

پروژه درس تحقیق در عملیات - مهدی جهانی - ۱۹۸۶۰۱۰

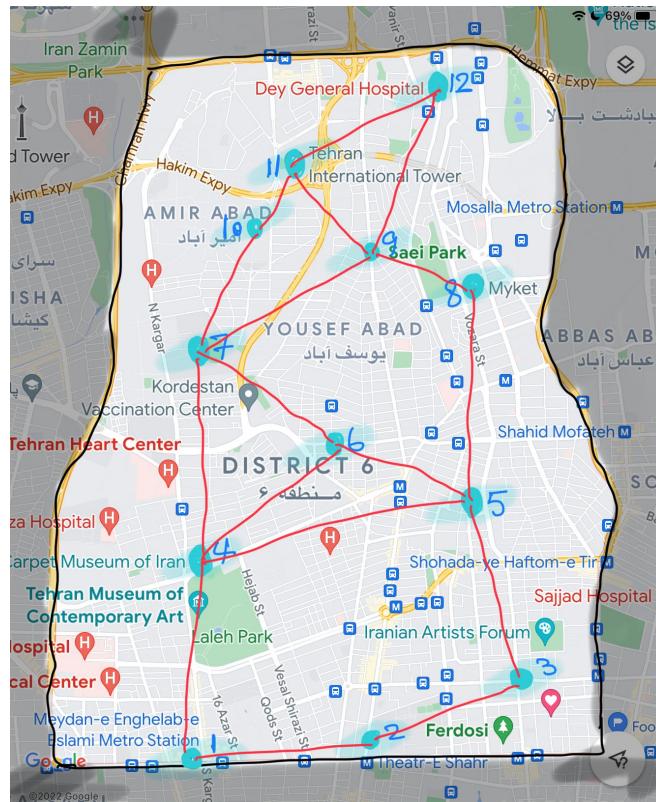
تعریف پروژه

پروژه تعریف شده در اینجا شباهت بسیار زیادی به مسئله «فروشنده دوره‌گرد» که در کلاس به عنوان اولین مثال از dynamic programming بررسی شد، دارد. برای حل این مسئله از همان الگو و اصول پیروی می‌کنیم. تفاوت این مسئله با نمونه‌ای که در کلاس حل شد در این است که گراف جهت‌دار است و وزن یال‌های رفت و برگشت بین دو راس لزوماً برابر نیست.

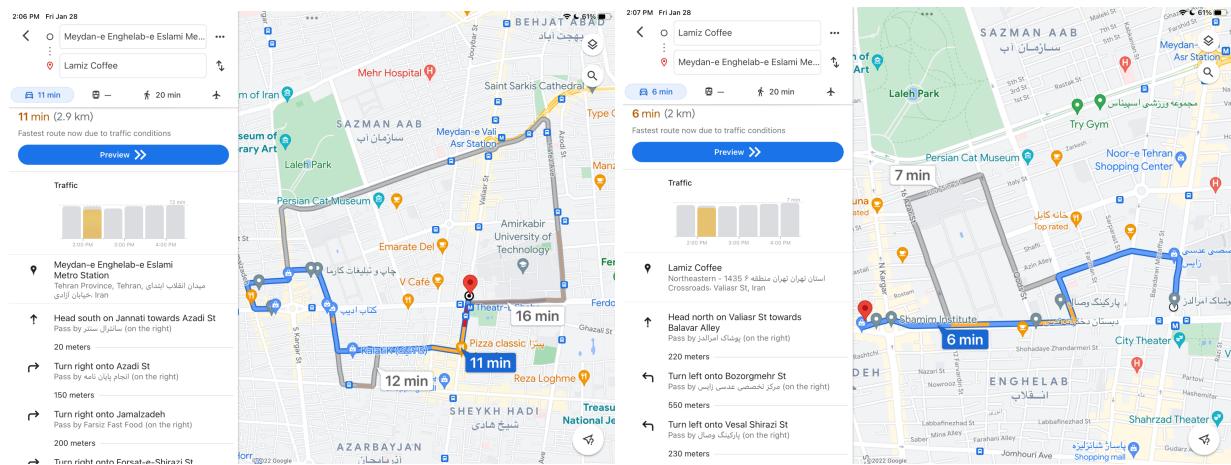
مدل‌سازی مسئله

با توجه به دستور پروژه مدل‌سازی این پروژه به صورت گراف و با کمک از سرویس google map انجام شده است. منطقه انتخاب شده در این پروژه منطقه ۶ تهران هست. از این منطقه ۱۲ نقطه انتخاب شده و تلاش کردیم تا توزیع این نقاط در منطقه ۶ به صورت یکنواخت بوده و تمام نواحی را پوشش دهد (شکل ۱). درست است که تمامی رئوس این گراف می‌توانند به یکدیگر وصل باشند و گراف کامل داشته باشیم اما برای این که مسئله بهینه‌سازی معنادار باشد، تنها بین برخی از رئوس یال در نظر گرفته‌ایم.

وزن هر یال در این گراف معادل فاصله زمانی است که برای رفتن از یک راس به راس دیگر در دنیای واقعی باید صرف شود. برای استخراج این داده از سرویس مسیریابی google map استفاده کرده‌ایم (یک نمونه از این وزن‌دهی در شکل ۲ و ۳ نشان داده شده است). به این ترتیب برای هر دو راس در گراف به دنبال کمینه کردن وزن مسیری هستیم که بین این دو وجود دارد.



نقاط انتخاب شده در منطقه ۶ و مدلسازی مستعله.



فاصله زمانی بین دو نقطه از نقاط انتخاب شده و تفاوت آنها

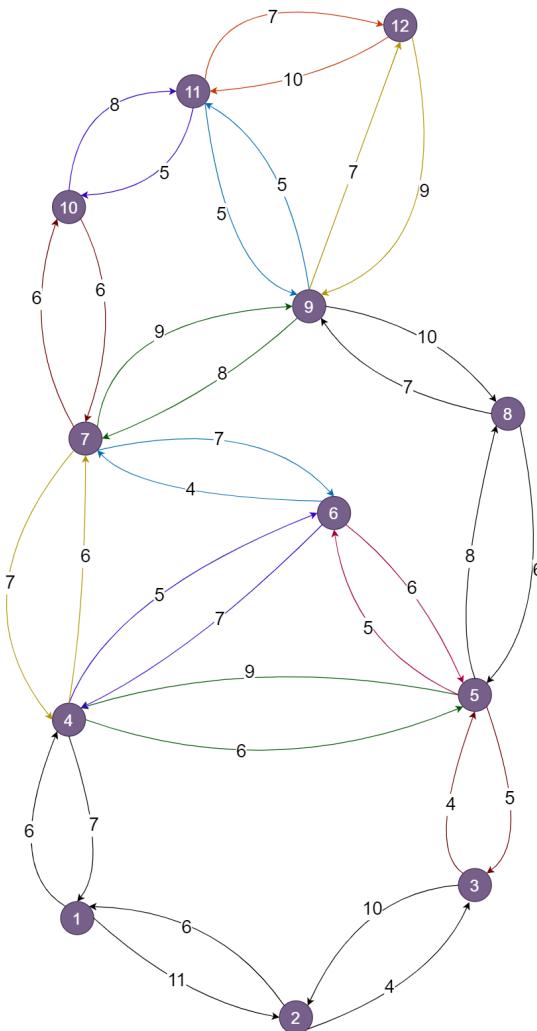
گراف نهایی و جزئیات نقاط انتخاب شده

با توجه به نقاط انتخاب شده و نوع مدلسازی مسئله در شکل ۴ جزئیات مربوط به نقاط انتخاب شده

و در شکل ۵ گراف نهایی که قصد داریم مسئله را روی آن حل کنیم، مشاهده می‌کنید.

Place_index	Place_name	Latitude, Longitude	Neighbors_index	Neighbors_weight
1	Meydan-e Enghelab-e Eslami Metro Station	35.70150614366656, 51.38992544653289	2,4	11,6
2	Lamiz Coffee	35.70201974849577, 51.40538126827055	1,3	6,4
3	Red Crescent Central Pharmacy	35.70687351286316, 51.41800665933752	2,5	10,4
4	Carpet Museum of Iran	35.71513413058044, 51.390711488047685	1,5,6,7	7,6,5,6
5	Yas Hospital	35.71865845546411, 51.414531047228	3,4,6,8	5,9,5,8
6	Salmas Park	35.72378818249789, 51.40231006856167	4,5,7	7,6,4
7	Faculty of Physical Education and Sport Sciences - University of Tehran	35.72940410910814, 51.390663315335956	4,6,9,10	7,7,9,6
8	Myket	35.734768670109375, 51.41482001057617	5,9	6,7
9	Yousef Abad Police Station	35.73747008332278, 51.40624573362557	7,8,11,12	8,10,5,7
10	Atomic Energy High School	35.73812161291542, 51.39324761920264	7,11	6,8
11	Tehran International Tower	35.74333003100125, 51.39851346165568	9,10,12	5,5,7
12	Dey General Hospital	35.748247519011464, 51.411520126817464	9,11	9,10

لیست دقیق نقاط انتخاب شده به همراه اطلاعات دقیق آن



گراف نهایی مدلسازی مسئله

حل مسئله

نحوه حل این مسئله شباهت زیادی به نمونههایی که در کلاس حل شده و در تکالیف و امتحان دیدیم دارد. برای حل آن از dynamic programming و اصل بهینگی بلمن استفاده میکنیم. پیش از پرداختن به حل مسئله ابتدا نگاهی به اصلی بهینگی بلمن و کلیت راه حل مسئله داریم.

اصلی بهینگی بلمن و برنامه‌ریزی پویا^۱

روش برنامه ریزی پویا انعطاف‌پذیری بالایی دارد و می‌توان از آن برای حل مسائل متنوعی استفاده کرد. همان‌طور که قبلًاً دیده‌ایم از این مسائل می‌توان به مسیریابی، صرف کمترین هزینه در جابه‌جایی بین دو شهر، برنامه‌ریزی موجودی، کنترل بهینه ... اشاره کرد. هدف اصلی در برنامه ریزی پویا محاسبه یک تابع هزینه در هر حالت است. وقتی نقشه این تابع هزینه به دست آید، قانون کنترل بهینه را می‌توان از یک حالت به حالت دیگر که هزینه را کمینه می‌کند به دست آورد. مسئله عمومی برنامه‌ریزی پویا را می‌توان به صورت زیر نمایش داد:

$$J(x[k]) = \min_{u[k]} [L(x[k], u[k], x[k + 1]) + J(x[k + 1])]$$

در بسیاری از موارد، عبارت بالا به صورت زیر اصلاح می‌شود:

$$J(x[k]) = \min_{u[k]} [L(x[k], u[k], x[k + 1]) + \gamma J(x[k + 1])]$$

عامل γ مقادیر آینده هزینه رو به جلو را کاهش می‌دهد. اگر $\gamma = 0$ ، آنگاه در الگوریتم، از مقدار هزینه رو به جلو کاملاً چشم‌پوشی می‌شود، در حالی که اگر γ بزرگ باشد، مقادیر آینده مورد توجه بیشتری قرار می‌گیرند. وجود γ در آن دسته از روش‌های عددی نیز مفید است که در آن‌ها مقدار

^۱ منبع: مجله فرادرس

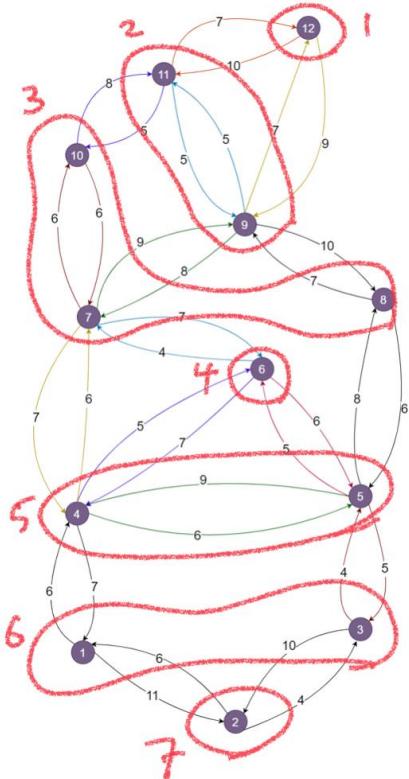
هزینه رو به جلو را برای شروع نمی‌دانیم و آن را به صورت بازگشتی تخمین می‌زنیم. ضریب γ اثرات مفروضات اشتباه اولیه را کاهش می‌دهد.

بنابراین، با استفاده از برنامه ریزی پویا حل مسئله کنترل بهینه به یافتن تابع هزینه رو به جلو تقلیل می‌یابد. این کار را می‌توان با حل رو به عقب از حالت مطلوب تا به همه مقادیر اولیه ممکن انجام داد. وقتی این تابع هزینه رو به جلو را یافتیم، قانون کنترل بهینه به یافتن مسیری می‌انجامد که تابع هزینه را از وضعیت فعلی به یک وضعیت آینده کمینه می‌کند. روش برنامه ریزی پویا یک الگوریتم بسیار عمومی است و می‌توان آن را به مسائل بهینه‌سازی اعمال کرد که در آن‌ها هزینه رو به جلو و هزینه جابه‌جایی از یک حالت به حالت دیگر، جمع شونده است. قبل از پرداختن به جزئیات الگوریتم، یک مثال ساده را بیان می‌کنیم.

اصل بهینگی بلمن پایه و اساس پیاده‌سازی این رویکرد در حل مسائل بهینه‌سازی است و بیان می‌کند که اگر یک راه حل بهینه برای یک مسئله پیدا کرده باشیم، هر زیرمجموعه‌ای از راه حل یافت شده، برای زیر مسئله متناظر خودش نیز جواب بهینه است.

حل دستی مسئله

رژن نمایه که بخواهیم از 2 تا برداشتم:



$$① F(12) = 0$$

$$② F(11) = 7, f(9) = 7$$

$$③ F(10) = F(11) + 8 = 15$$

$$F(7) = \min \left\{ F(9) + 9, F(10) + 6 \right\} = 16$$

$$F(8) = 7 + F(9) = 14$$

$$④ F(6) = 4 + f(7) = 20$$

$$⑤ F(4) = \min \left\{ 5 + F(6), 6 + F(7) \right\} = 22$$

$$F(5) = \min \left\{ 8 + F(8), 5 + F(6) \right\} = 22$$

$$⑥ F(1) = 6 + F(4) = 28, f(3) = 4 + F(5) = 26$$

$$⑦ F(2) = \min \left\{ 4 + F(3), 6 + F(1) \right\} = 30$$

Solution: $2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9 \rightarrow 12$

حل مسئله در حالت کلی (کدها)

- کتابخانه‌های استفاده شده:

برای خواندن فایل csv و کار کردن با آن از کتابخانه pandas استفاده کردیم. همچنین برای حل کردن معادله از کتابخانه PuLP استفاده کردیم.

- نوع تعریف و نگهداری رئوس گراف:

یک کلاس به اسم Node داریم که از آن برای تعریف کردن رئوس گراف استفاده می‌کنیم. این کلاس ویژگی‌های یک راس که اسم، مختصات، همسایه‌ها و وزن یال‌هایی که به همسایه‌هایش دارد را نگه می‌دارد.

```
In [59]: import csv
import pulp
import pandas as pd
from pulp import *

NODESNR = 12

class Node:
    def __init__(self, index, name, latitude, longitude, neighbors, arc_weights):
        self.index = index
        self.name = name
        self.latitude = latitude
        self.longitude = longitude
        self.neighbors = neighbors
        self.arc_weights = arc_weights
```

- نحوه تعریف گراف و کار با آن:

همهی عملیات مربوط به محاسبات در کلاس Calculations انجام می‌شود. در این تابع ابتدا فایل Dataset را می‌خوانیم و با استفاده از آن رئوس گراف را می‌سازیم. برای نگهداری یال‌های گراف یک آرایه‌ی دو بعدی به اسم arcs در نظر می‌گیریم. مقداری که در خانه (j,i) این آرایه قرار دارد، وزن یالی که از i به j می‌رود را ذخیره می‌کند. از آنجایی که قصد داریم کوتاهترین مسیر را پیدا کنیم، اگر یالی بین این دو راس نباشد، مقدار را ۱۰۰۰ قرار می‌دهیم.

```
class Calculations:
    def __init__(self):
        self.source = -1
        self.destination = -1
        self.nodes = []
        self.routes = []
        csv_file = pd.read_csv(r'Dataset.csv')

    # creating nodes in the corresponding graph
    for index, row in csv_file.iterrows():
        node = Node(int(row['Place_index']),
                    row['Place_name'],
                    float(row['Latitude']),
                    float(row['Longitude']),
                    list(map(int, row['Neighbors_index'].split(','))),
                    list(map(float, row['Neighbors_weight'].split(','))))
        )
        self.nodes.append(node)

    # creating a 2D list to store arcs and their corresponding weights. This list is not zero based.
    # arcs[i][j] returns the weight of the arc between nodes with index i and j. it returns 1000 if there's no edge b/w them
    self.arcs = [[1000 for _ in range(NODESNR+1)] for _ in range(NODESNR+1)]
    for node in self.nodes:
        for i in range(len(node.neighbors)):
            self.arcs[node.index][node.neighbors[i]] = node.arc_weights[i]
```

- مقداردهی اولیه مسئله و شروع حل:

یک مسئله بهینه‌سازی خطی تعریف می‌کنیم که هدف آن پیدا کردن مقدار \min است. سپس با استفاده از مقادیر مربوط به یالهای گراف متغیرهای این معادله را همان طور که دیده می‌شود مقداردهی می‌کنیم. در نهایت هدف مسئله را کمینه کردن `current_cost` مشخص می‌کنیم.

```
self.lpProblem = LpProblem("FindShortestRoute", LpMinimize)
current_cost = None

# define a list to store variables
self.lp_vars_list = [[0 for _ in range(NODESNUMBER)] for _ in range(NODESNUMBER)]

for i in range(NODESNUMBER):
    for j in range(NODESNUMBER):
        self.lp_vars_list[i][j] = LpVariable("x"+'_{0:02d}'.format(i)+'_{0:02d}'.format(j), None, None, cat="")

# updating variables with arcs weights
for i in range(NODESNUMBER):
    for j in range(NODESNUMBER):
        current_cost += self.lp_vars_list[i][j] * arcs[i][j]

# updating Problem with new variables list
self.lpProblem += current_cost, "minimizing current_cost"
```

- مشخص کردن constraint‌های مربوط به نقاط شروع و پایان:

وظیفه اصلی تابع `initialize_constraints` این است که با توجه به نقاط شروع و پایان که کاربر مشخص کرده است، معادلات محدودکننده (قیود) را مشخص کند. هر بار متغیری به اسم `left_hand_side` مشخص می‌کنیم و با توجه به شرایط مسئله (که متغیرها و وزن یالها آن را مشخص می‌کنند) قیود مسئله را بهروزرسانی می‌کنیم.

توجه: با توجه به آشنایی کم با PuLP من برای پیاده‌سازی این قسمت از تی‌ای درس و یک نفر دیگر کمک گرفتم.

```

# a function to set start and finish nodes as well as initial constraints
def initialize_constraints(self, src, dst):
    self.source = src
    self.destination = dst
    for i in range(NODESNUMBER):
        left_hand_side = None
        if i != self.source and i != self.destination:
            for j in range(NODESNUMBER):
                left_hand_side += self.lp_vars_list[i][j] - self.lp_vars_list[j][i]

        self.lpProblem += left_hand_side == 0, "constraint " + '_{0:02d}'.format(i)

    left_hand_side = None
    for j in range(NODESNUMBER):
        left_hand_side += self.lp_vars_list[self.source][j]

    self.lpProblem += left_hand_side == 1, "constraint " + '_{0:02d}'.format(self.source) + "out"

    # here we initialize constraints for source input
    left_hand_side = None
    for j in range(NODESNUMBER):
        left_hand_side += self.lp_vars_list[j][self.source]

    self.lpProblem += left_hand_side == 0, "constraint " + '_{0:02d}'.format(self.source) + "in"

    left_hand_side = None
    for j in range(NODESNUMBER):
        left_hand_side += self.lp_vars_list[j][self.destination]

    self.lpProblem += left_hand_side == 1, "constraint " + '_{0:02d}'.format(self.destination) + "in"

```

- محاسبه نهایی و نمایش جواب:

در نهایت وظیفه انجام محاسبات و پیمایش گراف به صورت سطح به سطح (همان‌طور که در شکل مربوط به حل دستی نشان دادیم) بر عهده تابع `retrieve_solution` است. این تابع در ابتدا از متغیرهایی که در مسئله تعریف کرده‌ایم، شروع و پایان را به دست می‌آورد. در یک حلقه `while` هر بار یکی از رؤوس را در یکی از سطوح‌های گراف بررسی می‌کند. تا زمانی که به مقصد نرسیده‌ایم، رؤوس یکی یکی به لیست `routes` اضافه می‌شوند.

```

def retrieve_solution(self):
    current_route = {}

    for variable in self.lpProblem.variables():
        if variable.varValue:
            destination_name = int(variable.name[5:7])
            source_name = int(variable.name[2:4])
            current_route[source_name] = destination_name

    current_position = self.source
    while current_position != self.destination:
        self.routes.append(current_position)
        current_position = current_route[current_position]

    self.routes.append(self.destination)
    print(self.routes)

    print("Shortest path is : ", value(self.lpProblem.objective))

```

- بررسی خروجی کد:

همان‌طور که دیده می‌شود، مشابه حل دستی که برای مسئله داشتیم، کد در حالتی که از ۲ بخواهیم به ۱۲ برویم، جواب درست را محاسبه کرده و نمایش می‌دهد.

```
Enter start point index
2
Enter destination point index
12
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

Result - Optimal solution found

Objective value:          30.00000000
Enumerated nodes:         0
Total iterations:          0
Time (CPU seconds):       0.00
Time (Wallclock seconds): 0.00

Option for printingOptions changed from normal to all
Total time (CPU seconds): 0.00   (Wallclock seconds): 0.00

Origin point is Lamiz Coffee. With 2 as index.
Summarized coordinates of this point are: ## 35.70201975, 51.40538127 ##

Destination point is Dey General Hospital. With 12 as index.
Summarized coordinates of this point are: ## 35.74824752, 51.41152013 ##

This is the shortest path: [2, 3, 5, 8, 9, 12]
Shortest path cost is : 30.0
```