به نام خدا

تمرین چهارم درس ساختمان داده‌ها و الگوریتم‌ها

چمران معینی

۹۹۳۱۰۵۳

۱- **صرافی صادق:** صادق که تازه از دانشکده فارغ التحصیل شده است، در حال راه اندازی یک صرافی آنلاین است. در این صرافی آنلاین، کاربران می‌توانند ارزهای مختلف را به هم تبدیل کنند. مثلا، آنها در ازای هر یک یورو، ۱.۱۳ دلار آمریکا دریافت کنند و یا در ازای هر دلار آمریکا، ۳.۶۷ درهم امارات دریافت کنند. نرخ تبدیل ارزها در سامانه قرار است هر ساعت در طول ساعات اداری به روز شود. با توجه به اینکه نرخ ها به صورت دستی وارد می‌شوند، امکان خطا در وارد کردن آنها وجود دارد. یکی از اتفاقاتی که در صورت خطا در وارد کردن نرخ تبدیل ارزها ممکن است رخ دهد، این است که افراد بتوانند صرفا با خرید و فروش ارزها، به سود دست پیدا کنند. مثلا اگر نرخ تبدیل یک یورو به دلار ۱.۱۳ و در عین حال نرخ تبدیل دلار به یورو نیز به جای ۰.۸۸ اشتباها ۰.۹۸ وارد شود، آنگاه فرد می تواند ۱ یورو را ابتدا به دلار تبدیل کرده و ۱.۱۳ دلار بگیرد و سپس ۱.۱۳ دلار را به یورو تبدیل کرده ۱.۱ یورو بگیرد و به این ترتیب ۱۰ درصد سود کند. صادق از شما خواسته است که در طراحی یک برنامه برای چک کردن چنین حالتی به او کمک کنید. برنامه شما باید نرخ تبدیل ارزهای مختلف به یکدیگر را به عنوان ورودی گرفته و بررسی کند که آیا مجموعه‌ای از تبدیل ارزها به یکدیگر (مثلا از A به B به C به D به A) وجود دارد که در با انجام آن، کاربر به سود دست یابد (مثلا با ۱ واحد A شروع کرده و در نهایت بیش از ۱ واحد از A به دست آورد؟) دقت کنید که صرافی لزوما امکان تبدیل تمامی ارزها به یکدیگر را فراهم نمی‌کند (مثلا ممکن است هیچ نرخی برای تبدیل مستقیم یورو به درهم امارات نباشد و یا اینکه حتی ممکن است که امکان تبدیل غیر مستقیم یورو به درهم هم فراهم نباشد (مثلا در یک روز، فقط بتوان درهم را به ارزهای دیگر تبدیل کرد و نه برعکس). اگر تعداد ارزهایی که در صرافی موجود است را با n و تعداد نرخ های تبدیل ارز به یکدیگر را با m نمایش دهیم، الگوریتم شما باید از زمان اجرای O(mn) برخوردار باشد. الگوریتم خود را شرح داده و بگویید چرا درست است.

اگر بخواهیم همه‌ی تبدیل‌های مختلف را محاسبه کنیم، بسیاری از محاسبات بارها و بارها تکرار می‌شوند. برای مثال در تبدیل «یورو به درهم به دلار»، و «یورو به درهم به دینار»، و تبدیل «یورو به درهم به ریال»، بارها تبدیل یورو به درهم را محاسبه می‌کنیم، که در مقیاس‌های بزرگ، این تکرارهای زیاد بسیار زمان‌گیر می‌شوند، پس به نظر می‌رسد برنامه‌نویسی پویا می‌تواند در یافتن الگوریتم مناسب، به ما کمک کند.

یک آرایه‌ی دو بعدی به نام dp با سایز n*n می‌سازیم تا مقادیر محاسبه شده را در آن ذخیره کنیم. خانه‌ی ستون i ام و سطر j ام، نشان‌دهنده‌ی این است که با تبدیل i به j ، چند واحد از آن را خواهیم داشت. پس ابتدا می‌توانیم یکی از قطرها را (که در تمام آن i == j است) را برابر با ۱ قرار دهیم.

گفته شده که m نرخ تبدیل عرض داریم. باید تمام آن‌ها را حداقل یک بار محاسبه و سپس ذخیره کنیم. اگر یک نرخ تبدیل، تبدیل ارز i به ارز j را به ما نشان دهد، در dp[i][j] آن را ذخیره می‌کنیم. سپس به سراغ j ها می‌رویم و هرجا که نرخ تبدیلی بود، دوباره آن را تبدیل می‌کنیم و در سطر مربوط در همان ستون می‌نویسیم. در هر ستون i ، آن‌قدر این کار را ادامه می‌دهیم که به i == j برسیم و اگر نتیجه مقداری غیر از ۱ بود، پس ایرادی در مقادیرمان وجود دارد.

یک راه حل دیگر می‌تواند این باشد که آرایه‌ی dp را m*m تعریف کنیم و در ستون اول از هر سطر، یکی از تبدیل‌ها را بنویسیم. ایده‌ی کلی این است که در هر یک از سطرها، مسیری که با تبدیلِ نوشته شده در ستون اول شروع شده است را، آن‌قدر ادامه بدهیم که شاید دوباره به همان ارزی که ابتدا داشتیم رسیدیم. اگر مقدار ۱ بود، که یعنی ایرادی در مقادیر داده شده، وجود دارد که باید اصلاح شود، اما اگر تمام سطرها، به ۱ رسیدند، یعنی مقادیرِ داده شده صحیح هستند.

۲- **اردوی قم:** قرار است دانشگاه اردوی یک روزه زیارت حرم حضرت معصومه (س) را به مناسبت شهادت حضرت زهرا (س) در هفته آینده برگزار کند. طاهره مسئول هماهنگی امور اردو است. یکی از اهداف ضمنی این اردوها، آشنایی بیشتر دانشجویان با یکدیگر است. برای همین، طاهره تصمیم دارد که افراد را به دو گروه تقسیم کند به گونه ای که در هر گروه، هیچ دو فردی قبلا با هم به اردو نرفته باشند. طاهره برای حل این مسئله (که آیا چنین کاری اصلا ممکن است و اگر بله، چطور می توان این کار را انجام داد) ایده زیر به ذهنش رسیده است. آیا این ایده درست است؟ اگر نه، چرا. و اگر بله، چطور می‌توان آن را ثابت کرد.

ایده: یکی از افراد را کنار بگذارد و ببیند که آیا می‌توان افراد باقی‌مانده را به دو گروه با شرط مورد نظر تقسیم کرد یا نه. اگر جواب برای افراد باقی مانده نه است، نتیجه بگیرد که برای کل افراد نیز جواب نه است. در غیر این صورت، جواب تقسیم افراد باقی‌مانده به دو گروه را در نظر گرفته و سعی کند فردی را که کنار گذاشته بود به یکی از دو گروه اضافه کند. اگر توانست، که یک جواب پیدا کرده است. اگر هم نمی‌توان این فرد را به هیچ یک از این دو گروه اضافه کرد، نتیجه بگیرد که جواب برای کل افراد نه است .

در ایده‌ی مطرح شده، آن‌قدر افراد کنار گذاشته می‌شوند که یا تمام می‌شوند (که یعنی جواب نه است) یا نهایتا تعدادی باقی می‌مانند که می‌توان آن‌ها را در دو گروه قرار داد، و پس از آن یکی یکی افراد کنار گذاشته را بررسی می‌کنیم و به یکی از این دو گروه اضافه می‌کنیم.

تا هنگامی که افرادی که اضافه می‌کنیم، قابلیتِ اضافه شدن را داشته باشند (یعنی گروهی باشد که در آن هیچ آشنایی نداشته باشند) پیش می‌رویم. نهایتا یا همه‌ی افراد گروه بندی می‌شوند، یا این که در این میان به موردی می‌رسیم که در هر دو گروه آشنا دارد، که در این صورت پاسخ نه خواهد بود.

این الگوریتم لزوما درست نیست، برای رد آن یک مثال نقض می‌آوریم.

فرض کنید ما سه نفر داریم، علی و محمد و صادق.

در نظر می‌گیریم که علی با محمد دوست باشد. همچنین علی با صادق دوست باشد. اما محمد و صادق با یکدیگر دوست نیستند.

فرض کنید ما ابتدا علی را کنار بگذاریم، سپس محمد و صادق را به دو گروه تقسیم می‌کنیم. حال آیا می‌توانیم علی را به یکی از گروه‌ها اضافه کنیم؟ خیر، زیرا در هر دو گروه دوستانی دارد. به این ترتیب الگوریتم مطرح شده، پاسخ نه می‌دهد، در حالی که می‌توان «علی» را در یک گروه و «محمد و صادق» را در گروه دیگر قرار داد تا به مطلوبِ سوال برسیم.

ایده‌ی مطرح شده در صورت سوال، اگر جواب بله برگرداند، جوابش صحیح است، اما ممکن است گاهی که جواب بله است، به اشتباه نه بازگرداند.

۳- **تیم هماهنگ**: نرگس به تازگی به عنوان مدیر پروژه برای پیاده سازی نرم افزار تلفن همراه کاربران شرکت انتخاب شده است و او باید اعضای تیم خود برای این کار را انتخاب کند. نرگس این اعتقاد را دارد که تیمی هماهنگ است که هر فرد آن، حداقل با k فرد دیگر تیم (به غیر از نرگس) قبلا در یک تیم کار کرده باشد. نرگس باید تیم خود را از بین n کارمند شرکت (به غیر از خودش) انتخاب کند. شرکت یک پایگاه داده دارد که اطلاعات همکاری افراد با یکدیگر در پروژه های مختلف را در خود دارد. برای استفاده از این سامانه، می‌توان از ۴ دستور زیر استفاده کرد:

الف) *مقدار دهی اولیه*: تمامی افراد قابل انتخاب در شرکت را به عنوان اعضای تیم اضافه می کند. پس از آن، می توان سه تابع دیگر را صدا زد. زمان اجرای این تابع $O(n^2)$ است.

ب) *تعداد افراد همکاری کرده با فرد x*: برای فرد x داده شده که هنوز در تیم است، تعداد افراد دیگر موجود در تیم را که x قبلا با آنها همکاری کرده است را باز می‌گرداند. زمان اجرای این تابع $O(1)$ است.

ج) *حذف فرد x از گروه*: فرد x را از تیم حذف می کند. پس از این کار، در محاسبه همکاری افراد (تابع ب)، این فرد دیگر در نظر گرفته نمی‌شود. اگر فرد x در هنگام حذف، با m نفر دیگر از افراد تیم موجود همکاری داشته است، زمان اجرای این تابع $O(m)$ خواهد بود.

د) *گرفتن لیست افراد موجود در تیم*: لیستی از افرادی که در حال حاضر در تیم هستند را بر می گرداند. اگر در حال حاضر p فرد در تیم هستند، زمان اجرای این تابع $O(p)$ خواهد بود.

به نرگس کمک کنید تا یک الگوریتم با زمان کلی $O(n^2)$ طراحی کند که تشخیص بدهد آیا نرگس می‌تواند تیمی با مشخصات مورد نظرش تشکیل بدهد یا نه. و اگر بله، بزرگترین تیم ممکن (از نظر تعداد افراد) را تشکیل دهد. دقت کنید که ممکن است بیش از یک تیم قابل تشکیل باشند و تشکیل یکی از آنها کافی است.

ابتدا تابع الف صدا می‌شود.

پس از آن، تابع ب را برای تمام اعضای دیگر صدا می‌کنیم و اگر مقداری که برای هر عضو برمی‌گرداند کمتر از k بود، تابع ج را برای آن فرد اجرا می‌کنیم.
سپس تابع د را صدا می‌زنیم و روی لیستی که باز گردانده است، همین فرآیند را دوباره تکرار می‌کنیم.

یا این فرآیند آن‌قدر تکرار می‌شود که دیگر هیچ عضوی در تیم باقی نماند که در این صورت یعنی چنین چیزی ممکن نیست، یا به جایی می‌رسم که هیچ عضوری در طول یک فرآیند حذف نمی‌شود، که یعنی اعضای باقیمانده، بزرگ‌ترین گروه ممکن هستند.

۴- **گزارش عملکرد**: کاظم چند سالی است که مدیر عامل شرکت است. در طول این مدت، آنها n قرارداد با مشتری ها امضا کرده‌اند و ارزش قرارداد iام برابر با v_i است. فرض کنید که قراردادها به ترتیب زمان امضا هستند، یعنی اگر $i<j$، آنگاه قرارداد iام قبل از قرارداد jام امضا شده است (هرچند ارزش قرارداد iام می‌تواند کمتر، مساوی، و یا بیشتر از قرارداد jام باشد.) قرار است هفته آینده او ارائه‌ای برای گروهی از سرمایه گذاران داشته باشد. کاظم می‌خواهد که به جای نمایش و ذکر تمامی قراردادها، زیر مجموعه‌ای از آنها را بیان کند که ارزش آنها اکیدا صعودی باشد. به عبارت دیگر، او میخواهد مجموعه ای از قراردادها مانند $i_1, i_2, ..., i_k$ را انتخاب کند که $i_1<i_2<...<i_k$ و همچنین $v_{i1}<v_{i2}<..._v_{ik}$. هدف کاظم از این کار ارائه یک نمای رو به رشد از شرکت (با توجه به افزایش ارزش قراردادها در طول زمان) است. یک الگوریتم برای کمک به کاظم طراحی کنید که بزرگ‌ترین زیرمجموعه قراردادها را که شرط مورد نظر را دارند، پیدا کند. زمان الگوریتم شما باید $O(n^2)$ باشد .

نکته: دقت کنید که این مسئله را می‌توان در زمان $O(nlgn)$ حل کرد. برای این تمرین لازم نیست این کار را بکنید، اما می‌توانید روی آن هم فکر کنید!

اولین راه حلی که برای حل این سوال به نظر می‌رسد، این است که تمام زیرمجموعه‌های ممکن را بررسی کنیم و اگر صعودی بودند، طول آن‌ها را محاسبه کنیم و سپس از بین نتایج، ماکسیمم را به عنوان نتیجه اعلام کنیم، اما این روش را نمی‌توان در O(n2) انجام داد. با کمی دقت، متوجه می‌شویم که در این روش، بسیاری از مواقع برخی از محاسبات چند بار انجام می‌شوند. برای مثال دنباله‌ی زیر را در نظر بگیرید:

1, 2, 4, 3, 6, 7

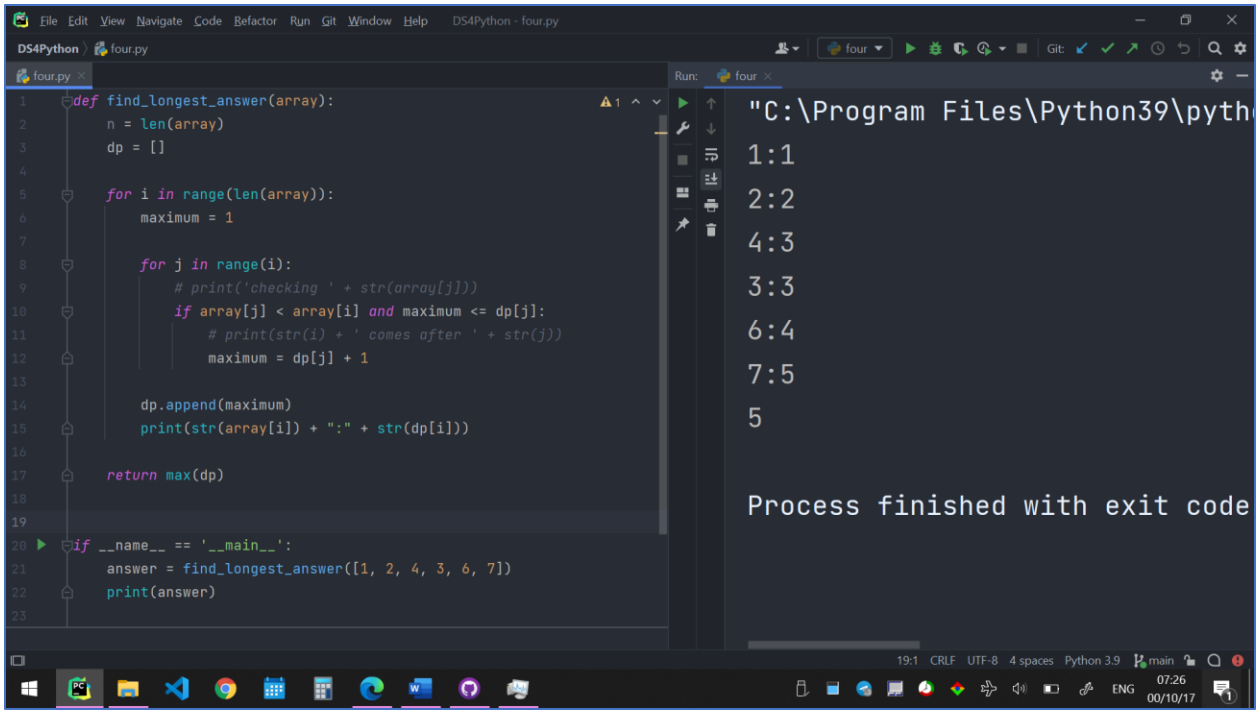
در این دنباله، می‌دانیم که طولانی‌ترین دنباله‌ی اکیدا صعودی که از ۱ تا ۴ وجود دارد، به طول ۳ است، اما در محاسبه‌ی طول 1, 2, 4, 6, 7 و 1, 2, 3, 6, 7 ، دوبار این شمارش تکرار شده است. پس از برنامه نویسی پویا کمک می‌گیریم تا برای مثال هرگاه که خواستیم اعداد چهار و چهار به بعد را بررسی کنیم، بدانیم که بلندترین دنباله‌ی اکیدا صعودیِ قبل از ۴ ، به طولِ ۳ است.

می‌دانیم که هر عضو، حداقل زیرمجموعه‌ای اکیدا صعودی به طول یک دارد (مجموعه‌ای که تنها شامل خود عدد باشد) ، پس ابتدا آرایه‌ی dp را به طول n تعریف می‌کنیم و تمام اعضای آن را ۱ می‌گذاریم. قرار است که عضو i ام از این دنباله، نشان‌دهنده‌ی این باشد که طولانی‌ترین دنباله‌ی اکیدا صعودی که به این عضو ختم می‌شود، چه طولی دارد.

این مقدار برای اولین عضو همواره ۱ است، یعنی همیشه dp[0]=1

برای عناصر بعدی، یک واحد به بزرگ‌ترین dp ی قبلی‌شان که مقدار v اش از v عنصر مورد نظر کوچک‌تر باشد، اضافه می‌کنیم. برای ۲ ، این مقدار یکی بیشتر از dp ی ۱ خواهد بود، یعنی همان ۲ . برای ۴ این مقدار یکی بیشتر از دی‌پیِ ۲ خواهد بود، یعنی ۳ ، برای ۳ هم این مقدار یکی بیشتر ای دی‌پیِ یک خواهد بود، یعنی ۳

برای ۶ ، این مقدار برابر دی‌پیِ ۴ به علاوه‌ی یک خواهد بود، یعنی همان ۵ و برای هفت هم دی‌پی۶ به علاوه‌ی یک را خواهیم داشت که برابر ۵ می‌شود.


File Edit View Navigate Code Refactor Run Git Window Help
DS4Python - four.py
DS4Python
four.py
four
Git:
four.py
def find_longest_answer(array):
n = len(array)
dp = []
for i in range(len(array)):
maximum = 1
for j in range(i):
# print('checking ' + str(array[j]))
if array[j] < array[i] and maximum <= dp[j]:
# print(str(i) + ' comes after ' + str(j))
maximum = dp[j] + 1
dp.append(maximum)
print(str(array[i]) + ":" + str(dp[i]))
return max(dp)
if __name__ == '__main__':
answer = find_longest_answer([1, 2, 4, 3, 6, 7])
print(answer)
Run:
four
"C:\Program Files\Python39\pyth
1:1
2:2
4:3
3:3
6:4
7:5
5
Process finished with exit code
19:1 CRLF UTF-8 4 spaces Python 3.9 main
ENG
07:26
00/10/17