CS 542 – Advanced Data Structures and Algorithms *Jon Turner*

Exam 1 Solution

2/14/2013

1. (10 points) Suppose you are given a graph $G=(V,E)$ with edge weights $w(e)$ and a minimum spanning tree T of G. Now, suppose a new edge $\{u,v\}$ is added to G. Describe (in words) a method for determining if T is still a minimum spanning tree for G.

   Examine the path in T from u to v. If any vertex on this path has weight larger than that of the new edge, then T is no longer an MST. We can modify T to obtain a new MST by removing the max weight edge on this path and replacing it with the new edge.

   Explain how your method can be implemented to run in $O(n)$ time if both G and T are provided as instances of the *wgraph* data structure.

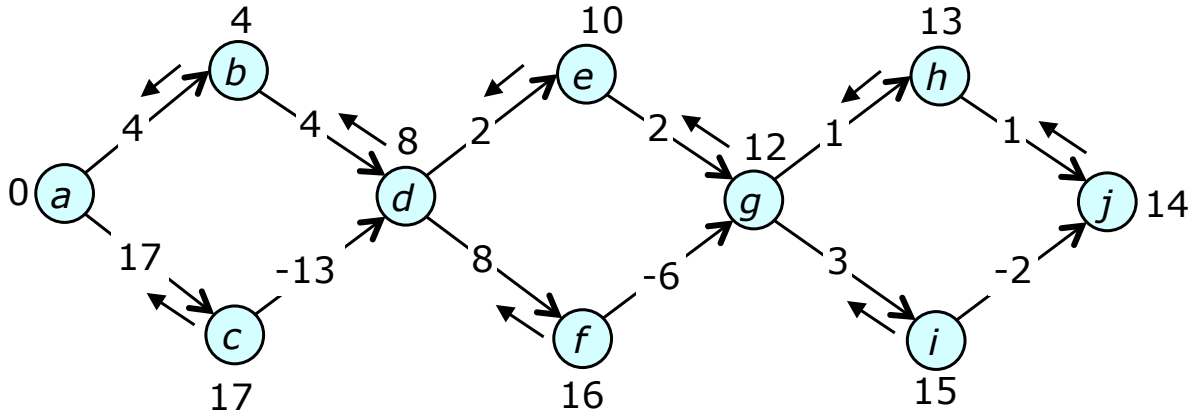
   Using the wgraph for T, we can do a recursive tree traversal in T, starting at vertex u. Once the traversal reaches v, we "unwind" the recursion, and as we do so, we look for the max weight edge along the u,v path.

   The runtime for a tree traversal is O(n) and the required changes to T can be done in constant time.

   Suppose that instead of a single edge, you are given a set of k new edges to add to G. For small enough k it makes sense to apply your algorithm repeatedly in order to update the MST, but if k is "too large", it's more efficient to re-compute the MST from scratch. How big does k have to be (as a function of m and n) in order for this to be a better choice? Assume that the MST is computed using Prim's algorithm with a d-heap, where $d=2$.

   When d=2, the running time for Prim's algorithm is O(m log n), so if kn grows faster than this, it makes sense to recompute from scratch. So, if k>(m/n) log n, it makes sense to recompute the MST.

2. (10 points) Suppose we apply Dijkstra's algorithm to the graph shown below, starting from vertex *a*.



List the first 7 vertices that are scanned.

a,b,d,e,g,h,j

In the diagram label each vertex with its tentative distance after the first 7 vertices are scanned and indicate the parent pointer of each vertex using an arrow pointing from each vertex to its parent.

List the next two vertices to be scanned.

i,j

List the next six.

f,g,h,j,i,j

What is the total number of scanning steps that Dijkstra's algorithm will perform on this graph?

The number of steps is 3+2(3+2(5))=29.

If you extended the graph by adding a fourth "diamond" to the left, with edge lengths of 8, 8, 32 and -24, how many steps would be Dijkstra's algorithm use for this graph?

3+2(29)=61

3. (10 points) Consider a Fibonacci heap containing an unmarked node x for which $p(x)$, $p^2(x)$, ..., $p^9(x)$ are all marked, but $p^{10}(x)$ is not (where $p(x)$ is the parent of x, $p^2(x)$ the grandparent, and so forth). Suppose that a *reducekey* operation is performed at x that makes the key of x smaller than the key of $p(x)$. Do any previously unmarked nodes become marked as a result of this operation? If so, which ones? Assume that $p^{10}(x)$ is not a tree root.

   $p^{10}(x)$ becomes marked.

   If k is the number of credits needed to maintain the credit invariant, in the amortized analysis before the *reducekey* operation, how many credits are needed after the operation.

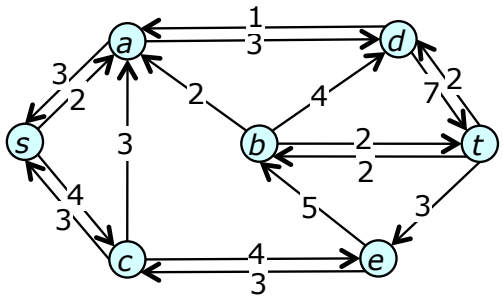
   The number of marked non-root nodes goes down by 8, while the number of trees goes up by 10. So, the number of credits needed to maintain the invariant goes down by 6 to k–6.

   Recall that during a *deletemin*, the basic step involves inserting a tree root into an array, at a position determined by its rank. In some steps, the current tree root "collides" with a tree root that was inserted earlier. In other steps, there is no collision. Suppose that a *deletemin* is done on this heap and that during the *deletemin,* there are 20 steps during which no collision occurs. Give an expression (in terms of $\phi=(1+5^{1/2})/2$) that represents a lower bound on the number of nodes in the heap. Justify your answer.

   If there 20 deletemin steps with no collision, then at the end of the deletemin, there must be some node with rank at least equal to 19. From the analysis we know that the number of nodes in a heap with rank k is at least ϕ^{k-2}, so ϕ^{17} in this case.

   We can actually get a better lower bound by noting that if there were 20 steps with no collision, then after the deletemin, the heap contains at least 20 trees with ranks of at least, 0, 1, 2, 3,...,19. Consequently, the number of nodes in the heap is at least $1+\phi+\phi^2+\cdots+\phi^{17}$.

4. (15 points) The residual graph shown below is for some flow f on a flow graph G.



What is the capacity of the edge connecting e and c in G? Justify your answer.

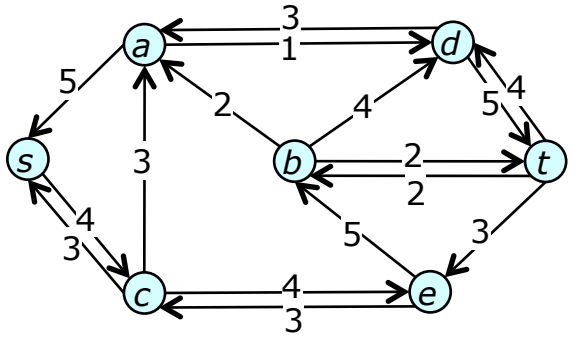
This edge has capacity 7 in the residual graph because the sum of the residual capacities in opposite directions is equal to the original capacity.

Is the edge in G directed from c to e or from e to c. Justify your answer. (Hint: consider the total incoming flow at e.) You may assume that there is no more than one edge joining any two vertices, but you should not assume anything about the direction of the edges at s and t (that is, G may have edges entering s or leaving t).

Consider vertex e. Since this is not the source or the sink, it must satisfy flow conservation. For each or its three incident edges, there are two possible flow values entering e. For the be edge, the inflow is either 0 or 5. For the et edge, the inflow is either 0 or -3. For the ce edge, the inflow is either 3 or -4. Since the sum of the inflows must be 0, the inflow on the be edge must be 0, the inflow on the et edge must be -3 and the inflow on the ce edge must be 3. This implies that the edge is directed from c to e.

Find a shortest augmenting path relative to f. Draw the residual graph that results from adding as much flow as possible to this path.

The path is sadt and the new residual graph appears below.



5. (10 points) Let R_f be a residual graph for a min-cost flow f, let p be a source-sink path in R_f with cost c and let q be a source-sink path in R_f with cost d. Let $f+$ be the flow we get when we add enough flow to p to saturate it and let R_{f+} be the resulting flow graph.

   Does R_f contain a negative cycle? Explain your answer.

   No, because f is a min-cost flow and min-cost flows cannot contain negative cycles.

   If R_{f+} has no negative cycle, what does that tell us about p? Explain.

   It implies that p is a min-cost augmenting path, because the absence of a negative cycle implies that R_{f+} is a min-cost flow and since we obtained f+ by augmenting along p, it follows that p is a min-cost augmenting path.

   If $d<c$ does R_{f+} contain a negative cycle. Explain.

   Yes. $d<c$ implies that p is not a min-cost augmenting path, so f+ must not be a min-cost flow, and the residual graph for a non min-cost flow must contain a negative cycle.

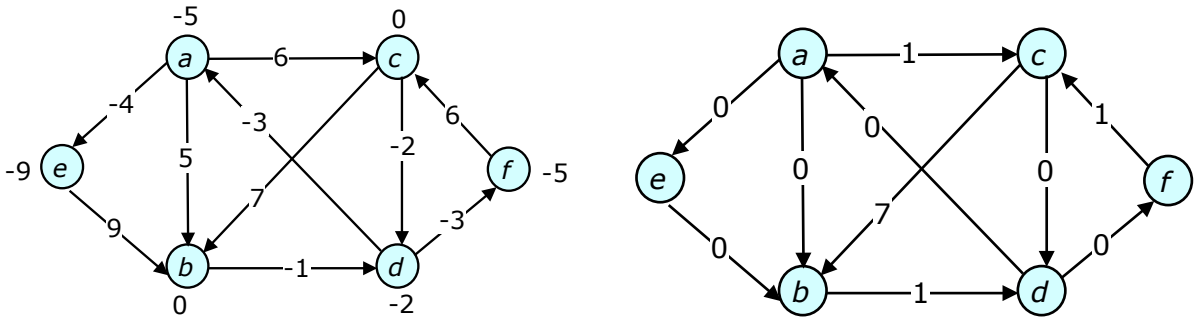
   If (u,v) is in R_{f+} but not in R_f, what does that tells us about (u,v)? Explain.

   It tells us that (v,u) is on p, since a new edge can only appears in the residual graph when flow is added in the opposite direction.

   If (u,v) is in R_f but not in R_{f+}, what does that tell us about $dist_f(u)$ and $dist_f(v)$ (where $dist_f(x)$ is the length of the shortest path from the source s to x in R_f)?

   This means that (u,v) is on p, so $dist_f(v)=dist_f(u)+cost(u,v)$

6. (15 points) The graph at left below has negative length edges. In the diagram at right, give transformed edge lengths that preserve the relative lengths of shortest paths while eliminating negative edge lengths.



The numbers next to the vertices in the left-hand graph represent the shortest path distances from an added source vertex with a zero cost edge to each original vertex. The right hand graph shows the transformed edge lengths computed using these distance values.

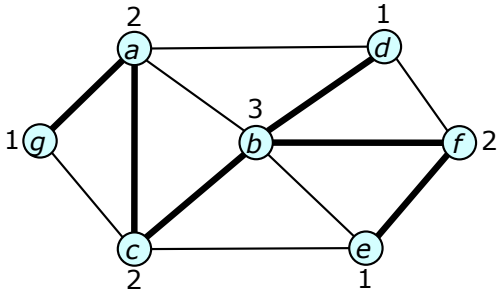
Let G be an arbitrary graph with negative length edges and let $length'(u,v)$ be the transformed edge length for (u,v). Suppose p is a path from x to y with $length(p)=-8$ and $length'(p)=5$. If q is another path from x to y with $length(q)=7$, what is $length'(q)$?

length'(q) = length(q) + (length'(p) - length(p)) = 7 + (5 - (-8)) = 20

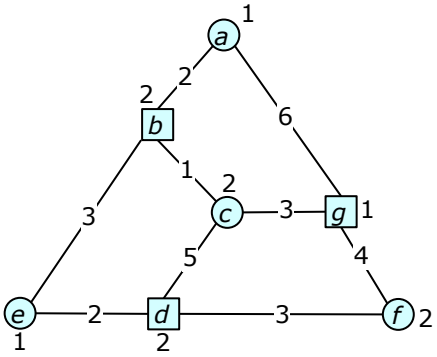
In the min-cost augmenting path algorithm (for the min-cost flow problem) using transformed edge costs, a new shortest path tree is computed during each step, and the shortest path distances are used to modify the edge costs. What is the total cost of the edges in the shortest path tree, using the newly modified costs? Explain.

The total cost is 0. For each edge (u,v), the new cost'(u,v) = cost(u,v) + dist(u) – dist(v). If (u,v) is an edge in the shortest path tree, then dist(v) = dist(u) + cost(u,v), and hence cost'(u,v)=0. Since this is true for all edges in the shortest path tree, the total cost is 0.

7. (15 points) In the degree-constrained subgraph problem, we are given a graph $G=(V,E)$ and a degree bound $b(u)$ for every vertex u. The objective is to find a subgraph of G in which every vertex u has at most $b(u)$ incident edges. In the graph at right, find a degree-constrained subgraph with 6 edges. Indicate the edges in the subgraph by making them heavier weight.



In the weighted version of the problem, each edge e has a weight $w(e)$ and we are interested in the degree-constrained subgraph of maximum weight. Describe (in words) an algorithm to solve this problem when the graph is bipartite. Use the graph shown at right to illustrate your solution. (You need not actually produce a max weight subgraph.) Note that the shapes of the vertices define the division of the vertices into subsets.



We solve the problem by converting it to a min-cost flow problem, as we did for the max weight matching problem. That is, we add a source vertex with an edge to every vertex in the "left subset" of vertices and a sink with an edge from every vertex in the "right subset". The edges (s,u) have capacity b(u) and cost 0. The edges (u,t) have capacity b(u) and cost 0. Each original edge {u,v} is turned into a directed edge from the "left vertex" u to v, with cost(u,v)=–wt(u,v) and cap(u,v)=1. We then add flow to the graph using the min-cost augmenting path method. The algorithm halts before finding a maximum flow if the next min-cost augmenting path has non-negative cost. At that point the edges in the "central" part of the graph that have positive flow define a degree-bounded subgraph of maximum weight.