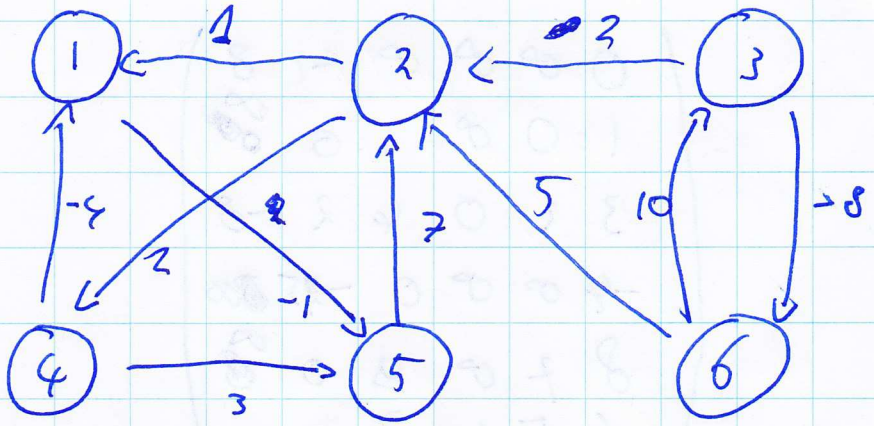
Assignment 10 –

I.



$D^{(0)}$:

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$D^{(1)}$:

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 6 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 3 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$D^{(2)}$:

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & 8 \\ 1 & 0 & \infty & 2 & \infty & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$D^{(3)}$:

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & 8 \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$D^{(4)}$:

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & 8 \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$D^{(5)}$:

$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & -2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$D^{(6)}$:

$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & 8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

II

Algorithm

```
Floyd-Warshall (n x n matrix P^(0), n x n matrix node^(0))
    for (k=1 to n)
    {
        D^(k) = new n x n matrix; node^(k) = new n x n matrix;
        for (i=1 to n)
            for (j=1 to n)
                if (D^(k-1)[i][k] + D^(k-1)[k][j] < D^(k-1)[i][j]) {
                    node^(k)[i][j] = k
                D^(k)[i][j] = min(...);
    }
    return node^(k);
```

node is an input matrix in this form:

$$node^{(0)} = \begin{pmatrix} - & 2 & - & - & 5 & - \\ 1 & - & - & - & 5 & 6 \\ 1 & 2 & - & 4 & 5 & 6 \\ - & - & 3 & - & - & - \\ - & 2 & - & - & - & - \\ - & - & - & - & 5 & - \end{pmatrix}$$

(node for shortest path to the node)

```
print nodes (n x n matrix node^(0))
    for (i=1 to n)
    {   for (j=1 to n)
        {
            if (i ≠ j) {
                print("Path: ")
    pi = i      while (D[i][j] ≠ j) node = D[i][j]
    pi = i      while (node ≠ j)
                while (D[i][j] ≠ j)
                    print(D[i][j])
                    i = D[i][j]

                i = pi
                newline
```

III Algorithm

```
Matrix-chain-multiply(A, s, i, j) {

    if (i == j)
        return A[i];

    if (i+1 == j)
        return A[i] · A[j];

matrixA = Matrix-chain-multiply(A, s, i, s[i,j])
matrixB = Matrix-chain-multiply(A, s, s[i,j], j)

return matrixA · matrixB
```

IV

Algorithm

```
Max_values (A, i, j)

    if (i == j)
        return (i, j)
    else
        right = max_values (A, i+1, j)
        left  = max_values (A, i, j-1)
        total = (i, j)

        return highest value of
               Count_total (A, right | left | total)

Count_total (A, (i, j))
    total = 0
    for (int k = 0 ; k < j ; k++)
        total += A[k]
    return total
```

b. We can compute M(j) from M(j-1) by taking the highest value of:

M(j-1) or
~~M(j-1).to~~ one of the arrays in between M(j-1).to to ~~M(j)~~ A[j] or
M(j-1).from to A[j] or
none at all

c. Algorithm

(i) We create an array of tuples called B. The first element of each tuple represents the position of the start of the max ~~count~~ sub-array and the second element the stop position. B[i] will be the tuple of the array with length i.

(ii) We will iterate over the array A and keep track of B[i] in the following way:
- (a) We update B[j] to a new tuple: (we take the largest
- (b) ~~if the sum of M(j-1) is greater~~ corresponding to b