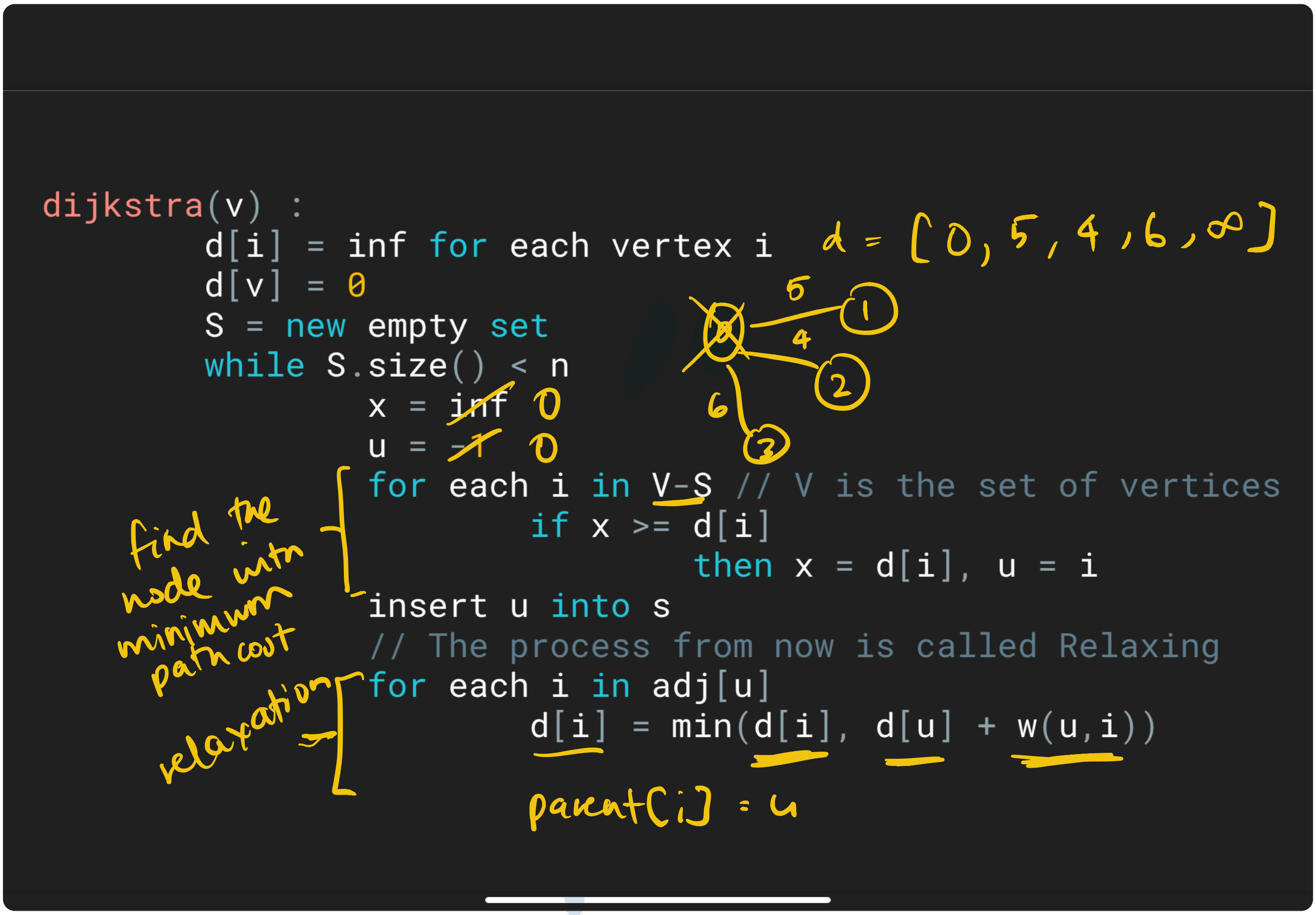
Coffee Shop. →



```
dijkstra(v) :
        d[i] = inf for each vertex i        d = [0, 5, 4, 6, ∞]
        d[v] = 0
        S = new empty set
        while S.size() < n
                x = ~~inf~~ 0
                u = ~~-1~~ 0
                for each i in V-S // V is the set of vertices
                        if x >= d[i]
                                then x = d[i], u = i
                insert u into s
                // The process from now is called Relaxing
                for each i in adj[u]
                        d[i] = min(d[i], d[u] + w(u,i))

                        parent[i] = u
```

find the node with minimum path cost

relaxation

WE'RE GOING TO LOOK AT DIJKSTRA 3 TIMES.

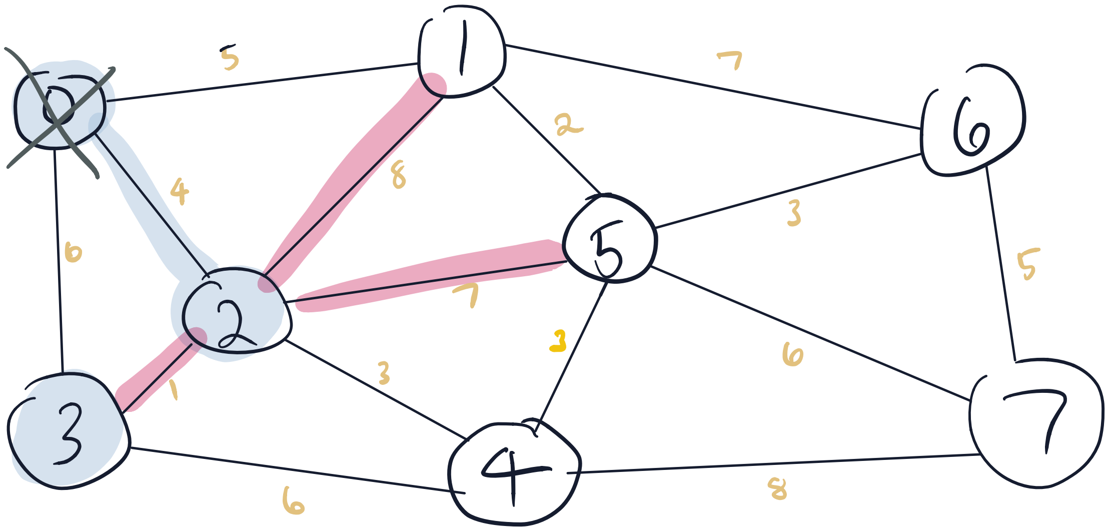
EACH TIME WE'LL INTRODUCE A "NEW" OBSERVATION.

1. Simulating the algorithm, mechanical robot.

2. Why is this correct? The "FRONTIER", when does it fail?

3. How do we classify this algorithm + complexity?

SSSP : Single Source Shortest Path



	0	1	2	3	4	5	6	7
dist	0	5	4	5	7	11	∞	∞

	0	1	2	3	4	5	6	7
parent	0	0	0	2	2	2	-1	-1

S : 0 2 3

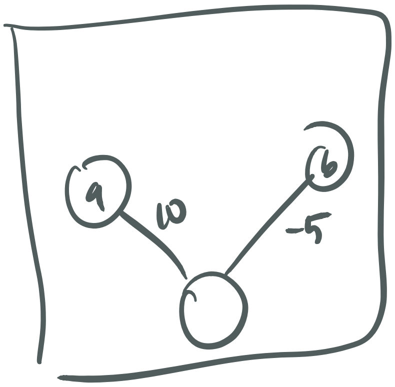
PSEUDO Code:

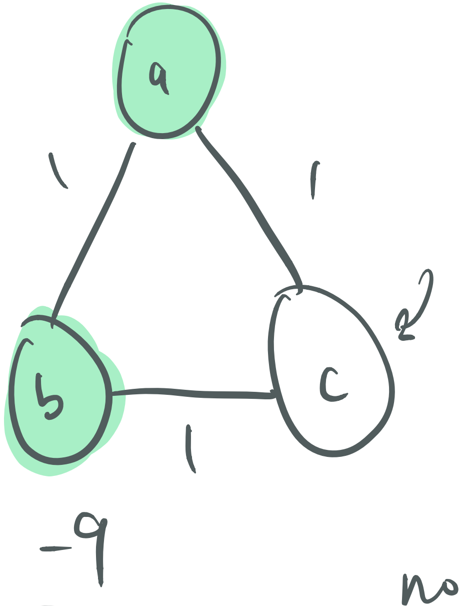
courtesy of 6.006
(a lil modified for us ü)

```
DIJKSTRA(G, start):
    D := {∞}  // init all ∞
    S := ∅    // haven't seen any nodes.
    D[start] = 0  // start ⇝ start = 0   δ(start, start) = 0
    P := InitParent()  // set all indices to some sentinel value

    while |S| < |G.Vertices| :
        next = extract_min(D)
        S := S ∪ {next}

        for v in Adj[next]
            Relax(G, P, D, next, v)


RELAX(G, P, D, u, v):
    if D[v] > D[u] + G[u][v]:
        update(D, v, D[u]+G[u][v])
        P[v] = u
```

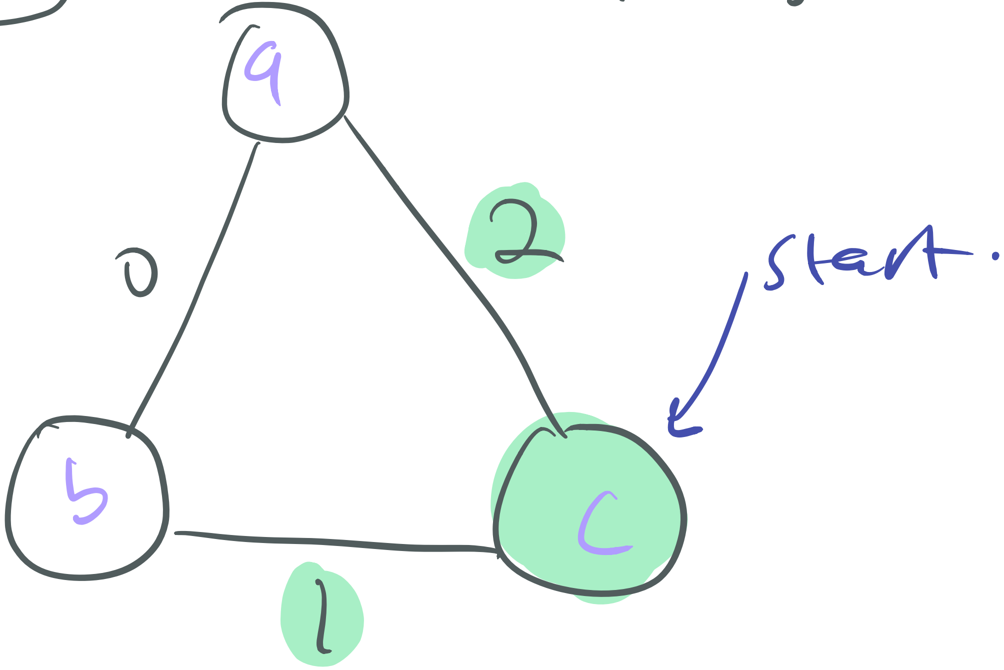
When do we fail?

Q: Can you think of a three node graph, that dijkstra fails to find the shortest path for from some start node?





non negative.



Mistakes to make:

① Not using nreachables instead using a local variable

→ don't know where to put next node in reachables

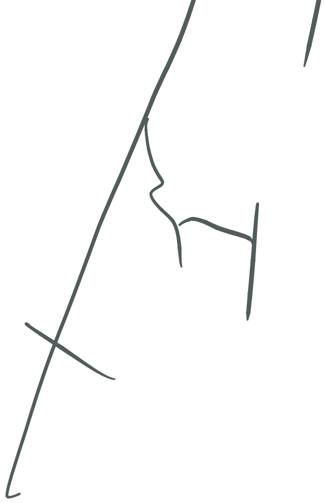
② Using nreachables += instead of =

→ double counting, we're threading nreachables, not accumulating it.

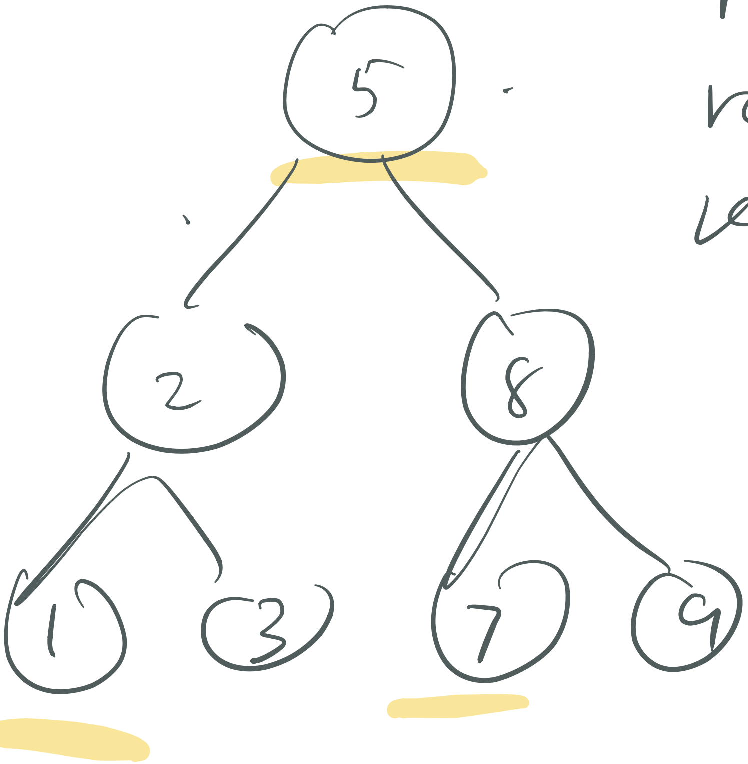
---

③ setting visited[i] at beginning vs. visited[i] on push.

④ Forgetting to set visited[start].



preorder



```
print(node)
recurse (left)
recurse (right)
```

inorder

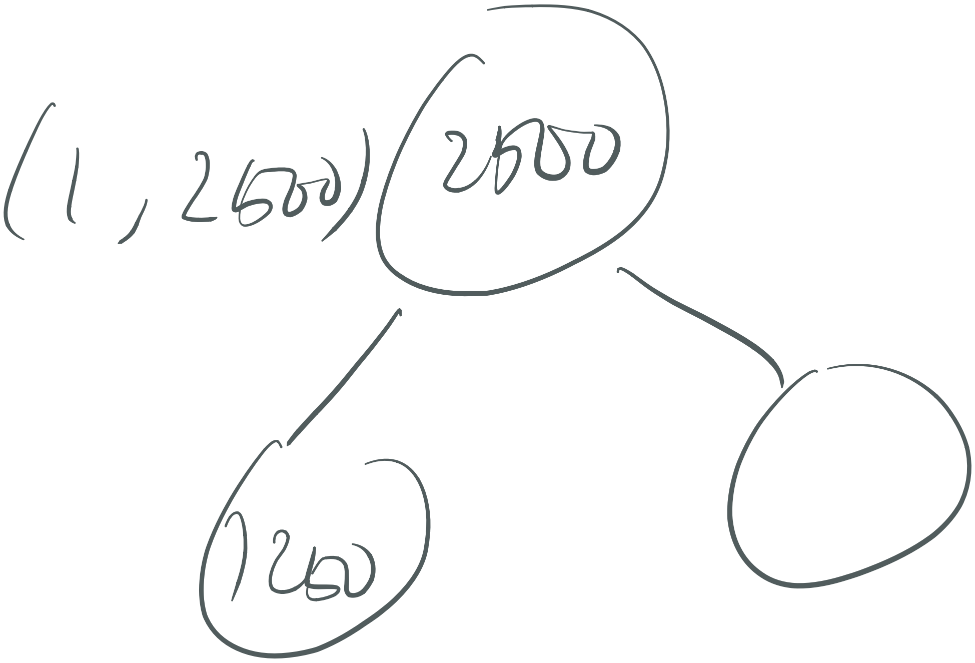
```
recurse l
print node
recurse right
```

preorder

postorder

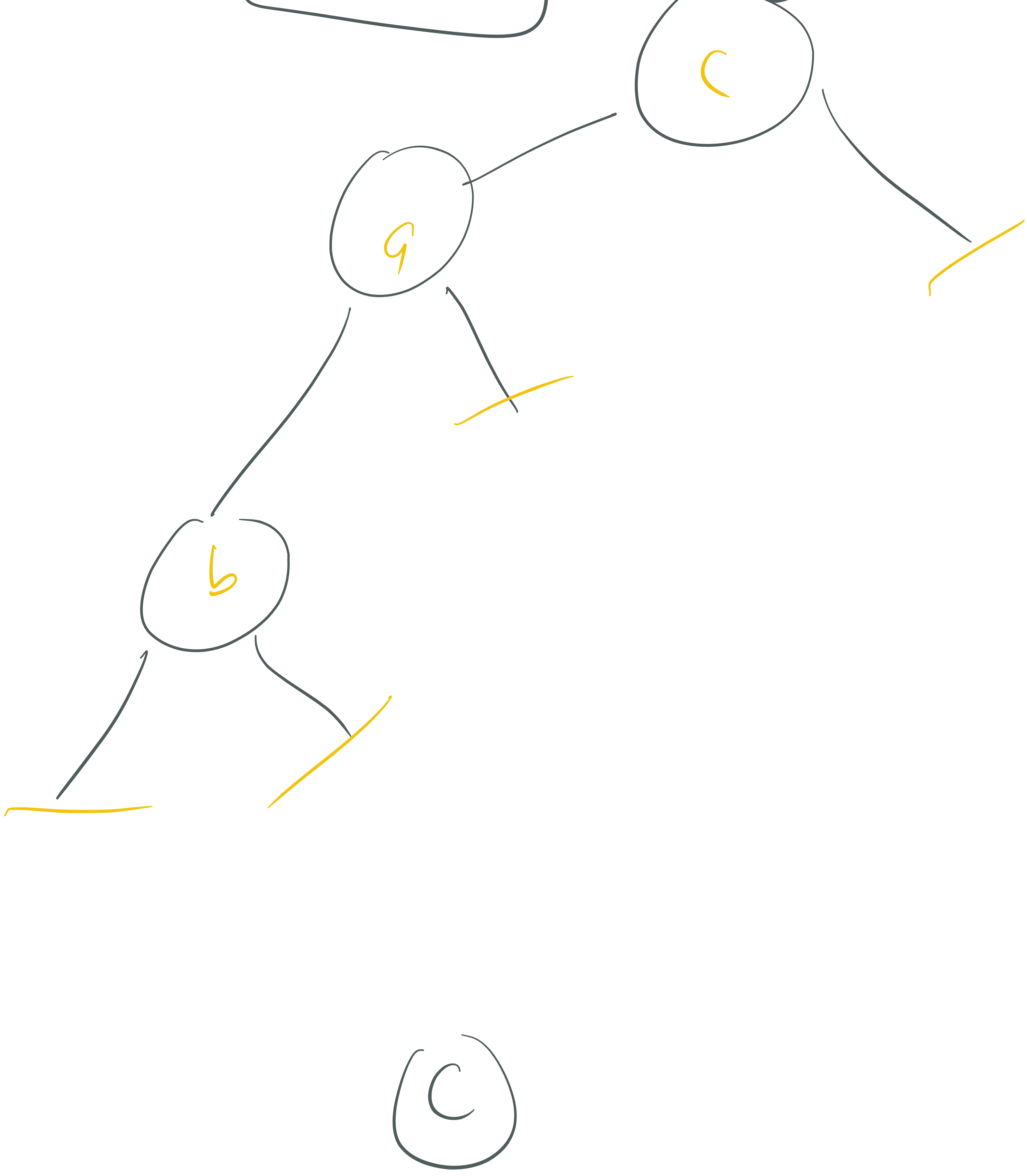
"prefix" order

[1, 5000)



2500, 1250, . . . , 5000

d
5000
n
n-1 nodes
n,


c
g
b
c