Problem 1.

$$\begin{tikzcd} c & a \arrow[l] \arrow[r, bend left] & b \arrow[l, bend left] \end{tikzcd} \qquad \begin{tikzcd} 4/5 & 1/6 \arrow[l] \arrow[r, bend left] & 2/3 \arrow[l, bend left] \end{tikzcd}$$

The given finishing times assume that the array of adjacency lists and the adjacency lists themselves are sorted alphabetically. The second DFS run would begin with vertex b and identify the whole graph as a strongly connected component when, in fact, there are two components $\{a, b\}$ and $\{c\}$.

Problem 2. It suffices to multiply all weights by -1, run Kruskal's or Prim's algorithm, and finally multiply all weights by -1 again. The minimum weight spanning tree found by Kruskal's or Prim's algorithm becomes a maximum weight spanning tree when the signs of the weights are inverted.

Problem 3. BFS can be used to find a minimum height spanning tree for a given source vertex because it finds the path with the smallest number of edges from the source (the root of the tree) to any other vertex (including the leaves of the tree) reachable from it, ensuring that a tree with minimal height is created. In order to find the minimum height spanning tree for *any* root, BFS has to be run repeatedly, once for each vertex, using each as the source once, and keep track of the minimum height tree that is found.

Problem 4.

$$\begin{tikzcd} a \arrow[r] \arrow[rr, bend left] & b & c \arrow[l] \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r] \arrow[rr, bend left] & b & c \end{tikzcd} \qquad \begin{tikzcd} a \arrow[rr, bend left] & b & c \arrow[l] \end{tikzcd}$$

Left = input graph, Middle = output tree of DFS (height 1), Right = max. height tree (height 2). Note that DFS *can* find a maximum height tree but it depends on the order of traversal of the graph and it is not clear how to make the "right" choice of which vertex to visit next at each step.

Problem 5. Only the loop that checks for negative cycles in the original pseudocode (lines 5-7 on page 651 in CLRS 3rd ed.) needs to be changed. Those three lines should be replaced by the following:

```
for i = 1 to |G.V|
   for each edge (u,v) $\in$ G.E
      if v.d > u.d + w(u,v)
         v.d = $-\infty$
```

Every vertex whose `.d` can still be improved after the main loop is part of a negative cycle. But not all vertices which are part of a negative cycle are necessarily found in a single loop over all edges. Rather, it could happen that in each loop over all edges, only one new vertex that is part of a negative cycle is identified. Therefore, in the worst case it is necessary to iterate over the edges `|G.V|` times.

Problem 6.

$$\begin{tikzcd} s \arrow[r, "2"] \arrow[rr, bend left, "1"] & a \arrow[r, "-2"] & b \arrow[r, "1"] & c \end{tikzcd} \qquad \begin{tikzcd} s \arrow[r, "4"] \arrow[rr, bend left, "3"] & a \arrow[r, "0"] & b \arrow[r, "3"] & c \end{tikzcd}$$

The shortest path from s to b in the original graph on the left is $< s, a, b, c >$ but in the modified graph on the right it is $< s, b, c >$. Running Dijkstra's algorithm directly on the left graph also yields an incorrect result because while it would find the correct shortest path from s to c, it would assign the wrong distance to c, 2 instead of 1, because when b's adjacency list is explored b has a distance of 1, not the final 0. This could lead to incorrect paths if the graph was larger.

Problem 7. See page 695 and 696 in the book for details on the matrix Π.

```
FLOYD-WARSHALL(W)
  $n = W$.rows
  $D^{(0)} = W$
  let $\Pi^{(0)} = (\pi_{ij}^{(0)})$ be a new $n \times n$ matrix
  for $i$ = 1 to $n$
    for $j$ = 1 to $n$
      if $W_{ij} < \infty$
        $\Pi_{ij}^{(0)} = i$
      else
        $\Pi_{ij}^{(0)}$ = NIL
  for $k$ = 1 to $n$
    let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
    let $\Pi^{(k)} = (\pi_{ij}^{(k)})$ be a new $n \times n$ matrix
    for $i$ = 1 to $n$
      for $j$ = 1 to $n$
        if $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
          $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
          $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$
        else
          $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
          $\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)}$
  return $D^{(n)}, \Pi^{(n)}$

PRINT-PATH($\Pi$, $u$, $v$)
  if $u$ == $v$
    print $u$
  else if $\Pi[u][v]$ == NIL
    print 'there is no path from $u$ to $v$'
  else
    PRINT-PATH($\Pi, u, \Pi[u][v]$)
    print $v$
```

Extra Credit 1. The modified version of Disjkstra's algorithm computes incorrect results when there are multiple paths between the source and a target vertex and those paths have different lengths. That is because in this case a different total weight will be added to the different paths – the same weight per edge but for a different number of edges. So the algorithm will compute correct results if for all pairs of vertices all paths connecting them consist of the same number of edges. Note that this condition is sufficient but a little stricter than necessary.

Extra Credit 2.

$D^{(0)}$	1	2	3	4	5	6
1	0	∞	1	∞	∞	∞
2	2	0	6	3	4	5
3	∞	∞	0	∞	∞	-4
4	-3	∞	∞	0	∞	∞
5	∞	∞	∞	-2	0	-1
6	∞	∞	∞	7	∞	0

$D^{(1)}$	1	2	3	4	5	6
1	0	∞	1	∞	∞	∞
2	2	0	3	3	4	5
3	∞	∞	0	∞	∞	-4
4	-3	∞	-2	0	∞	∞
5	∞	∞	∞	-2	0	-1
6	∞	∞	∞	7	∞	0

$D^{(2)}$	1	2	3	4	5	6
1	0	∞	1	∞	∞	∞
2	2	0	3	3	4	5
3	∞	∞	0	∞	∞	-4
4	-3	∞	-2	0	∞	∞
5	∞	∞	∞	-2	0	-1
6	∞	∞	∞	7	∞	0

$D^{(3)}$	1	2	3	4	5	6
1	0	∞	1	∞	∞	-3
2	2	0	3	3	4	-1
3	∞	∞	0	∞	∞	-4
4	-3	∞	-2	0	∞	-6
5	∞	∞	∞	-2	0	-1
6	∞	∞	∞	7	∞	0

$D^{(4)}$	1	2	3	4	5	6
1	0	∞	1	∞	∞	-3
2	0	0	1	3	4	-3
3	∞	∞	0	∞	∞	-4
4	-3	∞	-2	0	∞	-6
5	-5	∞	-4	-2	0	-8
6	4	∞	5	7	∞	0

$D^{(5)}$	1	2	3	4	5	6
1	0	∞	1	∞	∞	-3
2	-1	0	0	2	4	-4
3	∞	∞	0	∞	∞	-4
4	-3	∞	-2	0	∞	-6
5	-5	∞	-4	-2	0	-8
6	4	∞	5	7	∞	0

$D^{(6)}$	1	2	3	4	5	6
1	0	∞	1	4	∞	-3
2	-1	0	0	2	4	-4
3	0	∞	0	3	∞	-4
4	-3	∞	-2	0	∞	-6
5	-5	∞	-4	-2	0	-8
6	4	∞	5	7	∞	0

Extra Credit 3.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$p[i]$	a	a	b	a	a	b	b	a	a	b	a	a	a	b	a	b
$\pi(i)$	0	1	0	1	2	3	0	1	2	3	4	5	2	3	4	0

Extra Credit 4. Subproblem solutions, path to optimum bold and italic:

		E	X	P	E	D	I	T	I	O	U	S
	0	***1***	***2***	3	4	5	6	7	8	9	10	11
P	1	1	2	***2***	3	4	5	6	7	8	9	10
E	2	1	2	3	***2***	3	4	5	6	7	8	9
R	3	2	2	3	***3***	3	4	5	6	7	8	9
N	4	3	3	3	4	***4***	4	5	6	7	8	9
I	5	4	4	4	4	5	***4***	5	5	6	7	8
C	6	5	5	5	5	5	5	***5***	6	6	7	8
I	7	6	6	6	6	6	5	6	***5***	6	7	8
O	8	7	7	7	7	7	6	6	6	***5***	6	7
U	9	8	8	8	8	8	7	7	7	6	***5***	6
S	10	9	9	9	9	9	8	8	8	7	6	***5***

Optimal alignment, using 2 deletions (D), 2 substitutions (S), and 1 insertion:

E	X	P	E	D	-	I	T	I	O	U	S
D	D			S	I		S				
-	-	P	E	R	N	I	C	I	O	U	S

Extra Credit 5. The edit distance allowing insertions, deletions, substitutions, *and* transpositions is called Levenshtein-Damerau distance (see Wikipedia) and is computed as follows:

$$LD(P,Q,i,j) = \begin{cases} \max(i,j) & \texttt{if}\ \min(i,j)=0 \\ \min\begin{cases} LD(P,Q,i-1,j)+1 \\ LD(P,Q,i,j-1)+1 \\ LD(P,Q,i-1,j-1)+1_{P[i]\neq Q[j]} \\ LD(P,Q,i-2,j-2)+1 \end{cases} & \texttt{if}\ \ i,j>1, P[i]=Q[j-1]\ \texttt{and}\ P[i-1]=Q[j] \\ \min\begin{cases} LD(P,Q,i-1,j)+1 \\ LD(P,Q,i,j-1)+1 \\ LD(P,Q,i-1,j-1)+1_{P[i]\neq Q[j]} \end{cases} & \texttt{otherwise} \end{cases}$$

$1_{P[i]\neq Q[j]}$ is 0 if $P[i]=Q[j]$ and 1 otherwise. The middle case covers transpositions, the other two are identical to Levenshtein distance. If there are at least 2 characters in both strings ($i,j>1$) and the last two characters in the two substrings are a transposition of each other ($P[i]=Q[j-1]$ and $P[i-1]=Q[j]$), then we have the option of using the optimal solution for the substrings that are two characters shorter ($LD(P,Q,i-2,j-2)$) and add just one operation. But of course even in this case of a possible transposition we still have the option of insertion, deletion, and substitution which explains the other subcases of the second main case. The third case remains like in Levenshtein distance.