In [ ]:
```
start = time.time()
RHS = A @ (B @ C)
end = time.time()
print(end - start)
```

```
0.026722192764282227
```

In [ ]:
```
start = time.time()
RHS = A @ (B @ C)
end = time.time()
print(end - start)
```

```
0.025452852249145508
```

In [ ]:
```
np.linalg.norm(LHS - RHS)
```

Out[ ]:
```
4.704725926477414e-10
```

In [ ]:
```
start = time.time()
D = A @ B @ C      #Evaluated as (A@B)@C or as A@(B@C)?
end = time.time()
print(end - start)
```

```
0.24454998970031738
```

From the above, we see that evaluating `(A@B)@C` takes around 10 times as much time as evaluating `A@(B@C)`, which is predicted from the complexities. In the last line, we deduce that `$A@B@C$` is evaluated left to right, as `$(A@B)@C$`. Note that for these particular matrices, this is the (much) slower order to multiply the matrices.

10.2. Composition of linear functions

Second difference matrix. We compute the second difference matrix on page 184 of VMLS.

In [ ]:
```
D = lambda n: np.c_[-np.identity(n-1), np.zeros(n-1)] +
↪  np.c_[np.zeros(n-1), np.identity(n-1)]
D(5)
```

Out[ ]:
```
array([[-1.,  1.,  0.,  0.,  0.],
       [ 0., -1.,  1.,  0.,  0.],
       [ 0.,  0., -1.,  1.,  0.],
```

```
       [ 0.,  0.,  0., -1.,  1.]])
```

```
In [ ]: D(4)
```

```
Out[ ]: array([[-1.,  1.,  0.,  0.],
               [ 0., -1.,  1.,  0.],
               [ 0.,  0., -1.,  1.]])
```

```
In [ ]: Delta = D(4) @ D(5) #Second difference matrix
        print(Delta)
```

```
[[ 1. -2.  1.  0.  0.]
 [ 0.  1. -2.  1.  0.]
 [ 0.  0.  1. -2.  1.]]
```

10.3. Matrix power

The kth power of a square matrix A is denoted A^k. In Python, this power is formed using `np.linalg.matrix_power(A,k)`. Let's form the adjacency matrix of the directed graph on VMLS page 186. Then let's find out how many cycles of length 8 there are, starting from each node. (A cycle is a path that starts and stops at the same node.)

```
In [ ]: A = np.array([[0,1,0,0,1], [1,0,1,0,0], [0,0,1,1,1], [1,0,0,0,0],
        ↪   [0,0,0,1,0]])
        np.linalg.matrix_power(A,2)
```

```
Out[ ]: array([[1, 0, 1, 1, 0],
               [0, 1, 1, 1, 2],
               [1, 0, 1, 2, 1],
               [0, 1, 0, 0, 1],
               [1, 0, 0, 0, 0]])
```

```
In [ ]: np.linalg.matrix_power(A,8)
```

```
Out[ ]: array([[18, 11, 15, 20, 20],
               [25, 14, 21, 28, 26],
               [24, 14, 20, 27, 26],
               [11,  6,  9, 12, 11],
               [ 6,  4,  5,  7,  7]])
```