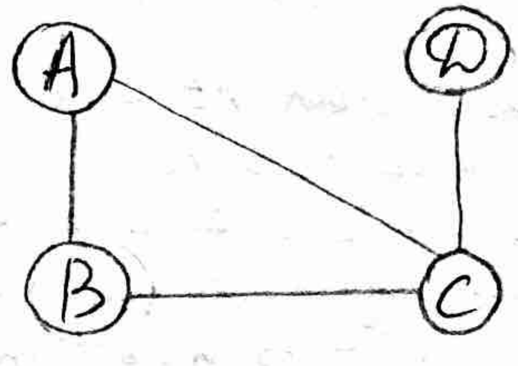
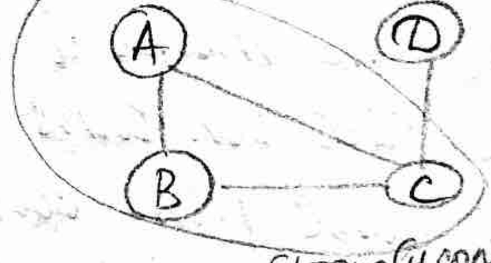
Problem #1



Consider the following graph with 4 vertices {A, B, C, D} and 4 edges A→B, B→C, C→A, C→D. During the first DFS run, D will always have the lowest finishing time. After that, we can assume that A has the highest finishing time. The strongly connected components consist of {A, B, C} & {D}. The incorrect algorithm will find the whole graph as one strongly connected component.

DFS of this graph starting at vertex A will be
⇒ A B C D



strongly connected component

Problem #2

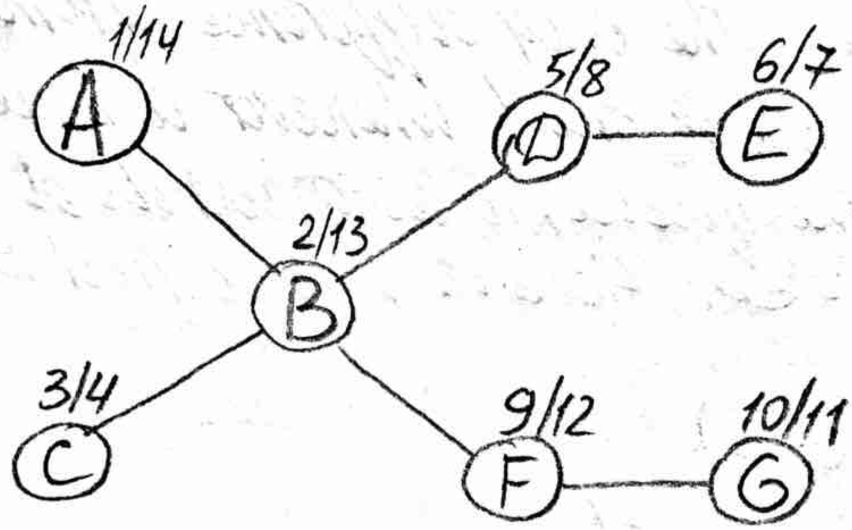
To create the spanning tree with maximum weight, we would have to start building a tree from the longest path into decreasing order.
So we can change Kruskal's algorithm as follows:

```
maximum-Kruskal (G, w)
A = ∅
for each vertex v ∈ G.V
        Make-Set(v)
sort edges of G.E into decreasing order by weight w
for each edge (u,v) ∈ G.E, taken in decreasing order by w
    if Find-Set(u) ≠ Find-Set(v)
            A = A ∪ {(u,v)}
            Union(u,v)
return A
```

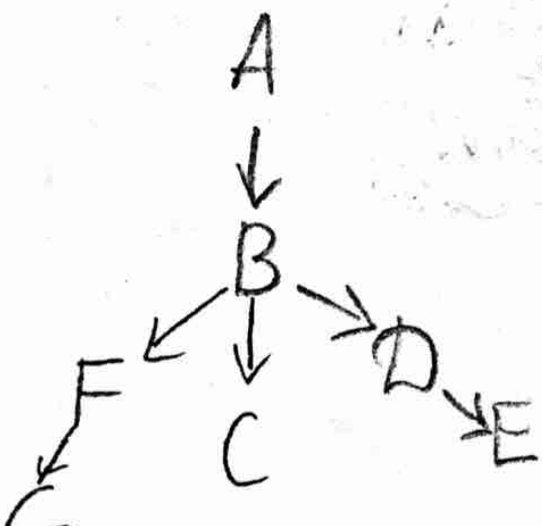
Problem #3 The way to find the spanning tree with minimum weight is to find the center of the graph (G), say 'a', so that the eccentricity of a is minimum among all the vertices of G. Also we make a tree with the blocks, where the root contains the vertex a and delete the edge from each block in such way that the height of the tree does not exceed the length of E(a). Hence the tree obtained from this algorithm is of minimum height.

Problem #4

Here is an example of a graph that proves that DFS doesn't work to find the maximum height spanning tree.



DFS



Actual

Problem #5

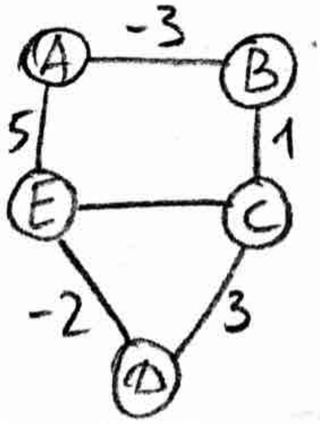
We can solve this problem by finding a node in the negative-weight cycle to set its weight to $-\infty$ and to run a BFS-like procedure on that node, setting the d values of reachable nodes to $-\infty$.
Here is the modified BFS:

```
ModifiedBFS(G, S)
    for i=1 to |V|[G|-1
        for each edge (u,v) ∈ E[G]
            RELAX(u, v, w)
    for each edge (u,v) ∈ E[G]
        if d[v] > d[u] + ∞[u,v]
            then d[v] = -∞
                dBFS(G, V)
```

dBFS - will have its d value set to $-\infty$, whenever a node is placed in the queue.

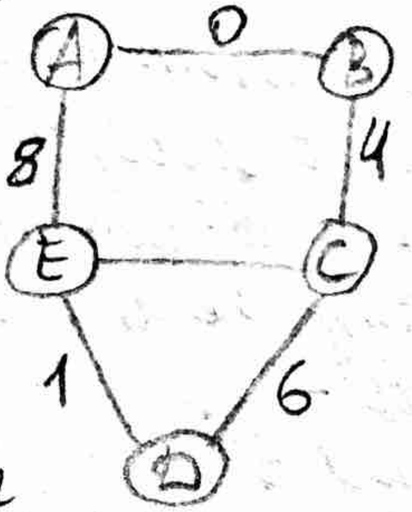
Problem #6

Here is an example of such graph



the shortest path from A to D right now is $A \to B \to C \to D$
(value = 1)

after adding 3 to all edges' weights we have graph with following edges' weights.



After the graph has been modified, shortest path from A to D is $A \to E \to D$ (value = 9) which proves that Dijkstra's algo doesn't work for graphs with negative weights.

Problem #7

Pseudocode for a function Print-Path(Π, i, j) will be as follows:

```
Print-Path(π, i, j)
  if i = j
     then print i
  else if π_ij = NIL
     print "no path from" i "to" j "exists".
  else Print-Path(π, i, π_ij)
  print j
```

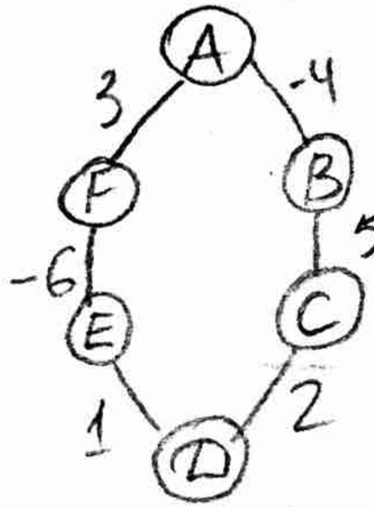
modified_Floyd-Warshall(W)

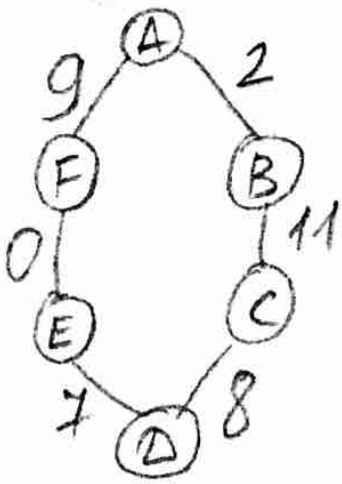
$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
            Print-Path$(D^{(k)}, i, j)$
return $D^{(n)}$

Extra credit #1

In order for a graph with negative weights to find only correct shortest paths, it needs to have the same # of paths from source to destination in <u>every</u> direction.

For example:



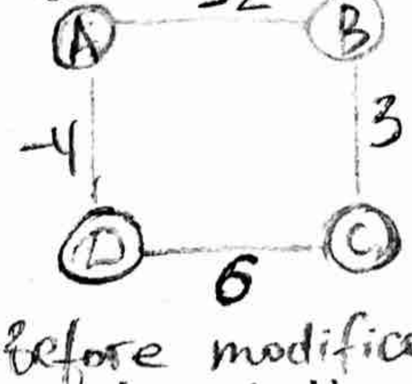


Shortest path before modification
$A \to F \to E \to D$
(value -2)

after modification
$A \to F \to E \to D$ (value 16)
is still the shortest path

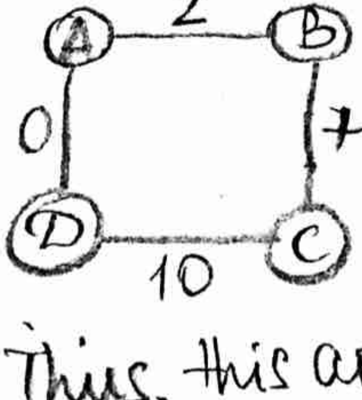
another example

A source
C - destination



before modification shortest path
$A \to B \to C$ (value 1)

after modification:
$A \to B \to C$ is still the Shortest path.



Thus, this argument is proven.

Extra credit 2

$$D^{(0)} = \begin{bmatrix} 0 & \infty & 1 & \infty & \infty & \infty \\ 2 & 0 & 6 & 3 & 4 & 5 \\ \infty & \infty & 0 & \infty & \infty & -4 \\ -3 & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & -2 & 0 & -1 \\ \infty & \infty & \infty & 7 & \infty & 0 \end{bmatrix}$$

$$D^{(1)} = \begin{bmatrix} 0 & \infty & 1 & \infty & \infty & -3 \\ 0 & 0 & 3 & 2 & 4 & 2 \\ \infty & \infty & 0 & 3 & \infty & -4 \\ -3 & \infty & -2 & 0 & \infty & \infty \\ -5 & \infty & \infty & -2 & 0 & -1 \\ 4 & \infty & \infty & 7 & \infty & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & \infty & 1 & 4 & \infty & -3 \\ -1 & 0 & 1 & 2 & 4 & -1 \\ 0 & \infty & 0 & 3 & \infty & -4 \\ -3 & \infty & -2 & 0 & \infty & -6 \\ -5 & \infty & -4 & -2 & 0 & -1 \\ 4 & \infty & 5 & 7 & \infty & 0 \end{bmatrix}$$

$$D^{(3)} = \begin{bmatrix} 0 & \infty & 1 & 4 & \infty & -3 \\ -1 & 0 & 0 & 2 & 4 & -3 \\ 0 & \infty & 0 & 3 & \infty & -4 \\ -3 & \infty & -2 & 0 & \infty & -6 \\ -5 & \infty & -4 & -2 & 0 & -8 \\ 4 & \infty & 5 & 7 & \infty & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & \infty & 1 & 4 & \infty & -3 \\ -1 & 0 & 0 & 2 & 4 & -4 \\ 0 & \infty & 0 & 3 & \infty & -4 \\ -3 & \infty & -2 & 0 & \infty & -6 \\ -5 & \infty & -4 & -2 & 0 & -8 \\ 4 & \infty & 5 & 7 & \infty & 0 \end{bmatrix}$$

$$D^{(5)} = \begin{bmatrix} 0 & \infty & 1 & 4 & \infty & -3 \\ -1 & 0 & 0 & 2 & 4 & -4 \\ 0 & \infty & 0 & 3 & \infty & -4 \\ -3 & \infty & -2 & 0 & \infty & -6 \\ -5 & \infty & -4 & -2 & 0 & -8 \\ 4 & \infty & 5 & 7 & \infty & 0 \end{bmatrix}$$

$$D^{(6)} = \begin{bmatrix} 0 & \infty & 1 & 4 & \infty & -3 \\ -1 & 0 & 0 & 2 & 4 & -4 \\ 0 & \infty & 0 & 3 & \infty & -4 \\ -3 & \infty & -2 & 0 & \infty & -6 \\ -5 & \infty & -4 & -2 & 0 & -8 \\ 4 & \infty & 5 & 7 & \infty & 0 \end{bmatrix}$$

Extra credit 3

Solution:

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
π[i]	0	1	0	1	2	3	0	1	2	3	4	5	2	3	4	0

page 4

Extra credit 4

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

We would have to make 5 replacements for these 2 words to be the same.

Extra credit #5 | The "new" edit distance is Damerau-Levenshtein distance, which considers insertion, deletion, substitution, and transposition of two adjacent characters. It's defined as follows:

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0, \\ \min\begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} & \text{if } i,j>1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \\ \min\begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

Where $d_{a,b}(i-2,j-2)+1$ corresponds to a transposition between two successive symbols.
Second case of recurrence relation is added due to considering transposition and this case would occur only when there is a case that 2nd last character of 'a' is same as the last character of b and vice versa.