**LAB 08: Dynamic Programming (Part 2)**

**III. Exercise**

**Warn-up**

1. Computing Longest Common Subsequence

def measure\_time\_1(func, N):

    runtime = []

    for n in N:

        start = time.time()

        f = func(n)

        stop = time.time()

        runtime.append(stop-start)

    return runtime

def measure\_time\_2(func, N1, N2):

    runtime = []

    for n in range (len(N1)):

        start = time.time()

        f = func(N1[n], N2[n])

        stop = time.time()

        runtime.append(stop-start)

    return runtime

def measure\_time\_3(func, N1, N2, N3):

    runtime = []

    for n in range (len(N1)):

        start = time.time()

        f = func(N1[n], N2[n], N3[n])

        stop = time.time()

        runtime.append(stop-start)

    return runtime

**Intermediate exercises**

1. Computing the transitive closure of a relation

class Graph:

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = [[0 for column in range(vertices)] for row in range(vertices)]

    def printSolution(self, reach):

        print("Following matrix transitive closure of the given graph")

        for i in range(self.V):

            for j in range(self.V):

                print("%d" % reach[i][j], end=' ')

            print("")

        print("")

        return

def Warshall(graph):

    reach = [[0 for i in range(graph.V)] for j in range(graph.V)]

    for i in range(graph.V):

        for j in range(graph.V):

            reach[i][j] = graph.graph[i][j]

    for k in range(graph.V):

        for i in range(graph.V):

            for j in range(graph.V):

                reach[i][j] = reach[i][j] or (reach[i][k] and reach[k][j])

    graph.printSolution(reach)

if \_\_name\_\_ == "\_\_main\_\_":

    g = Graph(4)

    g.graph = [[0, 1, 0, 1],

               [0, 0, 1, 0],

               [0, 0, 0, 1],

               [0, 0, 0, 0]]

    Warshall(g)

1. All pairs shortest path distances.

def printSolution(D):

    print("Following matrix is the shortest distances between every pair of vertices")

    for i in range(len(D)):

        for j in range(len(D)):

            print("%7d" % D[i][j], end=' ')

        print("")

    return

def Floyd(W):

    D = [[0 for i in range(len(W))] for j in range(len(W))]

    for i in range(len(W)):

        for j in range(len(W)):

            D[i][j] = W[i][j]

    for k in range(len(W)):

        for i in range(len(W)):

            for j in range(len(W)):

                D[i][j] = min(D[i][j], D[i][k] + D[k][j])

    printSolution(D)

    return

if \_\_name\_\_ == "\_\_main\_\_":

    W = [[0, 5, 999, 10],

         [999, 0, 3, 999],

         [999, 999, 0, 1],

         [999, 999, 999, 0]]

    Floyd(W)