Assignment #5, CS4/531

Due Date: Tue. Nov. 29, 2010

UNSUPPORTED SOLUTIONS RECEIVE NO CREDIT.
Total points: 46

1. (5 points) Find the strongly connected components of the graph $G = (V, E)$ in Fig. 1, by using the DFS based algorithm in section 22.5. Your solution should provide the following:

- Run DFS on G. Indicate the DFS trees constructed by DFS algorithm; indicate the $d[u]$ and $f[u]$ values for each vertex $u \in V$.
- Draw the graph G^T. Label the vertices in the reverse order of $f[u]$ values.
- Run DFS algorithm on G^T. Indicate the DFS trees constructed by the DFS algorithm.
- Indicate the strongly connected components of G.

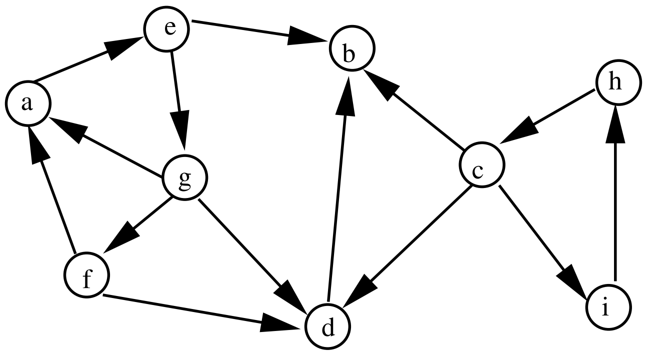

Figure 1: Graph for Strongly Connected Components Problem.

2. (6 pts) Consider the graph in Figure 2. Suppose that G is given by adjacency list representation where, for each vertex v, the neighbors of v in Adj(v) are in **alphabetical order**. Run the DFS-based biconnectivity algorithm on G with a as the starting vertex. You should:

1. Run DFS algorithm on G; indicate the DFS tree constructed;
2. Rename the vertices of G by integers in the order they are visited by the DFS algorithm;
3. Compute low(v) for each vertex v;
4. List the cut vertices found by the algorithm.

3. (3 pts) Run Dijkstra's algorithm on the directed graph in Figure 3, using the vertex a as the source. In the style of Figure 24.6 (page 596), show the $d[*]$ and the $\pi[*]$ values and the vertices in the set S after each iteration of the **while** loop.

4. (0 pts, so Don't hand-in the solution of this problem)
Run FASTER-ALL-PAIRS-SHORTEST-PATHS algorithm (page 691) on the directed graph in Fig 4. Your solution should show the matrices that result from each iteration of the loop.

5. (2+5 = 7 pts) Consider the flow network G shown in Fig. 5.

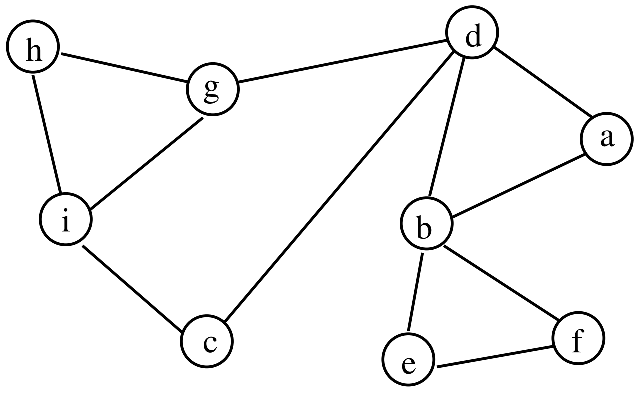

Figure 2: Graph for Biconnectivity Algorithm

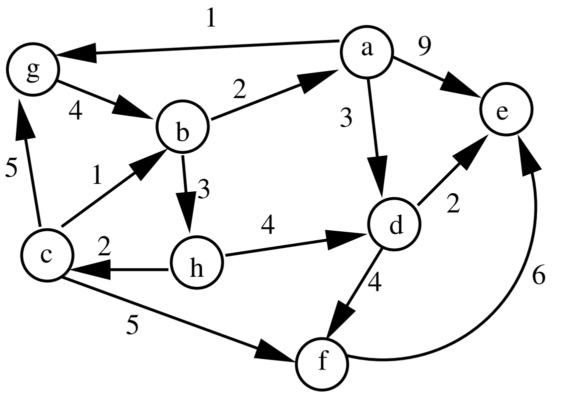

Figure 3: Graph for Dijkstra's Algorithm

1. What is the cut $(\{s, a, b\}, \{c, d, e, t\})$. What is the capacity of this cut?

2. Show the execution of Edmond-Karp algorithm on this network. Compute the max-flow of G. Show the progress of the algorithm after each iteration, (as shown in Figure 26.5 on Page 659).

6. (7 pts) We have three jars whose sizes are 4 pints, 7 pints and 10 pints respectively. The 4-pint and 7-pint jars start out full of water, the 10-pint jar is initially empty. We are allowed only one type of operation: pouring the contents of one jar into another, stopping only when the source jar is empty *or* the destination jar is full. We want to know if there exists a sequence of pouring that leaves exactly 2 pints water in the 7-pint jar or in the 4-pint jar.

This is a special case of a more general problem: we have k jars. The jar$_i$ has capacity a_i pints ($1 \leq i \leq k$, each a_i is a positive integer). Initially, the jar$_1$,jar$_2$,...,jar$_t$ ($1 \leq t < k$) are full, and the other jars are empty. We are allowed only the operation as above. We want to know if these exists a sequence of pouring that leaves b_i (where $1 \leq i \leq k$ and $b_i \leq a_i$) pint water in the jar$_i$.

Formulate this problem as a graph problem and describe how to solve it. More specifically, you should:

- Define a graph $G = (V, E)$ that describes the problem. The vertices and the edges of G may have labels. If so, describe the labels and their meanings.

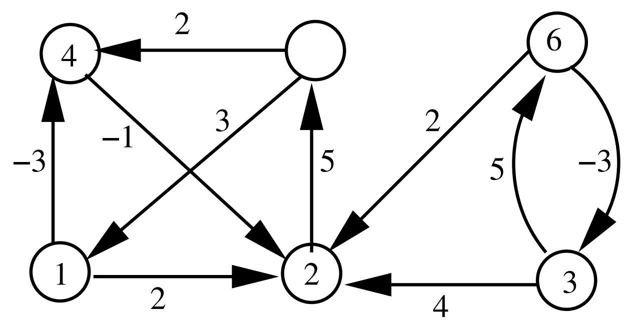

Figure 4: All Pairs Shortest Paths Problem

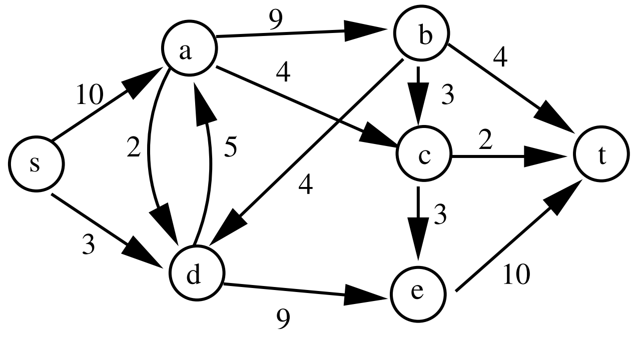

Figure 5: Max-Flow Problem

- Describe an algorithm for solving the problem. If an algorithm studied in class can be used, specify which one, and exactly how to use it.
- Analyze the run time.

You may describe your solution either for the general case, or for the special three jar case. (If you do the latter, the idea of your solution must be applicable to the general case.)

7. (6 pts) The CSE curriculum consists of n course, all of them are required. The *prerequisite graph* $G = (V, E)$ is a directed graph. Each vertex of G represents a course. If $i \to j \in E$, then the course i is a prerequisite of the course j (so you must complete the course i before you can take the course j). We assume G is a DAG (directed acyclic graph). (Without this assumption, no one can graduate!).

Describe an algorithm that, with the adjacency list representation of G as the input, computes the minimum number of semesters necessary to complete the curriculum. The run time of the algorithm should be $\Theta(n + m)$.

Note: Here we assume that a student can take any number of courses in any semester as long as the prerequisite requirements are satisfied. If there is a limit on the maximum number of courses she can take in one semester (say $k = 6$), the problem becomes much harder.

8. (5 pts) Let $T = (V, E)$ be a tree. In other words, T is an undirected graph, it is connected and contains no cycles. T is given by adjacency list representation. A vertex r of V is designated as the root. Then T becomes a rooted tree. Recall that a vertex u is an *ancestor* of another vertex v, if and only if u is on the path from r to v in T.

In some applications, we need to answer queries of the form "is u an ancestor of v" very often. In order to do this efficiently, you should:

- Preprocess the tree T in $O(n+m)$ time, to compute necessary information.
- Answer any query ("is u an ancestor of v"?) in $O(1)$ time by using information computed in the preprocessing.

Describe how this can be done.

9. (7 pts) Let $G = (V, E)$ be a dag (directed acyclic graph). Each vertex of G is either a computer, or a server. All servers are sinks (namely, out-degree $= 0$.) Each edge of G is a communication link (from a computer to either another computer, or to a server). We need to *secure some edges* so that the following conditions hold.

- For every pair c_i, s_j (where c_i is a computer and s_j is a server) such that there is a directed path from c_i to s_j, there is at least one path P from c_i to s_j so that at least one edge on P is a secure edge.
- Subject to the above condition, the number of secure edges is minimum.

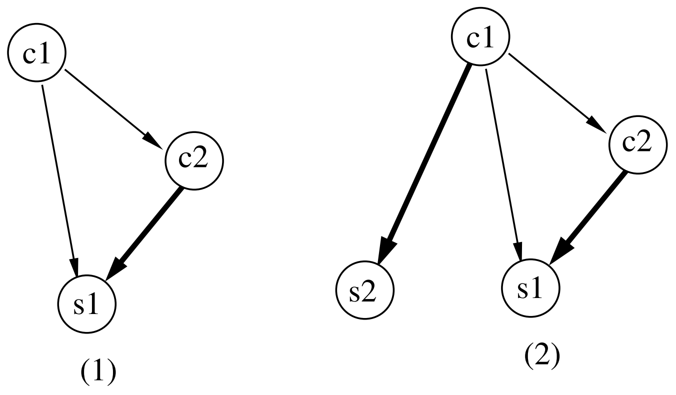

Figure 6: Graph for Dijkstra's Algorithm

For example, in Figure 6 (1), the graph G consists of two computers (c_1 and c_2) and one server s_1. Both computers can reach s_1 (there are two different paths from c_1 to s_1 and only one path from c_2 to s_1.) We only need to secure one edge $c_2 \rightarrow s_1$ (indicated by a thick arrow) to satisfy the above requirements.

On the other hand, in Figure 6 (2), the graph G consists of two computers (c_1 and c_2) and two servers (s_1 and s_2). Both computers can reach s_1. Only c_1 can reach s_2. In this case we need to secure two edges ($c_2 \rightarrow s_1$ and $c_1 \rightarrow s_2$) to satisfy the above requirements.

Describe an algorithm for solving this problem. Analyze its run time.