تمرین ۲ درس ساختمان داده

سید صالح اعتمادی
مریم سادات هاشمی

دانشگاه علم و صنعت ۹۸_۹۷

لطفا به نکات زیر توجه کنید:

- مهلت ارسال این تمرین شنبه چهاردهم مهر ماه ساعت ۱۱:۵۹ ب.ظ است.
- این تمرین شامل سوال های برنامه نویسی می باشد، بنابراین توجه کنید که حتماً موارد خواسته شده را رعایت کنید.
- نام شاخه، پوشه و پول ریکوست همگی دقیقا "A۲" باشد.
- در صورتی که به اطلاعات بیشتری نیاز دارید می توانید با ایدی تلگرام @maryam_sadat_hashemi در ارتباط باشید.

باتشکر.

۱ Maximum Pairwise Product

در این تمرین شما باید حداکثر حاصل ضرب دو عدد متمایز را در یک دنباله از اعداد صحیح غیر منفی پیدا کنید.

	5	6	2	7	4
5		30	10	35	20
6	30		12	42	24
2	10	12		7	4
7	35	42	14		28
4	20	24	8	28	

ورودی: دنباله ای از اعداد صحیح غیر منفی.
خروجی: حداکثر مقدار که می توان با ضرب دو عنصر مختلف از دنباله به دست آورد.

توجه: به محدویت ها و فرمت هایی که در داکیومنت اصلی تمرین توضیح داده شده، دقت کنید.

- محدودیت زمانی : **۵۰۰ میلی ثانیه**
- محدودیت حافظه: **۵۱۲ مگابایت**

۲ Naive Algorithm

در این مرحله، مشابه تمرین قبل یک پروژه به نام A۲ بسازید. یک تابع به اسم NaiveMax–PairWiseProduct بسازید که دنباله ی ورودی را به صورت لیست یا ارایه بگیرد. سپس مطابق با توضیحات داکیومنت اصلی، Naive Algorithm را در این تابع پیاده سازی کنید.

```
public static int NaiveMaxPairwiseProduct(List<int> numbers)
```

اکنون برای اطمینان از درستی الگوریتم خود، برای پروژه یک UnitTest بسازید. (مشابه کاری که در تمرین قبل انجام دادید.) اکنون باید به پروژه ی تستتان مطابق شکل زیر دو TestMethod به نام های GradedTest_Performance و GradedTest_Correctness را اضافه کنید.

```
[TestMethod()]
[DeploymentItem("TestData", "TestData")]
0 references | 0 changes | 0 authors, 0 changes
public void GradedTest_Correctness()
{
    TestCommon.TestTools.RunLocalTest(Program.Process);
}

[TestMethod(), Timeout(500)]
[DeploymentItem("TestData", "TestData")]
0 references | 0 changes | 0 authors, 0 changes
public void GradedTest_Performance()
{
    TestCommon.TestTools.RunLocalTest(Program.Process);
}
```

کار متد GradedTest_Correctness تنها بررسی درستی الگوریتم است و محدودیت زمانی را در این حالت در نظر نمی گیریم. اما در متد GradedTest_Performance شما علاوه بر اطمینان از درستی الگوریتم، از کارآمد بودن الگوریتم از لحاظ مدت زمان اجرایی نیز اطمینان حاصل کنید. به زبانی ساده تر با استفاده از این متد، مشخص می شود که برنامه ی شما کمتر از۵۰۰ میلی ثانیه اجرا می شود و سرعت قابل قبولی را در اجرای برنامه دارد.

در ضمیمه این فایل، testcase های لازم را در فولدر TestData قرار داده شده است که شما باید این فولدر را به برنامه ی تست خود اضافه کنید و باید برای تست برنامه تان از testcase های موجود در TestData استفاده کنید. برای این منظور باید همانند تمرین قبل، پروژه ی TestCommon که فایل های قبلا در اختیار شما قرار داده شده است؛ را به پروژه ی تست خود اضافه کنید و از کد زیر استفاده کنید.

```
TestCommon.TestTools.RunLocalTest(Program.Process);
```

متد RunLocalTest یک تابع را به عنوان ورودی دریافت می کند که ورودی و خروجی این تابع از نوع string است. بنابراین شما باید یک تابع process در پروژه ی اصلی خود داشته باشیدکه به عنوان ورودی به متد RunLocalTest بدهید. ساختار تابع process مطابق شکل زیر باید باشد:

```
public static string Process(string input)
{
    var inData = input.Split(new char[] { '\n', '\r', ' ' },
        StringSplitOptions.RemoveEmptyEntries)
         .Select(s => int.Parse(s))
         .ToList();

    return NaiveMaxPairwiseProduct(inData).ToString();
}
```

هدف از تابع proecess این است که دنباله ی مورد نظر را از ورودی می گیرد و سپس پردازش لازم روی دیتای ورودی انجام شود تا ورودی مطابق با نوع ورودی تابع الگوریتم شما باشد. برای مثال اگر تابع الگوریتم شما داده را به صورت لیست دریافت می کند شما باید داده را به صورت لیست در بیاورید تا بتوانید داده را به تابع الگوریتم خود بدهید. و در نهایت هم خروجی تابع process همان خروجی تابع الگوریتم شما خواهد بود.

اکنون شما موفق شدید تابع الگوریتم و process در پروژه ی اصلی و متد های Grad-edTest_Correctness و GradedTest_Performance را پیاده سازی کنید و باید الگوریتم خود را تست کنید.

اما تست GradedTest_Performance شکست می خورد. مشکل از کجاست؟

علت این موضوع در داکیومنت اصلی پروژه توضیح داده شده است. این مشکل در بخش بعد حل خواهد شد.

تذکر: شما می توانید متد های تست دیگری به غیر از دو متد هایی که در بالا ازشما

خواسته شده است، در پروژه ی تست خود داشته باشید و داده های دلخواه خود را امتحان کنید.

```
[TestMethod()]
0 | 0 references | 0 changes | 0 authors, 0 changes
public void NaiveMaxPairwiseProductTest()
{
    int mpwp = Program.NaiveMaxPairwiseProduct(new List<int>() {10, 14, 15, 9 });
    Assert.AreEqual(mpwp, 14 * 15);
}
```

۳ Fast Algorithm

در این بخش برای حل مشکل Naive Algorithm راهی مطرح شده است. توضیحات داکیومنت اصلی را بخوانید و مطمئن شوید که ایده ی آن را به خوبی متوجه شدید. سپس همانند بخش قبل یک تابع به نام FastMaxPairwiseProduct در پروژه ی خود بسازید و الگوریتم این بخش را در این تابع پیاده سازی کنید.

```
public static int FastMaxPairwiseProduct(List<int> numbers)
```

اکنون در تابع process تابع NaiveMaxPairWiseProduct با تابع FastMaxPair-wiseProduct جایگزین کنید و حالا این تابع جدید را تست کنید.

این دفعه علاوه بر GradedTest_Performance، تست GradedTest_Correctness هم شکست می خورد. یعنی می خواستیم با این الگوریتم زمان اجرای برنامه را کم کنیم اما حالا الگوریتم ما درست کار نمی کند و جواب اشتباه به ما می دهد. این دفعه مشکل چیست؟ و برای حل آن باید چه کاری انجام داد؟

برای اینکه متوجه شوید که دلیل این مشکل چیست و در چه حالتی این اتفاق رخ می دهد، از stress test استفاده می کنیم.

۴ Stress Testing

اکنون Stress Testing را معرفی می کنیم. یک روش برای تولید هزاران test با هدف پیدا کردن یک test case که راه حل شما در آن ناکام است.

stress test شامل چهار بخش است:

۱. اجرای الگوریتم شما.

۲. یک الگوریتم با آهسته از نظر زمانی اما با ارایه پاسخ صحیح برای یک مشکل مشابه.

۳. یک مولد تست تصادفی.

۴. یک حلقه بی نهایت که در آن تست جدید تولید می شود و در هر دو پیاده سازی الگوریتم به مقایسه نتایج می پردازد. اگر نتایج آنها متفاوت باشد، تست و هر پاسخ هر دو پیاده

سازی خروجی هستند، و برنامه متوقف می شود، در غیر این صورت حلقه تکرار می شود.

ایده stress test این است که دو پیاده سازی صحیح برای یک مسئله باید برای هر test case یک جواب بدهد (در صورتی که پاسخ به این مشکل منحصر به فرد باشد). اگر، با این حال، یکی از پیاده سازی ها نادرست باشد، پس تستی وجود دارد که پاسخ های دو پیاده سازی با هم متفاوت هستند. تنها در یک حالت این طوری نیست و آن زمانی است که برای هر دو پیاده سازی یک اشتباه مشابه وجود داشته باشد که چنین حالتی بعید است (مگر اینکه اشتباه جایی در دستورات ورودی / خروجی است که برای هر دو راه حل مشترک است). در واقع، اگر یک پیاده سازی صحیح باشد و دیگری اشتباه، حتما یک test case وجود دارد که پاسخ این دو پیاده سازی با هم متفاوت باشند. اگر هر دو پاسخ اشتباه بدهند، احتمالا یک تست وجود دارد که دو پیاده سازی نتایج متفاوتی را ارائه می دهند.

اکنون که با stress test آشنا شدید، با استفاده از توضیحات داکیومنت اصلی، برای دو الگوریتم Naive و Fast که در بخش های قبل پیاده سازی کردید؛ یک stress test بنویسید تا متوجه شوید که دنباله ی ورودی به چه صورت که باشد الگوریتم fast جوابی متفاوت از الگوریتم naive می دهد و فکر کنید که برای حل این مشکل چه تغییری باید در الگوریتم fast خود ایجاد کنید تا جواب هر دو الگوریتم یکسان و صحیح باشد.

```
[TestMethod()]
0 | 0 references | 0 changes | 0 authors, 0 changes
public void GradedTest_Stress()
```

۵ Even Faster Algorithm

اکنون شما توانسته اید، به کمک stress test الگوریتم fast خود را درست کنید و تمامی case test ها رو با موفقیت پشت سر بگذارید. توجه کنید چون حلقه ی while ای که در stress test نوشته اید یک حلقه ی بی نهایت است، اکنون این تست تا ابد تمام نخواهد شد. زیرا الگوریتم fast شما درست شده است و دیگر در تمامی test ها جواب یکسانی با الگوریتم naive خواهد داشت. پس برای جلوگیری از این کار شرط حلقه را به گونه ای بگذارید که حلقه برای ۵ ثانیه اجرا شود.

هوراااا اکنون شما توانستید همگی مراحل را با موفقیت بگذرانید. آیا می توانید بگویید مرتبه ی زمانی هر یک از الگوریتم ها naive و fast چیست؟

تمرین اختیاری ۱: چگونه می توان بزرگترین عنصر و دومین بزرگترین عنصر را در یک آرایه با تعداد مقایسه های $1.5n$ پیدا کرد؟ الگوریتم خود را شرح دهید و pseudo code آن را بنویسید.

تمرین اختیاری ۲: چگونه مسئله ی بالا را می توان با تعداد مقایسه های $n + [log2n] - 2$ حل کرد؟ الگوریتم خود را شرح دهید و pseudo code آن را بنویسید.

تمرین اختیاری ۳: اکنون ثابت کنید که مسئله ی تمرین اول را نمی توان کمتر از تعداد مقایسه های $n + [log2n] - 2$ حل کرد.

تمرین اختیاری ۴: سریعترین الگوریتم برای پیدا کردن بزرگترین و دومین بزرگترین و سومین بزرگترین عنصر در یک آرایه چیست؟