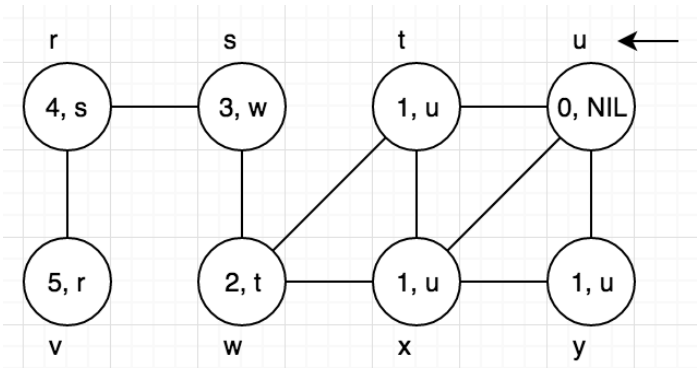
22.2-2

Vertex	d	π
u	0	NIL
t	1	u
x	1	u
y	1	u
w	2	t
s	3	w
r	4	s
v	5	r



22.2-4

If input graph is represented by adjacency matrix, then for each vertex u we de-queue, we have to check all vertices v to decide whether v is adjacent to u or not. The line 12 **for** each $v \in G.Adj[u]$ of original BFS algorithm should be modified to "for each v ∈ G.V". The runtime of this for-loop is O(V). So the total runtime is $O(V + V^2) = O(V^2)$.

22.2-8

Choose one node arbitrarily as source node, run BFS once to find a vertex ‘a’ such that a.d = max(G.V.d)
Reset all G.V.d values
Run BFS one more time with vertex ‘a’ as source node, find a vertex ‘b’ such that b.d = max(G.V.d)
The length of diameter is b.d, the diameter is from ‘a’ to ‘b’, the runtime is O(V + V) = O(V)

Proof of correctness. Note: use d(u, v) to determine the distance between u and v. Suppose the endpoints of the diameter are vertex a and vertex b. Run BFS once with any source node s from G.V

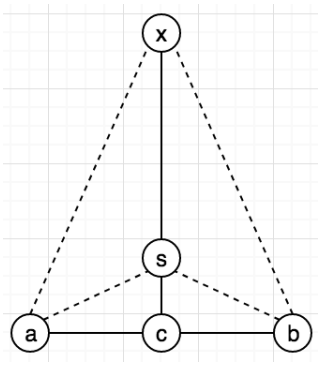
Declare lemma 1: Either a.d or b.d or both of them have the greatest d value, in other words, either a or b or both of them has greatest distance from s

Prove lemma 1 by contradiction. Suppose there exist a vertex x such that it is furthest from s. Then we have

$d(s, a) < d(s, x)$ ····· (1)

$d(s, b) < d(s, x)$ ····· (2)

Let vertex c be the one such that c is on the path from a to b, and d(s, c) is minimized. Then we have



$d(s, a) = d(s, c) + d(c, a)$ ····· (3)

$d(s, b) = d(s, c) + d(c, b)$ ····· (4)

$d(s, c) + d(c, a) + d(s, c) + d(c, b)$ ····· (5) = (3) + (4)

$= (d(c, a) + d(c, b)) + d(s, c) + d(s, c)$

$= d(a, b) + 2d(s, c)$ ·····(6)

$d(s, c) + d(c, a) + d(s, c) + d(c, b)$ ····· (7)

$= (d(s, c) + d(c, b)) + d(s, c) + d(c, a)$ ····· apply (4)

$= d(s, b) + d(s, c) + d(c, a)$ ····· apply (2)

$< d(s, x) + d(s, c) + d(c, a)$ ····· (8)

Declare lemma 2: $d(x, a) = d(x, s) + d(s, a) = d(x, s) + d(s, c) + d(c, a)$ ····· (9)

Prove lemma 2 by contradiction. Assume $d(x, a) < d(x, c) + d(c, a)$, which means there is at least one path from x to a which does not go through c. This implies that there is a cycle.

Contradiction.

Combine (5), (6) (8), (9), we get

$d(a, b) < d(a, b) + 2d(s, c) < d(s, x) + d(s, c) + d(c, a) = d(x, a)$

Contradiction, because we assume d(a, b) is maximal among all pairs.

Therefore, we can run BSF once with any source node from G.V to find one of the end point of the diameter. Then run BFS again with the endpoint as source node, we can find the other endpoint. Hence, the runtime is O(V + V) = O(V).

22.3-7

Define STACK stc in DFS(G), right before the for-loop of line 5

```
DFS-VISIT(G, u)
    stc.phsh_stack(u);
    while !stc.is_empty()
        v = stc.pop_stack();
        if v.color == GREY
            v.color = BLACK
            ++time
            v.f = time
            continue
        if v.color == WHITE
            v.color = GREY
            time++
            v.d = time
        for each w ∈ G.Adj[u]
            if w.color == WHITE
                w.π = v
                stc.push_stack(w)
```

22.4-2

For a given vectex u, u.outgoing_path is the number of path that points from u to other u's neighbors

```
number_of_simple_path(G, u, v)
    if u == v
        return 1
    else if u.outgoing_path != NIL
        return u.outgoing_path
    else
        for each w ∈ G.Adj[u]
            u.outgoing_path += number_of_simple_path(G, w, v)
        return u.outgoing_path
```

Since we have no cycles, we will never risk adding a partially completed number of paths. Moreover, we can never consider the same edge twice among the recursive calls. Therefore, the total number of executions of the for-loop over all recursive calls is O(V + E).