دانشکده مهندسی کامپیوتر

ساختمان داده

تمرین ۱۱*

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_A11
نام پروژه/پوشه/پول ریکوست	A11
مهلت تحویل	۱۴۰۰/۹/۲۷

*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 `A11` 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.

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

namespace A11.Tests
{
    [DeploymentItem("TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(2000)]
        public void SolveTest_Q1BinaryTreeTraversals()
        {
            RunTest(new Q1BinaryTreeTraversals("TD1"));
        }

        [TestMethod(), Timeout(1500)]
        public void SolveTest_Q2IsItBST()
        {
            RunTest(new Q2IsItBST("TD2"));
        }

        [TestMethod(), Timeout(1500)]
        public void SolveTest_Q3IsItBSTHard()
        {
            RunTest(new Q3IsItBSTHard("TD3"));
        }

        [TestMethod(), Timeout(10000)]
                // change the timeout to 6000 if you could solve it
        public void SolveTest_Q4SetWithRangeSums()
        {
            RunTest(new Q4SetWithRangeSums("TD4"));
        }

        [TestMethod(), Timeout(6000)]
        public void SolveTest_Q5Rope()
        {
            RunTest(new Q5Rope("TD5"));
        }

        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A11", p.Process, p.TestDataName, p.Verifier,
                        VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
                    excludedTestCases: p.ExcludedTestCases);
        }

    }
}
```

۱ Binary tree traversals

در این سوال به شما یک درخت دودویی داده می شود که باید پیمایش های in-order و pre-order و post-order برای این درخت را در خروجی نمایش دهید.
برای حل این سوال می توانید از توضیحات داخل ویدئو های درس استفاده کنید. اما دقت کنید که پیمایش های درخت را نباید بازگشتی پیاده سازی کنید. در این صورت زمان اجرای برنامه ی شما زیاد خواهد شد یا با ارور StackOverFlow مواجه خواهید شد.
گره های درخت دودویی از ایندکس 0 تا $n-1$ شماره گذاری شده اند و گره با ایندکس 0 ریشه ی درخت می باشد. همچنین شماره ی هر خط از فایل ورودی همان ایندکس گره می باشد. یعنی در خط اول از فایل ورودی گره با ایندکس صفر است که همان ریشه درخت نیز می باشد. خط دوم گره با ایندکس یک و خط سوم گره با ایندکس دو و الی آخر.
فرمت هر خط از ورودی به صورت زیر است:

key_i , $left_i$, $right_i$

که در واقع بیان کننده ی این است که $left_i$ و $right_i$ فرزند های چپ و راست گره key_i هستند. اگر گره ی key_i فرزند چپ یا راست نداشته باشد، $left_i$ یا $right_i$ برای این گره برابر -1 خواهد بود. در غیر این صورت ایندکس فرزند های چپ و راست خواهد بود. تمامی ورودی ها یک درخت دودویی می باشند. خروجی سه خط می باشد که فرمت آن به صورت زیر می باشد:

- خط اول پیمایش in-order
- خط دوم پیمایش pre-order
- خط سوم پیمایش post-order

ورودی نمونه	خروجی نمونه
4 1 2 2 3 4 5 -1 -1 1 -1 -1 3 -1 -1	1 2 3 4 5 4 2 1 3 5 1 3 2 5 4

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاسBinary tree traversals قرار دارد، بنویسید.

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

namespace A11
{
    public class Q1BinaryTreeTraversals : Processor
    {
        public Q1BinaryTreeTraversals(string testDataName) : base(testDataName) { }
        public override string Process(string inStr) =>
            TestTools.Process(inStr, (Func<long[][], long[][]>)Solve);

        public long[][] Solve(long[][] nodes)
        {
```

```
            throw new NotImplementedException();
        }
    }
}
```

۲ Is it a binary search tree؟

فرض کنید ساختار داده ای درخت جستجوی دودویی را از کتابخانه های زبان های برنامه نویسی مختلف، به شما داده اند و از شما خواستند که بررسی کنید، آیا این ساختار داده به درستی پیاده سازی شده است یا خیر. در حال حاضر یک برنامه وجود دارد که با عملیات های قرار دادن (Insert)، حذف (remove)، جستجو (search) اعداد صحیح را در این ساختار داده قرار می دهد و یک درخت جستجوی دودویی را می سازد و وضعیت های داخلی این درخت را خروجی می دهد. حالا شما باید تست کنید که آیا درخت ساخته شده یک درخت جستجو دودویی است یا خیر .
تعریف درخت جستجوی دودویی به صورت زیر است:

به ازای هر گره از درخت که مقدار آن x باشد، در این صورت به ازای هر گره ای که در زیر درخت سمت چپ وجود دارد، مقدارش باید کمتر از x باشد و به ازای هر گره ای که در زیر درخت سمت راست وجود دارد مقدارش باید بزرگتر از x باشد. بنابراین شما باید چک کنید که این شرط ها برای درخت جستجوی دودویی داده شده برقرار است یا خیر.

دقت کنید که اگر این شروط را برای هر گره و گره های زیر درختش به صورت بازگشتی چک کنید، بسیار کند خواهد بود و مدت زمان زیادی طول خواهد کشید. بنابراین شما باید یک الگوریتم سریعتر برای حل این سوال پیدا کنید.

تضمین می شود که داده های ورودی همگی یک درخت دودویی هستند یعنی ورودی یک درخت است و هر گره حداکثر دو فرزند دارد.

مانند سوال قبل، گره های درخت دودویی از ایندکس 0 تا $n-1$ شماره گذاری شده اند و گره با ایندکس 0 ریشه ی درخت می باشد. همچنین شماره ی هر خط از فایل ورودی همان ایندکس گره می باشد. یعنی در خط اول از فایل ورودی گره با ایندکس صفر است که همان ریشه درخت نیز می باشد. خط دوم گره با ایندکس یک و خط سوم گره با ایندکس دو و الی آخر.

فرمت هر خط از ورودی به صورت زیر می باشد:

key_i , $left_i$, $right_i$

که در واقع بیان کننده ی این است که$left_i$ و $right_i$ فرزند های چپ و راست گره key_i هستند. اگر گره ی key_i فرزند چپ یا راست نداشته باشد، $left_i$ یا $right_i$ برای این گره برابر -1 خواهد بود. در غیر این صورت ایندکس فرزند های چپ و راست خواهد بود. تمامی ورودی ها یک درخت دودویی می باشند.

مطابق شکل زیر، شما باید الگوریتم خود را در تابع Solve که در کلاس Is it a binary search tree؟ قرار دارد، بنویسید.

ورودی نمونه	خروجی نمونه
2 1 2 1 -1 -1 3 -1 -1	CORRECT

ورودی نمونه	خروجی نمونه
1 1 2 2 -1 -1 3 -1 -1	INCORRECT

ورودی نمونه	خروجی نمونه
0	CORRECT

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

namespace A11
{
    public class Q2IsItBST : Processor
    {
        public Q2IsItBST(string testDataName) : base(testDataName) { }

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

        public bool Solve(long[][] nodes)
        {
            throw new NotImplementedException();
        }
    }
}
```

۳ Is it a binary search tree؟ Hard version

در این سوال شما باید همان سوال قبلی را حل کنید اما در حالت کلی تر. یعنی در درخت جستجوی دودویی ممکن است گره هایی با مقدار های یکسان وجود داشته باشد. بنابراین تعریف درخت جستجوی دودویی به صورت زیر تغییر خواهد یافت:

به ازای هر گره از درخت که مقدار آن x باشد، در این صورت به ازای هر گره ای که در زیر درخت سمت چپ وجود دارد، مقدارش باید کمتر از x باشد و به ازای هر گره ای که در زیر درخت سمت راست وجود دارد مقدارش باید بزرگتر از x یا مساوی آن باشد. یعنی مقدار های مساوی همیشه در زیردرخت راست قرار می گیرند. بنابراین شما باید چک کنید که این شرط ها برای درخت جستجوی دودویی داده شده برقرار است یا خیر.

تضمین می شود که داده های ورودی همگی یک درخت دودویی هستند یعنی ورودی یک درخت است و هر گره حداکثر دو فرزند دارد. گره های درخت دودویی از ایندکس 0 تا $n-1$ شماره گذاری شده اند و گره با ایندکس 0 ریشه ی درخت می باشد. همچنین شماره ی هر خط از فایل ورودی همان ایندکس گره می باشد. یعنی در خط اول از فایل ورودی گره با ایندکس صفر است که همان ریشه درخت نیز می باشد. خط دوم گره با ایندکس یک و خط سوم گره با ایندکس دو و الی آخر.

فرمت هر خط از ورودی به صورت زیر می باشد:

key_i , $left_i$, $right_i$
که در واقع بیان کننده ی این است که$left_i$ و $right_i$ فرزند های چپ و راست گره key_i هستند. اگر گره ی key_i فرزند چپ یا راست نداشته باشد، $left_i$ یا $right_i$ برای این گره برابر -1 خواهد بود. در غیر این صورت ایندکس فرزند های چپ و راست خواهد بود. تمامی ورودی ها یک درخت دودویی می باشند.
اگر درخت داده شده یک درخت جستجوی دودویی باشد خروجی True و در غیر این صورت False می باشد.
برای حل این سوال کافی است الگوریتم سوال قبل را اندکی تغییر دهید.

ورودی نمونه	خروجی نمونه
2 1 2 1 -1 -1 3 -1 -1	CORRECT

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

namespace A11
{
    public class Q3IsItBSTHard : Processor
    {
        public Q3IsItBSTHard(string testDataName) : base(testDataName) { }

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

        public bool Solve(long[][] nodes)
        {
            throw new NotImplementedException();
        }
    }
}
```

۴ Set with range sums

در این سوال شما باید یک ساختار داده ای را پیاده سازی کنید که مجموعه ای از اعداد صحیح S را با عملیات مجاز زیر ذخیره کند:

- add(i) : این بدان معنی است که عدد صحیح i را به مجموعه ی S اضافه کنید. اگر در مجموعه از قبل وجود دارد، مجموعه نباید تغییر کند.
- del(i) : این بدان معنی است که عدد صحیح i را از مجموعه ی S حذف کنید. اگر در مجموعه وجود ندارد، اتفاق خاصی نمی افتد.
- find(i) : این بدان معنی است که بررسی کنید آیا عدد صحیح i در مجموعه ی S وجود دارد یا خیر.

- sum(l,r) : این بدان معنی است که مجموع همه ی المان های v را بدست آورید و خروجی دهید که $l <= v <= r$

فرض کنید که در ابتدا مجموعه ی S خالی است. همچنین اگرعملیات sum انجام شده باشد،x برابر با نتیجه آخرین عملیات sum است. در غیر این صورت برابر صفر خواهد بود. فرمت ورودی به صورت زیر است:

- "+ i" means add((i + x) mod M)
- "- i" means del((i + x) mod M)
- "? i" means find((i + x) mod M)
- "s l r" means sum((l + x) mod M, (r + x) mod M)

در خروجی شما باید نتیجه ی عملیات find و sum را پس بدهید. برای عملیات find اگر عدد $(i + x)modM$ را در مجموعه ی S پیدا کردید باید رشته ی Found و در غیر این صورت Not found را برگردانید. همچنین برای عملیات sum شما باید مجموع مقدار های v را برگردانید که $(l + x)modM <= v <= (r + x)modM$.
توجه کنید که پیاده سازی اولیه برای شما آورده شده است ولی زمان اجرای آن طولانی ست و شما باید پیاده سازی بهتری انجام دهید.

تایم‌اوت پیش‌فرض برای دستیابی به نمره کامل این سوال ۱۰ ثانیه است اما برای کسانی که کد آنها تمامی تست کیس ها را در ۶ ثانیه پاس کند نمره امتیازی در نظر گرفته شده است.

ورودی نمونه	خروجی نمونه
? 1 + 1 ? 1 + 2 s 1 2 + 1000000000 ? 1000000000 - 1000000000 ? 1000000000 s 999999999 1000000000 - 2 ? 2 - 0 + 9 s 0 9	Not found Found 3 Found Not found 1 Not found 10

ورودی نمونه	خروجی نمونه
? 0 + 0 ? 0 - 0 ? 0	Not found Found Not found

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

namespace A11
{
    public class Q4SetWithRangeSums : Processor
    {
        public Q4SetWithRangeSums(string testDataName) : base(testDataName)
        {
            CommandDict =
                        new Dictionary<char, Func<string, string>>()
                        {
                            ['+'] = Add,
                            ['-'] = Del,
                            ['?'] = Find,
                            ['s'] = Sum
                        };
        }

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

        public readonly Dictionary<char, Func<string, string>> CommandDict;

        public const long M = 1_000_000_001;

        public long X = 0;

        protected List<long> Data;

        public string[] Solve(string[] lines)
        {
            X = 0;
            Data = new List<long>();
            List<string> result = new List<string>();
            foreach (var line in lines)
            {
                char cmd = line[0];
                string args = line.Substring(1).Trim();
                var output = CommandDict[cmd](args);
                if (null != output)
                    result.Add(output);
            }
            return result.ToArray();
        }

        private long Convert(long i)
            => i = (i + X) % M;

        private string Add(string arg)
        {
            long i = Convert(long.Parse(arg));
            int idx = Data.BinarySearch(i);
            if (idx < 0)
                Data.Insert(~idx, i);
```

```
            return null;
        }

        private string Del(string arg)
        {
            long i = Convert(long.Parse(arg));
            int idx = Data.BinarySearch(i);
            if (idx >= 0)
                Data.RemoveAt(idx);

            return null;
        }

        private string Find(string arg)
        {
            long i = Convert(int.Parse(arg));
            int idx = Data.BinarySearch(i);
            return idx < 0 ?
                "Not found" : "Found";
        }

        private string Sum(string arg)
        {
            var toks = arg.Split();
            long l = Convert(long.Parse(toks[0]));
            long r = Convert(long.Parse(toks[1]));

            l = Data.BinarySearch(l);
            if (l < 0)
                l = ~l;

            r = Data.BinarySearch(r);
            if (r < 0)
                r = (~r - 1);
            // If not ~r will point to a position with
            // a larger number. So we should not include
            // that position in our search.

            long sum = 0;
            for (int i = (int)l; i <= r && i < Data.Count; i++)
                sum += Data[i];

            X = sum;

            return sum.ToString();
        }
    }
}
```

۵ Rope

در این سوال شما باید ساختار داده rope را پیاده سازی کنید که می تواند یک رشته را ذخیره کند و بخشی از این رشته (زیر رشته) را به طور موثری برش دهد و آن را در جای دیگری از رشته قرار دهد.
این یک سوال بسیار پیشرفته است، سخت تر از تمام سوالات پیشرفته ای که تا به حال در تمرین ها داشتیم، پس هرچه زودتر دست به کار شوید.
فرض کنید که رشته s را به شما داده اند و شما باید n تا query را بر روی این رشته پردازش کنید. هر query توسط سه عدد صحیح توصیف می شود: i ، j و k و به معنای آین است که زیر رشته $s[i...j]$ را از رشته S برش دهید و سپس آن را پس از k امین حرف در رشته باقی مانده درج کنید. اگر k برابر صفر باشد، زیر رشته در ابتدای رشته قرار داده می شود.
در خط اول ورودی رشته ی s قرار دارد و در n خط بعدی query هایی که باید بر روی رشته ی s انجام بدهید، قرار دارد. هر query با سه عدد توصیف می شود که در بالا توضیح داده شد. پس از این که تمام query ها بر روی رشته ی s انجام شد، باید نتیجه را در خروجی چاپ نمایید.

ورودی نمونه	خروجی نمونه
hlelowrold 2 1 1 2 6 6 7	helloworld

ورودی نمونه	خروجی نمونه
abcdef 2 0 1 1 4 5 0	efcabd

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

namespace A11
{
    public class Q5Rope : Processor
    {
        public Q5Rope(string testDataName) : base(testDataName) { }

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

        public string Solve(string text, long[][] queries)
        {
            throw new NotImplementedException();
        }
    }
}
```