LAB 10

1.ASSEMBLY LANGUAGE SEQUENCING

def ass(a, t, e, x):

n = len(a[0])

T1 = [0] \* n

T2 = [0] \* n

T1[0] = e[0] + a[0][0]

T2[0] = e[1] + a[1][0]

for j in range(1, n):

T1[j] = min(T1[j-1] + a[0][j], T2[j-1] + t[1][j-1] + a[0][j])

T2[j] = min(T2[j-1] + a[1][j], T1[j-1] + t[0][j-1] + a[1][j])

final\_time = min(T1[n-1] + x[0], T2[n-1] + x[1])

return final\_time

n = int(input("Enter the number of stations: "))

a = [[], []]

t = [[], []]

for i in range(n):

time = int(input("Enter node value for line 1: "))

a[0].append(time)

for i in range(n):

time = int(input("Enter node value for line 2: "))

a[1].append(time)

for i in range(n-1):

time = int(input("Transfer value for line 1 to 2: "))

t[0].append(time)

t[0].insert(0, 0)

for i in range(n-1):

time = int(input("Transfer value for line 2 to 1: "))

t[1].append(time)

t[1].insert(0, 0)

e = []

for i in range(2):

time=int(input("Enter the entry value: " ))

e.append(time)

x = []

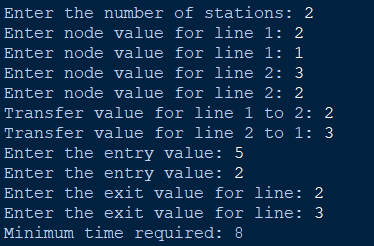
for i in range(2):

time=int(input("Enter the exit value for line: "))

x.append(time)

print("Minimum time required:", ass(a, t, e, x))

OUTPUT:



2.FLOYD WARSHELL

def flo(g):

n=len(g)

d=[list(i) for i in g]

for i in range(n):

for j in range(n):

for k in range(n):

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

return d

n=int(input("No of edges: "))

a=[[0]\*n for \_ in range(n)]

for i in range(n):

for j in range(n):

x=float(input("enter the graph: "))

a[i][j]=x

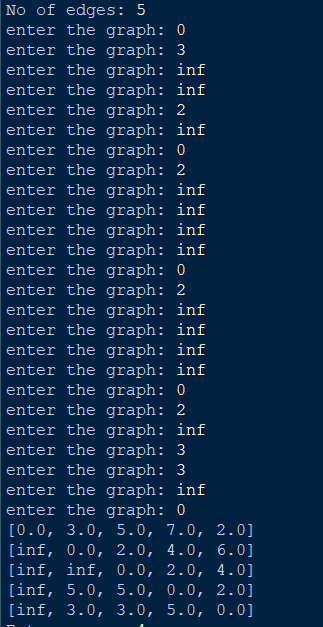
result=flo(a)

if result:

for i in result:

print(i)

OUTPUT:



3.knapsack

def k(w, c, key,n):

n = len(w)

dp = [[0 for \_ in range(key + 1)] for \_ in range(n + 1)]

for i in range(1, n + 1):

for j in range(1, key + 1):

if w[i - 1] <= j:

dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i - 1]] + c[i - 1])

else:

dp[i][j] = dp[i - 1][j]

p = dp[n][key]

res = []

j = key

for i in range(n, 0, -1):

if p <= 0:

break

if p != dp[i - 1][j]:

res.append(i - 1)

p -= c[i - 1]

j -= w[i - 1]

return res, dp[n][key]

w,c=[],[]

n=int(input("Enter range: "))

for i in range(n):

x=int(input("Enter Weight: "))

w.append(x)

y=int(input("Enter cost: "))

c.append(y)

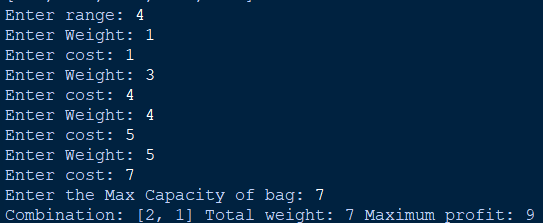
key=int(input("Enter the Max Capacity of bag: "))

r, p = k(w, c, key,n)

tw = sum(w[i] for i in r)

print("Combination:", r, "Total weight:", tw, "Maximum profit:", p)

OUTPUT:



4.bellman-ford

def bf(v, edges, s):

d = [float("inf")] \* v

d[s] = 0

for \_ in range(v - 1):

for u, v, w in edges:

if d[u] != float("inf") and d[u] + w < d[v]:

d[v] = d[u] + w

for u, v, w in edges:

if d[u] != float("inf") and d[u] + w < d[v]:

print("Graph contains negative weight cycle")

return None

return d

v = int(input("Enter the number of vertices: "))

e = int(input("Enter the number of edges: "))

edges = []

print("Enter each edge as source, destination, weight (space-separated):")

for \_ in range(e):

u, v, w = map(int, input().split())

edges.append((u, v, w))

s = int(input("Enter the source vertex: "))

distances = bf(v, edges, s)

if distances is not None:

print("Vertex Distance from Source")

for i in range(v):

print(f"{i}\t\t{distances[i]}")

OUTPUT:

