Ruolin Li 31764160

CPSC 320: Graph Play*

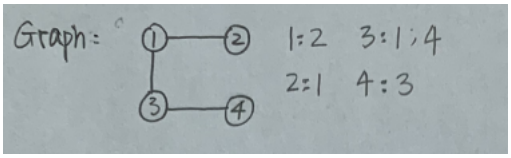
Here you'll gain experience with design and analysis of graph algorithms, starting with graph search algorithms and then moving on to finding the diameter of a graph.

1 Graph Search Algorithms

We use $G = (V, E)$ to denote a graph, where V is a finite set of *nodes* and E is a set of *edges*. Often we use n to denote the number of nodes, and assume that $V = 1, 2, \ldots, n$.

Edges are pairs (u, v) of adjacent nodes. In this worksheet, edges are undirected: there is no implied ordering ("arrow") from u to v. Usually we assume that $u = v$, in which case the graph is *simple*. G may or may not be connected. For this worksheet, all edges are unweighted. A handy data structure for describing the edges of a graph is *adjacency lists*: there is one list per node, and the list for node u includes exactly those nodes v that are "neighbours" of u, that is, such that $(u, v) \in E$.

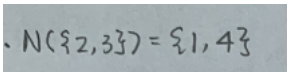
1. **Illustrate the notation defined above, for a graph with node set $V = \{1, 2, 3, 4\}$.**



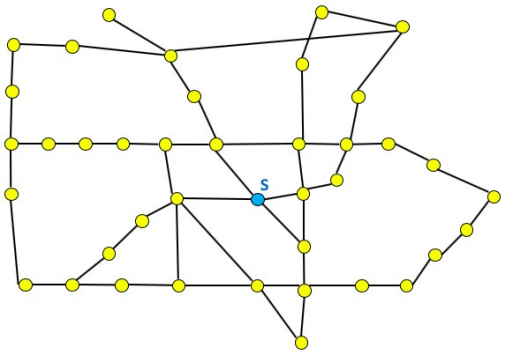
2. More generally, for any subset $A \subseteq V$, let

$$N(A) = \{ \text{ neighbours of } A\} = \{v / \in A \mid v \text{ is adjacent to some } u \in A\}.$$

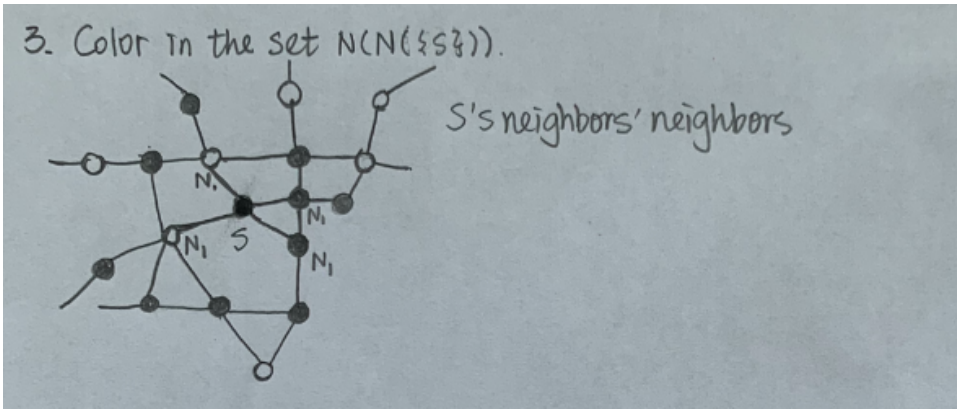
Write down $N(\{2, 3\})$ for your graph above.



3. Here's a graph, derived from the London Underground, where nodes represent stations, edges are between stations that are adacent stops of a train (trains go in both directions so edges are undirected). The node marked



s represents the Picadilly Circus stateion. **Colour in the set $N(N(\{s\}))$.**

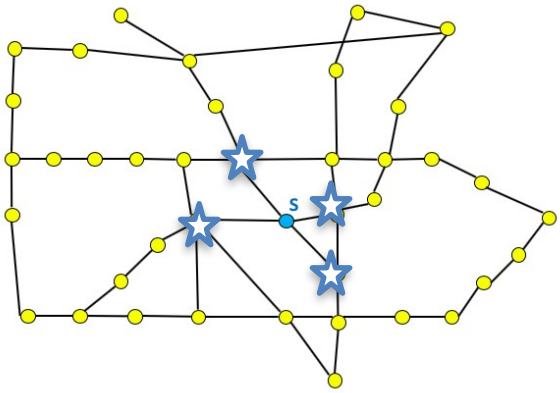


Mystery Definition

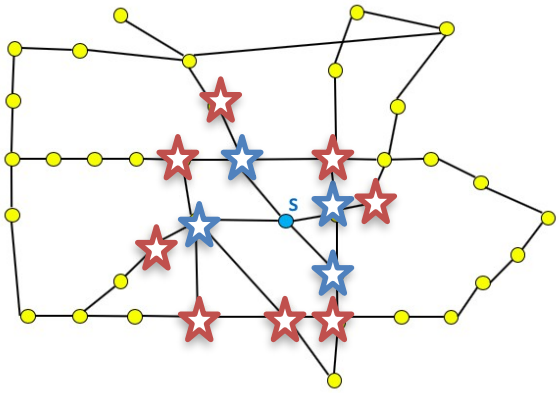
1. Here's a mystery inductive definition, pertaining to a *connected* graph $G = (V, E)$ and a node s of V.

$L_0 \leftarrow \{s\}$
$d \leftarrow 1$
while $L_{<d} \neq V$ **do** $\quad t > L_{<d}$ is defined as $\cup_{0 \le i < d} L_i$
$\quad L_d \leftarrow N(L_{d-1}) - L_{<d}$
$\quad d \leftarrow d + 1$

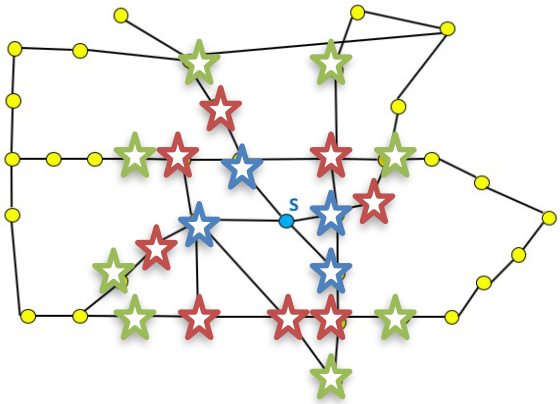
Trace through the definition by annotating the following graphs:



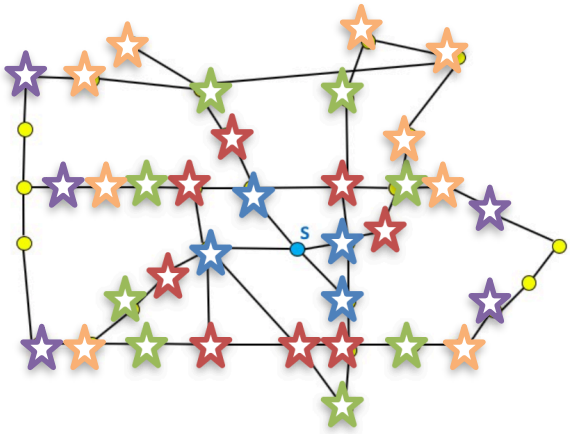
Circle the nodes in set L_1.



Circle the nodes in set $L_{<2}$ using one colour, and those in L_2 using another.



Circle the nodes in set $L_{<3}$ using one colour,
Circle the nodes in sets L_4, L_5, and so on, and those in L_3 using another.

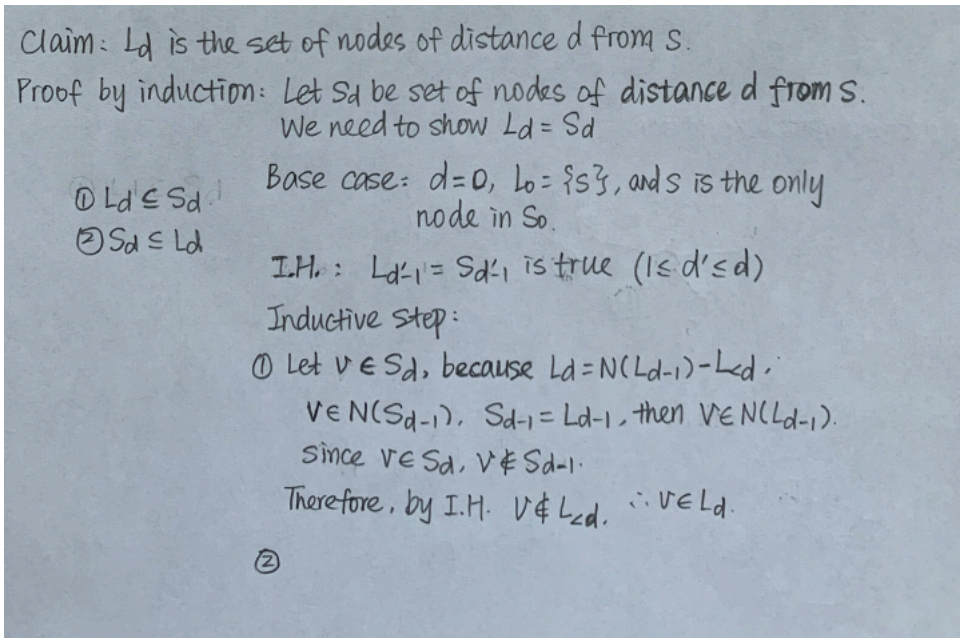


using different colours.

4. Can you describe the set L_d in a different way, using plain English?

L_d represents = { v ∈ *V: shortest path distance from s to v is d}*

5. Prove that your English definition is equivalent to the inductive definition above.



∈

6. Here is a pseudocode implementation of the mystery definition, that constructs the sets L_d.

```
procedure Construct-Level-Sets(s)
    for each v ∈ [1..n] do
        explored[v] ← false t> v is not yet in a level set
    L_0 ←{s}; explored[s] ←true; d ←1
    while L_<d /= V do
        L_d ←∅
        for each u  L_(d−1) do
            for each v adjacent to u do
                if explored[v] == false then
                    add v to L_d; explored[v] ←true

        d ←d + 1
```

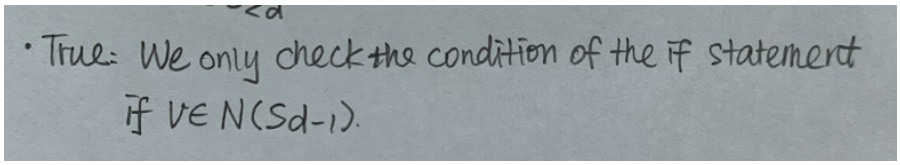
To establish that this is a correct implementation, each of the following statements attempt to relate the implementation code to the sets S_d. Recall that S_d is the set of nodes of distance d from s. If the statement is not true modify it slightly so that it is true.

- At the *start* of iteration $d \geq 1$ of the **while** loop, for any node v, explored[v] is true if and only if $v \in S_d$.

No.

- Within iteration d of the **while** loop, the **if** statement is executed if and only if $v \in N(S_{d-1})$.

True.



- When the algorithm completes, all nodes in V are explored.

Yes.

∈

7. We'll make one small change to the `Construct-Level-Sets` algorithm, to record the parent u of each node v once explored[v] becomes true:

```
procedure Construct-Level-Sets'(s)
for each v ∈ [1..n] do
    explored[v] ← false  t> v is not yet in a level set
L0 ←{s}; explored[s] ←true; d ←1
while L<d /= V do
    Ld ←∅
    for each u    Ld−1 do
        for each v adjacent to u do
            if explored[v] == false then
                add v to Ld; explored[v] ←true
                p[v] ←u                    t> this is the new line of code!
    d ←d + 1
```

8. Let P be the set containing the undirected edges $(v, p[v])$, for all $v \in V$.

   - **Explain briefly why the graph (V, P) is connected.**

   - **How many edges are in the graph (V, P)?**

     n-1

     Because every nodes has exactly one parent, except for the node s.

   - A beautiful theorem, stated in Tardos and Kleinberg (page 78, (3.2)) is that if G is an undirected graph on n nodes, then any two of the following statements implies the third.

     (i) G is connected.

     (ii) G is acyclic, i.e., is a tree.

     (iii) G has $n - 1$ edges.

     Use this theorem, plus the previous two facts to show that (V_s, P) is a tree. This is the breadth first search tree!

∈

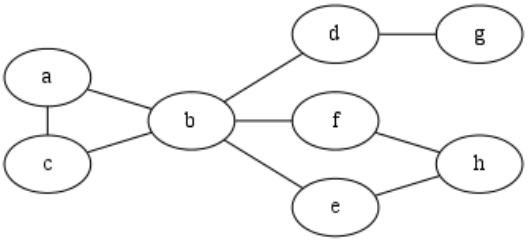
9. We'll make one final change to our algorithm, so that we compute the shortest path distance $d_s[v]$ of each node v from s. A nice side-effect is that we can use this distance array instead of the "explored" array to keep track of explored nodes. **Fill in the missing condition of the if statement.**

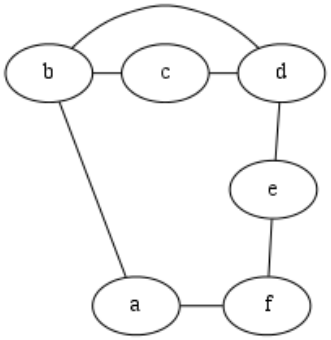
```
procedure BFS(s)
  for each v ∈ [1..n] do
      d_s[v] ←∞
  L_0 ←{s}; d_s[s] ←0; d ←1
  while L_<d / = V do
      L_d ←∅
      for each u      L_{d−1}  do
          for each v adjacent to u do

              if d_s[v] = ∞ then          t> fill in the missing condition!
                  add v to L_d; d_s[v] ←d p[v]
                  ←u
      d ←d + 1
```

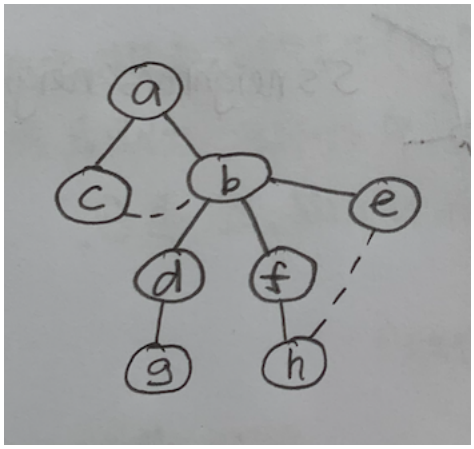
10. Suppose you want to find the route with the fewest stops from Picadilly Station to every other station of the London Underground. **How could you do it?**

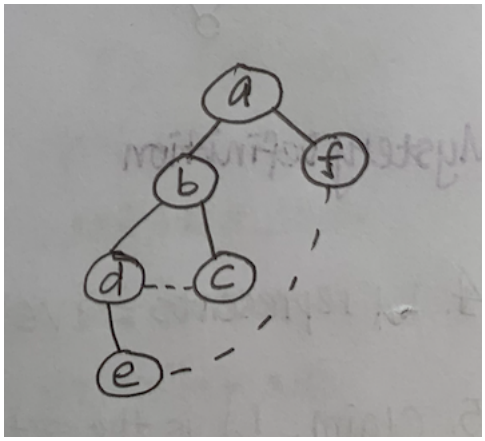
Run BFS algorithm on the underground map, and the s node is the Picadilly Station.

11. Draw the tree BFS(*a*) for each of the following graphs. Then, for each edge (*u, v*) that is *not* in the tree, connect *u* and *v* with a dashed edge.







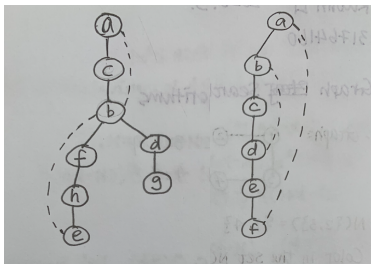


←
←

12. Here is an algorithm to construct a *depth-first search tree* of $G = (V, E)$, starting from node s.

```
procedure DFS(s)
    for each i ∈ [1..n] do explored[v] ←false
    Helper-DFS(s)

procedure Helper-DFS(u)
    explored[u]      true
    for each v adjacent to u do
        if  explored[v] is false then
            p[v]     u
            Helper-DFS(v)
```

Draw sample trees that could be produced by DFS(a) for each of the graphs of part 11. Again, for each edge (u, v) that is *not* in the tree, connect u and v with a dashed edge.



(a) **(b)**

13. More generally (i.e., not just for the examples above), consider the *level* of a node in the trees bfs(s) or dfs(s) as the distance of the node from s.

- Could a bfs tree have a dashed edge from node s to a node at level 3?

No, if there is a dashed edge between node s and node at level 3, then this node should be in the level below node s instead of level 3.

- Could a dfs tree have dashed edges between nodes at the same level?

No, because dfs will explore all the node up to the deepest level, if two nodes are in the same level, then depth first search will list one of the node first and then explore the children of this node until all the child nodes are listed, then it goes back to the remaining same level node, and connect it with solid line with the original same level node.