ECE368 Homework #11

IMPORTANT: Write your user (login) ID at the TOP of EACH page. Also, be sure to *read* and *sign* the *Academic Honesty Statement* that follows:

"In signing this statement, I hereby certify that the work on this exercise is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exercise and will be subject to possible disciplinary action."

Printed Name: **Solution**

Login: **Solution**

Signature:

Please acknowledge those people who have helped you with this homework.

# of Question	Credits

1. Dijkstra's

We are given a directed graph $G=(V,E)$ on which each edge $(u,v)\in E$ has an associated value $r(u,v)$, which is a real number in the range $0\le r(u,v)\le 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u,v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Solution:

To find the most reliable path between s and t, run Dijkstra's algorithm with edge weights $w(u,v) = -\lg r(u,v)$ to find 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 the reliabilities of its edges.

Here's why this method works. 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 $s \overset{p}{\leadsto} t$ such that $\prod_{(u,v)\in p} r(u,v)$ is maximized. This is equivalent to maximizing $\lg(\prod_{(u,v)\in p} r(u,v)) = \sum_{(u,v)\in p} \lg r(u,v)$, which is in turn equivalent to minimizing $\sum_{(u,v)\in p} -\lg r(u,v)$. (Note: $r(u,v)$ can be 0, and $\lg 0$ is undefined. So in this algorithm, define $\lg 0 = -\infty$.) Thus if we assign weights $w(u,v) = -\lg r(u,v)$, we have a shortest-path problem.

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

Alternate answer

You can also work with the original probabilities by running a modified version of Dijkstra's algorithm that maximizes the product of reliabilities along a path instead of minimizing the sum of weights along a path.

In Dijkstra's algorithm, use the reliabilities as edge weights and substitute

- max (and EXTRACT-MAX) for min (and EXTRACT-MIN) in relaxation and the queue,
- $\times$ for $+$ in relaxation,
- 1 (identity for $\times$) for 0 (identity for $+$) and $-\infty$ (identity for min) for ∞ (identity for max).

For example, the following is used instead of the usual RELAX procedure:

RELAX-RELIABILITY(u, v, r)
if $d[v] < d[u] \cdot r(u,v)$
  then $d[v] \leftarrow d[u] \cdot r(u,v)$
    $\pi[v] \leftarrow u$

This algorithm is isomorphic to the one above: It performs the same operations except that it is working with the original probabilities instead of the transformed ones.

2. **Dijkstra's**

Suppose we change line 4 of Dijkstra's algorithm to the following:

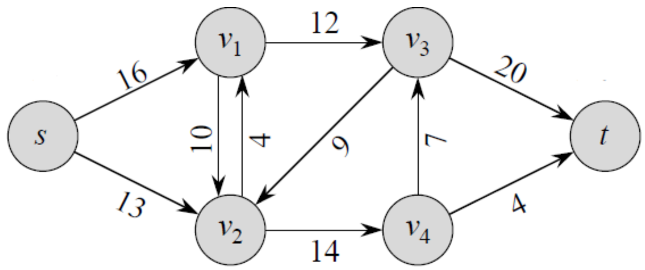
4 while $|Q| > 1$

This change causes the while loop to execute $|V| - 1$ times instead of $|V|$ times. Is this proposed algorithm correct?

Solution:

Yes, the algorithm still works. Let u be the leftover vertex that does not get extracted from the priority queue Q. If u is not reachable from s, then $d[u] = \delta(s, u) = \infty$. If u is reachable from s, there is a shortest path $p = s \sim x \rightarrow u$. When the node x was extracted, $d[x] = \delta(s, x)$ and then the edge (x, u) was relaxed; thus, $d[u] = \delta(s, u)$.

3. **Max flow graph**. Following the example shown in the lecture slides, show how *Ford-Fulkerson* works on the following graph to find the max s-t flow. In each step: show the flow on the input graph; show the residual graph; highlight the augmenting path on the residual graph. Describe how you identify the augmenting path in each step.



Answer:
There are multiple ways depending how you choose augmenting paths. Here is one possible execution.

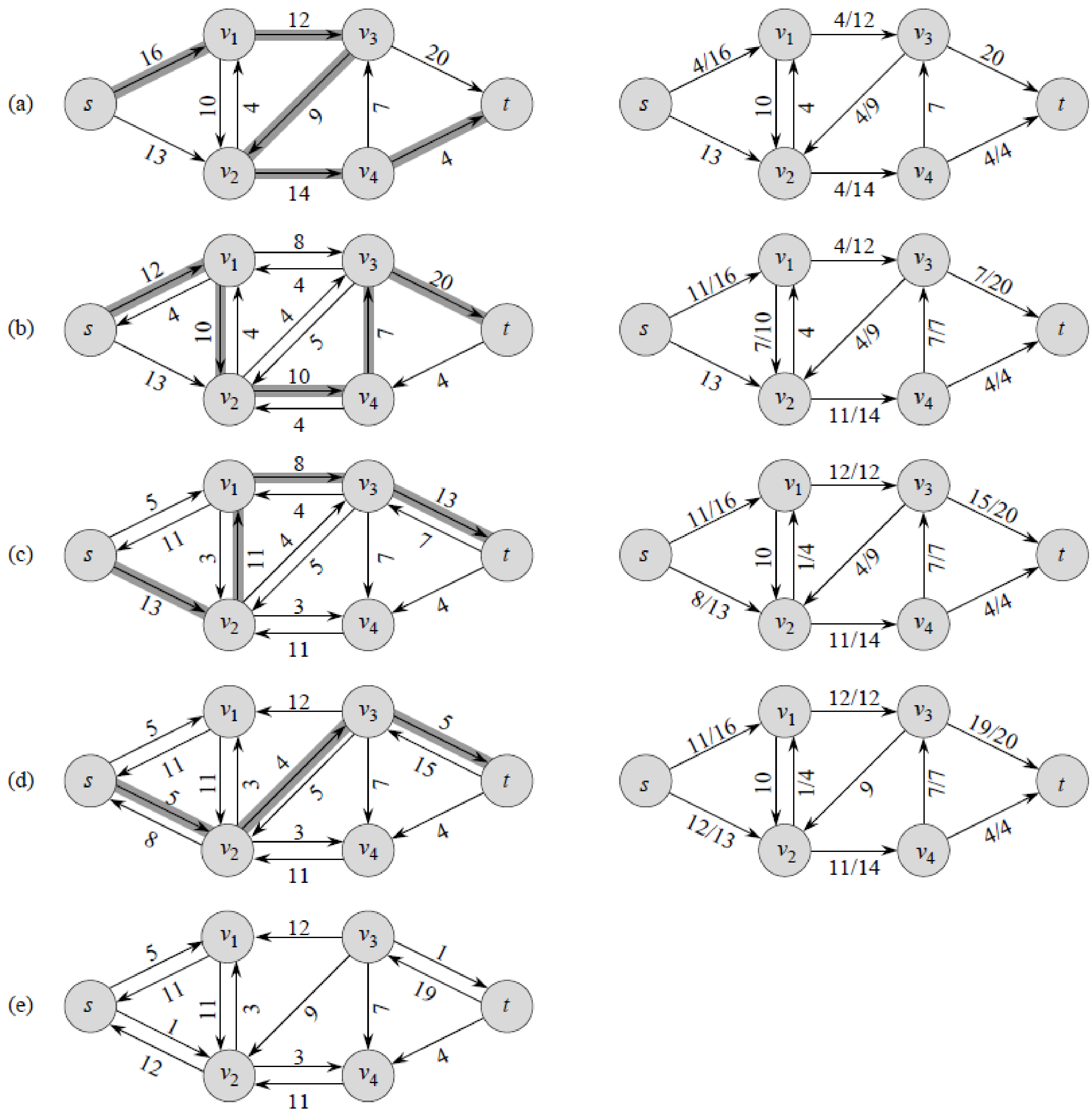


Figure 26.5 The execution of the basic Ford-Fulkerson algorithm. **(a)–(d)** Successive iterations of the **while** loop. The left side of each part shows the residual network G_f from line 4 with a shaded augmenting path p. The right side of each part shows the new flow f that results from adding f_p to f. The residual network in (a) is the input network G. **(e)** The residual network at the last **while** loop test. It has no augmenting paths, and the flow f shown in (d) is therefore a maximum flow.

4. **Min spanning Tree.**
Suppose that we represent the graph G = (V, E) as an adjacency matrix. Give a simple implementation of Prims algorithm for this case that runs in O(V^2) time.

Solution. If Graph $G = (V, E)$ is represented as an adjacency matrix, for an vertex u, to find its adjacent vertices, instead of searching the adjacency list, we search the row of u in the adjacency matrix. We assume that the adjacency matrix stores the edge weights, and those unconnected edges have weights 0. The Prim's algorithms is modified as:

Algorithm 1: MST-PRIM2(G, r)

```
for each u ∈ V[G] do
    key[u] = ∞;
    π[u] = NIL;
end
key[r] = 0;
Q=V[G];
while Q ≠ ∅ do
    u=EXTRACT-MIN(Q);
    for each v ∈ V[G] do
        if A[u,v] ≠ 0 and v ∈ Q and A[u,v] < key[v] then
            π[v] = u;
            key[v] = A[u, v];
        end
    end
end
```

The outer loop (while) has $|V|$ variables and the inner loop (for) has $|V|$ variables. Hence the algorithm runs in $O(V^2)$.
Remarks There are several ways to implement Prim's algorithm in $O(V^2)$ algorithm:

(a) Using the priority queue as above;

(b) Using an array so each time extracting the minimum by one-by-one comparison, which takes $O(V)$ time;

(c) Converting the adjacency matrix into adjacency list representation in $O(V^2)$ time, then using the implementation in textbook.

All above methods run in $O(V^2)$ time. ■

5. Min spanning tree

Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut V_1, V_2, and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.
Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

Solution. We claim that the algorithm will fail. A simple counter example is shown in Figure 1. Graph $G = (V, E)$ has four vertices: $\{v_1, v_2, v_3, v_4\}$, and is partitioned into subsets

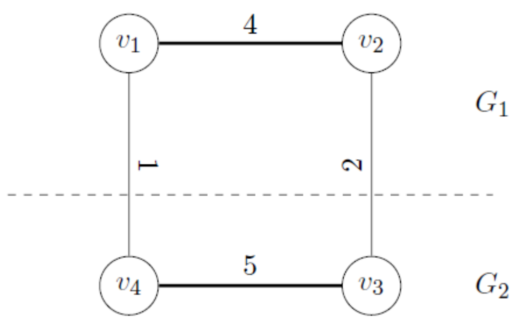


Figure 1: An counter example.

$G1$ with $V_1 = \{v_1, v_2\}$ and G_2 with $V_2 = \{v_3, v_4\}$. The minimum-spanning-tree(MST) of G_1 has weight 4, and the MST of G_2 has weight 5, and the minimum-weight edge crossing the cut (V_1, V_2) has weight 1, in sum the spanning tree forming by the proposed algorithm is $v_2 - v_1 - v_4 - v_3$ which has weight 10. On the contrary, it is obvious that the MST of G is $v_4 - v_1 - v_2 - v_3$ with weight 7. Hence the proposed algorithm fails to obtain an MST. $\blacksquare$