Algorithms (6470) HW05

Alex Darwiche

July 24, 2024

Answers

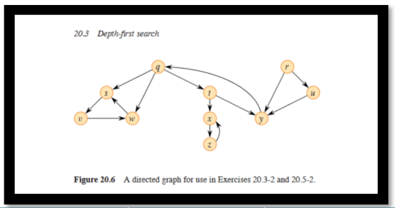
Q1

(a) Yes, the $u.\pi$ can change for some $u \in V$. Assuming node u is at depth i. 2 nodes at depth $i-1$ could both have an edge that connects to node u. The $u.\pi$ thus depends on which of those nodes you encounter first in your BFS. Thus, the order of the adjaceny matrix CAN affect $u.\pi$

(b) No, $u.d$ cannot change. A BFS looks through all the nodes away from the source, at each depth away from the source. This means that a BFS, even with ordering changed, will never change the $u.d$ of a node, because it would have found the shortest path in the first place, regardless of order.

Q2

(a) DFS on Figure 20.6:


20.3 Depth-first search
q
r
s
t
u
v
w
x
y
z
Figure 20.6 A directed graph for use in Exercises 20.3-2 and 20.5-2.

Node	color	u.d	u.f	u.pi
q	Black	1	16	NIL
r	Black	16	19	NIL
s	Black	2	7	q
t	Black	8	15	q
u	Black	17	18	r
v	Black	3	6	s
w	Black	4	5	v
x	Black	9	12	t
y	Black	13	14	t
z	Black	10	11	x

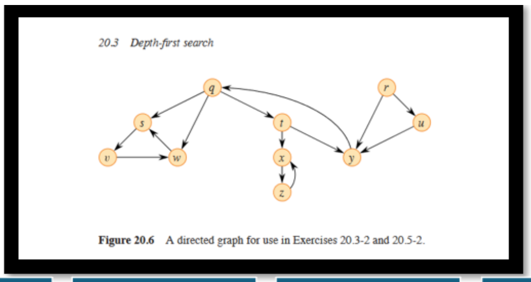

20.3 Depth-first search

Figure 20.6 A directed graph for use in Exercises 20.3-2 and 20.5-2.

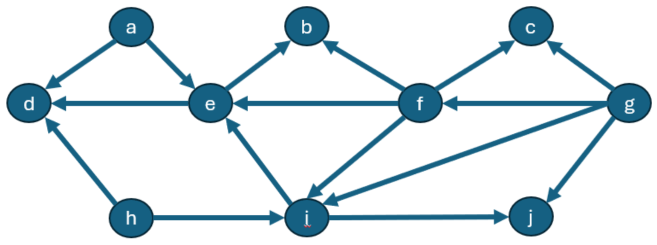
Tree Edge	Back Edge	Forward Edge	Cross Edge
qs	yq	qw	uy
sv	zx		ry
vw	ws		
qt			
tx			
xz			
ty			
ru			

Q3

(a) This will always be true. If there is a path between nodes u and v, and u.d $<$ v.d, then depth first search will search through all of u's child nodes until it explores the child that eventually connects to v. This means that v will always be a descendant of u, given the above assumptions.

Q4

(a) Topological Sort - alphabetical order

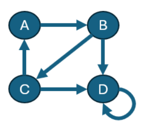


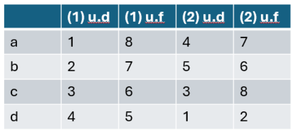
a,d,e,b,c,f,i,j,g,h

Q5

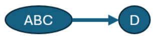
(a) The first G1 shows how this algorithm can find the strongly connected components in a graph and distinguish them from other components. However, the second example illustrates how this cannot always work. The node B and C cannot reach one another if you begin at node C. This is not shown in the tables however, and illustrates why this algorithm does not work for all graphs.

G1

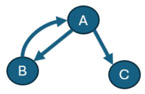


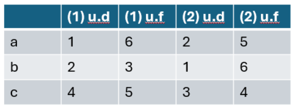


	(1) u.d	(1) u.f	(2) u.d	(2) u.f
a	1	8	4	7
b	2	7	5	6
c	3	6	3	8
d	4	5	1	2

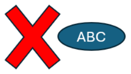


G2



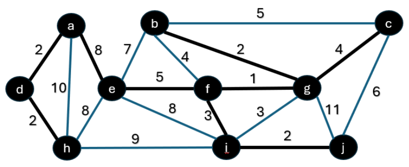


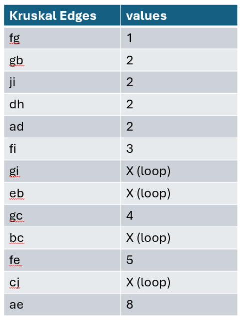
	(1) u.d	(1) u.f	(2) u.d	(2) u.f
a	1	6	2	5
b	2	3	1	6
c	4	5	3	4



Q6

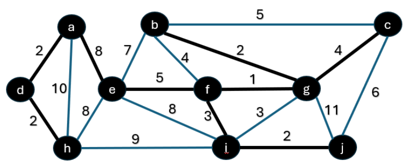
(a) Kruskal Algorithm

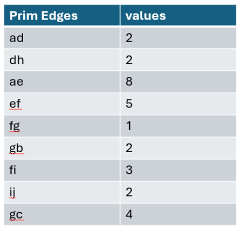




Kruskal Edges	values
fg	1
gb	2
ji	2
dh	2
ad	2
fi	3
gi	X (loop)
eb	X (loop)
gc	4
bc	X (loop)
fe	5
cj	X (loop)
ae	8

(b) Prim Algorithm





Prim Edges	values
ad	2
dh	2
ae	8
ef	5
fg	1
gb	2
fi	3
ij	2
gc	4

Q7

(a) To prove that an minimum weighted edge between u and v is part of some minimum spanning tree of G, I will use contradiction.

(1) Assumption: There is a minimum spanning tree that does not include the edge from u to v, called S1.

(2) Next, we will add in the edge from u to v to this tree.

(3) This will now form a cycle in the graph.

(4) To remove a loop, we first can look at all the different edges that make up the loop. A loop implies that every vertex in the loop has at least 2 edges coming from it. To remove this loop, we simply need to remove one of the edges that makes up said loop. To do this and maintain minimum spanning, we should remove the edge with the highest weight.

(5) Now we remove an edge other than the one from u to v.

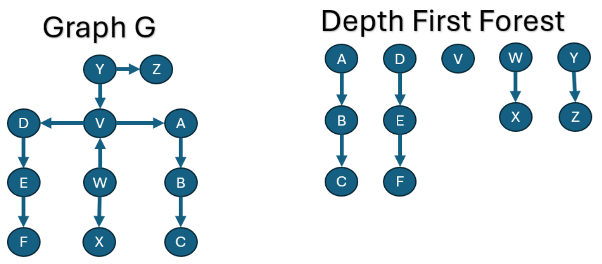
(6) This leaves us with 2 different spanning trees. S1 and S2, where S2 is the minimum spanning tree that now includes the edge from u to v.

(7) weight(S1) - weight(S2) $>$ 0 or weight(S1) $>$ weight(S2) (THIS IS A CONTRADICTION)

(8) This shows a contradiction as you can find a tree with a lower weight by including the edge from u to v. So by contradiction, the edge from u to v must be in some minimum spanning tree of graph G.

Q1 (Graduate students only)

(a) Graph G



Q2 (Graduate students only)

(a) This proof will be quite similar to the one used earlier in question 7.

(1) First, assume that there exists a minimum spanning tree for graph G.

(2) Now, we want to remove and edge and add an edge to this minimum spanning tree.

(3) Case 1: Adding this edge increased the weight of the tree. This means that the new tree is not "minimum" spanning tree.

(4) Case 2: Adding this edge decreased the weight of the tree. This means that the old tree is not "minimum" spanning tree.

(5) In either case, we are able to conclude that once we've absolutely minimized the weight on the minimum spanning tree, any changes to the tree will only increase the weight of it, thus making it not minimum spanning. This means that there is only 1

(b) For a contradiction proof.

(1) Assume MST2 exist with an edge that is not in MST1, such that w(MST2) = w(MST1).

(2) We will add an edge and remove an edge to form another spanning tree.

(3) Case 1: We add an edge that increases the weight of MST2, such that w(MST2) > w(MST1)

(4) Case 2: We add an edge that decreases the weight of MST2, such that w(MST2) < w(MST1)

(5) Both cases are a contradiction, thus there is only 1 minimum spanning tree when all edges are unique.