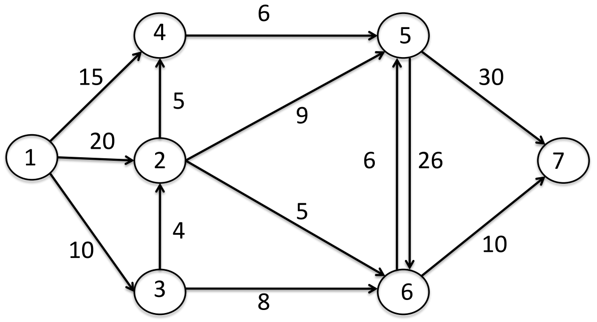
Prof. Salma Elmalaki | Design and Analysis of Algorithms
Fall 2019 | EECS215

Homework 4 - Solution sketch

Topic: Network Flow (Ch.7)

1. *(55 Points)* **Finding Max Flow.** Consider the directed graph shown in the figure below. The numbers on the edges indicate capacities.



   (a) *(20 Points)* Find the max-flow from the source (node 1) to the sink (node 7).

   (b) *(15 Points)* Identify the min-cut corresponding to the max-flow you found in (a).

   (c) *(20 Points)* Now assume that the capacity of edge 2-5 changes from 9 to 3. Find the max flow on this new graph *without* recomputing it from scratch, but starting from the solution you found in (a) and incrementally updating it.

      i. Describe an algorithm that does that incremental update.

      ii. Argue that it indeed finds the optimal solution (max flow for the new graph).

      iii. Analyze its running time (it should be less than running Ford-Fulkerson from scratch).

      iv. Run your algorithm and report the new max flow.

Answer:

(a) Let's apply the Ford-Fulkerson algorithm. We indicate with G the original graph and with G_F the residual graph. The detailed steps of the algorithm are illustrated in Fig. 2-10 in the two pages at the end. G and G_F indicate the original graph and the residual graph, respectively. The black font (in G) indicates the link capacity, while gray font indicates the flow associated with the specific link. The augmenting paths (chosen on G_F) are shown in red. The max flow has value 28.

(b) Fig. 1 illustrates the min-cut as well as the flow value assigned to each link. The min-cut consists of edges $\{4-5, 2-5, 2-6, 3-6\}$ and has capacity 28, which, as expected, equals the max flow value.

In general, the min-cut can be found by running BFS on the residual graph, and identifying the saturated forward edges.

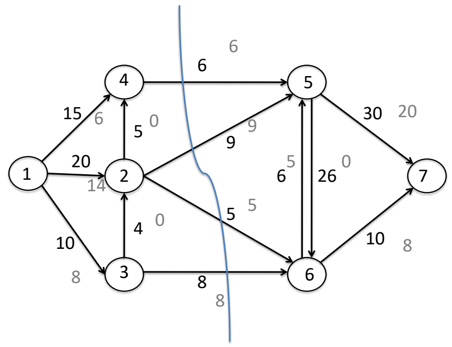


Figure 1: Min-cut and max-flow.

(c) Let $\Delta c = 9 - 3 = 6$ be the decrease in the capacity of the affected edge. Let *head* and *tail* be the head and tail of the affected flow respectively. In this case, head is node 2 and tail is node 5.

i. *Algorithm:*

- Start from the max flow found previously. Consider the residual graph G_F shown in Fig. 10.
- Reduce the flow on edge 2-5 by $\Delta c = 9 - 3 = 6$.
- Find an augmenting path P_{21}, on G_F, from head to the source. Send (i.e., push back) flow $max\{flow(P_{21}), \Delta c\}$. Repeat finding augmenting paths and pushing back flow this way, until the total decrease is Δc.
- Find an augmenting path P_{75}, on G_F, from the sink to the tail. Send (i.e., push back) flow $max\{flow(P_{75}), \Delta c\}$. Repeat finding augmenting paths and pushing back flow, until the total decrease is Δc.
- Because the original edge was part of the min-cut, we can stop now [why? hint: see answer (ii) below]. If the affected flow was not part of the min-cut, then we should run the augmenting paths algorithm on the residual graph until we saturate it.

ii. The new flow (argue that this is still a flow [...])has value Δc 28-6=22. in this example, this is also the new capacity of the cut $\{4 - 5, 2 - 5, 2 - 6, 3 - 6\}$ and therefore is the new max flow.

iii. The complexity of finding augmenting paths from the ends of the affected edge to the source and sink, is the complexity of running BFS on the residual graph twice, once from head to source and once from tail to sink, i.e., $O(m + n)$. "Augmenting" each path takes $O(n)$, as each path has at most n-1 nodes. Updating G_F takes $O(m)$. We will need at most Δc such iterations. So the total running time is $O(m \cdot \Delta c)$.

iv. In this particular example, $\Delta c = 9 - 3 = 6$ and the new flow is the same as before, with the following changes on three edges:

$$f(1-2) = 14 - 6 = 8, \quad f(2-5) = 9 - 3 = 6, \quad f(5-7) = 20 - 6 = 14.$$

The new max flow has value 28-6=22.

Below are some intermediate steps of finding the max flow.

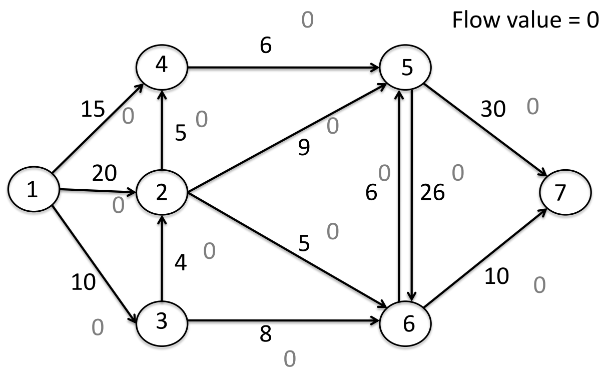


Figure 2: Ford-Fulkerson algorithm: G initialized

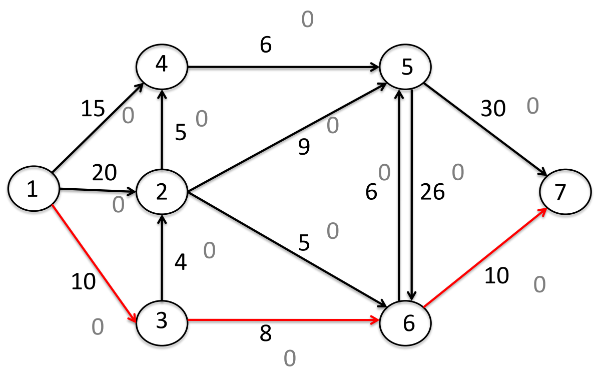


Figure 3: Ford-Fulkerson algorithm: G_F at iteration 1

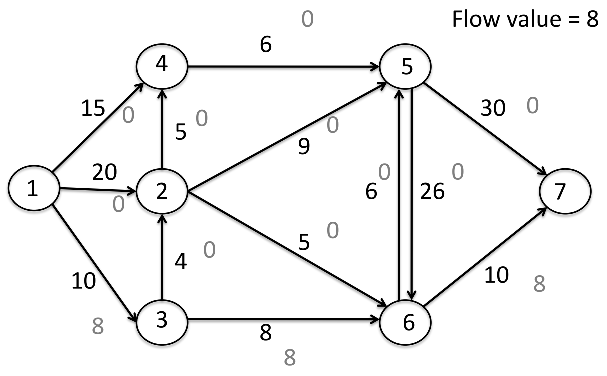
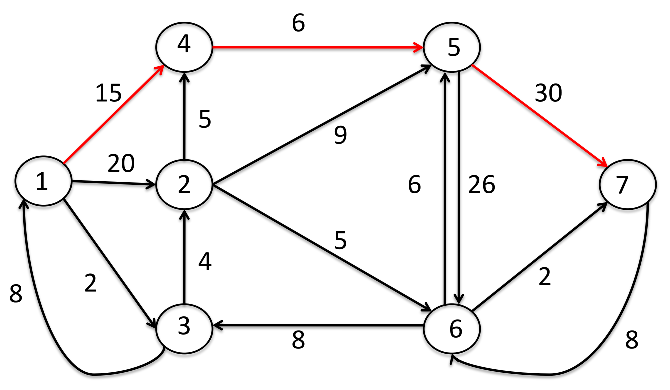
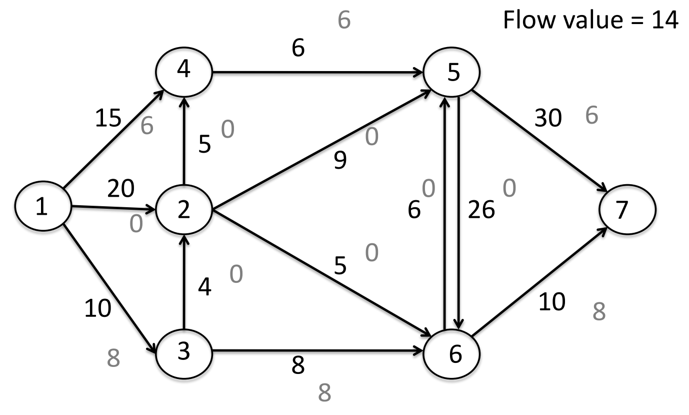


Figure 4: Ford-Fulkerson algorithm: G at iteration 1





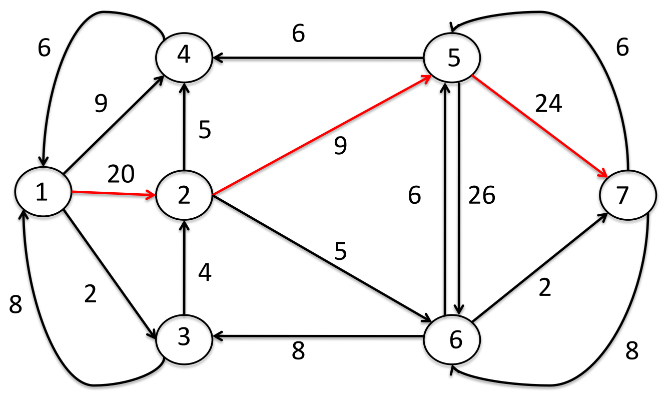


Figure 7: Ford-Fulkerson algorithm: G_F at iteration 3

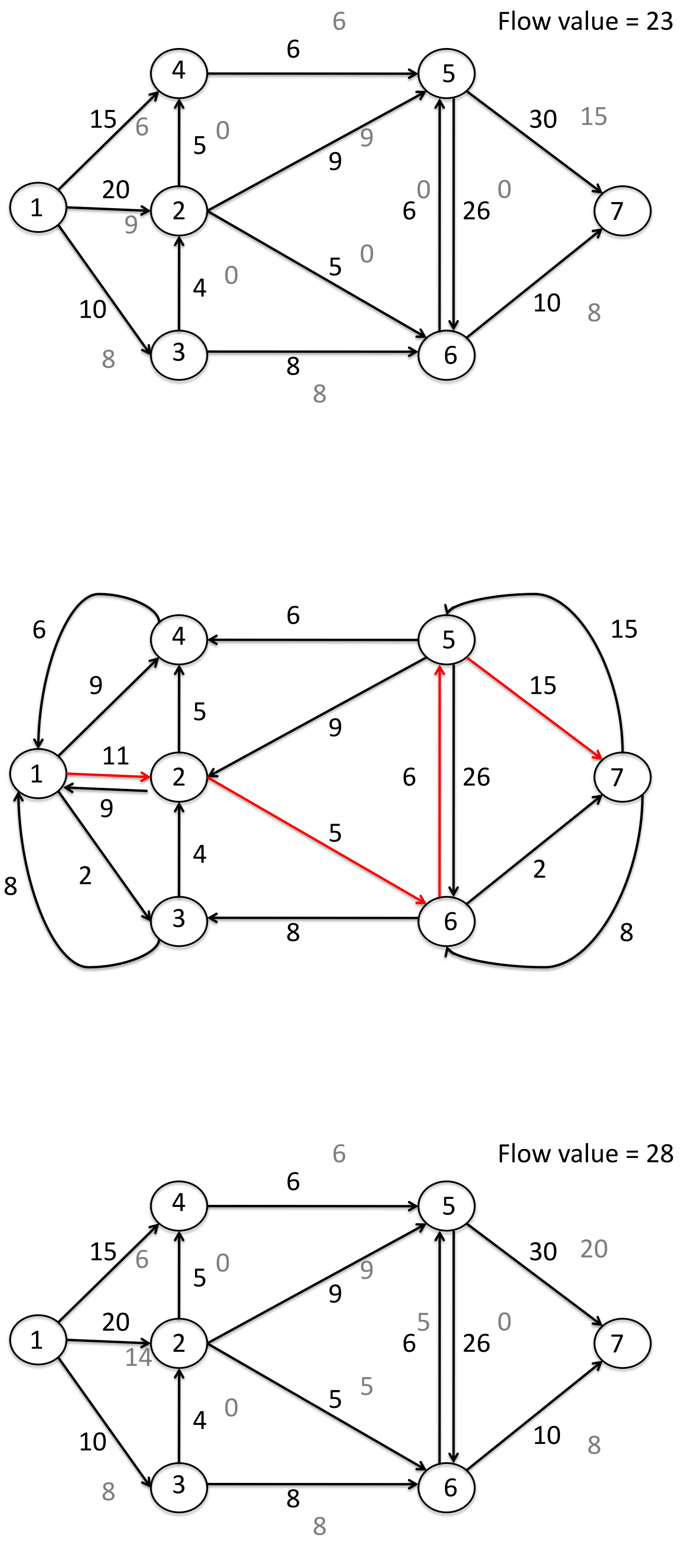


Figure 10: Ford-Fulkerson algorithm: output

2. *(25 Points)* **KT Ch.7, Ex.7** Consider a set of mobile computing clients who need to be connected to one of several possible base stations. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

   For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a *range parameter* $r-$ a client can only be connected to a base station that is within distance r. There is also a *load parameter* L–no more than L clients can be connected to any single base station.

   (a) *(10 Points)*Design a polynomial-time algorithm for the following problem. Given the positions of a set for clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station, subject to the range and load conditions.

   (b) *(10 Points)*Write the condition of the feasible solution (i.e. connecting all clients to base stations).

   (c) *(5 Points)*Analyze the running time of your algorithm.

   Answer:

   (a) We build our flow network as below: There is a node v_i for each client i, a node w_j for each base station j and an edge (v_i, w_j) of capacity 1 if client i is withing range of base station j. We then connect a super-source s to each of the client nodes by and enge of capacity 1, and we connect each of the base station nodes to a super-sink t by an edge of capacity L.

   (b) We claim that there is a feasible way to connect all clients to base stations if and only if there is an $s-t$ flow of value n. If there is a feasible connection, then we send one unit of flow from s to t along each of the paths s, v_i, w_j, t where client i is connected to base station j. This does not violate the capacity condition, in particular on the edge (w_j, t), due to the load constraints. Conversely, if there is a flow of value n, then there is one with integer values. We connect client i to base station j if the edge (v_i, w_j) carries one unit of flow, and we observe that the capacity condition ensures that no base station is overloaded. *Note:* The idea is similar to the problem we solved in the discussion session.

   (c) The running is the time required to solve a max-flow problem on a graph with $O(n + k)$ nodes and $O(nk)$ edges.

3. *(20 Points)* **KT Ch.7, Ex.24.** Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative edge capacities $\{c_e\}$. Give a polynomial-time algorithm to decide whether G has a *unique* minimum s-t cut (i.e., an s-t of capacity strictly less than that of all other s-t cuts).

   Answer:

   This solution is based on ideas from Ex.23.: Run a Max-Flow algorithm and a BFS on the residual graph G_f to get a cut (A^*, B^*). Every node in A^* has to be in the source side of every cut. Assume there is a node $v \in A^*$ that is in the sink side of a cut (A', B'). This leads to a contradiction, because there is a path from s to v (since $v \in A^*$), but this would mean there is a crossing edge in G_f's (A', B') cut which contradicts with the definition of the minimum cut.

   Symmetric argument can be built by finding nodes that can reach t in G_f to get a minimum cut (A_*, B_*). Every node in B_* has to be in the sink side of every cut.

After finding A^* and B_*, define $C = V - \{A^*, B_*\}$. Nodes in C can be on either side of a minimum cut. If $|C| > 0$, then there is no unique Min-Cut. if $|C| = 0$ then $(A^*, B^*) = (A_*, B_*)$ which means the Min-Cut is unique.

4. *(Extra Credit: 10 Points)* **KT Ch.7, Ex.20.** (Hint: This problem can be solved using maximum-flow or using circulation. Any one of them will be accepted but you need to precisely mention your formulation in either cases.)

   Answer:

   (a) We build our flow network as below: There is a source s, a node x_i representing each balloon i, a node z_i representing each condition c_i, and a sink t. There are edges (s, x_i) of capacity 2, (x_i, z_j) of capacity 1 whenever $c_j \in S_i$ (balloon i measures condition j), and edges (z_j, t) of capacity k. We then test whether the maximum $s - t$ flow has value nk.

   The FF algorithm to find a max flow has running time $O(|E|C)$, where $|E|$ is number of edges and C is the total capacity of edges out of s. Here $|E| = O(mn)$ and $C = 2m$, so the running time is $O(m^2n)$. An alternate but related solution would place a lower bound of k on each edge of the from (z_j, t). One would then place a demand of $-nk$ on s and nk on t, and test whether there is a feasible circulation.

   (b) Break all edges (x_i, z_j). Insert new nodes y_{pj} for each sub-contractor p and condition c_j. Add an edge (x_i, y_{pj}) of capacity 1 when $c_j \in S_i$ and balloon i is produced by sub-contractor p. Add and edge (y_{pj}, c_j) of capacity $k-1$. Again, test whether the maximum $s - t$ flow has value nk.

   As in part (a), the running time is the same.