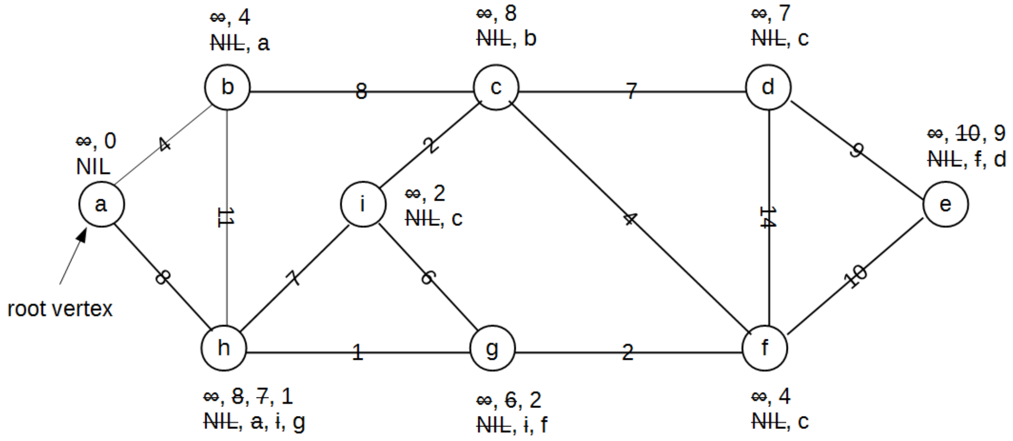
1. We first note that if T and T' be spanning trees in G, given any edge $e' \in T' - T$, there exists an edge $e \in T - T'$ such that $(T - \{e\}) \cup \{e'\}$ is also a spanning tree. To verify this property. Adding e' into T results in a unique cycle. There must be some edge in this cycle that is not in T' (since otherwise T' must have a cycle). Call this edge e. Then deleting e restores a spanning tree, since connectivity is not affected, and the number of edges is restored to $n - 1$.

   Suppose there are two MST's T_1 and T_2. By the above observation, we should be able to change from T_1 to T_2 by exchanging edges and never increasing the weight of the tree. But, since edge weights are distinct, any exchange would change the weight of the tree, and since T_1 is a MST it would have to increase the weight, a contradiction.

2. Prim's algorithm is run and illustrated as follows:



   The set of edges of the final MST is $\{(a,b),(b,c),(c,i),(c,f),(f,g),(g,h),(c,d),(d,e)\}$

3. The edges adding to the MST in the following order in set A:

$$A = \{(h,g),(c,i),(g,f),(a,b),(c,f),(c,d),(a,h),(d,e)\}$$

   Note the order of (c,i) and (g,f) can be switched. This is also the case for (a,b) and (c,f). (a,h) can be replaced by (b,c).

4. Prim's MST algorithm will work with negative weights. Review the pseudo-code of Prim's algorithm, the arguments used do not depend on the weight being positive.

5. Multiply the weights of all the edges by -1. Since both Kruskals and Prims algorithms work for positive as well as negative weights, we can find the minimum spanning tree of the new graph. This is the same as the maximum spanning tree of the original graph.

6. (a) Edges are processed in the following lexicographic order in each pass:

$$(u,v),\quad (u,x),\quad (u,y),\quad (v,u),\quad (x,v),\quad (x,y),\quad (y,v),\quad (y,z),\quad (z,u),\quad (z,x)$$

   The Bellman-Ford algorithm with source vertex y

	$d[\ ]$					$\pi[\ ]$				
pass	u	v	x	y	z	u	v	x	y	z
0	∞	∞	∞	0	∞	NIL	NIL	NIL	NIL	NIL
1	8	7	9	0	2	z	y	z	NIL	y
2	5	6	9	0	2	v	x	z	NIL	y
3	4	6	9	0	2	v	x	z	NIL	y
4	4	6	9	0	2	v	x	z	NIL	y
tst	4	6	9	0	2	v	x	z	NIL	y

The algorithm returns **TRUE**.

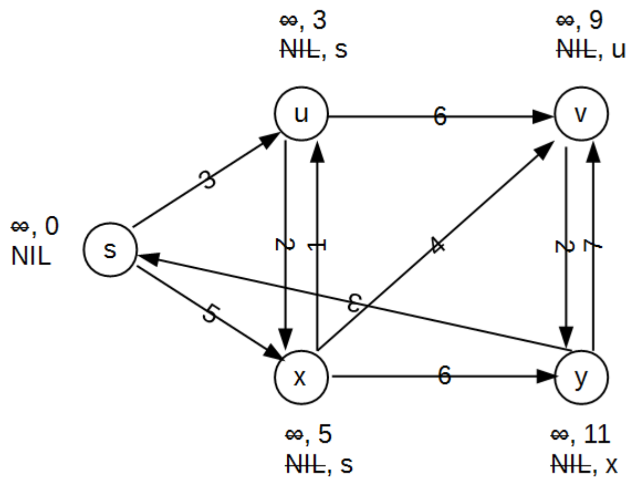
(b) The Bellman-Ford algorithm with source vertex z, with edge weight of (u, v) changed to 4.

	$d[\ ]$					$\pi[\ ]$				
pass	u	v	x	y	z	u	v	x	y	z
0	∞	∞	∞	∞	0	NIL	NIL	NIL	NIL	NIL
1	6	∞	7	∞	0	z	NIL	z	NIL	NIL
2	6	4	7	2	0	z	x	z	u	NIL
3	2	4	7	2	0	v	x	z	u	NIL
4	2	2	7	-2	0	v	y	z	u	NIL
tst	0	2	7	-2	0	v	y	z	u	NIL

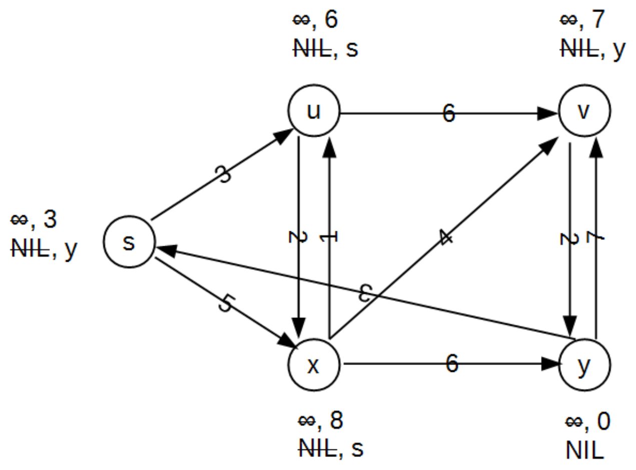
We see that 5th iteration still changes the distances, so there is a negative cycle in this graph. We can find by looking at the graph $y \to v \to u \to y$. The algorithm returns **FALSE**.

7. Dijkstra's algorithm:is run and illustrated as follows:

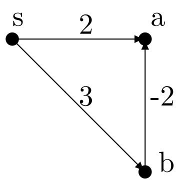
   (a) Dijkstra's algorithm:is run and illustrated as follows:



   (b)



8. Dijkstra's algorithm gives the wrong answer on the graph below, incorrectly deciding that the shortest path from s to a is the direct edge of length 2, instead of the path via b of total length $3 - 2 = 1$.



Further detail: the proof of Dijkstra fails because it is not true that $d[u] = \delta(s, u)$ at the time u is inserted into S. In particular, when a is inserted, $d[a] = 2 \neq \delta(s, a) = 1$. The proof fails because it assumes that adding edges to a path can only increase its weight, which is false if there are negative-weight edges.

9. 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 is an explanation. Because the probabilities are independent, the probability that a path will not fail is the product of the probabilities that its edge will not fail. We want to find a path such that $\Pi_{(u,v)\in p} r(u, v)$ is maximized. This is equivalent to maximizing

$$\lg \left(\Pi_{(u,v)\in p} r(u, v) \right) = \sum_{(u,v)\in p} \lg r(u, v),$$

which is equivalent to minimizing $\sum_{(u,v)\in p}(-\lg r(u, v))$. Note that $r(u, v)$ can be 0, and $\lg 0$ is undefined. So in this algorithm, define $\lg 0 = -\infty$. Thus if we assign weigths $w(u, v) = -\lg r(u, v)$, 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 are nonnegative, and we can use Dijkstra's algorithm to find the shortest paths from s in $O(E + V \lg V)$ time.

Alternative answer: You can also work with original probability 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 weight 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 relaxation procedure.

```
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 transformed ones.

10.
- The minimum spanning tree does not change. Since, each spanning tree contains exactly $n - 1$ edges, the cost of each tree is increased $n - 1$ and hence the minimum is unchanged.
- The shortest paths may change. In the following graph, the shortest path from a to d changes from $p = a \to b \to c \to d$ to $p = a \to d$ if each edge weight is increased by 1: