Report
Assignment 4

Name - Parul Bansal
Roll No - B14116

Question 1 (Maximum-Flow Problem)

Psuedo Code of augment(f,p)

```
Let b = bottleneck (P , f )
For each edge (u, v) ∈ P
    If e = (u, v) is a forward edge then
        increase f (e) in G by b
    Else ( (u, v) is a backward edge, and let e = (v, u) )
        decrease f (e) in G by b
    Endif
Endfor
Return( f )
```

Psuedo Code of Max-Flow(G,s,t)

```
Initially f (e) = 0 for all e in G
While there is an s - t path in the residual graph G_f
    Let P be a simple s - t path in G_f
    f'= augment (f , P)
    Update f to be f'
    Update the residual graph G_f to be G_f'
Endwhile
Return f
```

Observations

- The result of augment (f , P) is a new flow f' in G, obtained by increasing and decreasing the flow values on edges of P.
- Algorithm implemented above is also called *Ford-Fulkerson Algorithm*.
- Time complexity of *Ford-Fulkerson algorithm* is O(EC), where C is the max_flow.
- To find an s-t path in G_f , we use breadth-first search.

Question 2 (0-1 Knapsack Problem)

Psuedo Code Knapsack(n, W)

```
Array M[0 . . . n, 0 . . . W]
Initialize M[0, w] = 0 for each w = 0, 1, . . . , W
For i = 1, 2, . . . , n
        For w = 0, . . . , W
                Use the recurrence
                  If w < w_i then OPT (i, w) = OPT (i − 1, w). Otherwise
                  OPT (i, w) = max( OPT (i − 1, w), v_i+ OPT (i − 1, w − w_i )).
                to compute M[i, w]

        Endfor
  Endfor
Return M[n, W]
```

Observations

- The Knapsack(n, W) Algorithm correctly computes the optimal value of the problem, and runs in O(nW) time.
- Dynamic Programming is used in above algorithm.

Question 3 (Dijkstra's algorithm)

Psuedo Code Dijkstra(Graph, Source)

```
create vertex set Q
for each vertex v in Graph:
        dist[v] ← INFINITY
     prev[v] ← UNDEFINED
        add v to Q

dist[source] ← 0
while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u:
                alt ← dist[u] + length(u, v)
                if alt < dist[v]:
                        dist[v] ← alt
                  prev[v] ← u

return dist[], prev[]
```

Observations

- Time complexity of above algorithm is O(ELogV)) .
- The code finds shortest distances from source to all vertices.
- Dijkstra’s algorithm doesn’t work for graphs with negative weight edges.
- It is a greedy algorithm.