به نام خدا

پروژه دوم بهینه سازی ترکیبی ابولفضل عرب انصاری، محمدرضا باطنی، علی ثابت

این پروژه توسعه یافته شده تمرین سوم است. قید ها و پارامتر ها و مجموعه ها و توابع هدف از روی آن پیاده سازی شدند.

تفاوتی که در اینجا هست این است که ممکن است بازه ها ۱ ساعت و نیمه باشند و ما برای حل کردن این مشکل پارامتر p(i, h) را تعریف کرده ایم. این پارامتر زیر مجموعه های p(i) را نشان میدهد که در آن زمان هایی که با هم اشتراک دارند میایند. یعنی مثلا اگر بازه های زمانی ۷:۴۵ تا ۹:۱۵ و ۸ تا ۱۰ و ۱۰:۴۵ تا ۱۲:۱۵ باشند، بازه های ۷:۴۵ تا ۹:۱۵ و ۸ تا ۱۰ در زیر مجموعه اول میفتند و بازه های ۹:۱۵ تا ۱۰:۴۵ و ۸ تا ۱۰ هم در زیرمجموعه دوم میفتند. زیر مجموعه هم به صورت پارامتر بدین ترتیب تعریف شده که اگر h در زیرمجموعه i ام وجود داشته باشدp(i,h) برابر ۱ و در غیر این صورت ۰ میشود.

همچنین ساخته شدن p(i,h) به صورت اتوماتیک و توسط پایتون انجام میشود (در ادامه توضیح داده میشود) یعنی شما در لیست h_times بازه های زمانی دلخواهتان را به صورت لیستی از زوج مرتب ها تعریف میکنید و ساعت ها را به عدد تبدیل میکنید. یعنی مثلا اگر بازه های ۸ تا ۱۰ و ۹:۱۵ تا ۱۰:۴۵ مدنظر باشند لیست به صورت زیر تعریف میشود:

h_times = [(8,10), (9.25,10.75)]

همچنین تعدادی درس هم بودند که نباید تداخل داشته باشند و آنها با s مشخص میشدند. برای راحتی کاربر، آن هم با استفاده از s_python ورودی گرفته میشود و s(j,c) اتوماتیک ساخته میشود. ساختار s_python هم به صورت زیر است:

s_python = [(‘1’,’3’,’7’),(‘1’,’2’),(‘6’,’7’,’8’)]

و قید ها به صورتی تغییر میکنند که روی تمام اعضای یک زیر مجموعه i جمع زده شود.

همچنین برای ورودی دادن a و b از lecturer_courses و lecturer_times استفاده شده. بدین صورت که هر بار اضافه شدن عنصر به lecturer_courses معادل دادن اطلاعات یک استاد هست و i امین tuple ای که append میشود، برابر مجموعه درس هایی ست که استاد i ام درس میدهد و از روی آن a(c,l) ساخته میشود.

هر بار اضافه شدن عنصر به lecturer_times هم معادل دادن زمان های حضور یک استاد هست و i امین tuple ای که append میشود، برابر مجموعه ای از string هاست که بیانگر زمان های حضور روز های مختلف هفته است و اگر k امین عنصر استرینگ j ام یک باشد یعنی استاد i ام در روز j ام و ساعت k ام اعلام آمادگی کرده است. قابل توجه است که ساعت kام طبق ساعت های اعلام شده در h_times است. همچنین از روی این lecturer_times ، b(l,d,h) ساخته میشود.

بقیه پارامتر های مسئله هم سرراست داده شده اند.

برای مثال یک نوع دیتای ورودی مسئله مانند زیر است:

```
c_python = [str(c) for c in range(1,19)]
l_python = [str(l) for l in range(1,7)]
d_python = [str(d) for d in range(1,6)]
h_time = [(8,10),(10,12),(13,15),(15,17),(7.75,9.25),(9.25,10.75),(10.75,12.25),(13.5,15),(15,16.5)]

s_tuples =
[("1","2","8","9","18"),("2","3","10","11","18"),("3","4","5","12","13","14"),("7","8","16","17"),("5","12","13
","14","15","17")]
nc = [1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3] #len(nc) = c
hh_python = '3'
dd_python = '5'
k_python = 2

lecturer_courses = [] #a(c,l)
lecturer_courses.append(("1","2","18"))#lecturer 1
lecturer_courses.append(("3","8","9"))#lecturer 2
lecturer_courses.append(("4","10","11"))
lecturer_courses.append(("5","6","12"))
lecturer_courses.append(("7","13","14"))
lecturer_courses.append(("15","16","17"))

lecturer_times = [] #b(l,d,h)
lecturer_times.append(("111100000","111100000","111100000","111100000","111100000"))
lecturer_times.append(("111100000","000000000","111100000","000000000","111100000"))
lecturer_times.append(("111000000","111000000","111000000","111000000","111000000"))
lecturer_times.append(("110000000","110000000","110000000","110000000","110000000"))
lecturer_times.append(("000000000","111100000","000000000","110000000","000000000"))
lecturer_times.append(("110000000","110000000","110000000","110000000","000000000"))
```

(این دیتا همان دیتای بیان شده در صورت سوال (موجود در کتاب) است.)

کمی درباره نحوه تولید p(i,h) از روی h_times توضیح میدهیم.

بدین ترتیب که هر زیرمجموعه i را متناظر با یک بازه کوچک زمانی میگیریم. یعنی اگر زمان ها ۸–۱۰ و ۹:۱۵–۱۰:۴۵ و ۱۰–۱۲ باشد بازه ها را متناظر با ۸–۹:۱۵ و ۹:۱۵–۱۰ و ۱۰–۱۰:۴۵ و ۱۰:۴۵–۱۲ میگیریم.

برای اینکار تمام نقاط انتهایی بازه های زمانی را در یک لیست میریزیم، تکراری های آنها را جدا میکنیم و به ترتیب مرتب میکنیم.

برای فهمیدن اینکه یک بازه در بازه دیگر میفتد یا خیر هم از تابع does_share استفاده میکنیم.

برای ساختن s_python هم چک میکنیم که آیا درس c در s_tuples[j] میفتد یا خیر، اگر افتاد s(j,c) را ۱ میگذاریم و در غیر این صورت ۰ میگذاریم.

برای ران شدن همه ی موارد باید دو کد پایتون app.py, time_changer.py و سه فایل گمز در یک فولدر و کنار هم قرار گرفته باشند.

time_changer.py

از این اسکریپت برای برگشت زمان سیستم عامل ویندوز به زمان لایسنس گمز استفاده کردیم.
به طوری که در ابتدا زمان را به عقب می برد، مدل گمز ران می شود و سپس زمان را برمیگردانیم.
این کار سه بار برای ران کردن مدل های z1, z2, z3 انجام شد.

برای استفاده از این اسکریپت باید ابتدا ماژول pywin32 را با پایتون ۳.۶ نصب کنید. برای این کار از دستور زیر می توان استفاده کرد:

```
py -3.6 pip install pywin32
```

دقت شود که لحظه ی برگشت زمان سیستم باید کد توسط دسترسی administrator ران شده باشد.

Run models Z1

حال برای قراردادن داده ها در دیتابیس گمز به جای استفاده از دستور export با همان روش های توصیه شده کد ها را مستقل از آدرس کردیم.
سپس تایم را به عقب برگرداندیم و مدل z1.gms را ران می کنیم.
مقدار خروجی z1 را پرینت میکنیم و سپس چک می شود که اگر model state ای که از گمز دریافت کردیم برابر یک بود پیغام optimal solution found چاپ شود.
در مدل هم قیدی که تعداد کلاس های تایم بعد از نهار کمینه شود تحت عنوان قید constz1 به مدل وارد شد ولی با مقدار نامنفی w آن را نرم کردیم.
برای روند لود داده ها هم در این فایل گمز و در دوفایل بعدی همه ی داده ها توسط پایتون در دیتابیس قرار خواهند گرفت.

Run model Z2

در ابتدا مقدار z1 که از حل مدل فایل z1.gms دریافت کردیم را درون دیتا بیس قرار میدهیم.
سپس دوباره ران شدن را مستقل از آدرس کردیم و دوباره روند عقب بردن تایم، ران شدن و آپدیت تایم را تکرار می کنیم.
فایل z2.gms موجود را ران می کنیم.
در فایل z2.gams این بار علاوه بر داده های قبل مقدار z1 را هم لود میکنیم و آن قید نرم که تعداد کلاس هایی که بیش تر از یک جلسه دارند و جلساتشان در دو روز متوالی قرار میگیرد را کمینه خواهیم کرد. این قید تحت عنوان constz2 به مدل تحمیل شد.
سپس طبق روال قبل این مقدار z2 به پایتون داده خواهد شد.

Run model Z3

در این قسمت z2 دریافتی از حل مدل قبل در دیتابیس ریخته می شود سپس دوباره روند ران کردن را تکرار میکنیم.
فایل z3.gms ران می شود.
قید نرم این مدل به متوازی بودن و متناظر بودن ساعت های یک درس اشاره دارد.
در نهایت مقدار z3 حاصل از حل مدل چاپ خواهد شد.

برای مثال کد را برای یک دیتا ران میکنیم و مشاهده میکنیم که مقدار هر سه تابع هدف ۰ است یعنی هر سه شرط برقرار شده:

```
day_of_week = {'1': 'Saturday', '2': 'Sunday', '3': 'Monday', '4': 'Tuesday', '5': 'Thursday'}
time_of_day = {'1': '8-10', '2': '10-12', '3': '13-15', '4': '15-17', '5': '7:45-9:15', '6': '9:15-10:45',
               '7': '10:45-12:15', '8': '13:30-15', '9': '15-16:30'}

# Dataset 6
lecturer_courses = []
lecturer_times = []

# number of courses
c_python = [str(c) for c in range(1,28)]

# number of instructors
l_python = [str(l) for l in range(1,11)]

# number of days
d_python = [str(d) for d in range(1,6)]

# times
h_time = [(8,10),(10,12),(13,15),(15,17),(7.75,9.25),(9.25,10.75),(10.75,12.25),(13.5,15),(15,16.5)]

# course sets that can not conflict
s_tuples=[('3','9','10','12'),('2','11','13'),('2','4','6','20','24'),('1','2','3','5','10','17','20'),
          ('2','7','8','10','15','20','24'),('6','9','14','15','17','18'),('20','22','24'),('19','21','23')]

# number of times that course holds in a week (courses at position x holds y times)
nc = [2,2,1,2,2,2,2,1,2,2,1,1,2,1,2,2,1,1,1,2,2,1,1]

# courses that instructor can present
lecturer_courses.append(("1","2","3"))
lecturer_courses.append(("4","5","6"))
lecturer_courses.append(("7","8","9"))
lecturer_courses.append(("10","11","12"))
lecturer_courses.append(("13","14","15"))
lecturer_courses.append(("16","17","18"))
lecturer_courses.append(("19","20","21"))
lecturer_courses.append(("22","23","24"))
lecturer_courses.append(("25","26","27"))
lecturer_courses.append(("28","29","30"))

# boolean that shows which times instructor can present courses
lecturer_times.append(("011110000","000111101","011110000","000111101","110000111"))
lecturer_times.append(("000001010","000001110","000001010","000001110","001100010"))
lecturer_times.append(("000100000","100010000","000100000","100010000","010010100"))
lecturer_times.append(("010100000","111000110","010100000","111000110","110011010"))
lecturer_times.append(("010011101","001110110","010011101","001110110","101001110"))
lecturer_times.append(("110001111","001101100","110001111","001101100","101101010"))
lecturer_times.append(("000011110","111000011","000011110","111000011","001111001"))
lecturer_times.append(("001010011","011000101","001010011","011000101","000101011"))
lecturer_times.append(("000001101","001111110","000001101","001111110","001001110"))
lecturer_times.append(("101011000","001101110","101011000","001101110","001001010"))

# first time of afternoon
hh_python = '3'
# last day of teaching
dd_python = '5'
# number of classes
k_python = 50
```

```
final result1:
0.0
model state is:  1.0
Optimal Solution Found!!!
final result z2:
0.0
model state is:  1.0
Optimal Solution Found!!!
final result z3:
0.0
model state is:  1.0
optimal solution found!!!
Saturday :      8-10(16)        10-12(12, 15)   15-17(1, 7, 11) 7:45-9:15(14, 20, 23)
:15-10:45(19, 25)       10:45-12:15(27) 13:30-15(5, 18, 24)     15-16:30(22, 26)
Sunday :        7:45-9:15(9)    10:45-12:15(6, 13)      13:30-15(4, 10) 15-16:30(2)
Monday :        10-12(15)       15-17(1, 7)     13:30-15(24)
Tuesday :       7:45-9:15(8)    9:15-10:45(25)  10:45-12:15(3, 6, 13)   13:30-15(5)
5-16:30(2)
Thursday :      8-10(16)        15-17(17)       7:45-9:15(9, 20)        9:15-10:45(19)
3:30-15(4, 10)  15-16:30(21)
PS C:\GAMS\win64\25.1\apifiles\Python\api_36>
```

```
Saturday :      8-10(16)        10-12(12, 15)   15-17(1, 7, 11) 7:45-9:15(14, 20, 23)       9:15-10:45(19, 25)      10:45-12:15(27) 13:30-15(5, 18, 24)     15-16:30(22, 26)
Sunday :        7:45-9:15(9)    10:45-12:15(6, 13)      13:30-15(4, 10) 15-16:30(2)
Monday :        10-12(15)       15-17(1, 7)     13:30-15(24)
Tuesday :       7:45-9:15(8)    9:15-10:45(25)  10:45-12:15(3, 6, 13)   13:30-15(5)         15-16:30(2)
Thursday :      8-10(16)        15-17(17)       7:45-9:15(9, 20)        9:15-10:45(19)      13:30-15(4, 10)  15-16:30(21)
```