CS 600 Homework 7 Solutions

C-15.2 Minimum Spanning Tree containing edge with largest weight

G is a weighted, connected, undirected, simple graph and e is the largest weight edge in G. The claim is that there is no minimum spanning tree of G that contains e. This claim is false. For instance, if G is already a tree, then even the largest-weight edge must belong in the minimum spanning tree.

A-15.2 Maximum Bandwidth of a path

We can model this problem using a graph. We associate a vertex of the graph with each switching center and an edge of the graph with each line between two switching centers. We assign the weight of each edge to be its bandwidth. Vertices that represent switching centers that are not connected by a line do not have an edge between them.

We use the same basic idea as in Dijstra's algorithm. We keep a variable $d[v]$ associated with each vertex. We initialize the d values of all vertices to 0, except for the value of the source (the vertex corresponding to a) which is initialized to ∞. We also keep a pi value associated with each vertex (that contains the predecessor vertex).

Assume that we have an edge (u, v). If $\mathsf{min}\{d[u], w(u, v)\} > d[v]$, then we should update $d[v]$ to $\mathsf{min}\{d[u], w(u, v)\}$ (because the path from a to u and then to v has bandwidth $\mathsf{min}\{d[u], w(u, v)\}$, which is more than the one we have currently).

The total running time is $O((n + m) \log n)$.

A-15.6 Communcation Network in Flash University

Use the Prim-Jarnik algorithm, but with a simpler implementation of the priority queue, Q. Namely, we can implement the priority queue as two doubly linked lists, one for the items with cost 10 and the other for items with cost 30. Insertions and updates can therefore be done in $O(1)$ time, as can the removal of an item with the smallest key.

Therefore the running time is $O(n + m)$.

R-16.2 Questions about Flow Network (Figure 16.6a)

What are the forward and backward edges of augmenting path π?

Forward edges are $(s, v_2), (v_2, v_3), (v_1, v_4), and (v_4, t)$. The backward edge is (v_1, v_3).

Augmenting paths with respect to flow f?

Notice that the paths (v_3, t) and (v_2, v_5, t) are at full capacity, so the only way to possible increase the flow is through $(v4, t)$. There are six such augmenting paths. The residual capacity of the path, $\delta_f(\pi) = \min_{e \in \pi} \delta_f(e)$ is shown after the turnstile ($\vdash$).

$$(s, v_1, v_4, t) \vdash 2$$
$$(s, v_1, v_3, v_4, t) \vdash 3$$
$$(s, v_2, v_3, v_4, t) \vdash 3$$
$$(s, v_2, v_3, v_1, v_4, t) \vdash 3$$
$$(s, v_3, v_4, t) \vdash 1$$
$$(s, v_3, v_1, v_4, t) \vdash 1$$

Maximum flow in N?

The maximum flow in N is 15.

C-16.7 Algorithm to determine whether a graph is bipartite

Start from some vertex v and perform a breadth-first-search. If there are any non-tree edges joining vertices on the same level, then the graph is not bipartite (for we would have just found an odd-length cycle).

If, after performing this test, there is an unvisited vertex, w, we repeat this algorithm with w. We continue this process until we have determined that the graph is not bipartite or we have visited all its vertices. If we have visited all its vertices and found no odd-length cycles, then the graph is bipartite.

The running time is $O(n + m)$, since we are performing a BFS on each connected component.

A-16.2 Puppy adoptions

Create a bipartite graph, with residents and puppies as its vertices and include an edge (i, j) if resident i would like to adopt puppy j. Give each such edge a capacity of 1.

Next, create a source node, s, and connect it with an edge to each resident, giving each such edge a capacity of 3. Finally, create a sink node, t, and connect every puppy to it with an edge, giving each such edge a capacity of 1.

Now perform a maximum flow algorithm on this network. The edges used for flow between residents and puppies will indicate which puppies each resident adopts, and the contraints on the edges from s guarantee that no resident can apopt more than 3 puppies.