## CSCI406 Dynamic Programming Group Project

Contributors: Jane Lockshin, Jacob McBee, Naveen Shankar Mitikiri, Matthew Peters

2. To develop an optimal solution to this problem, a good approach is to realize that at any given step, the optimal cost is equal to the cost of doing operations in the present city plus the cost of relocating to this city from any other city. With this information in mind, we can solve it through the following recursive relation:

|  |  |
| --- | --- |
|  | The number of months up to where the current optimal path is being calculated, including , a month previous to the first month in any given dataset |
|  | The number of cities in the dataset, 0 indexed |
|  | The opt\_path function, which takes an arbitrary city and a month |
|  | The cost of operating at city during the month of |
|  | The cost of relocating from city to city |

Basis Step:

Recursive Step:

ii. Dynamic Programming Algorithm:  
 function **opt\_path\_dp**( = cities, = months, = fixed costs):  
 // initialize a ( x ) matrix with 0s  
 M = [0 for i in | |][0 for j in (| |) ]

// initialize first column of matrix with starting costs from each city for i in ||:  
 M[i][0] = [i][0]

// populate cells column-major  
 for j in | |:  
 for i in | |:  
 costs = []  
 for c in | |:  
 costs += M[(i+c) % | |][j-1] + [i][(i+c) % | |]  
 M[i][j] = min(costs) + [i][j]

(M[i][j] is the min value of the (cell in the previous column (j -1) + its transportation costs to city ) + cost at city in present month (loop over all cities from previous month to find the lowest cost at the th city to finalize matrix).

iii. Traceback Algorithm:

b) Complexity of Algorithm:

Using dynamic programming, the complexity of our algorithm is . This is because of a double for-loop which first iterates over every month, and as a sub-step of each month iteration iterates over every city. If we had instead gone with the typical recursive approach, our complexity would have been .

c) Implementation:

def opt\_path\_dp(cities, months, fixed\_costs):

*""" DP function that takes in cities, months*

*and returns the lowest cost/optimal path """*

# Initialize matrix

M = [[0 for j in range(len(months[0]))] for i in range(len(cities))]

# Initialize first column (starting costs from each city)

for i in range(len(cities)):

M[i][0] = months[i][0]

# Populate cells column-major

nc = len(cities)

for j in range(1, len(months[0])):

for i in range(nc):

costs = []

for c, nothing in enumerate(cities):

costs.append(M[(i + c) % nc][j-1] + fixed\_costs[i][(i + c) % nc])

min\_cost\_0 = min(costs)

min\_cost\_0 += months[i][j]

M[i][j] = min\_cost\_0

# Traceback

last\_costs = []

for i in range(len(cities)):

last\_costs.append(M[i][-1])

min\_cost = min(last\_costs)

mc\_loc = last\_costs.index(min\_cost)

trace = cities[mc\_loc]

for m, cost in reversed(list(enumerate(M[0][:-1]))):

temp\_arr = []

for c in range(len(cities)):

temp\_arr.append(M[c][m] + fixed\_costs[c][mc\_loc])

temp\_min = min(temp\_arr)

mc\_loc = temp\_arr.index(temp\_min)

trace = cities[mc\_loc] + " " + trace

return min\_cost, trace

def main():

city\_names = []

paths = []

fixed\_costs = []

additional\_city = raw\_input("New city name: [Enter N when done] ")

while additional\_city != "N" and additional\_city != "n":

city\_path = raw\_input("Enter month-by-month cost for city (use spaces): ")

city\_names += [additional\_city]

paths.append([int(x) for x in city\_path.split(',')])

additional\_city = raw\_input("New city name: [Enter N when done] ")

for i in range(len(city\_names)):

transport\_costs = []

for j in range(len(city\_names)):

fij = int(raw\_input("Cost to move from " + city\_names[i] + " to " + city\_names[j] + ": "))

transport\_costs.append(fij)

fixed\_costs.append(transport\_costs)

ans = opt\_path\_dp(city\_names, paths, fixed\_costs)

print(ans[0])

print(ans[1])

main()

d) Optimal Solution:

145

NY NY DEN DEN DEN DEN DEN LA LA LA LA LA