PES UNIVERSITY

**Department of Computer Science and Engineering**

**PES UNIVERSITY**

**UE19CS251: Design and Analysis of Algorithm (4-0-0-4-4)**

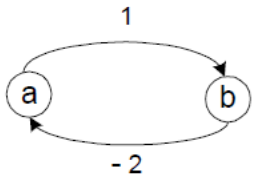
Q.1 what does dynamic programming have in common with divide-and conquer?
Solution:
a. Both techniques are based on dividing a problem's instance into smaller instances of the same problem.
b. Typically, divide-and-conquer divides an instance into smaller instances with no intersection whereas dynamic programming deals with problems in which smaller instances overlap. Consequently, divide-and-conquer algorithms do not explicitly store solutions to smaller instances and dynamic programming algorithms do.

Q.2 Write pseudocode of the bottom-up dynamic programming algorithm for the knapsack problem
Solution:

```
Bottom up Dynamic knapsack 0/1 (v[1..n], w[1..n], n, W)
for j = 0 to W do
  cost[0, w] = 0
for i = 1 to n do
  cost[i, 0] = 0
  for j = 1 to W do
    if wi ≤ j then
      if vi + cost[i-1, w-wi] then
        cost[i, w] = vi + cost[i-1, w-wi]
      else cost[i, w] = cost[i-1, w]
    else
      cost[i, w] = cost[i-1, w]
```

Q.3 Give an example of a graph with negative weights for which Floyd's algorithm does not yield the correct result

Solution:




Floyd's algorithm will yield:

$$D^{(0)} = \begin{bmatrix} 0 & 1 \\ -2 & 0 \end{bmatrix} \qquad D^{(1)} = \begin{bmatrix} 0 & 1 \\ -2 & -1 \end{bmatrix} \qquad D^{(2)} = \begin{bmatrix} -1 & 0 \\ -3 & -2 \end{bmatrix}$$

None of the four elements of the last matrix gives the correct value of the shortest path, which is, in fact, $-\infty$ because repeating the cycle enough times makes the length of a path arbitrarily small.

Q.4Enhance Floyd's algorithm so that shortest paths themselves, not just their lengths, can be found

Solution:

**Algorithm** *FloydEnhanced*$(W[1..n, 1..n])$
//Input: The weight matrix $W$ of a graph or a digraph
//Output: The distance matrix $D[1..n, 1..n]$ and

```
//          the matrix of intermediate updates P[1..n, 1..n]
D ← W
for i ← 1 to n do
    for j ← 1 to n do
        P[i, j] ← 0 //initial mark
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if D[i, k] + D[k, j] < D[i, j]
              D[i, j] ← D[i, k] + D[k, j]}
              P[i, j] ← k
```