دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۸*

مبین داریوش همدانی
بابک بهکام کیا
سید صالح اعتمادی

نیم‌سال اول ۱۴۰۱-۱۴۰۰

ایمیل/تیمز	m_dariushhamedani@comp.iust.ac.ir babak_behkamkia@comp.iust.ac.ir
نام شاخه	fb_A8
نام پروژه/پوشه/پول ریکوست	A8
مهلت تحویل	۱۴۰۰/۹/۶

*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 عبداله‌پور، 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. همچنین پروژه تست متناظر آن را ساخته و مطابق راهنمای تمرین یک فایل‌ها را در پوشه متناظر اضافه کرده و تنظیمات مربوط به کپی کردن TestData به پوشه خروجی را در تنظیمات پروژه تست قرار دهید. دقت کنید که پروژه TestCommon فقط یکبار باید در ریشه گیت موجود باشد و نباید در هر تمرین مجدد کپی شود. برای روش ارجاع به این پروژه به تمرین شماره یک مراجعه کنید.

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.

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.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TestCommon;

namespace A8.Tests
{
    [DeploymentItem("TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(300)]
        public void SolveTest_Q1CheckBrackets()
        {
            RunTest(new Q1CheckBrackets("TD1"));
        }

        [TestMethod(), Timeout(1400)]
        public void SolveTest_Q2TreeHeight()
        {
            RunTest(new Q2TreeHeight("TD2"));
        }

        [TestMethod(), Timeout(500)]
        public void SolveTest_Q3PacketProcessing()
        {
            RunTest(new Q3PacketProcessing("TD3"));
        }

        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A8", p.Process, p.TestDataName, p.Verifier,
                VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
                excludedTestCases: p.ExcludedTestCases);
        }
    }
}
```

۱ Check brackets in the code

فرض کنید در ورودی رشته S که مجموعه ای از $bracket$ ها است به شما داده ایم و شما باید چک کنید که آیا ترتیب آنها درست است یا نه.
اگر مشکلی وجود نداشت در خروجی $Success$ چاپ کنید و در غیر این صورت مکان اولین براکت اشتباه را چاپ کنید. (ایندکس از یک شروع می شود)

- $1 \leq |S| \leq 10^5$

ورودی نمونه	خروجی نمونه
`[]`	`Success`

ورودی نمونه	خروجی نمونه
`{}[]`	`Success`

ورودی نمونه	خروجی نمونه
`[()]`	`Success`

ورودی نمونه	خروجی نمونه
`(())`	`Success`

ورودی نمونه	خروجی نمونه
`{[]}()`	`Success`

ورودی نمونه	خروجی نمونه
`{`	`1`

ورودی نمونه	خروجی نمونه
`{[}`	`3`

ورودی نمونه	خروجی نمونه
foo(bar);	Success

ورودی نمونه	خروجی نمونه
foo(bar[i);	10

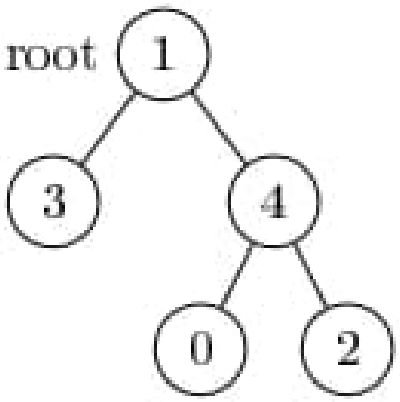
```
using System;
using System.Collections.Generic;
using System.Text;
using TestCommon;

namespace A8
{
    public class Q1CheckBrackets : Processor
    {
        public Q1CheckBrackets(string testDataName) : base(testDataName)
        {}

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

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

۲ Compute tree height

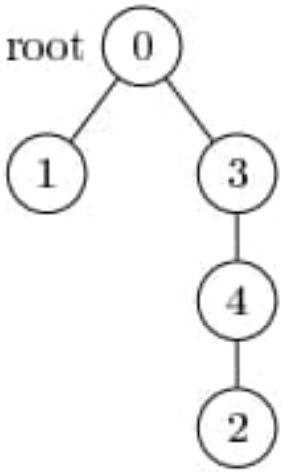
در ورودی به شما یک درخت داده می شود. وظیفه شما این است که ارتفاع این درخت را محاسبه کنید و در خروجی چاپ کنید. در خط اول ورودی به شما عدد n که نشان دهنده تعداد گره ها است، داده می شود. خط بعدی شامل n عدد از -1 تا $n-1$ که هرکدام نشان دهنده والدین آن گره است. اگر عدد i ام برابر -1 باشد، بدین معنی است که گره i ام $root$ این گراف است. تضمین می شود که در این گراف فقط یک $root$ وجود دارد.

- $1 \leq n \leq 10^5$

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



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




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestCommon;

namespace A8
{
    public class Q2TreeHeight : Processor
    {
        public Q2TreeHeight(string testDataName) : base(testDataName)
        {}

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

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

۳ Network packet processing simulation

یک سری بسته های شبکه به شما داده می شوند و وظیفه شما شبیه سازی پردازش آنها است. بسته ها به ترتیب خاصی به دست شما می رسند. برای هر بسته شماره i ، زمان رسیدن آن A_i و زمان پردازش آن توسط پردازنده P_i (هر دو در میلی ثانیه) داده می شود . تنها یک پردازنده وجود دارد و بسته های دریافتی را به ترتیب ورود آنها پردازش می کند. اگر پردازنده شروع به پردازش برخی از بسته‌ها کند، تا زمانی که پردازش این بسته تمام نشود، قطع نمی‌شود یا متوقف نمی‌شود و پردازش بسته i دقیقاً P_i میلی‌ثانیه طول می‌کشد.

کامپیوتری که بسته ها را پردازش می کند دارای یک بافر شبکه با اندازه ثابت S است. وقتی بسته ها می رسند، قبل از پردازش در بافر ذخیره می شوند. با این حال، اگر بافر هنگام ورود بسته پر باشد (بسته‌های S هستند که قبل از این بسته وارد شده‌اند و کامپیوتر پردازش هیچ یک از آنها را تمام نکرده است)، حذف می‌شود و اصلا پردازش نمی‌شود.

اگر چندین بسته به طور همزمان وارد شوند، ابتدا همه آنها در بافر ذخیره می شوند (برخی از آنها ممکن است به همین دلیل حذف شوند – آنهایی که بعداً در ورودی توضیح داده شده است). کامپیوتر بسته ها را به ترتیب رسیدن آنها پردازش می کند و به محض اینکه پردازش بسته قبلی را تمام کرد، پردازش بسته های موجود بعدی از بافر را آغاز می کند. اگر در نقطه ای کامپیوتر مشغول نباشد و هیچ بسته ای در بافر وجود نداشته باشد، کامپیوتر فقط منتظر می ماند تا بسته بعدی برسد. توجه داشته باشید که یک بسته از بافر خارج می شود و به محض اینکه کامپیوتر پردازش آن را تمام کرد، فضای بافر را آزاد می کند.

ورودی: خط اول ورودی شامل اندازه بافر S و تعداد ورودی n است. در هر کدام از n خط بعدی، زمان ورود A_i و زمان پردازش P_i (هر دو بر حسب میلی ثانیه) برای یک بسته آمده است. تضمین شده است که توالی زمان های ورودی کاهشی نیست .

خروجی: اگر بسته حذف شود در خروجی عدد -1 چاپ شود در غیر این صورت زمان شروع پردازش آن بسته را چاپ کنید.

- $1 \leq S \leq 10^5$
- $0 \leq n \leq 10^5$
- $0 \leq A_i \leq 10^6$
- $0 \leq P_i \leq 10^3$
- $1 \leq i \leq n-1$
- $A_i \leq A_i + 1$

خروجی نمونه	ورودی نمونه
	1 0

خروجی نمونه	ورودی نمونه
0	1 1 0 0

خروجی نمونه	ورودی نمونه
0 -1	1 2 0 1 0 1

هر دو بسته در زمان 0 می رسند. اما به دلیل اینکه بافر شبکه دارای اندازه یک است، بسته دوم هیچوقت پردازش نمی شود.

خروجی نمونه	ورودی نمونه
0 1	1 2 0 1 1 1

اولین بسته در زمان 0 می رسد و بلافاصله پردازش می شود. بعد از اتمام پردازش آن در زمان 1 ، دومین بسته می رسد و بدون مشکل پردازش می شود.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestCommon;

namespace A8
{
    public class Q3PacketProcessing : Processor
    {
        public Q3PacketProcessing(string testDataName) : base(testDataName)
        {}

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

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