به نام خدا

دانشگاه تهران

دانشکده مهندسی

برق و کامپیوتر

درس تحقیق در عملیات

پروژه پایانی

نام و نام خانوادگی	فاطمه نائینیان
شماره دانشجویی	810198479

فهرست

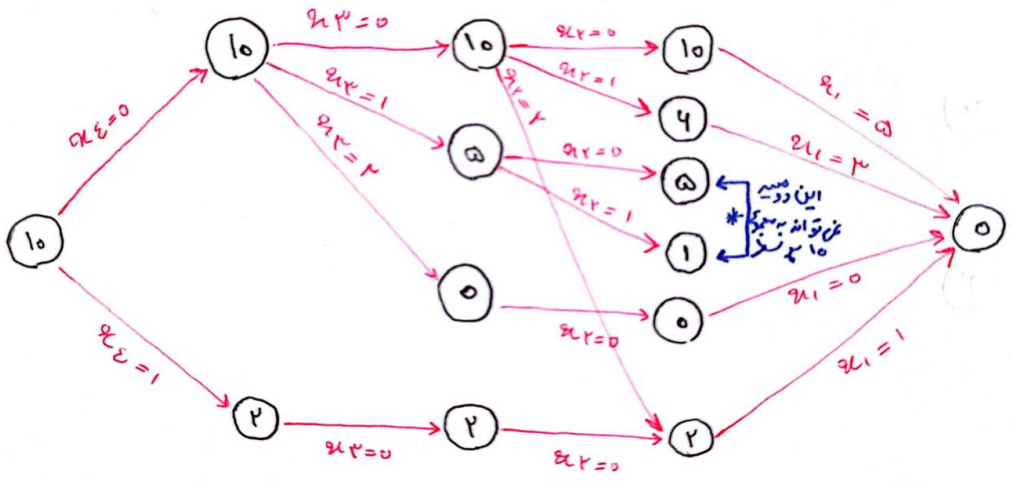
پاسخ ۱

$$Find \sum_{\forall(x_1,x_2,x_3,x_4)} \frac{(x_1+x_2+x_3+x_4)!}{x_1!\,x_2!\,x_3!\,x_4!}$$

$$for\ \forall(x_1,x_2,x_3,x_4)\ st.\ 2x_1+4x_2+5x_3+8x_4=10\ ,\forall(x_1,x_2,x_3,x_4)\geq 0\ \&\ Integers$$

همانند روشهایی که در درس یادگرفتیم ابتدا باید یک مدل مناسب برای مسئله پیدا کنیم. پنج متغیر داریم که در هر مرحله باید یکی از انها را انتخاب کنیم و حالت هایی که میتواند تولید کند را پیدا کنیم. به ازای متغیر چهارم، به دلیل وجود ضریب 8 دو حالت خواهیم داشت. به ازای متغیر سوم، به دلیل ضریب 5 سه حالت داریم اما همانطور که در شکل زیر پیداست اگر x3=1 شود به حالت اعداد فرد میرسیم که چون بقیه متغیر ها زوج هستند نمیتوان تساوی را برقرار کرد پس به نظر می اید که متغیر سوم فقط دو حالت 0 و 2 دارد. سایر حالت های باقی مانده را نیز برای متغیر اول و دوم به دست می اوریم. (برای راحتی کار و جلوگیری از تولید مقدار زیادی حالت و پیچیدگی، از متغیری که ضریب ان بزرگتر بود شروع کردیم.)

در نهایت مدل زیر به دست می اید.



حال به ازای مسیر های به دست امده باید مقدار تابع هدف را پیدا کنیم.

$$\sum \frac{(x_1+x_2+x_3+x_4)!}{x_1!\,x_2!\,x_3!\,x_4!}$$
$$= \frac{(1+0+0+1)!}{1!\,0!\,0!\,1!}+\frac{(1+2+0+0)!}{1!\,2!\,0!\,0!}+\frac{(0+0+2+0)!}{0!\,0!\,2!\,0!}$$
$$+\frac{(3+1+0+0)!}{3!\,1!\,0!\,0!}+\frac{(5+0+0+0)!}{5!\,0!\,0!\,0!}=2+3+1+4+1=11$$

بنابراین مقدار تابع هدف باید برابر 11 شود.

حال با کمک پایتون پیاده سازی میکنیم.

برای اینکار ابتدا یک تابع تعریف میکنیم که همه حالت های ممکن را پیدا میکند. نحوه کار این تابع به گونه ای است که به ازای هر متغیر اگر مقدار cost باقی مانده که در ابتدا برابر 10 است، کمتر از ضریب ان بود، تغییری نمیکند. اما زمانی که به اولین متغیری برسد که cost از ضریب ان بیشتر است، به اندازه ضریب از ان کم میکند و به متغیر یک واحد اضافه میکند و ادامه می دهد. این کار را ان قدر ادامه میدهد تا همه حالت ها پیدا شود. دلیل اینکه این نحوه پیاده سازی داینامیک است همین موضوع ارتباط بین متغیر هاست. به گونه ای که اولین متغیر انتخاب شده بر روی تعداد متغیر های بعدی تاثیر دارد.

```
possible_ans = []
def states(x1, x2, x3, x4, const):
  # if the cost gets 0 and if its not repeated, we add it to possible answers
  # as we explained in report, states where cost is an odd number cannot be a solution so they will be illuminated
  if(const == 0 and [x1, x2, x3, x4] not in possible_ans):
    possible_ans.append([x1, x2, x3, x4])

  if(const >= 8):  # find a new state where x4+1 and cost-8
    states(x1, x2, x3, x4+1, const-8)
  if(const >= 5):  # find a new state where x3+1 and cost-5
    states(x1, x2, x3+1, x4, const-5)
  if(const >= 4):  # find a new state where x2+1 and cost-4
    states(x1, x2+1, x3, x4, const-4)
  if(const >= 2):  # find a new state where x1+1 and cost-2
    states(x1+1, x2, x3, x4, const-2)
```

دلیل اینکه به ترتیب از متغیر چهارم به اول نوشته ام به این دلیل است که مشابه شکل رسم شده عمل کند و در ابتدا متغیر چهارم را مقدار دهی کند.

برای جلوگیری از مشکلات حافظه از یک متغیر global برای ذخیره سازی همه جواب ها استفاده شده است که در حال کلی استفاده از متغیر global توصیه نمی شود.

در ادامه یک تابع دیگر برای محاسبه مخرج تابع هدف به شکل زیر تعریف می شود و در نهایت مقدار تابع هدف را به دست می اوریم. میبینیم با مقدار دستی هم خوانی دارد.

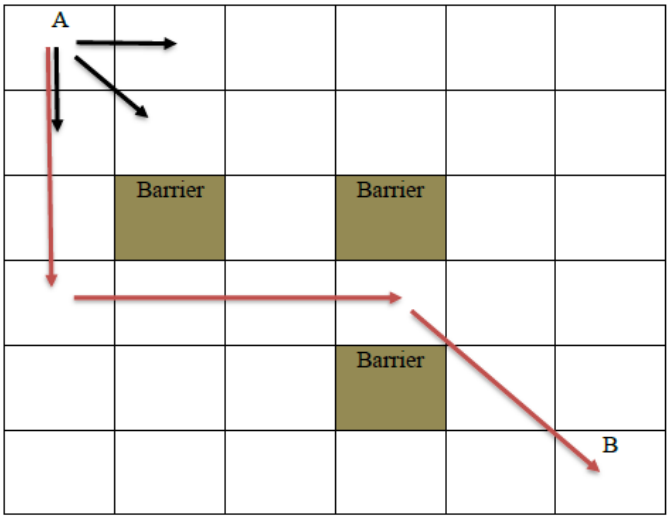
```
def factor(ans):  # this function helps us to calculate factorial for each element of a solution
  fact = 1
  for x in ans:
    fact = fact * math.factorial(x)
  return fact
```

```
states(0, 0, 0, 0, 10) # at first all possible solutions are found
sigma = 0
for ans in possible_ans:
  sigma += math.factorial(sum(ans)) / factor(ans)  # in this loop we find the goal

sigma
```

```
11.0
```

پاسخ ۲



ایده ای که برای حل این سوال میتوانیم بزنیم این است که برای هر خانه از جدول یک cost تعریف کنیم که این cost برابر تعداد مسیر های ممکن از ان خانه باشد. با توجه به شکل بالا و متن سوال نحوه حرکت فقط به سمت پایین، اریب و راست است بنابراین در هر خانه، به تعداد روشهایی که بتوان به سه همسایه ان خانه برسد، میتواند از ان خانه به مقصد برسد. به عبارتی:

همسایه 1	خانه مدنظر
همسایه 2	همسایه 3

دلیل انتخاب این سه همسایه به علت نحوه حرکت است چون فقط فقط پایین، اریب و راست میتواند حرکت کند پس از سایر همسایه ها نمیتوانیم به این خانه برسیم. بنابراین از نقطه پایان مشخص شده B شروع میکنیم و عدد 1 را به ان نسبت میدهیم. حال همه مسیر های ممکن را برای همه خانه ها به دست می اوریم. روش کار اینگونه است که برای هر خانه تعداد مسیر های سه همسایه را به شکل بالا جمع میزنیم. خانه های مانع نیز تعداد مسیر 0 خواند داشت. در نهایت جدول زیر به دست می اید.

318	172	86	36	11	1
94	52	34	16	9	1
42	0	18	0	7	1
26	16	10	8	5	1
6	4	2	0	3	1
1	1	1	1	1	1

با توجه به این جدول در نهایت برای رسیدن از A به B تعداد 318 مسیر وجود دارد.

برای پیاده سازی این مدل کافیست تا یک جدول متجسم شویم که x ان از 1 تا 6 و y ان نیز از 1 تا 6 تغییر میکند. سپس نقطه نهایی یعنی B را مختصات (1,1) و A را مختصات (6,6) فرض میکنیم. سپس مختصات خانه های مانع را نیز به عنوان یک شرط به تابع میدهیم که راه های عبوری ان را برابر صفر میکند. یک مختصات current تعریف میکنیم که در ابتدا ان را با مختصات A تعریف میکنیم . حال همانند دنباله فیبوناتچی، برای اینکه بتوانیم همه مسیر های عبوری از A با مختصات (6,6) را داشته باشیم به مسیر های عبوری از مختصات (5,5) , (5,6) , (6,5) نیاز داریم بنابراین در حالت کلی برای به دست اوردن مسیر های (xC,yC) به (xC-1,yC) و (xC,yC-1) و (xC-1,yC-1) نیاز داریم. این وابستگی بین متغیر ها که سبب می شود همه به B وابسته باشند و هیچ کدام به A وابسته نباشند و مسیر های میانی از هم تاثیر بپذیرند همان داینامیک بودن مسئله است.

```
# C is the current position which at first will be initialiezed with A position, and B is the goal position
def possible_route(xC=6, yC=6, xB=1, yB=1):
  if xC==xB or yC==yB : # goal state and all possible states to it with x=1 or y=1 has only one route
    return 1
  elif (xC==3 and yC==2) or (xC==3 and yC==4) or (xC==5 and yC==4) : # Barrier
    return 0
  else:
    return possible_route(xC-1, yC) + possible_route(xC, yC-1) + possible_route(xC-1, yC-1) # each state depend on 3 other state
```

```
possible_route()
```

```
318
```

میبینیم همانند حل دستی مقدار 318 را دریافت کردیم.

پاسخ ۳

هنگامی که در گوگل عبارتی اشتباه مینویسیم و گوگل عبارت درست را به ما پیشنهاد میدهد در واقع گوگل بین همه عبارت ها توانسته بهترین عبارتی را پیدا کند که کمترین cost ممکن را داشته است. این cost میتواند بر اساس عوامل مختلفی سنجیده شود. برای نمونه میتواند بین همه کلمات هم سایز، کلمه ای را انتخاب کند که بیشترین حروف مشترک و یا بیشترین بخش های یکسان را دارد را پیدا کند. سپس بین همه کلمات پیدا شده کلمه ای را پیشنهاد میدهد که نیازمند کمترین تغییرات برای رسیدن به عبارت درست باشد. بدین ترتیب گوگل عبارت درست را پیدا میکند و پیشنهاد میدهد.

الف) در این بخش میخواهیم با کمک سه عملگر حذف کردن با هزینه 1 و وارد کردن با هزینه 1 و جایگزین کردن با هزینه 2 ، کمترین هزینه ممکن برای رسیدن از رشته S1 به S2 را پیدا کنیم. برای اینکار همانند مثال فیبوناتچی نیازمند اثر بخش های قبل هستیم. این عملیات این گونه است که از انتهای هردو رشته شروع به بررسی میکنیم. اگر دو حرف یکسان باشند، از روی انها عبور میکند و اگر یکسان نباشند، سه حالت را بررسی میکند حالت اول اینکه با جایگزینی به حرف مدنظر برسد و حالت دوم اینکه با حذف به حرف مد نظر برسد ، و حالت سوم اینکه با وارد کردن به حرف مدنظر برسد. در حالت اول 2 واحد به هزینه اضافه می شود و در دو حالت بعدی 1 واحد اضافه می شود . حال بین این سه حالت مینیمم میگیرد و کمترین را پیدا میکند. زمانی الگوریتم متوقف می شود که به انتهای حداقل یک عبارت برسیم. وقتی به انتهای یک عبارت برسیم یعنی در عبارت دیگر n حرف را بررسی نکرده ایم که یا نیازمند وارد کردن است یا نیازمند حذف کردن و این یعنی هزینه هر حرف برابر 1 است و یعنی هزینه n به هزینه کل اضافه می شود. در نهایت کد به شکل زیر می شود که هزینه نهایی به ازای دو عبارت داده شده در سوال در ان نمایش داده شده است. مدل سازی داینامیک این مسئله همانطور که در بالا مسیر ان توضیح داده شد، ارتباط سه حالت است که بین انها مینیمم گرفته می شود و این اثر در مراحل بعدی نیز خودش را نشان میدهد.

```python
def find_min_cost(s1_pos, s2_pos):
  # when one of them reach the first of string, so max(s1_pos,s2_pos)  is the number of letters we need to insert
  if s1_pos == 0 or s2_pos == 0 :
    return max(s1_pos, s2_pos)
  # when the next letter is similar and need no change
  elif s1[s1_pos-1] == s2[s2_pos-1] :
    return find_min_cost(s1_pos-1, s2_pos-1)
  # when is not the end and next letters are not similar, we need to check (i-1,j-i) , (i-1,j) , (i,j-1)
  # in case we find two similar letter and if not, we need to delete, insert or replace
  else:
    return min(find_min_cost(s1_pos-1, s2_pos-1)+2, find_min_cost(s1_pos-1, s2_pos)+1, find_min_cost(s1_pos, s2_pos-1)+1)

find_min_cost(len(s1), len(s2))
```

```
3
```

ب) در این بخش میخواهیم طول بزرگترین زیر سری مشترک دو عبارت S1 و S2 را پیدا کنیم. به این معنی که حساب کنیم چند حرف مشترک بین انها وجود دارد. برای این کار همانند بخش قبل از انتهای دو رشته شروع میکنیم و هرجا دو حرف برابر بودند تعداد را به علاوه 1 میکنیم و باقی رشته ها را بررسی میکنیم. اگر دو حرف برابر نبودند ادامه میدهیم و سپس برای اینکه همه حالت ها را سنجیده باشیم در هر موقعیت هر حرف را به دو حرف کنونی و قبلی میسنجیم و سپس بین تعداد حرف مشترک انها ماکسیمم میگیریم. در نهایت کد ان به شکل زیر می شود. نحوه مدلسازی ان همین ارتباطیست که بین سه حالت وجود دارد و بین سه حالت ماکسیمم میگیریم و این ماکسیمم روی مراحل بعدی اثر میگذارد.

```python
def similar_letters(s1_pos, s2_pos):
  # when one of them reach the end of string and find no similar letter
  if s1_pos == 0 or s2_pos == 0 :
    return 0
  # when the next letter is similar so number of similar letters add by 1
  elif s1[s1_pos-1] == s2[s2_pos-1] :
    return similar_letters(s1_pos-1, s2_pos-1)+1
  # we need to move on and check the next letters, (i-1,j-1), (i-1,j), (j-1,i) , so we will check the whole string
  else:
    return max(similar_letters(s1_pos-1, s2_pos-1), similar_letters(s1_pos-1, s2_pos), similar_letters(s1_pos, s2_pos-1))

similar_letters(len(s1), len(s2))
```

```
4
```