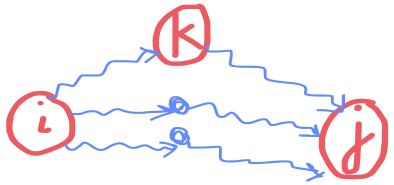
1. (25 points) Following the notes/slides/book, explain All-Source-Shortest-Paths by edges DP using matrix multiplication trick. Write pseudocode for ASSP-Fast and the corresponding Extending-SP procedures.

Solution:
To find the shortest path for ALL SOURCE SP

STEP 1 : Characterising the OPSOL

All subpaths of a shortest path are shortest paths
let us take the SP from i to j via k (k is the node that separates SP into 2 parts (m) edge that connects k to j and (m-1) edge.
Since i to j is the shortest path, the path k to j also has to be the shortest path between k to j .



STEP 2 : Recursive Defination

$c[i][j]^m$ is the minimum weight of any path from vertex i to j that contains at most m edges.
$c[i][j]^m$ = min $c[i][j]^{m-1}$, $c[i][k]^{m+1}$ + w[k,j]

STEP 3 : Pseudocode

ExtendedSP (c^{m-1} , w)

```
for i = 1 to n
   for j=1 to n
      dist = ∞
       for k = 1 to n
           dist = min { c[i][j]^(m-1) , c[i][k]^(m-1) + w[k][j] }
            c[i][j]^m = dist

 return c
```


Slow-All-Pairs Shortest-Paths (W)

n = w.rows

C^1 = W

For m = 2 to n - 1
  C^m = Extended-SPC^{m-1}, W)

return C^{n-1}

Improving the Runtime:
Our goal is not to compute all C^m matrices, but only C^{n-1}. If there is no negative cycle , then - $C^m = C^{n-1}$. we can compute C^{n-1} with only log(n-1) matrix products by using repeated squaring.

Because each log(n-1) matrix products is taking $O(n^3)$,algorithm runs in $\theta(n^3 log n)$

Fast-All-Pairs Shortest-Paths (W)

n = w.rows

C^1 = W
m=1

while m< n - 1
  C^{2m} = Extended-SP(C^m, C^m)
  m=2m

return C^m

---

2. (25 points) Exercise 24.1-3.

Solution:
Bellman Ford Algorithm is used to find the shortest path from a source vertex s to any vertex v in a weighted graph, by iteratively relaxing the edges. By relaxation , we can improve the ESP distance. To make Bellman Ford algorithm to terminate in m+1 passes, we can use a counter and initialise it to 0 . This counter will be "tracking" or "remembering" if any vertex v has been relaxed or not, basically the pass made so far.At every pass, we increment the counter by 1 .
If it has been relaxed, we can wait and check if it gets updated again. If it get updated means it was relaxed again. However, if it does not get updated, we can stop because there will be no more changes to its distance value.

Why will it terminate at m+1 pass? Because after the m^{th} pass we do not have any vertices which need to be relaxed.

---

3. (25 points) Exercise 24.2-2.

Solution:

DAG-Shortest-Path(G, w, s)

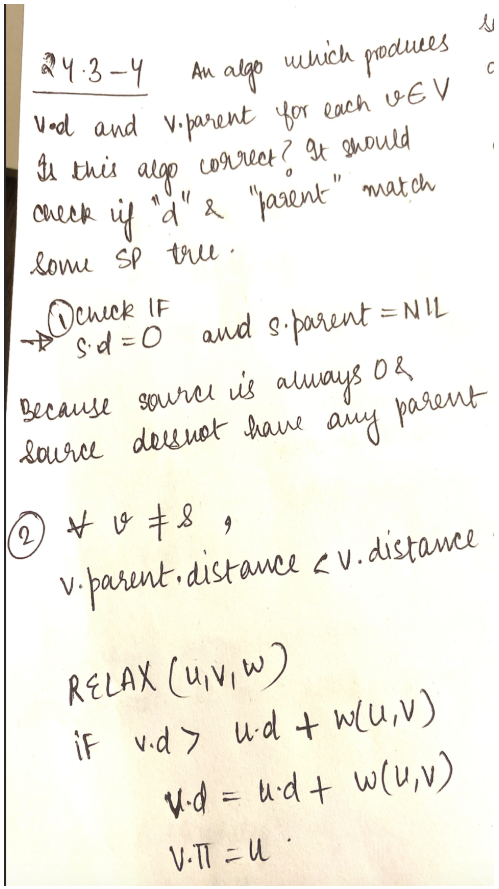
```
topologically sort the vertices of G
Initalise-Single-Source(G, s )

for first V-1 vertices taken in topologically sorted order
    for each vertex v belongs to G.Adj[u ]
        Relax( u, v, w )
```

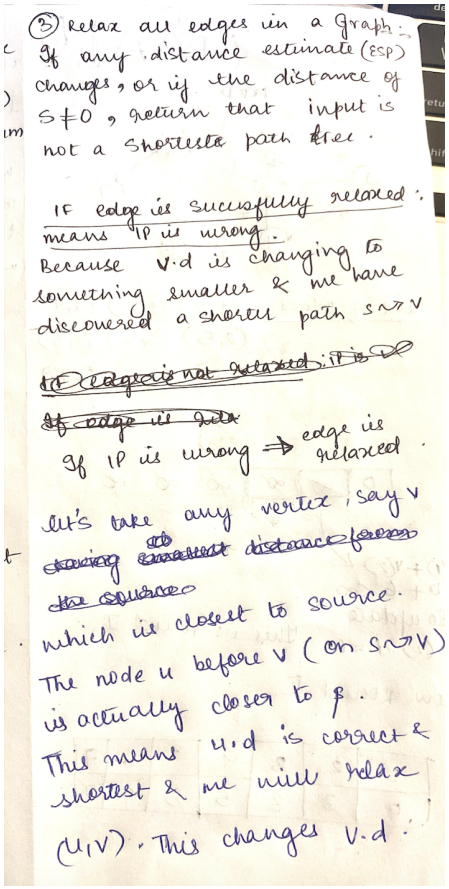
Changing line 3 in the program to "for first V-1 vertices in topologically sorted order" , it would alter the way vertices are processed in the algorithm. Algorithm would then process only the first V-1 vertex in topological order, this means the last vertex will not be reached or will not be visited. As a result, the shortest path of the last vertex will not be computed.

---

4. (25 points) Exercise 24.3-4.

Solution:
Uploading hand written answer, due to lack of time..





5. (Extra Credit 30 points) Problem 24-2.

6. (30 points) Explain in few lines the concept of transitive closure.

Solution:

Given a directed graph G, transitive closure is the “reachability” of each node to other node. We represent “ 1” as the node can reach to other node and “ 0 or ∞ “ if it cannot. Formally, we check if the **vertex j** is “reachable” from vertex i ∀ (i, j) in the graph. “reachable” means having a path , this is done using a reachability matrix.

To compute transitive closure in G , we assign weight of 1 to each edge of E and run the Floyd-Warshall algorithm. If for a vertex (i, j) , a path exists, we get D_ij < n , else D_ij = ∞ .

This is done in $O(n^3)$

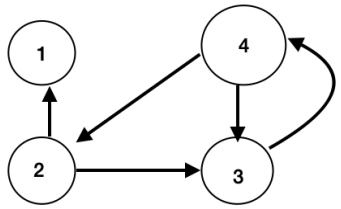
We can also use AND , OR operators to calculate transitive closure in $O(n^3)$

T[i][j] = 0 if i != j and (i, j) not belong to E ,
1 if i= j or (i, j) belong to E

for k > = 1 ,

$T[i][j]^k$ = $T[i][j]^{k-1}$ **OR** ($T[i][k]^{k-1}$ **AND** $T[k][j]^{k-1}$)

Using AND , OR Example from the book -



1	0	0	0
0	1	1	1
0	1	1	0
1	0	1	1

T(0)

1	0	0	0
0	1	1	1
0	1	1	0
1	0	1	1

T(1)

1	0	0	0
0	1	1	1
0	1	1	1
1	0	1	1

T(2)

1	0	0	0
0	1	1	1
0	1	1	1
1	1	1	1

T(3)

1	0	0	0
1	1	1	1
1	1	1	1
1	1	1	1

T(4)

---

7. (20 points) Exercise 25.1-6 (the book uses different notation for the matrices). Also explain how to use this result in order to display all-pair shortest paths, enumerating intermediary vertices (or edges) for each path.

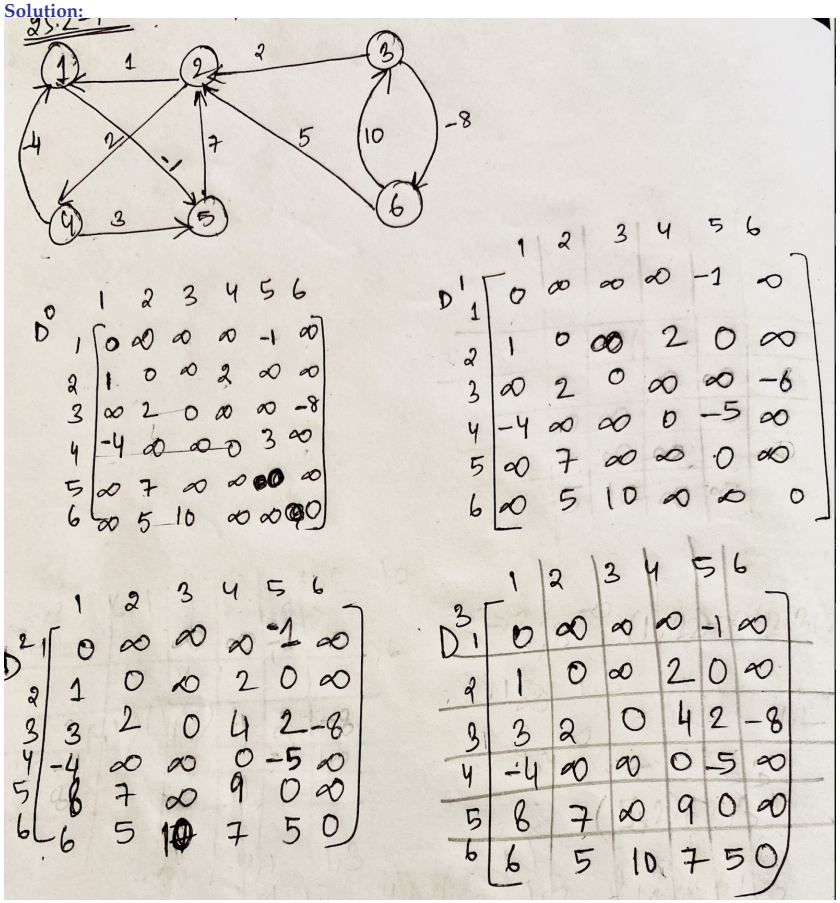
Solution:

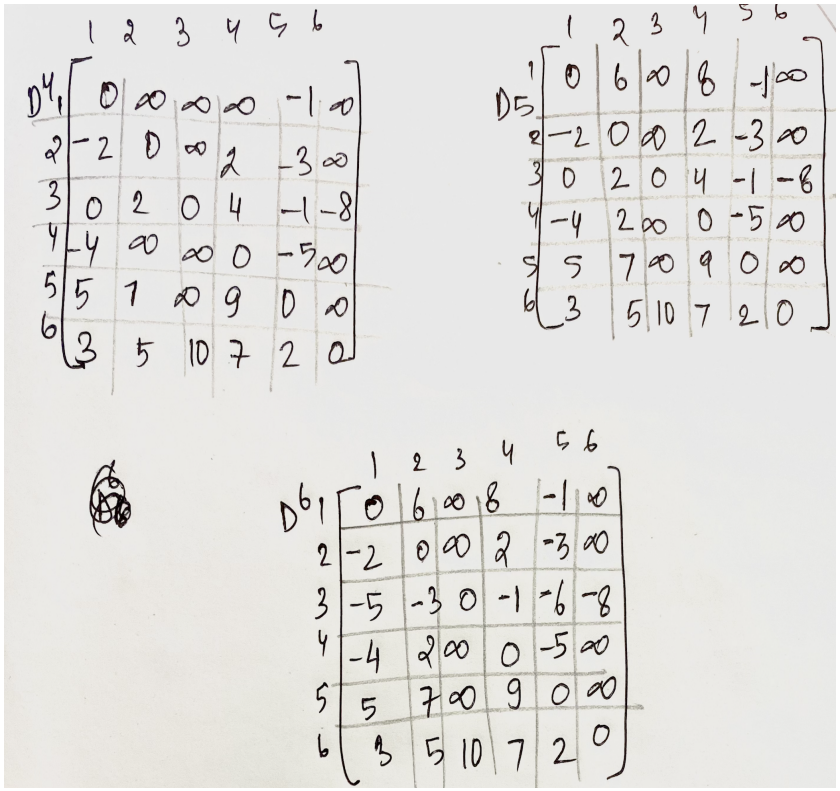
//collaborated from a site
(1) We initialise a Predecessor Matrix P for edge pairs (i,j) and initialise all its values to -1 .
(2)Now we check L matrix for every (i,j) pair. If an edge exists for (i,j) in L where L(i, j) = w then P(i,j) = i , this would mean SP from $i -> j$ is direct.
(3)If P(i, j) is -1, set P(i, j) to value of P(i, k) because SP from $i -> j$ goes through k vertex.
(4) Repeat till all values are filled in P and no more values are updating in P matrix.

---

8. (20 points) Exercise 25.2-1.

Solution:





9. (20 points) Exercise 25.2-4.

Solution:

Flyod Warshall computes SP between every pair of vertices which go via an intermediate vertex and this SP is computed at every iteration.

In the Question, "dropping superscripts" would mean we only store the values of dij at each iteration. Because we only need value of dij(k) to compute dij(k+1) and we dont need to keep a track of those values. We are computing the weight of a shortest path from to $i->j$ where intermediate vertices 1..k . If we use d_{ik}^{k}, we will use subpath from i to k . Also, k cannot be an intermediate vertex between i to k , otherwise it would means cycle exists.

By dropping superscripts, it would not change our final result because we are dropping info we dont need for the final solution. This implies $O(n^2)$ space is required to store dij at every iteration, instead of the O(n^3) required by Floyd Warshall.