به نام خداوند علم و دانش

دانشکده مهندسی کامپیوتر

برنامه نویسی پیشرفته(پایتون)

تمرین دوم (پایتون مقدماتی + oop)

دکتر مرضیه داودآبادی

زمستان 1403

طراحان تمرین: امیر سلیمی و مبینا هشیاری پور و مهدادچراغی

- در صورت وجود هرگونه ابهام به طراح پیام دهید.
- باتوجه به وجود تاخیر 10 روزه، امکان تاخیر تحت هیچ شرایطی امکان پذیر نیست.
- انجام تمرین ها تک نفره می‌باشد.
- زبان برنامه نویسی پایتون است.
- موارد ارسال شده به صورت آنلاین تحویل گرفته خواهند شد.
- ددلاین تمرین: 18 فروردین ساعت 23:59
- برای دیدن تست کیس های نمونه متنوع به کوئرای درس بپیوندید.
- https://quera.org/course/add_to_course/course/20444/
- Password: ap4032
- لینک تلگرام طراحان:
- آقای امیر سلیمی | https://telegram.me/IamAmirsalimi
- خانم مبینا هشیاری پور | https://telegram.me/mobinahhh
- آقای مهداد چراغی | https://telegram.me/GMinfinite

01.یافتن دو عدد گم شده (سلیمی)

در یک سرزمین پر از اعداد، همه آن‌ها زوج بودند، به‌جز دو عدد خاص که تنها مانده بودند! این دو عدد منحصربه‌فرد هیچ جفتی نداشتند، و حالا وظیفه‌ی شماست که آن‌ها را پیدا کنید!

چالش: یک آرایه از اعداد صحیح به شما داده می‌شود که در آن همه‌ی اعداد به‌جز دو عدد خاص دقیقاً دو بار ظاهر شده‌اند. شما باید این دو عدد را که فقط یک بار در آرایه حضور دارند، پیدا کنید. الگوریتم شما نباید شامل لوپ تو در تو و حافظه اضافی باشد (فقط با استفاده از همان لیست اعداد ورودی باید به جواب برسید)

ورودی

یک آرایه از اعداد صحیح که در آن تمام اعداد به‌جز دو عدد خاص، دقیقاً دو بار تکرار شده‌اند.

خروجی

یک لیست شامل دو عدد گمشده که فقط یک بار در آرایه ظاهر شده‌اند.

مثال ۱:

Input: [2, 4, 6, 8, 10, 2, 6, 10]

Output: [4, 8]

مثال ۲:

Input: [1, 2, 3, 5, 1, 2, 3, 7]

Output: [5, 7]

02.ضرب دو عدد بدون استفاده از عملگر ضرب(سلیمی)

شما باید دو عدد صحیح aو b را بدون استفاده از عملگر * (ضرب) در هم ضرب کنید! اما چالش اصلی اینجاست: شما فقط می‌توانید از جمع (+)، تفریق (-)، و توابع بازگشتی (Recursion) استفاده کنید.

اما این بار فقط با تکرار ساده نمی‌توانید حلش کنید! باید راهی باهوش‌تر پیدا کنید که تعداد دفعاتی که تابع خودش را صدا می‌زند، کمتر شود!

ورودی

دو عدد صحیح a و b که می‌توانند مثبت یا منفی باشند.

مثال ۱:

Input: multiply(15, 17)

Output: 255

خروجی

یک عدد صحیح که حاصل ضرب a×b را نشان می‌دهد.

قوانین:

✅ نباید از عملگر * استفاده کنید!

✅ تابع شما باید بازگشتی (Recursive) باشد.

✅ اگر یکی از اعداد منفی باشد، نتیجه را به درستی محاسبه کنید.

✅ نباید تعداد زیادی فراخوانی بازگشتی بیهوده انجام شود!

03.یافتن کوچک‌ترین زیررشته(سلیمی)

یک متن خیلی طولانی و یک لیست از کلمات مشخص به شما داده شده است. شما باید کوتاه‌ترین زیررشته‌ای از متن را پیدا کنید که همه‌ی کلمات در لیست را حداقل یک بار شامل شود (با همان تعداد تکرار).

نمی‌توانید از روش‌های ساده‌ی brute force استفاده کنید، زیرا متن بسیار بزرگ است! باید از دیکشنری (dictionary) و تکنیک‌های پیشرفته‌ای مثل sliding window استفاده کنید!

ورودی ها:

text (یک رشته که شامل جمله‌ی اصلی است)

word_list (یک لیست از کلمات که باید در زیررشته وجود داشته باشند)

خروجی ها

- کوچک‌ترین زیررشته‌ای از text که تمام کلمات word_list را شامل می‌شود.
- اگر چنین زیررشته‌ای وجود ندارد، مقدار "" (رشته‌ی خالی) برگردانید.

مثال ها

مثال ۱:

```
"text = "this is a test string containing a test
word_list = ["test", "a"]
Output: "a test"
```

"a test"کوچک‌ترین زیررشته‌ای است که شامل "test" و "a" است!

مثال 2:

```
"text = "hello world programming is fun word_list = ["python",
"fun"]
Output: ""
```

🚫 چون کلمه "python" در جمله نیست، خروجی "" است.

04.قیمت بیت‌کوین(چراغی)

در صرافی ها آخرین قیمت بیت کوین در انتهای هر روز تهیه می شود و بر روی این داده ها تحلیل های مختلف انجام میگیرد. فرض کنید میخواهیم در هر روز "دوره ی قیمت" را پیدا کنیم. بدین صورت که اگر قیمت بیتکوین در روز i برابر p_i باشد دوره ی قیمت در روز i برابر است با تعداد روزهای بلافاصله قبل از (i و شامل خود i) که قیمت بیت کوین کمتر یا مساوی p_i باشد. اگر قیمت بیتکوین در n روز متولی در آرایه ی n تایی به نام P ذخیره شده باشد میخواهیم در خروجی آرایه ی n تایی S را محاسبه کنیم که $S[i]$ دوره ی قیمت در روز i باشد.

ورودی ها:

$$1 \leq n \leq 10^5 \, , 1 \leq p[i] \leq 10^9$$

در خط اول عدد n و در خط بعدی n عدد که عدد i ام قیمت بیتکوین در روز i ام است.

خروجی ها

آرایه ی n تایی S که $S[i]$ دوره ی سهام در روز i ام را نشان میدهد.

برنامه دنباله مورد نظر را محاسبه کرده و به عنوان خروجی نمایش می‌دهد.

مثال‌ها

ورودی

```
7
[10,7,2,4,1,6,9]
```

خروجی

```
[1,1,1,2,1,4,6]
```

05. سیستم فرودگاه(چراغی)

توجه شود که این سوال تست کیس ندارد و دستی اصلاح خواهد شد. قابل ذکر است که برخی جزئیات مانند اینکه دقیقا چه متنی مثلا برای اعلام خروج نوشته شود اهمیت ندارد.

در این سوال هدف بر این است که شما سیستمی شبیه مدیریت پرواز ها در فرودگاه بسازید. در ابتدای برنامه ترمینال را پاک کنید (با استفاده از کتابخانه os) و منوی برنامه را نشان دهید.

منوی شما باید شامل دو آپشن باشد: ورود و خروج. برای مثال:

```
+------------------------------+
|         AirPortSystem        |
| 1.Login                      |
| 2.Exit                       |
+------------------------------+
```

و سپس از یوزر ورودی بخواهد.

اگر 2 وارد شد برنامه پیغام خروج نشان دهد و پایان یابد. اگر 1 وارد شد وارد بخش لاگین شود، و اگر هرچیز دیگری وارد شد پیغام بدهد که ورودی اشتباه است و دوباره ورودی بخواهد.

بخش لاگین

اگر یوزر 1 وارد کرد برنامه وارد بخش لاگین میشود. ترمینال بار دیگر پاک شده و درخواست نام کاربری و رمز عبور کند. فرض میکنیم تنها نام کابری قابل قبول "admin" و تنها رمز قابل قبول "1admin" می باشد (میتوانید نام کاربری و رمز عبور دلخواه تعریف کنید).

سپس اطلاعات ورودی پردازش میشود اگر نام کاربری اشتباه بود پیغام نمایش داده شده که نام کاربری اشتباه است و برنامه پایان یابد. اگر رمز اشتباه بود، پیغام اینکه رمز اشتباه است نشان داده شده و برنامه پایان یابد. اگر هر دو مورد درست بود برنامه وارد بخش ادمین بشود.

بخش ادمین

در این بخش دوباره ترمینال پاک شده و منو با چهار گزینه نمایش داده شود:

- اضافه کردن پرواز
- حذف پرواز
- نمایش پرواز ها
- خروج

و دوباره از یوزر درخواست ورودی کند. در صورت اشتباه بودن ورودی پیغام خطا و درخواست ورودی جدید شود، در صورت وارد کردن 4 برنامه پیغام خروج نمایش داده و پایان یابد. در صورت وارد کردن 1 یا 2 یا 3 دوباره ترمینال پاک شده و عملیات مناسب انجام گردد.

پرواز

هر پرواز شامل 3 ویژگی است:

1. شماره پرواز
2. مبدا پرواز
3. مقصد پرواز

اضافه کردن پرواز

ترمینال پاک شده و به ترتیب درخواست شماره پرواز، مبدا و مقصد میکند و بعد از اضافه کردن پرواز پیغام مناسب چاپ شده و برنامه دوباره منوی ادمین را نشان میدهد.

حذف پرواز

ترمینال پاک شده و برنامه درخواست شماره پرواز میکند. اگر شماره پرواز بین پرواز ها بود، پرواز را حذف میکند و پیغام مناسب را نشان میدهد و اگر نبود خطا نشان میدهد. بعد از انجام عملایت دوباره منوی ادیمن نمیاش داده میشود.

لیست پرواز ها

ترمینال پاک شده و لیست پرواز ها را نشان میدهد. ابتدا شماره پرواز سپس مبدا و مقصد پرواز.

لیست پرواز ها در ابتدای برنامه خالی است.

در این سوال دست شما برای اضافه کردن آپشن های دیگر باز است. مثلا لیست پرواز ها در فایلی ذخیره شود یا کاربر تعریف شود که فقط بتواند لیست پرواز ها را ببیند. خواسته ی اصلی سوال مواردی بود که گفته شد.

06.بمب ساعتی (چراغی)

توجه شود که این سوال تست کیس ندارد و دستی اصلاح خواهد شد. قابل ذکر است که برخی جزئیات مانند اینکه دقیقا چه متنی مثلا برای اعلام خروج نوشته شود اهمیت ندارد.

در این تمرین میخواهیم بازی بمب ساعتی بسازیم.

ابتدا یک عدد بین 5 تا 20 به صورت رندوم به عنوان تایمر بمب انتخاب میشود و تایم بمب ساعتی را چاپ میکند. سپس از لیستی از رنگ ها بین 3 تا 8 رنگ را به عنوان سیم های بمب انتخاب میکند و یکی از این سیم ها به صورت رندوم سیمی است که خنثی کننده است.

نکته مهم:

رنگ های سیم ها باید به گونه ای باشند که دو حرف پشت سر هم تکراری نیاید. برای مثال رنگ green نباید در بین رنگ های سیم ها باشد چون شامل دو e پشت هم است. به جای آن میتوان نوشت . green رنگی مانند purple مورد قبول است چون دو p پشت سر هم نیامده اند.

پس از انتخاب رنگ ها لیست رنگ های بمب چاپ شود و تایمر شروع شود. با استفاده از تابع input__timed که کد آن به شما داده میشود درخواست نام سیمی که میخواهیم ببریم خواسته شود. اگر ورودی وارد شد، چک شود اگر سیم درست بود بمب خنثی شده و با نمایش پیغام مناسب برنامه پایان میابد و اگر سیم درست نبود یا ورودی ای در طی 1 ثانیه وارد نشد پیغام مناسب نمایش داده شده و از زمان بمب یک ثانیه کم میشود و زمان باقی مانده چاپ شده و دوباره نام سیم های بمب چاپ شده و بتوان نام سیم بعدی را وارد کرد.

اگر زمان به صفر برسد بووووووووم!!!

تابع timed_input

```
import time
import random
import sys
import os
def timed_input(prompt):
timeout=3
print(prompt, end="", flush=True)
"" = user_input
start_time = time.time()
if os.name == "nt":  # Windows
import msvcrt
last_key = None
while time.time() - start_time < timeout:
if msvcrt.kbhit():
char = msvcrt.getwch()
if char == "\r":
break
if char.isprintable() and char != last_key:
user_input += char
last_key = char
time.sleep(0.1)
print("\n", end="", flush=True)
else:  # Linux/macOS
import termios, fcntl
```

```
()fd = sys.stdin.fileno
old_settings = termios.tcgetattr(fd)
[:]new_settings = old_settings
new_settings[3] = new_settings[3] & ~termios.ICANON
fcntl.fcntl(fd, fcntl.F_SETFL, os.O_NONBLOCK)
termios.tcsetattr(fd, termios.TCSANOW, new_settings)
:try
last_key = None
:while time.time() - start_time < timeout
(1)char = sys.stdin.read
:if char
:"if char == "\n
break
:if char.isprintable() and char != last_key
user_input += char
last_key = char
(0.1)time.sleep
:finally
termios.tcsetattr(fd, termios.TCSANOW, old_settings)
print("\n", end="", flush=True)
return user_input.strip() if user_input else None
```

ورودی تابع متنی است که میخواهیم در ترمینال چاپ شود.

این تابع بدین گونه کار میکند که اگر در عرض 3 ثانیه ورودی ای وارد شد، ورودی را برمیگرداند در غیر این صورت None برمیگرداند. (دلیل 3 ثانیه بودن کمک به تایپ است و در عمل همانند یک ثانیه حس میشود).

نحوه استفاده برای مثال:

```
wire = timed_input("enter wire name: ")
```

07.مدیریت وسایل نقلیه(هشیاری پور)

شما مسئول توسعه یک سیستم ساده برای مدیریت حمل و نقل شهری هستید. این سیستم باید قادر باشد ایستگاه‌ها، مسیرها و وسیله‌های نقلیه را مدیریت کند. علاوه بر این، کاربران باید بتوانند از ایستگاه‌ها به مقصد خود سفر کنند و سیستم باید مسیر بهینه را با توجه به وضعیت سوخت وسیله نقلیه پیدا کند. هدف این تمرین این است که شما یک مدل ساده برای مدیریت ایستگاه‌ها، مسیرها، وسیله‌های نقلیه و سفرها طراحی کنید.

جزئیات:

۱ .مدیریت ایستگاه‌ها:

- هر ایستگاه باید دارای شناسه و نام باشد.
- ایستگاه‌ها می‌توانند با یکدیگر از طریق مسیرهایی به هم متصل شوند که این مسیرها شامل مسافت هستند.

۲ یافتن کوتاه‌ترین مسیر:

- شما باید الگوریتمی برای پیدا کردن کوتاه‌ترین مسیر از یک ایستگاه به ایستگاه‌های دیگر پیدا کنید.
- گراف شما باید شامل گره‌ها (ایستگاه‌ها) و یال‌ها (مسیرها با مسافت) باشد.

۳ وسیله نقلیه:

- هر وسیله نقلیه دارای شناسه، نام و میزان سوخت است.
- وسیله نقلیه باید قادر به طی کردن مسافت‌ها باشد تا بتواند سفر را آغاز کند.

۴ سفر:

- سفر باید شامل اطلاعات مربوط به وسیله نقلیه و مسیر (ایستگاه مبدا و مقصد) باشد.
- سفر باید ابتدا چک شود که وسیله نقلیه سوخت کافی دارد یا خیر، سپس مسیریابی انجام شود.

کلاس‌ها و ویژگی‌ها:

۱ .کلاس **Station** :

ویژگی‌ها:

- station_id شناسه ایستگاه
- station_name نام ایستگاه
- connected_stations لیستی از ایستگاه‌هایی که به این ایستگاه متصل هستند به همراه مسافت‌ها

متدها:

- add_connection(station, distance): افزودن یک ایستگاه و مسافت به لیست ایستگاه‌های متصل

۲ .کلاس **Route**

ویژگی‌ها:

- start_station ایستگاه مبدا
- end_station ایستگاه مقصد

- distance مسافت بین دو ایستگاه

متدها:

- **str()** نمایش اطلاعات مسیر

۳ .کلاس Vehicle

ویژگی‌ها:

- vehicle_id شناسه وسیله نقلیه
- vehicle_name نام وسیله نقلیه
- fuel میزان سوخت

متدها:

- add_fuel(amount) افزودن سوخت
- use_fuel(amount) مصرف سوخت
- can_travel(distance) بررسی اینکه وسیله نقلیه قادر به طی کردن مسافت معین است یا نه

۴ .کلاس Travel

ویژگی‌ها:

- travel_id شناسه سفر
- vehicle وسیله نقلیه‌ای که سفر را انجام می‌دهد
- route مسیری که باید طی شود
- statusوضعیت سفر شامل "Not started", "In progress", "Completed"

متدها:

- start_travel() آغاز سفر
- complete_travel() پایان سفر

۵ .تابع find_shortest_path یافتن کوتاه‌ترین مسیر:

ویژگی‌ها:

- start_station ایستگاه مبدا
- stations لیست تمامی ایستگاه‌ها

متدها:

- find_shortest_path(): اجرای الگوریتمی برای پیدا کردن کوتاه‌ترین مسیر از ایستگاه مبدا به ایستگاه‌های دیگر (می‌توانید از الگوریتم دایجکسترا یا هر الگوریتم مناسب دیگری برای این کار استفاده کنید)

ورودی

ورودی‌ها به صورت دستورات مختلف وارد می‌شوند که شامل اطلاعات ایستگاه‌ها، مسیرها، وسیله‌های نقلیه و سفرها است:

۱. ثبت ایستگاه‌ها:

- register_station <station_id> <station_name>: ثبت یک ایستگاه جدید با شناسه و نام.

۲ ثبت مسیرها:

- add_route <start_station_id> <end_station_id> <distance>: ثبت یک مسیر جدید بین دو ایستگاه با مسافت مشخص.

۳ ثبت وسیله نقلیه:

- register_vehicle <vehicle_id> <vehicle_name> <fuel>: ثبت یک وسیله نقلیه جدید با شناسه، نام و سوخت.

۴ یافتن کوتاه‌ترین مسیر:

- find_shortest_path <start_station_id>: پیدا کردن کوتاه‌ترین مسیرها از ایستگاه مبدا به دیگر ایستگاه‌ها.

۵ آغاز سفر:

- start_travel <travel_id> <vehicle_id> <start_station_id> <end_station_id>: شروع سفر از ایستگاه مبدا به مقصد با استفاده از وسیله نقلیه مشخص.

۶ پایان سفر:

- complete_travel <travel_id>: تکمیل سفر و تغییر وضعیت آن.

۷ .خروج از برنامه:

- exit برای خروج از برنامه.

خروجی

برای هر دستور، سیستم باید پیغام مناسبی را نمایش دهد:

۱ ثبت ایستگاه‌ها و مسیرها:

- Station registered successfully

+Route added successfully

۲ ثبت وسیله نقلیه:

+Vehicle registered successfully

۳ یافتن کوتاه‌ترین مسیر:

- نمایش مسافت‌های کوتاه از ایستگاه مبدا به دیگر ایستگاه‌ها.

۴ آغاز سفر:

- Travel started successfully
- اگر وسیله نقلیه سوخت کافی نداشته باشدNot enough fuel for the trip :

۵ پایان سفر:

- Travel completed successfully

08.مدیریت زمان(هشیاری پور)

در این تمرین، شما باید یک سیستم مدیریت زمان طراحی کنید. هر وظیفه (Task) دارای نام و مدت‌زمانی است که برای انجام آن لازم است. هدف این است که تمام وظایف همنام را با هم جمع کرده و سپس آن‌ها را بر اساس زمان موردنیازشان از کمترین به بیشترین مرتب کنیم.

جزئیات

۱ کلاس Time

این کلاس نمایانگر یک زمان مشخص است و شامل سه ویژگی زیر می‌باشد:

- hours تعداد ساعت‌ها
- Minutes تعداد دقیقه‌ها
- Seconds تعداد ثانیه‌ها

متدهای کلاس Time

- **init**(self, hours, minutes, seconds): مقدار ساعت، دقیقه و ثانیه را دریافت کرده و یک شیء جدید ایجاد می‌کند.
- **str**(self): زمان را به‌صورت رشته‌ای در قالب "HH:MM:SS" نمایش می‌دهد.

- Overloading اپراتورها:

- جمع زمان‌ها (+): این اپراتور دو زمان را با هم جمع می‌کند. (مثلاً "1:30:20" + "2:10:40" = "3:41:00")

- مقایسه زمان‌ها (==, >, <) این اپراتورها امکان مقایسه دو زمان را فراهم می‌کنند.

۲ .کلاسTask

این کلاس نمایانگر یک وظیفه است که شامل موارد زیر است:

- task_name نام وظیفه مثلاً "shop", "homework", "sleep"

- time مدت‌زمان موردنیاز برای انجام این وظیفه (یک شیء از کلاسTime)

متدهای کلاسTask

- **str**(self) نمایش وظیفه به‌صورت "نام وظیفه" HH:MM:SS

- Overloading اپراتور + : اگر دو وظیفه نام یکسانی داشته باشند، زمان آن‌ها با هم جمع می‌شود.

- مقایسه وظایف (<, >) این اپراتورها بر اساس زمان کل، وظایف را مقایسه می‌کنند.

۳ ورودی و عملیات‌های موردنیاز

برنامه ورودی را دریافت کرده و وظایف مشابه را با هم جمع می‌کند. سپس لیست وظایف را بر اساس زمان موردنیازشان مرتب کرده و نمایش می‌دهد.

ورودی

هر خط از ورودی شامل نام یک وظیفه و زمان موردنیاز برای انجام آن به فرمت زیر است

```
<task_name> <hours>:<minutes>:<seconds>
```

اگر یک وظیفه بیش از یک‌بار در ورودی ظاهر شود، مقدار زمان‌های آن‌ها باید با هم جمع شوند.

در انتها، همه وظایف بر اساس زمان موردنیازشان از کمترین به بیشترین مرتب شده و نمایش داده می‌شوند.

خروجی

برنامه پس از پردازش ورودی، ابتدا تمام زمان‌های مشابه را با هم جمع می‌کند، سپس لیست وظایف را بر اساس زمان مرتب کرده و نمایش می‌دهد.

09.اورژانس بیمارستان(هشیاری پور)

بیمارستان دارای یک بخش اورژانس است که بیماران را براساس شدت بیماری (priority) پذیرش می‌کند. هرچه شدت بیماری بیشتر باشد (عدد بالاتر)، بیمار زودتر پذیرش می‌شود. برای مدیریت این فرآیند، بیمارستان از یک صف اولویت‌دار Priority Queue استفاده می‌کند.

برای پیاده‌سازی این سیستم باید یک کلاس EmergencyRoom طراحی کنید که قابلیت مدیریت بیماران اورژانسی را داشته باشد. این کلاس باید از یک Priority Queue برای مرتب‌سازی بیماران بر اساس شدت بیماری استفاده کند.

جزئیات:

ویژگی‌های کلاس EmergencyRoom

افزودن بیمار(add_patient(name, priority))

- بیمار جدیدی با نام و شدت بیماری مشخص به صف اضافه شود.
- عدد بزرگ‌تر = اولویت بالاتر (مثلاً بیماری با اولویت 5 زودتر از بیماری با اولویت 3 پذیرش می‌شود).

پذیرش بیمار(next_patient())

- بیماری که بالاترین اولویت را دارد از صف حذف شود و نام او برگردانده شود.
- اگر صف خالی بود، مقدار None برگردانده شود.

لیست بیماران در صف(get_all_patients())

- لیستی از بیماران به ترتیب اولویت (از زیاد به کم) برگردانده شود.
- فرمت خروجی[(name1, priority1), (name2, priority2), ...] :

عملیات‌های مورد نیاز

مدیریت بیماران اورژانس

- add_patient <name> <priority> - افزودن بیمار به صف.
- next_patient - پذیرش بیمار با بالاترین اولویت.
- get_all_patients - نمایش لیست بیماران به ترتیب اولویت.