distance[edge.toVertex] = distance[root] + 1
          queue[writeIndex++] = edge.toVertex

    return distance[readIndex]
```

The weighted case: Johnson's Algorithm

We would love Dijkstra, but edgeweights maybe $\in \mathbb{R}$



$\Rightarrow$ use $\mu_s(s, v) \hat{=} l[v]$

How to check for neg. cycles in undirected graphs?

New weights of G':

$$w'(u,v) = w(u,v) + h[u] - h[v] \geq 0 \quad (?) \checkmark$$

$$\boxed{\Leftrightarrow \quad w(u,v) \geq h[v] - h[u]}$$

$\Rightarrow \forall v \in V(G')$: dijkstra(v)

$\Rightarrow$ get diameter