600.363/463
Solution to Midterm Exam 2
11/24/2003

1. (15 points)

Give an algrithm 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 $|E|$.

Answer:
An undirected graph is acyclic (i.e., a forest) iff a DFS yields no back edges. Since back edges are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree, so no back edges means there are only tree edges, so there is no cycle.
So we can simply fun DFS. If find a back edge, there is a cycle. The complexity is $O(V)$ instead of $O(E + V)$. Since if there is a back edge, it must be found before seeing $|V|$ distinct edges. This is because in a acyclic (undirected) forest, $|E| \leq |V| + 1$

2. (15 points)

Prove or disaprove (by counter-example) the following conjectures about finish numbering $f(u)$ of vertices u after DFS in a directed graph $G = (V, E)$.

a. $v = arg(\min f(u))$ is loacted at the strongly connected component (SCC) which is a sink of the strongly connected component graph.

Answer:
Wrong. Consider such a graph that has edges and vertices: $a \to b \to c \to a$ and $a \to d$. the SCC graph should be $(abc) \to d$. But DFS may make $f(c)$ be the minimum value and c is not located on the sink of SCC graph.

$b.w = arg(\max f(u))$ is loacted at the strongly connected component (SCC) which is a source of the strongly connected component graph.

Answer:
Right. Lemma 22.14 says: If C and C' are distinct strongly connected components in directed graph $G = (V, E)$, and there is an edge $(u, v) \in E$, where $u \in C$, and $v \in C'$, then $f(C) > f(C')$.
So suppose $w \in C$. since $f(w)$ is the maximum among all the edges, so

$f(C) > f(C')$ for any other SCC C' in the SCC graph. So there is no edge from other SCC to C in the SCC graph. This means C is a source.

c. Which vertex (v or w) should be used for discovery of strongly connected componens? based this on answers from a and b.

Answer:
w should be used for discovery of strongly connected component.

3.(10 points)

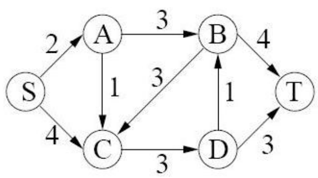
Design an efficient algorithm to find a spanning tree for a connected weighted undirected graph $G = (V, E)$ such that the weight of the maximum-weight edge in the spanning tree is minimized. Prove its correctness.

Answer:

The Minimum Spanning Tree is actually what we need. Suppose the maximum weighted value of the MST is e_1, and the maximum weighted edge of another spanning tree T is e_2. If $w(e_1) > w(e_2)$, we can remove e_1 from MST and cut the graph into two parts. From T, we can find an edge e^* linking these two parts. As e_2 is the maximum weighted edge in T, $w(e^*) \leq w(e_2) < w(e_1)$. Remove e_1 from MST and add e^* into it, we get a new spanning tree T^*, and the weight of T^* is less than MST. This is a contradiction. So it is proved that the MST's maximum weighted edge is minimized.

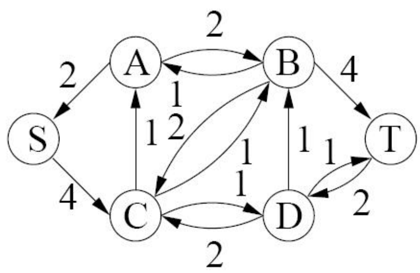
4.

In the flow network illustrated below, each directed edge is labeled with its capacity. We are using the Ford-Fulkerson algorithm to find the maximum flow. The first augmenting path is S-A-C-D-T, and the second augmenting path is S-A-B-C-D-T.



a. Draw the residual network after we have updated the flow using these two augmenting paths (in the order given).

Answer:

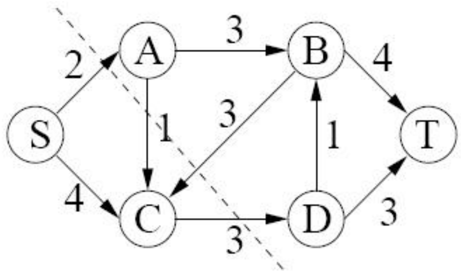


b. List all of the augmenting paths that could be chosen for the third augmentation step.

Answer:
$S \to C \to A \to B \to T$, $S \to C \to B \to T$, $S \to C \to D \to B \to T$, and $S \to C \to D \to T$.

c. What is the numerical value of the maximum flow? Draw a dotted line through the original graph to represent the minimum cut.

Answer:



5. (27 points)

Let $G = (V, E)$ be a flow network with source s, sink t, and suppose each edge $e \in E$ has capacity $c(e) = 1$. Assume also, for convenience, that $|E| = \Omega(V)$.

a. Suppose we implement the Ford-Fulkerson maximum-flow algorithm by using depth-first search to find augmenting paths in the residual graph. What is the worst-case running time of this algorithm on G?

Answer:
The running time is: $O(VE)$ because: the running time is $O(E|f^*|)$, and $|f^*| = O(V)$ because the flow is bounded by thbe capacity of any cut, and the capacity of the cut $(s, V - s)$ is $O(V)$ because there are $O(V)$ edges leaving s, and every edge has capacity=1.

b. Supose a maximum flow for G has been computed, and a new edge with unit capacity is added to E. Describe how the maximum flow can be efficiently updated. (*Note:* It is not the value of the flow that must be updated, but the flow itself.) Analyze your algorithm.

Answer:
Just execute one more iteration of the Ford-Fulkerson algorithm. The new edge in E adds a new edge to the residual graph, so look for an augmenting path and update the flow if a path is found.
The time is $O(V + E) = O(E)$ if find path with depth-first or breadth-first search.
Only 1 iteration is needed because adding the capacity-1 edge increases the capacity of the min cut by at most 1. Thus the mas flow (which = the min cut capacity) increases by at most 1. Since all edge capacities are 1, any augmentation increases flow by 1, so only 1 augmentation can be needed.

c. Suppose a maximum flow for G has been computed, but an edge is now removed from E. Describe hjow the maximum flow can be efficiently updated. Analyze your algorithm.

Answer:
Let the removed edge by (u, v).
If (u, v) has no flow, we don't need to do anything.
If (u, v) has flow, the network flow must be updated.
There is now 1 more unit of flow coming into u than is going out (and 1 more unit going out of v than is coming in). Te idea is to try to reroute this unit of flow so that it goes out of u and into v via some other path. If that is not possible, we must reduce the flow from s to u and from v to t by 1 unit.
So look for an augmenting path from u to v. (*Note:* not from s to t)
If there is such a path, augment the flow along that path.
If there is no such path, reduce the flow from s to u by augmenting the flow from u to s. That is, find an augmenting path $u \leadsto s$ and augment the flow along that path. (There definitely is such a path, because there is flow from s to u.) Similarly augment the path from t to v.
THe time is: $O(V + E) = O(E)$ if finding path with DFS or BFS.

6. (18 points)

Provide a polynomial time algorithm to find a negative weight cycle in a directed weighted graph that has negative edges.

Answer:
The Bellman-Ford algorithm can detect whether there is any negative weight cycle. It will return false when there is negative weight cycle. So we can use Bellman-Ford to help find the negative weight cycle.
Frist run the Bellman-Ford. If it returns true, there is no negative weight cycle.
If it returns false, there are negative weight cycles in the graph. On line 5-7 of Bellman-ford, when it return false, it finds an edge (u, v) that satisfies $d[v] > d[u] + w(u, v)$. So if we trace u's parent So vertex u must be on the negative weight cycle. We may find the negative cycle by looking at u's parent, which is $\pi(u)$). We keep tracing the vertex's parent until a negative weight cycle is found.