Assignment 7

Benjamin Jakubowski

July 14, 2016

1. BFS on figure 22.3 starting from u

Using u as a source in the graph shown in CLRS Figure 22.3 yields the following values for d and u.

Vertex	d	π
u	0	`Nil`
t	1	u
x	1	u
y	1	u
w	2	t[1]
s	3	w
r	4	s
v	5	r

[1]Note this assumes the adjacency list for $G.E[u]$ is $[t, x, y]$, such that t is the first vertex enqueued from u and as such is the parent for w.

2. Efficient algorithm to compute the diameter of a tree

An algorithm to compute the diameter of a tree is:

function DIAMETER(T)
  Pick a vertex s from T.V
  Let u be the last vertex discovered in BFS(T, s)
  Let v be the last vertex discovered in BFS(T, u)
  return d(u, v) (i.e. v.d following second BFS)

We now prove the correctness of this algorithm.
Let x and y be vertices in $T.V$ such that $d(x, y) = diam(T)$, and let s be the arbitrary vertex selected from $T.V$ to initialize the first BST.

Then let u be the last vertex discovered in BFS(T, s), such that (following the first BFS)

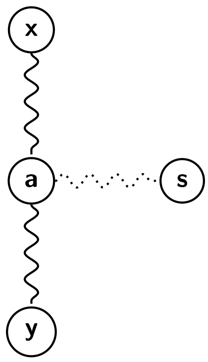
$$u.d = \max_{w \in T.V} w.d$$

The let v be the last vertex discovered in BFS(T, u), such that (following the second BFS)

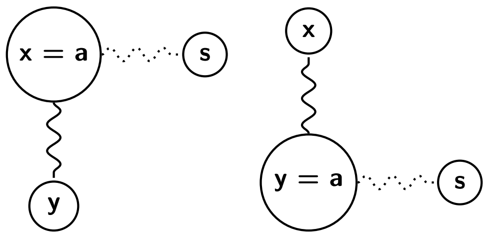
$$v.d = \max_{w \in T.V} w.d$$

Our claim is that $v.d = d(u, v) = diam(T)$. Before we begin, note since T is a tree, there is a single simple path connecting any two vertices in T. Thus, going forward, mention of paths and distances between two vertices assume this uniqueness. Additionally, note if we pick s to be either x or y, then our algorithm clearly return $d(x, y)$ and is correct. Hence, assume we pick some $s \neq x, y$.

Now consider the path between x and y, and it's relationship to s. Note the paths from s to x and from s to y must both include some a in $x \sim y$, $a \neq x, y$.

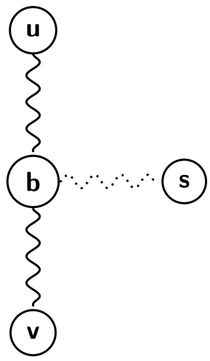


Otherwise, $diam(T) \neq d(x, y)$, since we can construct a longer path (either the path $s \sim x = a \sim y$ or $s \sim y = a \sim x$, and again these are then unique simple paths $s \sim y$ or $s \sim x$ since T is a tree).



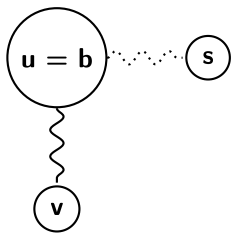
Additionally, note while we assume $s \neq x, y$, it is possible $s = a$.

Next, consider the path between u and v, and it's relationship to s. Note if $s \neq u, v$ the paths $s \sim u$ and $s \sim v$ must both include some b in the interior of $u \sim v$.

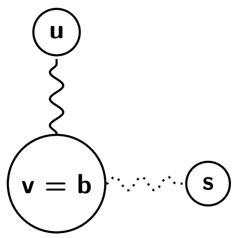


If it did not, we would have one of two contradictions:

- $d(s,u) \neq \max_{w \in T.V} d(s,w)$, since we can construct a longer path $s \sim u = b \sim v$. This contradictory path is illustrated below



- $d(u,v) \neq \max_{w \in T.V} d(u,w)$, since we can construct a longer path $u \sim v = b \sim s$. This contradictory path is also shown below.



Now we proceed to show this yields $diam(t) = d(u,v)$.
First note

$$d(s,b) + d(b,u) \geq d(s,a) + d(a,x)$$

otherwise the first BFS does not find u. Additionally,

$$d(b,v) \geq d(b,a) + d(a,y)$$

otherwise the second BFS does not find v.

But then combining these inequalities yields

$$\begin{aligned} d(s,b) + \underbrace{d(b,u) + d(b,v)}_{=d(u,v)} &\geq d(s,a) + d(b,a) + \underbrace{d(a,x) + d(a,y)}_{d(x,y)} \\ \implies \qquad d(s,b) + d(u,v) &\geq d(x,y) + d(s,a) + d(b,a) \end{aligned}$$

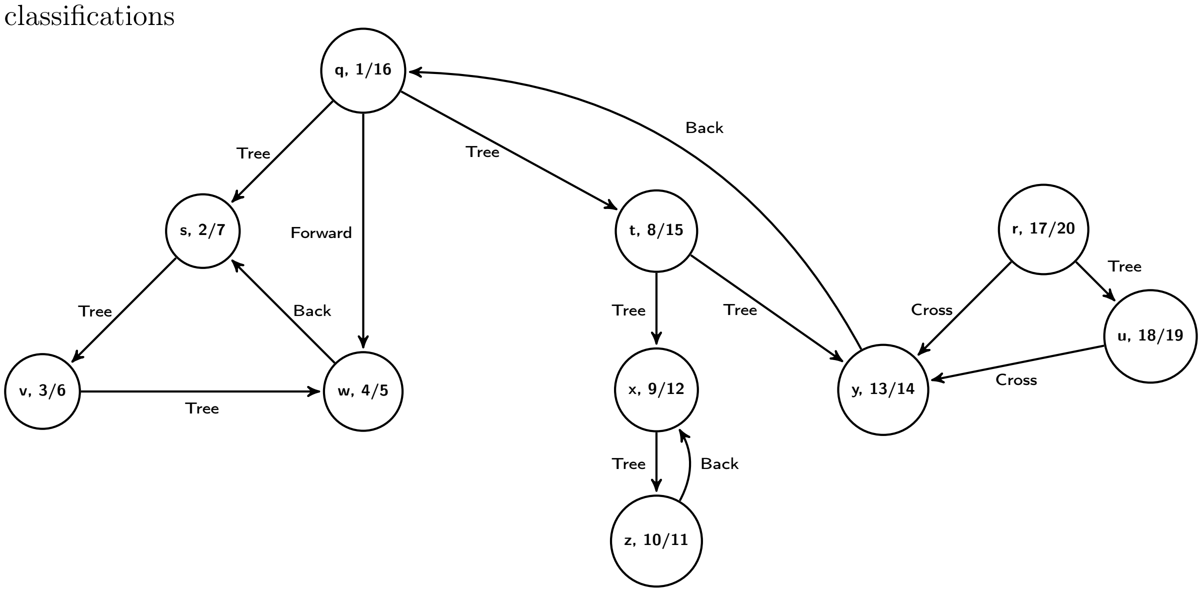
But note $d(s,b) \leq d(s,a) + d(b,a)$ since $s \sim a \sim b$ is a (potentially non-simple) path $s \sim b$.
This, we conclude

$$\begin{aligned} d(s,b) + d(u,v) &\geq d(x,y) + d(s,a) + d(b,a) \\ \implies \qquad d(u,v) &\geq d(x,y) \end{aligned}$$

But by assumption $d(x,y) = diam(T)$, so $d(x,y) = \max_{s,t \in T.V} d(s,t)$. Hence, $d(x,y) = d(u,v) = diam(T)$.
Next we consider the running time of the algorithm. Since it is involves running BFS twice, it is just $O(BFS) = O(V+E)$.

3. DFS on graph of Figure 22.6

DFS on the graph of Figure 22.6 produces the discovery and finishing times and edge classifications



4. Ordering of vertices produced by `Topological-Sort`

When `Topological-Sort` isrun on the DAG of Figure 22.8, under the assumption of Exercise 22.3-2, the following ordering is obtained:

1. p
2. n
3. o
4. s
5. m
6. r
7. y
8. v
9. x
10. w
11. z
12. u
13. z
14. t

5. Determining whether or not there's a cycle

Consider an undirected graph $G = (V, E)$. We give an $O(V)$ algorithm that determines whether or not the graph contains a cycle. We first make and prove a couple of claims:

Claim 1: Any acyclic graph has $|E| \leq |V| - 1$ edges.

First, note that the graph G clearly has a cycle if $|E| \geq |V|$. To see this, note that if a connected graph has $|E| = |V| - 1$, then G is acyclic. Therefore, a disconnected acyclic graph has $|E| < |V| - 1$, since it can be constructed by removing connecting edges from a connected acyclic graph. Thus, any acyclic graph has $|E| \leq |V| - 1$ edges.

Claim 2: If $|E| < |V|$, $\texttt{DFS}(G) = O(|V|)$

Recall DFS is $O(V + E)$. However, if $|E| < |V|$, then substitution yields

$$O(V + E) = O(V + V) = O(V)$$

With these two claims in mind, we present an $O(V)$ algorithm for detecting cycles in an undirected graph G. The algorithm essentially:

1. Checks if $|E| \geq |V|$. If so, then G has cycles
2. If not, it runs a modified DFS (which runs in $O(|V|)$- see claim 2), returning true if a cycle is found (i.e. there is a back edge from u to v).

   The algorithm is presented below:

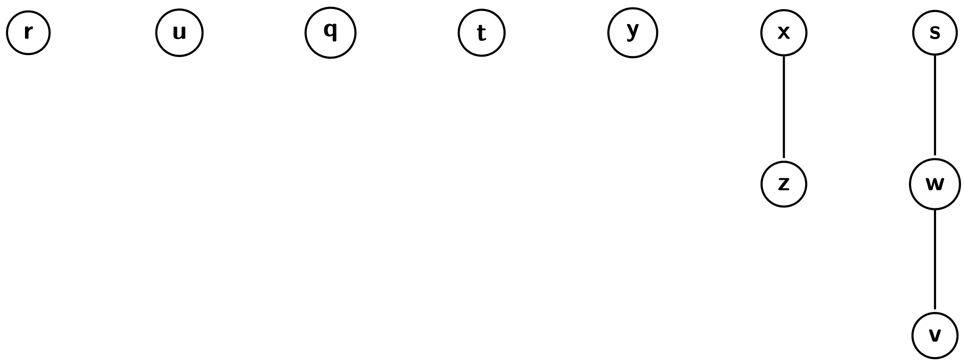
```
function HAS_CYCLES(G)
    V = 0                                          ▷ Counter for number of vertices
    E = 0                                          ▷ Counter for number of edges
    for v in G.V do
        V++
    for v in G.V do
        for e in G.E[v] do
            E++
            if E ≥ V then
                return true
    return modified_DFS(G)                         ▷ Only reached if E < V
function MODIFIED_DFS(G)
    for u in G.V do
        u.color == White
        u.π = Nil
    time = 0
    for u in G.V do
        if u.color == white then
            if modified_DFS_visit(G,u) then
                return true
    return false                                   ▷ Only reached if no cycles
function MODIFIED_DFS_VISIT(G, u)
    time = time + 1
    u.d = time
    u.color = Gray
    for v in G.E[u] do
        if v.color == White then
            v.π = u
            if modified_DFS_visit(G, v) then
                return true
        else                                       ▷ v.color != White
            if (v != u.π) or (v.π != u) then       ▷ (u,v) must complete a cycle
                return True
    u.color = black
    time = time + 1
    u.f = time
    return false
```

6. Strongly-Connected-Components on graph of Figure 22.6

First, the finishing times are given in the graph shown for problem three. The nodes finish (in decreasing order):

1. r
2. u
3. q
4. t
5. y
6. x
7. z
8. s
9. v
10. w

When DFS is run a second time, visiting the vertices in this order, we obtain the following forest of strongly connected components:



7. BFS tree edge classification

a. BFS of undirected graph

First, throughout we assume without loss of generality u is discovered before v. If not, swap the labels "u" and "v" and the proof holds.

A.1. No back edges and no forward edges

Consider a BFS tree of an undirected graph. Consider two vertices u and v connected by edge (u, v). This cannot be a forward edge, since if v is a descendant of u and there exists an edge (u, v), $v \in G.Adj[u]$ and so v is discovered in the inner loop on lines 12-17 in the BFS algorithm given on page 595 of CLRS. Thus, (u, v) must be a tree edge and not a forward edge. Finally, since we are considering an undirected graph, back edges and forward edges are the same; thus we immediately have that there are also no back edges.

A.2. A tree edge (u, v) has $v.d = u.d + 1$

Since v is in $G.Adj[u]$, v is discovered in the inner loop on lines 12-17 in the BFS algorithm given on page 595 of CLRS. Thus, on line 15 we assign $v.d = u.d + 1$.

A.3. A cross edge (u, v) has $v.d = u.d$ or $u.d + 1$

Cross edges are edges between vertices in in the BFS tree, where one vertex is not an ancestor of the other in the tree. First, note an edge (u, v) is a cross edge if an only if it is not a tree edge (since we have previously shown there are no forward or back edges in an undirected graph's BFS tree).
Since (u, v) is not a tree edge, v must have been discovered (enqueued) from some vertex $x \neq u$. This vertex x must have been enqueued and dequeued before u; otherwise, v would have been discovered from u and (u, v) would be a tree edge.
But then at some point (recalling the assumption u is discovered before v) the queue Q must have contained

$$Q = [v_{start}, \cdots, x, \cdots, u, \cdots, v, \cdots, v_{end}]$$

Then, since u and v were in the queue Q at the same time, by lemma 22.3

$$v_{start}.d + 1 \geq v_{end}.d \geq v.d \geq u.d \geq v_{start}$$

so

$$v.d = \begin{cases} u.d \\ u.d + 1 \end{cases}$$

B. BFS of directed graph

Again we assume without loss of generality u is discovered before v.

B.1. No forward edges

Consider a BFS tree of a directed graph. Consider two vertices u and v connected by edge (u, v). This cannot be a forward edge, since if v is a descendant of u and there exists an edge (u, v), $v \in G.Adj[u]$ and so v is discovered in the inner loop on lines 12-17 in the BFS algorithm given on page 595 of CLRS. Thus, (u, v) must be a tree edge and not a forward edge.

B.2. A tree edge (u, v) has $v.d = u.d + 1$

Since v is in $G.Adj[u]$, v is discovered in the inner loop on lines 12-17 in the BFS algorithm given on page 595 of CLRS. Thus, on line 15 we assign $v.d = u.d + 1$.

B.3. A cross edge (u, v) has $v.d \leq u.d + 1$

Again, cross edges are edges between vertices in in the BFS tree, where one vertex is not an ancestor of the other in the tree. Let (u, v) be a cross edge. Then again since (u, v) is not a tree edge, v must have been discovered before u was dequeued; otherwise, v would have been discovered from u and (u, v) would be a tree edge.
But then $v.d \leq u.d + 1$, since at the time u is dequeued v is either

- **Case 1**: In the queue, but then by lemma 22.3 u and v were in the queue at the same time (with v after u), so $v.d \leq u.d + 1$.
- **Case 2**: Already out of the queue, but then clearly by the monotonicity of d values in the queue $v.d \leq u.d$.

In either case, $v.d \leq u.d + 1$.

B.4. A back edge (u, v) has $0 \leq v.d \leq u.d$

if (u, v) is back edge, then v must have been discovered before u; otherwise, when u is dequeued v would still be white, and then v would be discovered from u, making (u, v) a tree edge.
But then at the time u is enqueued, by corollary 22.4, we clearly have $v.d \leq u.d$. Finally, note for all $w \in V, w.d \geq 0$. Thus, we have

$$0 \leq v.d \leq u.d$$