Fundamental Algorithms
CSCI-GA.1170-001/Summer 2016

Solution to Homework 7

Problem 1 (CLRS 22.2-2). (1 point) Show the d and π values that result from running breadth-first search on the undirected graph of Figure 22.3, using vertex u as the source.

Solution: Running BFS on the graph of Figure 22.3 gives:

u	$u.d$	$u.\pi$
r	4	s
s	3	w
t	1	u
u	0	`null`
v	5	r
w	2	t
x	1	u
y	1	u

Problem 2 (CLRS 22.2-8). (3 points) The *diameter* of a tree $T = (V, E)$ is defined as $\max_{u,v \in V} \delta(u, v)$, that is, the largest of all shortest-path distances in the tree. Give an efficient algorithm to compute the diameter of a tree, prove the algorithm correct, analyze the running time.

Solution: We first note that in a tree (an acyclic connected graph), any two vertices are connected by exactly one path, and thus $\delta(u, v) = d(u, v)$ for all u and v. Our main tool for finding shortest paths is BFS. Intuitively, the closer BFS starts to the center of the graph, the more it can underestimate the diameter. However, if we first find the farthest vertex (a leaf node in the tree) and run a second BFS from there, the longest of the resulting paths should correspond to the diameter of the tree.

Assuming the existence of procedure `BFS(T, v)`, performing breadth-first search of tree `T` from vertex `v` and returning the last discovered vertex, the algorithm would run in $O(V + E)$ (same as BFS) and could be coded as follows:

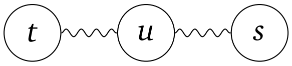
```
def tree-diameter(T):
    s = arbitrary-vertex(T)
    u = BFS(T, s)
    v = BFS(T, u)
    return distance(u, v)
```

Let us prove the algorithm correct.

Let the first BFS run from vertex s with vertex u discovered last.
Let the second BFS run from vertex u with vertex v discovered last.
Let a and b be any two nodes such that $d(a, b)$ is the diameter of T.
Let t be the first node on $a \rightsquigarrow b$ (the path from a to b) discovered by the first BFS.

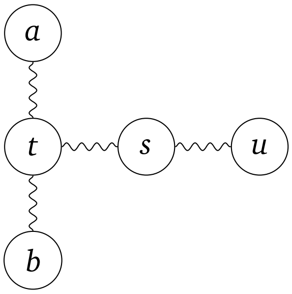
We first show that $d(u,b) = d(a,b)$. Consider three possible cases:

1. s is not on $t \rightsquigarrow u$ and t is not on $s \rightsquigarrow u$:

   But then u would not be the last vertex discovered.

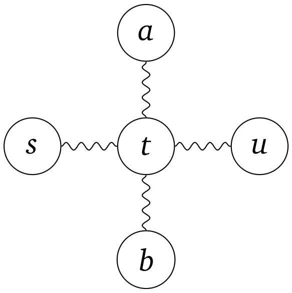
2. s is on $t \rightsquigarrow u$:

   Then $d(t,u) \geq d(s,u)$. At the same time, $d(s,u) \geq d(s,a)$, as otherwise u would not be the last vertex discovered. $s \rightsquigarrow a$ includes $t \rightsquigarrow a$, and so $d(s,a) \geq d(t,a)$. Thus, $d(t,u) \geq d(t,a)$ and, adding $d(b,t)$ to both sides, $d(b,u) \geq d(b,a)$.

   But $d(a,b)$ is a diameter, so $d(b,u) \leq d(a,b)$. Thus, $d(u,b) = d(a,b)$.

3. t is on $s \rightsquigarrow u$:

   $d(t,u) \geq d(t,a)$, as otherwise u would not be the last vertex discovered. At the same time, $d(t,a) \geq d(t,u)$, as otherwise diameter $d(b,a) < d(b,u)$. Thus, $d(t,u) = d(t,a)$ and, adding $d(b,t)$ to both sides, $d(u,b) = d(a,b)$.

$d(u,v) \leq d(a,b)$ because $d(a,b)$ is a diameter, and $d(u,v) \geq d(u,b)$ because otherwise v would not be the last vertex discovered by the second BFS. Thus, $d(u,b) \leq d(u,v) \leq d(a,b)$.

Recall that we showed $d(u,b) = d(a,b)$. Thus, $d(a,b) \leq d(u,v) \leq d(a,b)$ or $d(u,v) = d(a,b)$.

Problem 3 (CLRS 22.3-2). (1 point) Show how depth-first search works on the graph of Figure 22.6. Assume that the `for` loop of lines 5-7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.

Solution: We know that when DFS first explores an edge (u, v), the color of v and discovery time of u and v allow us to determine edge type:

- White v indicates a tree edge.
- Gray v indicates a back edge.
- Black v with $u.d < v.d$ indicates a forward edge.
- Black v with $u.d > v.d$ indicates a cross edge.

Running DFS on the graph of Figure 22.6 gives the following vertex information:

u	$u.d$	$u.f$	$u.\pi$
q	1	16	`null`
r	17	20	`null`
s	2	7	q
t	8	15	q
u	18	19	r
v	3	6	s
w	4	5	v
x	9	12	t
y	13	14	t
z	10	11	x

And the following classification of edges:

Edge	Type
(q, s)	tree
(q, t)	tree
(q, w)	forward
(r, u)	tree
(r, y)	cross
(s, v)	tree
(t, x)	tree
(t, y)	tree
(u, y)	cross
(v, w)	tree
(w, s)	back
(x, z)	tree
(y, q)	back
(z, x)	back

Problem 4 (CLRS 22.4-1). (1 point) Show the ordering of vertices produced by `Topological-Sort` when it is run on the dag of Figure 22.8, under the assumption of Exercise 22.3-2.

Solution: Running DFS on the graph in Figure 22.8 gives:

u	$u.d$	$u.f$	$u.\pi$
m	1	20	null
n	21	26	null
o	22	25	n
p	27	28	null
q	2	5	m
r	6	19	m
s	23	24	o
t	3	4	q
u	7	8	r
v	10	17	y
w	11	14	v
x	15	16	v
y	9	18	r
z	12	13	w

With the associated topological ordering: $p, n, o, s, m, r, y, v, x, w, z, u, q, t$.

Problem 5 (CLRS 22.4-3). (2 points) Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of $|E|$.

Solution: Consider the following points:

- By Theorem 22.10, in a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge. Finding a back edge indicates a cycle immediately. Finding no back edges indicates that all edges are tree edges. Recall that tree edges are edges in the depth-first forest and thus cannot form a cycle, and a graph with only tree edges is acyclic.
- When DFS first explores an edge (u, v), gray v indicates a back edge. Note that in an undirected graph we need to treat (u, v) and (v, u) as the same edge.
- By Theorem B.2, in an acyclic undirected forest, $|E| \leq |V| - 1$ (with $|E| = |V| - 1$ indicating a connected graph). Thus, the algorithm will never process more than $|V|$ distinct edges before running out of edges or finding a back edge, and the running time is $O(V)$.
- We don't need to keep track of predecessors and discovery/finishing times, but we need a way to tell whether we are seeing an edge for the first time.

This approach can be realized in the following modification of the DFS algorithm:

```
def has_cycle(G):
    for u in G.V:
        u.color = white
    seen = {}
    for u in G.V:
        if u.color == white:
            if has_cycle_visit(G, u):
                return true
    return false

def has_cycle_visit(G, u):
    u.color = gray
    for v in G.adj[u]:
        if (u, v) not in seen:
            seen.add((u, v))
            seen.add((v, u))
            if v.color == gray:
                return true

        if v.color == white:
            if has_cycle_visit(G, v):
                return true

    u.color = black
    return false
```

Problem 6 (CLRS 22.5-2). (1 point) Show how the procedure `Strongly-Connected-Components` works on the graph of Figure 22.6. Specifically, show the finishing times computed in line 1 and the forest produced in line 3. Assume that the loop of lines 5-7 of DFS considers vertices in alphabetical order and that the adjacency lists are in alphabetical order.

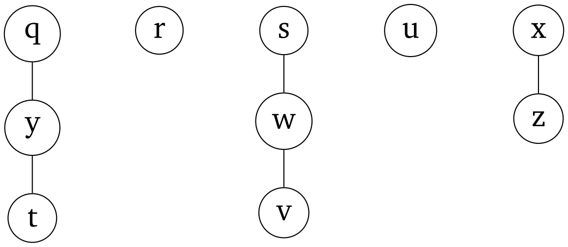
Solution: Running DFS on the graph of Figure 22.6 gives (same as in Problem 3):

u	$u.d$	$u.f$	$u.\pi$
q	1	16	`null`
r	17	20	`null`
s	2	7	q
t	8	15	q
u	18	19	r
v	3	6	s
w	4	5	v
x	9	12	t
y	13	14	t
z	10	11	x

DFS on the transposed graph gives:

u	$u.d$	$u.f$	$u.\pi$
q	5	10	`null`
r	1	2	`null`
s	15	20	`null`
t	7	8	y
u	3	4	`null`
v	17	18	w
w	16	19	s
x	11	14	`null`
y	6	9	q
z	12	13	x

Or, graphically:



Problem 7 (CLRS 22-1). (2 points) A depth-first forest classifies the edges of a graph into tree, back, forward, and cross edges. A breadth-first tree can also be used to classify the edges reachable from the source of the search into the same four categories.

Solution:

(a) Prove that in a breadth-first search of an undirected graph, the following properties hold:

1. There are no back edges and no forward edges.

   A forward edge (u, v), by definition, connects vertex u to a descendant v in the breadth-first tree. BFS explores all edges of u, including (u, v), before exploring edges of any descendants of u, so (u, v) must be a tree edge. Analogous argument (with u and v reversed) can be made for back edges.

2. For each tree edge (u, v) we have $v.d = u.d + 1$.

   (u, v) is a tree edge when BFS sets $v.\pi = u$. At the same time the algorithm sets $v.d = u.d + 1$ and never changes $v.d$ and $u.d$ again.

3. For each cross edge (u, v), we have $v.d = u.d$ or $v.d = u.d + 1$.

   Consider the BFS queue right before dequeuing u. (u, v) being a cross edge indicates that at this time v has already been enqueued (otherwise (u, v) would be a tree edge) and thus u, the head of the queue, is coming before v in the queue. If v_r is the tail of the queue, by Lemma 22.3: $v.d \leq v_r.d \leq u.d + 1$ or $v.d \leq u.d + 1$. At the same time,

u has been enqueued before v, and so by Corollary 22.4: $u.d \leq v.d$. Combining both results shows that either $v.d = u.d$ or $v.d = u.d + 1$.

(b) Prove that in a breadth-first search of a directed graph, the following properties hold:

1. There are no forward edges.

   Forward edges occur when DFS encounters edge (u, v), where v has been discovered after u, but has been explored from a vertex other than u. This cannot happen in BFS, as all neighbors of u, including v, would have been explored from u, making (u, v) a tree edge.

2. For each tree edge (u, v), we have $v.d = u.d + 1$.

   (u, v) is a tree edge when BFS sets $v.\pi = u$. At the same time the algorithm sets $v.d = u.d + 1$ and never changes $v.d$ and $u.d$ again.

3. For each cross edge (u, v), we have $v.d \leq u.d + 1$.

   $v.d > u.d + 1$ would only be possible if the algorithm does not explore edge (u, v) when visiting u, which cannot happen.

4. For each back edge (u, v), we have $0 \leq v.d \leq u.d$.

   The distances cannot be negative, so $v.d \geq 0$ for all vertices v. A back edge (u, v), by definition, connects vertex u to an ancestor v in the breadth-first tree, and v cannot be farther from the root than its descendant u.