دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۸*

نگار زین العابدین
محمد مصطفی رستم خانی
سید صالح اعتمادی

نیم‌سال اول ۰۰-۹۹

ایمیل/تیمز	mo_rostamkhani97@comp.iust.ac.ir n_zeyn@comp.iust.ac.ir
نام شاخه	fb_A8
نام پروژه/پوشه/پول ریکوست	A8
مهلت تحویل	۹۹/۹/۱

*تشکر ویژه از خانم مریم سادات هاشمی که در نیم‌سال اول سال تحصیلی ۹۷-۹۸ نسخه اول این مجموعه تمرین‌ها را تهیه فرمودند. همچنین از اساتید حل‌تمرین نیم‌سال اول سال تحصیلی ۹۹-۹۸ سارا کدیری، محمد مهدی عبداللهپور، مهدی مقدمی، مهسا قادران، علیرضا مرادی، پریسا یل‌سوار، غزاله محمودی و محمدجواد میرشکاری که مستند این مجموعه تمرین‌ها را بهبود بخشیدند، متشکرم.

توضیحات کلی تمرین

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A8 بسازید.

۲. کلاس هر سوال را به پروژهی خود اضافه کنید و در قسمت مربوطه کد خود را بنویسید. هر کلاس شامل دو متد اصلی است:

- متد اول: تابع `Solve` است که شما باید الگوریتم خود را برای حل سوال در این متد پیاده سازی کنید.
- متد دوم: تابع `Process` است که مانند تمرین های قبلی در `TestCommon` پیاده سازی شده است. بنابراین با خیال راحت سوال را حل کنید و نگران تابع `Process` نباشید! زیرا تمامی پیاده سازی ها برای شما انجام شده است و نیازی نیست که شما کدی برای آن بزنید.

۳. اگر برای حل سوالی نیاز به تابع های کمکی دارید؛ می توانید در کلاس مربوط به همان سوال تابع تان را اضافه کنید.

اکنون که پیاده سازی شما به پایان رسیده است، نوبت به تست برنامه می رسد. مراحل زیر را انجام دهید.

۱. یک `UnitTest` برای پروژهی خود بسازید.

۲. فولدر `TestData` که در ضمیمه همین فایل قرار دارد را به پروژهی تست خود اضافه کنید.

۳. فایل `GradedTests.cs` را به پروژهی تستی که ساخته اید اضافه کنید.

توجه:
برای اینکه تست شما از بهینه سازی کامپایلر دات نت حداکثر بهره را ببرد زمان تست ها را روی بیلد **Release** امتحان کنید، درغیر اینصورت ممکن است تست های شما در زمان داده شده پاس نشوند.

۱ پرانتزگذاری

دوست شما در حال ساخت یک ویرایشگر متن برای برنامه نویسان است. او در حال حاضر بر روی پیدا کردن خطاهایی که هنگام استفاده از انواع مختلف براکت ها ایجاد می شود؛ کار می کند. کد می تواند شامل هر براکت از مجموعه [] {} () باشد که که براکت های باز[،{،(هستند و براکت های بسته مربوط به آنها]،}،) هستند.
برای راحتی، ویرایشگر متن نه تنها باید به کاربر اطلاع دهد که یک خطا در استفاده از براکت وجود دارد، بلکه محل دقیق اشتباه در کد را نیز مشخص کند. نخستین اولویت شما این است که اولین براکت بسته که match نشده است را پیدا کنید که دو حالت ممکن است بوجود آید:

۱. یا قبل از آن هیچ براکت باز شده ای وجود ندارد مثل [در [()

۲. یا بستن اشتباه یک براکت باز مانند {در {[]().

اگر چنین اشتباهاتی وجود نداشته باشد، سپس باید اولین براکت بازی که بعد از آن هیچ براکت بسته ای که منطبق با آن نیست را پیدا کنید مانند) در [])){}.
در آخراگر هیچ اشتباهی در کد وجود نداشته باشد، ویرایشگر متن باید کاربر را مطلع کند که از براکت ها به درستی استفاده شده است.
به غیر از براکت ها، کد ممکن است شامل حروف بزرگ، کوچک و لاتین، ارقام و علامت های نگارشی باشد. همه ی براکت های موجود در کد باید به صورت جفت های match شده باشند. براکت ها می توانند تو در تو باشند مانند ([foo[bar]) و یا به صورت جداگانه باشند مانند .f(a،b)-g[c]
دقت کنید که براکت [با براکت]، {با براکت }، و (با براکت) match می شود.
اکنون شما باید به دوست تان برای این که این ویژگی را به ویرایشگر متنش اضافه کند؛ کمک کنید. ورودی شما یک رشته S است که شامل حروف کوچک و بزرگ لاتین، ارقام و علائم نگارشی و مجموعه برکت های [] {} () است.اگر در S از براکت ها به درستی استفاده شده است، خروجی -۱ را بدهد. در غیر این صورت، محل اولین براکت که به درستی match نشده است را در خروجی نمایش دهد برای مثال :

خروجی نمونه	ورودی نمونه
-1	{[]}()

خروجی نمونه	ورودی نمونه
1	{

خروجی نمونه	ورودی نمونه
3	{[}

```
1 reference
public class Q1CheckBrackets : Processor
{
    0 references
    public Q1CheckBrackets(string testDataName) : base(testDataName)
    {}

    0 references
    public override string Process(string inStr) =>
        TestTools.Process(inStr, (Func<string, long>)Solve);

    1 reference
    public long Solve(string str)
    {
        throw new NotImplementedException();
    }
}
```

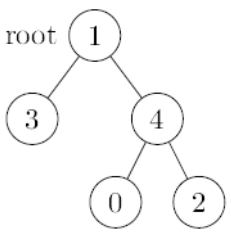
۲ ارتفاع درخت

در این سوال، هدف شما این است که از درخت(tree) استفاده کنید. شما باید یک توصیف درخت را از ورودی بخوانید، ساختار داده ای درخت را بر روی آن اجرا کنید، درخت را ذخیره کرده و عمق یا ارتفاع درخت را محاسبه کنید.

دقت کنید که ارتفاع یا عمق درخت را بیشترین فاصله ی ریشه درخت تا برگ ها تعریف می کنند. تست های این کلاس لزوما درخت دودویی نیست و هر درخت دلخواهی می تواند باشد. در فایل های ورودی اولین خط تعداد گره های درخت یعنی n است و خط بعدی گره های درخت است.خروجی هم یک عدد به عنوان ارتفاع درخت می باشد. برای مثال:

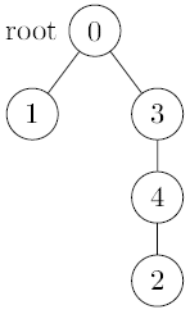
ورودی نمونه	خروجی نمونه
5 4 -1 4 1 1	3

ورودی بدین معناست که ۵ گره با اعداد از ۰ تا ۴ وجود دارد.اولین عدد ۴ است که در جایگاه ۰ قرار گرفته است که به معنی این است که گره ۰ فرزند گره ۴ است. عدد بعدی -۱ است که در جایگاه ۱ قرار دارد یعنی گره ۱ ریشه است. هر جا -۱ دیدید یعنی آن گره ریشه ی درخت است. عدد سوم ۴ است که در جایگاه ۲ قرار دارد یعنی گره ۲ فرزند گره ۴ است به همین ترتیب گره ۳ فرزند گره ۱ و گره ۴ فرزند گره ۱ است. یعنی شکل درخت به صورت زیر خواهد شد:



که ارتفاع آن ۳ می باشد. مثال دیگر:

ورودی نمونه	خروجی نمونه
5 -1 0 4 0 3	4




```
1 reference
public class Q2TreeHeight : Processor
{
    0 references
    public Q2TreeHeight(string testDataName) : base(testDataName)
    {}

    0 references
    public override string Process(string inStr) =>
        TestTools.Process(inStr, (Func<long, long[], long>)Solve);

    1 reference
    public long Solve(long nodeCount, long[] tree)
    {
        throw new NotImplementedException();
    }
}
```

۳ شبیه‌سازی شبکه کامپیوتری

فرض کنید که شما یک سری از بسته های شبکه ای را دریافت کرده اید. وظیفه شما این است که پردازش این بسته ها را شبیه سازی کنید. برای هر بسته شماره ی i، شما می دانید که زمانی که بسته رسیده است A_i است و مدت زمانی که پردازشگر آن را پردازش می کندP_i است. (هر دو در مبنای میلی ثانیه هستند). تنها یک پردازنده وجود دارد و بسته های دریافتی را به ترتیب ورود آنها پردازش می کند. اگر پردازنده شروع به پردازش بسته ای بکند، آن را قطع یا متوقف نمی کند تا پردازش این بسته را پایان دهد، و پردازش بسته i دقیقا P_i میلی ثانیه طول می کشد.

کامپیوتری که بسته ها را پردازش می کند؛ دارای یک بافر شبکه به اندازه ثابت S است. هنگامی که بسته ها وارد می شوند، قبل از پردازش، در بافر ذخیره می شوند.

```
1 reference
public class Q3PacketProcessing : Processor
{
    0 references
    public Q3PacketProcessing(string testDataName) : base(testDataName)
    {}

    0 references
    public override string Process(string inStr) =>
        TestTools.Process(inStr, (Func<long, long[], long[], long[]>)Solve);

    1 reference
    public long[] Solve(long bufferSize,
        long[] arrivalTimes,
        long[] processingTimes)
    {
        throw new NotImplementedException();
    }
}
```

با این حال، اگر بافر در هنگام رسیدن بسته پر باشد (یعنی S بسته در بافر وجود دارد که قبل از این بسته وارد بافر شده اند و کامپیوتر هیچکدام از آنها را هنوز پردازش نکرده است)، آن بسته اصطلاحا drop می شود یعنی اصلا وارد بافر نمی شود بنابراین هیچ پردازشی هم روی آن انجام نخواهد شد.

اگر چندین بسته هم زمان وارد بافر شوند، ابتدا همه در بافر ذخیره می شوند(البته به هر تعدادی که بافر خالی باشد در بافر ذخیره می شود و بقیه بسته ها drop می شود) سپس کامپیوتر شروع به پردازش بسته ها براساس ترتیب زمان وارد شدن آن ها می کند. به محض اینکه پردازش یک بسته تمام شد، سراغ بسته ی بعدی می رود.

اگر در برخی موارد کامپیوتر مشغول نباشد و در بافر هیچ بسته ای وجود نداشته باشد، در این مدت کامپیوتر منتظر بسته ی بعدی می ماند.

توجه داشته باشید که به محض این که پردازش کامپیوتر بر روی یک بسته تمام شود؛ بسته بافر را ترک می کند و فضایی که در بافر اشغال کرده است را آزاد می کند.

خط اول ورودی شامل اندازه بافر S است. خطوط بعدی حاوی دو عدد است. خط i_th حاوی زمان ورود A_i و زمان پردازش P_i (هر دو در میلی ثانیه) بسته i_th است. توجه داشته باشید که زمان ورود بسته ها به صورت صعودی است.

در فایل خروجی هر خط نشان دهنده ی خروجی برای هر بسته است که خروجی برای هر بسته، یا زمانی است که پردازنده شروع به پردازش آن بسته کرده است ، یا در صورت drop شدن بسته -۱ می باشد.(ترتیب خروجی برای بسته ها باید به همان ترتیبی باشد که بسته ها در ورودی داده شده است).

به مثال هایی که برای این سوال در داکیومنت اصلی آمده است، حتما توجه فرمایید.