دانشکده مهندسی کامپیوتر

طراحی و تحلیل الگوریتم‌ها

تمرین ۱۱*

اساتید حل تمرین: علیرضا مرادی، مرسده ایرانی
تهیه و تنظیم مستند: مریم سادات هاشمی

استاد درس: سید صالح اعتمادی

نیم‌سال دوم ۹۹-۹۸

تلگرام	@Alireza1044 @imercedeh
نام شاخه	fb_A11
نام پروژه/پوشه/پول ریکوست	A11
مهلت تحویل	۱۳۹۹/۳/۱۲

*تشکر ویژه از اساتید حل‌تمرین مریم سادات هاشمی، بنفشه کریمیان، مهسا سادات رضوی، امیر خاکپور، سهیل رستگار و علی آل‌یاسین که در نیم‌سال دوم سال تحصیلی ۹۷-۹۸ نسخه اول این مجموعه تمرین‌ها را تهیه فرمودند.

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 A11;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A11.Tests
{
    [DeploymentItem("TestData", "A11_TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(2000)]
        public void SolveTest_Q1CircuitDesign()
        {
            RunTest(new Q1CircuitDesign("TD1"));
        }

        [TestMethod(), Timeout(4000)]
        public void SolveTest_Q2FunParty()
        {
            RunTest(new Q2FunParty("TD2"));
        }

        [TestMethod(), Timeout(3000)]
        public void SolveTest_Q3SchoolBus()
        {
            RunTest(new Q3SchoolBus("TD3"));
        }

        [TestMethod(), Timeout(4000)]
        public void SolveTest_Q4RescheduleExam()
        {
            RunTest(new Q4RescheduleExam("TD4"));
        }

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

    }
}
```

۱ طراحی مدار مجتمع [۱]

VLSI یا $VeryLarge - ScaleIntegration$ فرآیند ایجاد یک مدار یک‌پارچه با ترکیب هزاران ترانزیستور در یک تراشه واحد است. شما می‌خواهید یک لایه از یک مدار یک‌پارچه را طراحی کنید. شما دقیقا می‌دانید کدام ماژول‌ها در این لایه استفاده می‌شود و کدام یک از آن‌ها باید توسط سیم‌ها متصل شود. سیم‌ها همه در یک لایه قرار می‌گیرند، اما آن‌ها نمی‌توانند با یکدیگر تقاطع داشته باشند. همچنین، هر سیم تنها می‌تواند تنها در یکی از دو جهت چپ یا راست خم شود. اگر دو ماژول را با یک سیم وصل کنید، انتخاب جهت خم شدن سیم، موقعیت سیم را تعیین می‌کند. شما باید موقعیت هر سیم را به گونه‌ای تنظیم کنید که هیچ دو سیمی با یکدیگر متقاطع نباشد.

این مسئله را می‌توان به مسئله‌ی ۲-SAT کاهش داد (یک مورد خاص از مسئله‌ی SAT که در آن هر گزاره شامل دقیقا ۲ متغیر است). هر سیم i را با متغیر باینری x_i نشان می‌دهیم که مقدار ۱ یعنی سیم به سمت راست خم شده‌است و مقدار ۰ یعنی سیم به سمت چپ خم شده‌است. همچنین، برخی از جفت سیم‌ها در برخی مواقع تقاطع دارند. به عنوان مثال، اگر سیم ۱ به سمت چپ خم شود و سیم ۲ به سمت راست خم شود، آن‌ها با یکدیگر متقاطع می‌شوند. ما می‌خواهیم فرمولی بنویسیم که تنها در صورتی که هیچ دو سیمی با یکدیگر متقاطع نباشند، satisfied شود.

در این مورد، عبارت $(x_1\ OR\ \overline{x_2})$ را به فرمول اضافه می‌کنیم که تضمین می‌کند که x_1 (اولین سیم به سمت راست خم شده) درست است یا $\overline{x_2}$ (سیم دوم به سمت چپ خم شده) درست است. بنابراین زمانی که فرمول satisfied است، حالتی که سیم ۱ به سمت چپ خم شده و سیم ۲ به سمت راست خم شده، هرگز اتفاق نمی‌افتد.

پس ما برای هر دو جفت سیم و موقعیت آن‌ها، یک گزاره به فرمول اضافه‌خواهیم کرد؛ اگر این سیم‌ها در هنگام قرار گرفتن در این موقعیت‌ها متقاطع شوند. البته، اگر برخی از جفت سیم‌ها در هر جفت موقعیت احتمالی متقاطع باشند، در این صورت نمی‌توانیم یک مدار طراحی کنیم. وظیفه شما این است که مشخص کنید، آیا ممکن است یک مدار طراحی کنیم و اگر بله، جهت خم شدن برای هر سیم را تعیین کنید.

ورودی یک فرمول ۲-CNF است. خط اول شامل دو عدد صحیح V و C است که به ترتیب تعداد متغیرها و تعداد گزاره‌ها در فرمول است. هر خط بعدی حاوی دو عدد صحیح غیر صفر i و j است که نشان‌دهنده‌ی یک جمله در فرم CNF است. اگر $i > 0$ باشد یعنی x_i و در غیر این صورت اگر $i < 0$ باشد، یعنی $\overline{x_i}$ و همین طور برای j است. به عنوان مثال یک خط "۲ ۳" یک گزاره $(x_3\ \text{OR}\ x_2)$ را نشان می‌دهد و خط "۱ -۴" $(\overline{x_4}\ \text{OR}\ x_1)$ ، خط "-۱ -۳" $(\overline{x_3}\ \text{OR}\ \overline{x_1})$ و خط "۰ ۲ "نمی‌تواند رخ‌دهد، زیرا i و j باید غیر صفر باشند.

اگر فرمول ۲-CNF در ورودی unsatisfied باشد، فقط کلمه "UNSATISFIABLE" را خروجی می‌دهد. اگر فرمول ۲-CNF در ورودی satisfied باشد، کلمه "SATISFIABLE" را در خط اول و انتساب متناظر متغیرها در خط دوم نمایش داده می‌شود. برای هر x_i، خروجی i اگر $x_i = 1$ یا $-i$ اگر $x_i = 0$. به عنوان مثال، اگر یک فرمول توسط انتساب $x_1 = 0$ ، $x_2 = 1$ و $x_3 = 0$ راضی باشد در این صورت خروجی به صورت ۳- ۲ ۱- خواهد بود.

ورودی نمونه	خروجی نمونه
3 3 1 -3 -1 2 -2 -3	SATISFIABLE 1 2 -3

[۱] Integrated Circuit Design

ورودی نمونه	خروجی نمونه
1 2 1 1 -1 -1	UNSATISFIABLE

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using TestCommon;
using Microsoft.SolverFoundation.Solvers;

namespace A11
{
    public class Q1CircuitDesign : Processor
    {
        public Q1CircuitDesign(string testDataName) : base(testDataName) { }

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

        public override Action<string, string> Verifier =>
            TestTools.SatAssignmentVerifier;

        public virtual Tuple<bool, long[]> Solve(long v, long c, long[][] cnf)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```

۲ برنامه‌ریزی یک مهمانی سرگرم‌کننده. [۲]

شما در حال برنامه‌ریزی یک مهمانی برای اعضای یک شرکت هستید. شما می‌خواهید باحال‌ترین افراد را دعوت کنید و برای هر یک از آن‌ها یک فاکتور شوخ‌طبعی تعریف کرده‌اید. هرچه فاکتور شوخ‌طبعی بیشتر باشد، شخص باحال‌تر است. شما می‌خواهید مجموع فاکتور شوخ‌طبعی همه افراد دعوت‌شده را به حداکثر برسانید. با این حال، شما نمی‌توانید همه را دعوت کنید، زیرا اگر رئیس مستقیم افراد دعوت‌شده نیز دعوت شود، ناخوشایند خواهد بود. بنابراین شما باید پیدا کنید که با دعوت کردن چه کسانی از اعضای شرکت مجموع فاکتور شوخ‌طبعی حداکثر می‌شود.

خط اول شامل یک عدد صحیح n است که نشان‌دهنده‌ی تعداد افراد در شرکت است. خط بعدی شامل n شماره f_i فاکتور شوخ‌طبعی هر یک از افراد شرکت است. هر خط بعدی ساختار وابستگی را توصیف می‌کند. هر کس به جز مدیر عامل این شرکت دقیقا یک رئیس مستقیم دارد. هیچ چرخه‌ای وجود ندارد: هیچ کس نمی‌تواند یک رئیس یک رئیس ... از رئیس خود باشد. هر یک از $n-1$ خط دارای دو عدد صحیح u و v است و شما می‌دانید که u رئیس v است یا بالعکس (شما واقعا نمی‌خواهید بدانید که کدام یک رئیس است، اما شما می‌توانید تنها یکی از آن‌ها یا هیچ‌کدام از آن‌ها را دعوت کنید).

[۲] Plan a Fun Party

در خروجی یک عدد به عنوان حداکثر مجموع فاکتور شوخ‌طبعی برمی‌گردانید.

ورودی نمونه	خروجی نمونه
1 1000	1000

ورودی نمونه	خروجی نمونه
2 1 2 1 2	2

ورودی نمونه	خروجی نمونه
5 1 5 3 7 5 5 4 2 3 4 2 1 2	11

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

namespace A11
{
    public class Q2FunParty : Processor
    {
        public Q2FunParty(string testDataName) : base(testDataName) { }

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

        public virtual long Solve(long n, long [] funFactors, long[][] hierarchy)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```

۳ اتوبوس مدرسه [۳]

یک اتوبوس مدرسه باید از صبح زود از انبار شروع کند، همه دانش‌آموزان را از خانه‌هایشان با یک نظم خاص بردارد، همه آن‌ها را به مدرسه برساند و به انبار بازگردد. شما زمان لازم برای رفتن از انبار به هر خانه، از هر خانه به هر خانه دیگر، از هر خانه به مدرسه و از مدرسه به انبار را می‌دانید. شما باید ترتیبی را تعریف کنید که کمترین زمان ممکن برای طی کردن این مسیرها نیاز باشد.

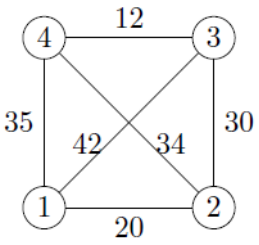
این نمونه‌ای از یک مسئله کامل $NP-complete$ کلاسیک به نام $Problem\ Traveling\ Salesman$ است. در این صورت می‌توانید شما این مسئله را یک گراف در نظر بگیرید که راس‌های آن خانه‌ها، مدرسه و انبار است و وزن یال‌ها زمان لازم برای رفتن از یک رأس به یک راس دیگر است. بعضی از رأس‌ها ممکن است با هیچ راسی در ارتباط نباشد.

خط اول شامل دو عدد صحیح n و m است که به ترتیب نشان‌دهنده‌ی تعداد رأس‌ها و تعداد یال‌ها در گراف است. رأس‌ها از ۱ تا n شماره‌گذاری شده‌اند. هر یک از m خط بعدی حاوی سه عدد صحیح u ، v و t است که نشان‌دهنده‌ی این است که از راس u به راس v می‌توان در زمان t رفت و بالعکس. از یک راس به خودش نیز یالی وجود ندارد.

اگر بتوان از یک راس شروع کرد به طوری‌که از تمامی راس‌های گراف عبور کنیم و هر راس را تنها یک بار ملاقات کنیم، دو خط در خروجی باید برگردانید. خط اول کمترین زمان ممکن برای طی کردن چنین مسیری می‌باشد و خط دوم ترتیب ملاقات راس‌ها در این مدت زمان است. اگر چنین مسیری در گراف وجود نداشته‌باشد، کافیست عدد ۱- را در خروجی برگردانید.

توجه داشته باشید که اگر یک گراف تنها شامل دو راس باشد، می‌توان با یالی که بین این دو راس وجود دارد از یک راس به راس دیگر رفت و دوباره از همان یال این مسیر را برگشت. پس در این حالت مسیر مورد نظر سوال در گراف وجود دارد.

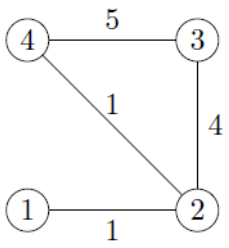
ورودی نمونه	خروجی نمونه
4 6 1 2 20 1 3 42 1 4 35 2 3 30 2 4 34 3 4 12	97 1 4 3 2



شکل ۱: نمونه اول

[۳] School Bus

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



شکل ۲: نمونه اول

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

namespace A11
{
    public class Q3SchoolBus : Processor
    {
        public Q3SchoolBus(string testDataName) : base(testDataName) { }

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

        public override Action<string, string> Verifier { get; set; } =
            TestTools.TSPVerifier;

        public virtual Tuple<long, long[]> Solve(long nodeCount, long[][] edges)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```

۴ تنظیم مجدد امتحانات [۴]

مسئول جدید آموزش دانشکده‌ی علوم کامپیوتر برنامه‌ای برای امتحان درس CS-۱۰۱ آماده‌کرده‌است. در واقع برای هر دانشجو یک تاریخ امتحان مشخص کرده‌است. با این حال، این برنامه‌ریزی دو مشکل اساسی دارد. مشکل اول این است که تاریخ امتحان برخی از دانشجویان که با هم دوست صمیمی هستند، در یک روز افتاده است و مشکل دوم این است که هیچ یک از دانشجویان در تاریخی که برای آن‌ها تعیین شده است ، نمی‌توانند در امتحان شرکت کنند. استادان سه تاریخ متفاوت برای این امتحان در نظر گرفته‌اند و این تاریخ‌ها قابل تغییر نیستند. تنها چیزی که می‌توان تغییر داد؛ اختصاص تاریخ امتحان به هر دانشجو است.

شما مطمئنا می‌دانید که هر دانشجو نمی‌تواند در تاریخی که هم اکنون برای او برنامه‌ریزی شده است، شرکت کند ولی قطعا در دو تاریخ دیگر می‌تواند در امتحان شرکت کند. همچنین، باید اطمینان حاصل کنید که هیچ دو دوست صمیمی تاریخ امتحان یکسانی نداشته باشند. شما باید تعیین کنید که آیا این امکان وجود دارد که برنامه‌ای برای امتحانات دانشجویان در نظر گرفت که این شرایط را داشته باشد یا خیر.

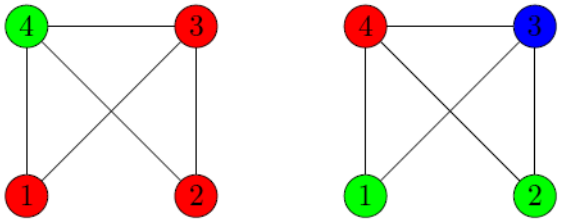
این مسئله را می‌توان به مسئله رنگ کردن گراف با سه رنگ کاهش داد. در این مسئله، یک گراف به شما داده می‌شود و هر رأس را می‌توان با یکی از ۳ رنگ ممکن، رنگ کرد. شما باید به هر رأس یک رنگ اختصاص دهید به طوری‌که متفاوت از رنگ راس‌های متصل به آن باشد. در اینجا رنگ‌ها متناظر با تاریخ امتحان هستند و راس‌ها متناظر با دانشجویان و یال‌ها متناظر با رابطه‌ی دوستی دانشجویان است.

خط اول شامل دو عدد صحیح n و m است که به ترتیب نشان‌دهنده‌ی تعداد رأس‌ها و تعداد لبه‌های گراف است. رأس‌ها از ۱ تا n شماره‌گذاری شده‌اند. خط بعدی شامل یک رشته به طول n است که فقط شامل حروف R ، G و B است که نماینده رنگ‌های فعلی هستند. برای هر موقعیت i در رشته، اگرR باشد یعنی رأس i قرمز رنگ است؛ اگرG باشد یعنی رأس i سبز رنگ است؛ اگرB باشد یعنی رأس i آبی رنگ است. این تخصیص رنگ فعلی است و هر یک از آن‌ها باید تغییر کند. هر یک از m خط بعدی حاوی دو عدد صحیح u و v است که نشان‌دهنده‌ی آن است که رأس u و v توسط یک یال بهم متصل می‌شوند (ممکن است که $v = u$).

اگر این مسئله را نمی‌توان حل کرد کافی است در خروجی "Impossible" را برگردانید و اگر قابل حل است، رشته جدیدی که رنگ هر راس را نشان می‌دهد را در خروجی نمایش دهید.

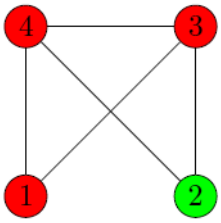
ورودی نمونه	خروجی نمونه
4 5 RRRG 1 3 1 4 3 4 2 4 2 3	GGBR

[۴] Reschedule the Exams



شکل ۳: نمونه اول

ورودی نمونه	خروجی نمونه
4 5 RGRR 1 3 1 4 3 4 2 4 2 3	Impossible



شکل ۴: نمونه دوم

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.SolverFoundation.Solvers;
using TestCommon;

namespace A11
{
    public class Q4RescheduleExam : Processor
    {
        public Q4RescheduleExam(string testDataName) : base(testDataName) { }

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

        public override Action<string, string> Verifier =>
```

```
            TestTools.GraphColorVerifier;


        public virtual char[] Solve(long nodeCount, char[] colors, long[][] edges)
        {
            // write your code here
            throw new NotImplementedException();
        }
    }
}
```