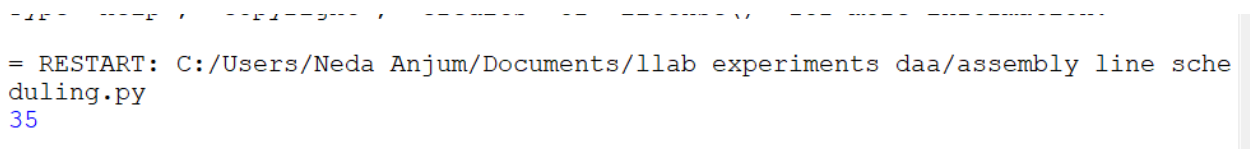
**Lab-11(18/6/24)**

**1.Assembly line scheduling:**

Code:

```
def fun(a, t, cl, cs, x1, x2, n):
    if cs == n - 1:
        if cl == 0:
            return x1
        else:
            return x2
    same = fun(a, t, cl, cs + 1, x1, x2, n) + a[cl][cs + 1]
    diff = fun(a, t, not cl, cs + 1, x1, x2, n) + a[not cl][cs + 1] + t[cl][cs + 1]
    return min(same, diff)
n = 4
a = [[4, 5, 3, 2], [2, 10, 1, 4]]
t = [[0, 7, 4, 5], [0, 9, 2, 8]]
e1 = 10
e2 = 12
x1 = 18
x2 = 7
x = fun(a, t, 0, 0, x1, x2, n) + e1 + a[0][0]
y = fun(a, t, 1, 0, x1, x2, n) + e2 + a[1][0]
print(min(x, y))
```

output:



```
= RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/assembly line sche
duling.py
35
```

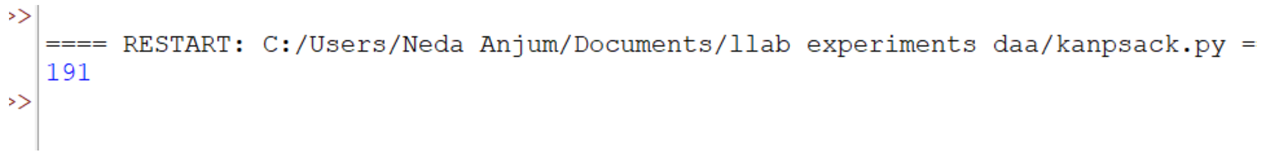
**2.kanpsack 0/1:**

Code:

```
def knapSack(W, wt, val, n):
        if n == 0 or W == 0:
                return 0

        if (wt[n-1] > W):
                return knapSack(W, wt, val, n-1)
        else:
                return max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                        knapSack(W, wt, val, n-1))
if __name__ == '__main__':
        profit = [20, 101, 70]
        weight = [10, 20, 30]
        W = 80
        n = len(profit)
        print (knapSack(W, weight, profit, n))
```

output:



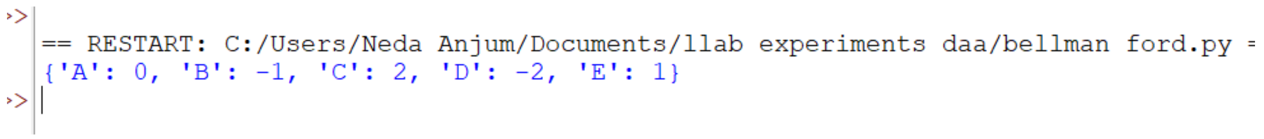
```
>>
==== RESTART: C:/Users/Neda Anjum/Documents/1lab experiments daa/kanpsack.py =
191
>>
```

**3. bellman ford algorithm:**

**Code:**

```
def bellman_ford(graph, source):
  distances = {vertex: float('inf') for vertex in graph}
  distances[source] = 0
  for _ in range(len(graph) - 1):
    for u in graph:
      for v, weight in graph[u].items():
        if distances[u] != float('inf') and distances[u] + weight < distances[v]:
                distances[v] = distances[u] + weight
    for u in graph:
        for v, weight in graph[u].items():
            if distances[u] != float('inf') and distances[u] + weight < distances[v]:
                raise ValueError("Graph contains negative weight cycle")
    return distances
graph = {
    'A': {'B': -1, 'C': 4},
    'B': {'C': 3, 'D': 2, 'E': 2},
    'C': {},
    'D': {'B': 1, 'C': 5},
    'E': {'D': -3}
}
source = 'A'
shortest_distances = bellman_ford(graph, source)
print(shortest_distances)
```

output:



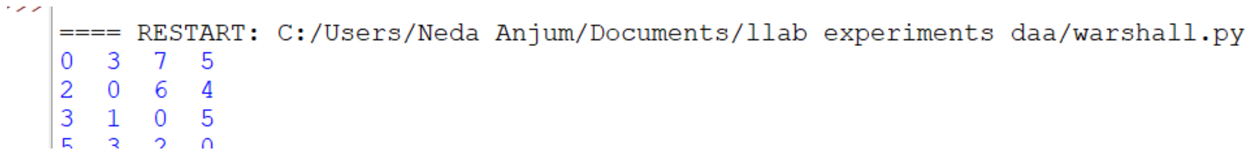
```
>
== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/bellman ford.py =
{'A': 0, 'B': -1, 'C': 2, 'D': -2, 'E': 1}
>
```

**4.floyds warshall algorithm:**

**Code:**

```
V = 4
INF = 999
def floyd_warshall(G):
    d = list(map(lambda i: list(map(lambda j: j, i)), G))
    for k in range(V):
        for i in range(V):
            for j in range(V):
                d[i][j] = min(d[i][j], d[i][k] + d[k][j])
    print_solution(d)
def print_solution(d):
    for i in range(V):
        for j in range(V):
            if(d[i][j] == INF):
                print("INF", end=" ")
            else:
                print(d[i][j], end="  ")
        print(" ")
G = [[0, 3, INF, 5],
     [2, 0, INF, 4],
     [INF, 1, 0, INF],
     [INF, INF, 2, 0]]
floyd_warshall(G)
```

output:



```
==== RESTART: C:/Users/Neda Anjum/Documents/llab experiments daa/warshall.py
0   3   7   5
2   0   6   4
3   1   0   5
5   3   2   0
```