CS161 Handout 15
Summer 2001 August 14, 2001

Solution Set 7: Graph Algorithms

1) Exercise 24.1 - 1 on page 503.

Theorem 24.1 shows this. Just let A be the empty set and S be any set containing u but not v.

2) Exercise 24.2 - 4 on page 510.

Kruskal's algorithm as given in the textbook requires $O(E \lg E)$ time:

$O(V)$ time to initialize.
$O(E \lg E)$ to sort the edges by weight.
$O(E\alpha(E,V))$ to process the edges (if a Fibonacci heap is used.)

If all edge weights are integers ranging from 1 to $|V|$, we can use counting sort (instead of a more generally applicable sorting algorithm) to sort the edges in $O(V+E) = O(E)$ time. (Note that $V = O(E)$ for a connected graph.) This speeds up the running time to take only $O(E\alpha(E,V))$ time; the time to process the edges, not the time to sort them, is now the dominant term. Knowledge about the weights won't help speed up any other part of the algorithm, since nothing besides the sort uses the weight values.
If the edge weights are integers ranging from 1 to a constant W, we can again use counting sort, which again runs in $O(W+E) = O(E)$ time, so we get the same asymptotic upper bound as above.

3) Exercise 24.2 - 5 on page 510.

The time taken by Prim's algorithm is determined by the speed of the queue operations. With the queue implemented as a Fibonacci heap, it takes $O(E + V \lg V)$ time.
Since keys in the priority queue are edge weights, it might be possible to implement the queue even more efficiently when there are restrictions on the possible edge weights. If the edge weights are integers to some constant W, we can speed up the algorithm by implementing the queue as an array $Q[0 \ldots W+1]$ (using the $W+1$ slot for the key = infinity), where each slots holds a doubly-linked list of vertices that that weight as their key. Then Extract-Min takes only constant time (just scan for the first non-empty slot), and Decrease-Key takes only constant time as well (just remove the vertex from the list it's in and insert it at the front of the list indexed by the new key.) This

gives a total running time of $O(E)$, which is the best possible asymptotic time (since $\Omega(E)$ edges must be processed).
If the edge weights are integers ranging from 1 to $|V|$, however, the array implementation doesn't help. Decrease-Key would still be reduced to constant time, but Extract-Min would now use $O(V)$ time, for a total running time of $O(E + V^2)$. We're better off sticking with the Fibonacci-heap implementation, which takes $O(E + V \lg V)$ time. To get any advantage out of the integer weights, you'd have to use data structures that we haven't studied, such as:

> the van Emde data structure mentioned in the introduction to Part V: upper bound $O(E + V \lg \lg V)$ for Prim's algorithm, or
> redistributive heaps (not in the book): $O\left(E + V\sqrt{\lg V}\right)$ for Prim's algorithm.

4) Exercise 25.1 - 4 on page 526.

Whenever Relax sets π for some vertex, it also reduces the vertex's d value. Thus, if $\pi[s]$ gets set to some non-NIL value, $d[s]$ is reduced from its initial value of 0 to some negative number. But $d[s]$ is the weight of some path from s to s, which is a cycle including s. That would mean there is a negative-weight cycle.

5) Exercise 25.2 - 4 on page 531.

To find the most reliable path between s and t, run Dijsktra's algorithm with edge weights of $w(u,v) = -\lg r(u,v)$ to find the shortest paths from s in $O(E + V \lg V)$ time. The most reliable path is the shortest path from s to t, and that path's reliability is the product of its edges' reliabilities.

Because the probabilities are independent, the probability that a path will not fail is the product of the probabilities that its edges will not fail. We want to find a path p from s to t such that $\prod_{(u,v)\ p} r(u,v)$ is maximized. This is equivalent to maximizing $\lg \prod_{(u,v)\ p} r(u,v) = \sum_{(u,v)\ p} \lg r(u,v)$, which is equivalent to minimizing $-\sum_{(u,v)\ p} \lg r(u,v)$. (Note that $r(u,v)$ can be 0 and that $\lg 0$ is undefined, but for this problem we can just define $\lg 0 = -\infty$.) Thus, if we assign weights $w(u,v) = -\lg r(u,v)$, we have a shortest paths problem.

Since $\lg 1 = 0$, $\lg x \le 0$ for all $x \le 1$, and we have that $\lg 0 = -\infty$, and all weights are then nonnegative. Therefore, we can use Dijkstra's algorithm to find the shortest paths from s in $O(E + V \lg V)$ time.

6) Exercise 25.3 - 3 no page 535.

The proof of Lemma 25.12 shows that for every v, d[v] has attained its final value after **length**(any shortest path to v) iterations of Bellman-Ford. Thus, after m passes, Bellman-Ford can terminate. We don't know m in advance, so we can't make the algorithm loops exactly m times and then terminate. But if we adjust the algorithm to stop when nothing changes any more, it will stop after m + 1 iterations.

```
Bellman-Ford-M-Plus-One(G,w,s)
   Initialize-Single-Source(G,s)
   changes = true
   while changes = true
      do changes = false
         for each edge (u,v) in E[G]
            do Relax-M(u,v,w)


Relax-M(u,v,w)
   if d[v] > d[u] + w(u,v)
      then  d[v] = d[u] + w(u,v)
            π[v] = u
            changes = true
```

The test for a negative-weight cycle has been removed above, because this version of the algorithm will never escape the while loop unless all of the d values stop changing. If you want to terminate, you'd have to keep track of how many times the outer while loop executes, and make sure it never exceeds |V|.

7) Problem 25.-4 on page 547.

I'm assuming that all vertices are reachable from the source.

(a) Since all the weights are nonnegative, use Dijkstra's algorithm. I'm the priority queue as an array Q[0... |E| +1], where Q[i] is a list of vertices v for which d[v] = i. Initialize d[v] for all vertices not equal to s to |E| + 1 instead of infinity, so that all vertices have a place in Q.

The |V| Extract-Mins can be done in O(|E|) time total, and decreasing a d value during relaxation can be one in constant time, for a total running time of O(E).

1. When d[v] decreases, just add v to the front of the list in Q[d[v]].
2. Extract-Min removes the head of the list of the first nonempty slot in Q. To do Extract-Min without scanning the whole Q, keep track of the smallest i for which Q[i] is nonempty. The key point is that when d[v] decreases due to a relaxation of edge (u, v), d[v] remains greater than or equal to d[u], so it never moves to an earlier slot of Q

than the one that had u, the previous minimum. Thus Extract-Min can always scan upward in the array, taking a total of O(E) extra time for all Extract-Min calls.

(b) For all (u,v) E, we have $w_1(u,v)$ $\{0,1\}$, so $\delta_1(u,v) \le |V| - 1 \le |E|$. Use part (a) to get the $O(E)$ time bound.

(c) The i bits of $w_i(u,v)$ consist of the i - 1 bits of $w_{i-1}(u,v)$ followed by one more bit. If that low-order bit is 0, then $w_i(u,v) = 2w_{i-1}(u,v)$, if it is 1, then $w_i(u,v) = 2w_{i-1}(u,v) + 1$.

Notice the following two properties of shortest paths:

1. If all edge weights are multiplied by a factor c, the all shortest-path weights are multiplied by c as well.
2. If all edge weights are increased by at most c, the all shortest path weights are increased by at most $c(|V| - 1)$, since all shortest paths are simple and have at most $|V| - 1$ edges.

The lowest possible value for $w_i(u,v)$ is $2w_{i-1}(u,v)$, so by observation 1, the lowest possible value for $\delta_i(s,v)$ is $2\delta_{i-1}(s,u)$.
The highest possible value for $w_i(u,v)$ is $2w_{i-1}(u,v) + 1$, so using both observations, the highest possible value for $\delta_i(s,v)$ is $2\delta_{i-1}(s,u) + |V| - 1$.

(d) We have that

$$\begin{aligned}\hat{w}_i(u,v) &= w_i(u,v) + 2\delta_{i-1}(s,u) - 2\delta_{i-1}(s,v)\\ &\ge 2w_{i-1}(u,v) + 2\delta_{i-1}(s,u) - 2\delta_{i-1}(s,v)\\ &\ge 0\end{aligned}$$

The second line follows from part (c). The third line follows from Lemma 25.3: $\delta_{i-1}(s,v) \le \delta_{i-1}(s,u) + w(u,v)$.

(e) Observe that if we compute $\hat{w}_i(p)$ for any path $p : u \quad v$, the $\delta_{i-1}(s,t)$ terms cancel for every intermediate vertex t on the path. Thus, $\hat{w}_i(p) = w_i(p) + 2\delta_{i-1}(s,u) - 2\delta_{i-1}(s,v)$. These δ terms will be of minimum w_i weight and of minimum $\hat{w}_i$ weight between u and v. Letting u = s, we get

$$\hat{\delta}_i(s,v) = \delta_i(s,v) + 2\delta_{i-1}(s,s) - 2\delta_{i-1}(s,v) = \delta_i(s,v) - 2\delta_{i-1}(s,v)$$

Combining the result $\delta_i(s,v) = \hat{\delta}_i(s,v) + 2\delta_{i-1}(s,v)$ with $\delta_i(s,v) \le 2\delta_{i-1}(s,u) + |V| - 1$ gives us $\hat{\delta}_i(s,v) \le |V| - 1 \le |E|$.

(f) To compute all $\delta_i(s,v)$ from $\delta_{i-1}(s,v)$ for all v V in $O(E)$ time:

1. Compute the weights $\hat{w}_i(u,v)$ in $O(E)$ time (as shown in part (d).)
2. By part (e), $\hat{\delta}_i(s,v) \le |E|$. so use part (a) to compute all $\hat{\delta}_i(s,v)$ in $O(E)$ time.

3. Compute all $\delta_i(s,v)$ from $\hat{\delta}_i(s,v)$ and $\delta_{i-1}(s,v)$ as shown in part (e) in $O(V)$ time.

To compute all $\delta(s,v)$ in $O(E\lg W)$ time

1. Compute $\delta_1(s,v)$ for all $v \quad V$. As shown in part (b), this takes $O(E)$ time.
2. For each $i = 2,3,\ldots,k$, compute all $\delta_i(s,v)$ from $\delta_{i-1}(s,v)$ in $O(E)$ as shown above. This computes $\delta(s,v) = \delta_k(s,v)$ in time $O(Ek) = O(E\lg W)$.