



دانشگاه علم و صنعت ایران

دانشکده مهندسی کامپیوتر

طراحی و تحلیل الگوریتم‌ها

تمرین ۱۱ \*

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

استاد درس: سید صالح اعتمادی

نیم‌سال دوم ۹۸-۹۹

@Alireza1044 @imercedeh	تلگرام
fb_A11	نام شاخه
A11	نام پروژه/پوشه/پول ریکوست
۱۳۹۹/۳/۱۷	مهلت تحویل

\*تشکر ویژه از اساتید حل تمرین 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 بنویسید.
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، در غیر اینصورت ممکن است تست های شما در زمان داده شده پاس نشوند.

```

1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2 using A11;
3 using System;
4 using System.Collections.Generic;
5 using System.Linq;
6 using System.Text;
7 using System.Threading.Tasks;
8 using TestCommon;
9
10 namespace A11.Tests
11 {
12     [DeploymentItem("TestData", "A11_TestData")]
13     [TestClass()]
14     public class GradedTests
15     {
16         [TestMethod(), Timeout(2000)]
17         public void SolveTest_Q1CircuitDesign()
18         {
19             RunTest(new Q1CircuitDesign("TD1"));
20         }
21
22         [TestMethod(), Timeout(4000)]
23         public void SolveTest_Q2FunParty()
24         {
25             RunTest(new Q2FunParty("TD2"));
26         }
27
28         [TestMethod(), Timeout(3000)]
29         public void SolveTest_Q3SchoolBus()
30         {
31             RunTest(new Q3SchoolBus("TD3"));
32         }
33
34         [TestMethod(), Timeout(4000)]
35         public void SolveTest_Q4RescheduleExam()
36         {
37             RunTest(new Q4RescheduleExam("TD4"));
38         }
39
40         public static void RunTest(Processor p)
41         {
42             TestTools.RunLocalTest("A11", p.Process, p.TestDataName,
43                 p.Verifier, VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
44                 excludedTestCases: p.ExcludedTestCases);
45         }
46     }
47 }
48

```

## ۱ طراحی مدار مجتمع<sup>۱</sup>

VLSI یا *VeryLarge – ScaleIntegration* فرآیند ایجاد یک مدار یک پارچه با ترکیب هزاران ترانزیستور در یک تراشه واحد است. شما می‌خواهید یک لایه از یک مدار یک پارچه را طراحی کنید. شما دقیقاً 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 مسئله SAT-2 کاهش داد (یک مورد خاص از مسئله SAT که در آن هر گزاره شامل دقیقاً ۲ متغیر است). هر سیم  $i$  را با متغیر باینری  $x_i$  نشان می‌دهیم که مقدار ۱ یعنی سیم به سمت راست خم شده است و مقدار ۰ یعنی سیم به سمت چپ خم شده است. 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  $(x_1 \text{ OR } \bar{x}_2)$  را به فرمول اضافه می‌کنیم که تضمین می‌کند که  $x_1$  (اولین سیم به سمت راست خم شده) درست است یا  $\bar{x}_2$  (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 اضافه خواهیم 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 2-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  $\bar{x}_i$  و همین طور برای  $j$  است. به عنوان مثال یک خط "۳ ۲" یک گزاره  $(x_3 \text{ OR } x_2)$  را نشان می‌دهد و خط "۱- ۴"  $(\bar{x}_4 \text{ OR } x_1)$ ، خط "۱- ۳"  $(\bar{x}_3 \text{ OR } \bar{x}_1)$  و خط "۲ ۰" نمی‌تواند رخ دهد، زیرا  $i$  و  $j$  باید غیر صفر باشند.

اگر فرمول 2-CNF در ورودی unsatisfied باشد، فقط کلمه "UNSATISFIABLE" را خروجی می‌دهد. اگر فرمول 2-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  $x_i = 0$  باشد، 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.

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

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using TestCommon;
5 using Microsoft.SolverFoundation.Solvers;
6
7 namespace A11
8 {
9     public class Q1CircuitDesign : Processor
10    {
11        public Q1CircuitDesign(string testDataName) : base(testDataName) { }
12
13        public override string Process(string inStr) =>
14            TestTools.Process(inStr,
15                (Func<long, long, long[] [], Tuple<bool, long[]>>)Solve);
16
17        public override Action<string, string> Verifier =>
18            TestTools.SatAssignmentVerifier;
19
20        public virtual Tuple<bool, long[]> Solve(long v, long c, long[] [] cnf)
21        {
22            // write your code here
23            throw new NotImplementedException();
24        }
25    }
26 }

```

## 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 دقیقاً 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).

در خروجی یک عدد به عنوان حداکثر مجموع فاکتور شوخ طبعی برمی گردانید.

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using TestCommon;
7
8 namespace A11
9 {
10     public class Q2FunParty : Processor
11     {
12         public Q2FunParty(string testDataName) : base(testDataName) { }
13
14         public override string Process(string inStr) =>
15             TestTools.Process(inStr, (Func<long,long[],long[][] , long>)Solve);
16
17         public virtual long Solve(long n, long [] funFactors, long[][] hierarchy)
18         {
19             // write your code here
20             throw new NotImplementedException();
21         }
22     }
23 }

```

### 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.

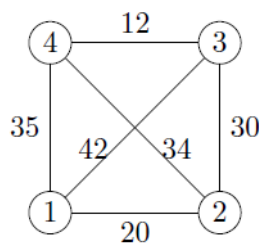
این نمونه‌ای از یک مسئله کامل  $NP - complete$  کلاسیک به نام *Problem Traveling Salesman* است. در این صورت می‌توانید شما این مسئله را یک گراف در نظر بگیرید که رأس‌های آن خانه‌ها، مدرسه و انبار است و وزن یال‌ها زمان لازم برای رفتن از یک رأس به یک رأس دیگر است. بعضی از رأس‌ها ممکن است با هیچ رأسی در ارتباط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

خط اول شامل دو عدد صحیح  $n$  و  $m$  است که به ترتیب نشان‌دهنده 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 نشان‌دهنده این است که از رأس  $u$  به رأس  $v$  می‌توان در زمان  $t$  رفت و بالعکس. از یک رأس به خودش نیز یالی وجود ندارد.

اگر بتوان از یک رأس شروع کرد به طوری که از تمامی رأس‌های گراف عبور کنیم و هر رأس 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 رأس‌ها در این مدت زمان است. اگر چنین مسیری در گراف وجود نداشته‌باشد، کافیت عدد ۱- را در خروجی برگردانید.

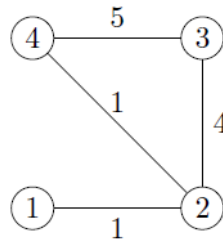
توجه داشته باشید که اگر یک گراف تنها شامل دو رأس باشد، می‌توان با یالی که بین این دو رأس وجود دارد از یک رأس به رأس دیگر رفت و دوباره از همان یال این مسیر را برگشت. پس در این حالت مسیر مورد نظر سوال در گراف وجود دارد.

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

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



شکل ۲: نمونه اول

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using TestCommon;
7
8 namespace A11
9 {
10     public class Q3SchoolBus : Processor
11     {
12         public Q3SchoolBus(string testDataName) : base(testDataName) { }
13
14         public override string Process(string inStr)=>
15             TestTools.Process(inStr, (Func<long, long[][], Tuple<long, long[]>>)Solve);
16
17         public override Action<string, string> Verifier { get; set; } =
18             TestTools.TSPVerifier;
19
20         public virtual Tuple<long, long[]> Solve(long nodeCount, long[][] edges)
21         {
22             // write your code here
23             throw new NotImplementedException();
24         }
25     }
26 }

```



## ۴ تنظیم مجدد امتحانات ۴

مسئول جدید آموزش دانشکده‌ی علوم کامپیوتر برنامه‌ای برای امتحان درس CS-۱۰۱ آماده کرده‌است. 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 مطمئن می‌دانید که هر دانشجو نمی‌تواند در تاریخی که هم اکنون برای او برنامه‌ریزی شده است، شرکت کند ولی قطعاً 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.

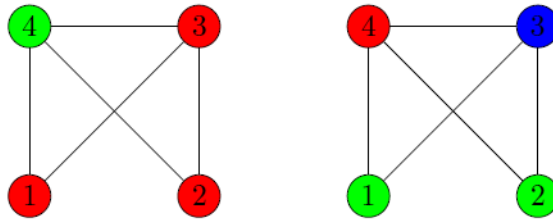
این مسئله را می‌توان به مسئله رنگ کردن گراف با سه رنگ کاهش داد. در این مسئله، یک گراف به شما داده می‌شود و هر رأس را می‌توان با یکی از ۳ رنگ ممکن، رنگ کرد. شما باید به هر رأس یک رنگ اختصاص دهید به طوری که متفاوت از رنگ راس‌های متصل به آن باشد. در اینجا رنگ‌ها متناظر با تاریخ امتحان هستند و راس‌ها متناظر 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، اگر  $R$  باشد یعنی رأس  $i$  قرمز رنگ است؛ اگر  $G$  باشد یعنی رأس  $i$  سبز رنگ است؛ اگر  $B$  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.

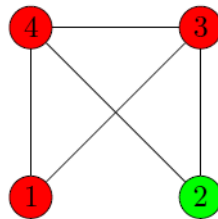
ورودی نمونه	خروجی نمونه
4 5 RRRG 1 3 1 4 3 4 2 4 2 3	GGBR

Reschedule the Exams<sup>۴</sup>



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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using Microsoft.SolverFoundation.Solvers;
7 using TestCommon;
8
9 namespace A11
10 {
11     public class Q4RescheduleExam : Processor
12     {
13         public Q4RescheduleExam(string testDataName) : base(testDataName) { }
14
15         public override string Process(string inStr) =>
16             TestTools.Process(inStr, (Func<long, char[], long[][], char[]>) Solve);
17
18         public override Action<string, string> Verifier =>

```

```
19         TestTools.GraphColorVerifier;
20
21
22     public virtual char[] Solve(long nodeCount, char[] colors, long[][] edges)
23     {
24         // write your code here
25         throw new NotImplementedException();
26     }
27 }
28 }
```