



دانشگاه علم و صنعت ایران

دانشکده مهندسی کامپیوتر
درس ساختمان داده
تمرین ۱۲ *

محمد مهدی عبدالله پور

سارا کدیری

سید صالح اعتمادی

نیم سال اول ۹۹-۹۸

@m9m8a	تلگرام
@Codeiry	
fb_A12	نام شاخه
A12	نام پروژه/پوشه/پول ریکوست
۹۸/۱۰/۱۴	مهلت تحویل

توضیحات کلی تمرین

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A12 بسازید.
۲. کلاس هر سوال را به پروژه 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 پروژه خود بسازید.
 ۲. فولدر TestData که در ضمیمه همین فایل قرار دارد را به پروژه تست خود اضافه کنید.
 ۳. فایل GradedTests.cs را به پروژه 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.

در این سری از تمرین، علاوه بر فایل ها با پسوند txt که تست کیس های سوالات بودند و شما از آن ها برای تست کدتان استفاده می کردید، فایل هایی با پسوند webgraphviz نیز در پوشه TestDat وجود دارد. شما با استفاده از این فایل ها می توانید گراف های کوچکتر از ۱۰۰ گره را به صورت تصویری در این سایت مشاهده کنید.

```

1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2 using TestCommon;
3
4 namespace A12.Tests
5 {
6     [DeploymentItem("TestData")]
7     [TestClass()]
8     public class GradedTests
9     {
10         [TestMethod(), Timeout(100)]
11         public void SolveTest_Q1MazeExit()
12         {
13             RunTest(new Q1MazeExit("TD1"));
14         }
15
16         [TestMethod(), Timeout(4000)]
17         public void SolveTest_Q2AddExitToMaze()
18         {
19             RunTest(new Q2AddExitToMaze("TD2"));
20         }
21
22         [TestMethod(), Timeout(300)]
23         public void SolveTest_Q3Acyclic()
24         {
25             RunTest(new Q3Acyclic("TD3"));
26         }
27
28         [TestMethod(), Timeout(10000)]
29         public void SolveTest_Q4OrderOfCourse()
30         {
31             RunTest(new Q4OrderOfCourse("TD4"));
32         }
33
34         [TestMethod(), Timeout(500)]
35         public void SolveTest_Q5StronglyConnected()
36         {
37             RunTest(new Q5StronglyConnected("TD5"));
38         }
39
40         public static void RunTest(Processor p)
41         {
42             TestTools.RunLocalTest("A12", p.Process, p.TestDataName, p.Verifier,
43                 VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
44                 excludedTestCases: p.ExcludedTestCases);
45         }
46     }
47 }

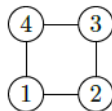
```

۱ پیدا کردن راه خروجی^۱

Maze یک شبکه مستطیل شکل از سلول ها و خانه هاست. در این سوال شما باید بررسی کنید که با شروع از یک نقطه‌ی خاص از maze آیا مسیری برای خروج از آن وجود دارد یا خیر. برای این منظور، می‌توانید maze را یک گراف غیر جهت دار در نظر بگیرید که خانه‌های maze رأس این گراف هستند. اگر دو خانه با هم مجاور باشند و هیچ دیواری بین آن‌ها نباشند، با یک یال غیر جهت دار به هم متصل می‌شوند. بنابراین، برای بررسی این که آیا بین دو خانه داده شده در maze مسیری وجود دارد یا خیر، کافی است، چک کنیم که بین دو رأس متناظر در گراف مسیر وجود دارد یا خیر.

خط اول فایل ورودی، تعداد گره‌های گراف (یعنی n) را مشخص می‌کند. هر یک از n خط بعدی، شامل دو گره است که بدین معنی است که این دو گره توسط یک یال بهم متصل شدند. در خط آخر هم گره‌های u و v قرار دارد که الگوریتم شما باید چک کند آیا در گراف داده شده مسیری از گرهی u به گرهی v وجود دارد یا خیر. اگر مسیری وجود داشت خروجی الگوریتم یک می‌باشد و در غیر این صورت صفر.

خروجی نمونه	ورودی نمونه
1	4 1 2 3 2 4 3 1 4 1 4



همانطور که در شکل بالا معلوم است دو مسیر از گره ۱ به گره ۴ وجود دارد: ۱-۲-۳-۴ و ۱-۴.

```

1 using System;
2 using System.Collections.Generic;
3 using TestCommon;
4
5 namespace A12
6 {
7     public class Q1MazeExit : Processor
8     {
9         public Q1MazeExit(string testDataName) : base(testDataName) { }
10
11         public override string Process(string inStr) =>
12             TestTools.Process(inStr, (Func<long, long[][], long, long, long>)Solve);
13
14         public long Solve(long nodeCount, long[][] edges, long StartNode, long EndNode)
15         {
16             throw new NotImplementedException();
17         }
18     }
19 }

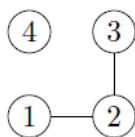
```

۲ اضافه کردن راه خروج

فرضیات سوال قبل را در نظر بگیرید. در این سوال شما باید مطمئن شوید که در یک maze هیچ منطقه‌ی مرده‌ای وجود ندارد. یعنی حداقل یک مسیر خروج از هر خانه وجود دارد که ما را به بیرون از Maze می‌رساند. برای این کار، اجزای متصل در گراف غیر جهت دار را پیدا کنید و اطمینان حاصل کنید که در هر جز متصل، یک راه خروج به بیرون از maze وجود دارد. در این سوال یک گراف با n گره به شما داده می‌شود و شما باید تعداد اجزای متصل این گراف را پیدا کنید.

خط اول فایل ورودی، تعداد گره‌های گراف (یعنی n) را مشخص می‌کند. هر یک از n خط بعدی، شامل دو گره است که بدین معنی است که این دو گره توسط یک یال بهم متصل شدند. در فایل خروجی هم یک عدد که نشان دهنده‌ی تعداد اجزای متصل گراف است، می‌باشد.

ورودی نمونه	خروجی نمونه
4 1 2 3 2	2



همان طور که در شکل بالا مشخص است، تعداد اجزای متصل در این گراف دو تا می‌باشد.

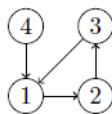
```
1 using System;
2 using System.Collections.Generic;
3 using TestCommon;
4
5 namespace A12
6 {
7     public class Q2AddExitToMaze : Processor
8     {
9         public Q2AddExitToMaze(string testDataName) : base(testDataName) { }
10
11         public override string Process(string inStr) =>
12             TestTools.Process(inStr, (Func<long, long[][], long>)Solve);
13
14         public long Solve(long nodeCount, long[][] edges)
15         {
16             throw new NotImplementedException();
17         }
18     }
19 }
```

۳ بررسی چارت درسی رشته کامپیوتر^۳

فرض کنید که برنامه درسی رشته علوم کامپیوتر را به شما داده اند که لیستی از درس های این رشته است که پیش نیاز هر درس را هم مشخص می کند. وظیفه شما این است که در این لیست چک کنید که وابستگی درس ها بهم ایجاد دور یا cycle نمی کند. بدین منظور، یک گراف جهت دار در نظر بگیرید که گره های این گراف درس ها هستند و یال ها وابستگی درس ها به یکدیگر را نشان می دهند. یعنی اگر درس u پیش نیاز درس v باشد، از راس u به راس v یک یال جهتدار در گراف متناظر وجود دارد. پس از ساخت چنین گرافی شما کافی است چک کنید که در این گراف دور وجود نداشته باشد.

خط اول ورودی، تعداد گره های گراف را مشخص می کند و در هر یک از خط های بعدی، دو راس وجود دارد که نشان دهنده وجود یک یال جهتدار از راس اول به راس دوم می باشد. اگر در گراف دور وجود داشته باشد، خروجی ۱ و در غیر این صورت صفر می باشد.

خروجی نمونه	ورودی نمونه
1	4 1 2 4 1 2 3 3 1



همانطور که در شکل بالا معلوم است، دور ۳۱۲۳ در گراف وجود دارد. بنابراین خروجی یک خواهد بود.

```

1 using System;
2 using System.Collections.Generic;
3 using TestCommon;
4
5 namespace A12
6 {
7     public class Q3Acyclic : Processor
8     {
9         public Q3Acyclic(string testDataName) : base(testDataName) { }
10
11         public override string Process(string inStr) =>
12             TestTools.Process(inStr, (Func<long, long[][], long>)Solve);
13
14         public long Solve(long nodeCount, long[][] edges)
15         {
16             throw new NotImplementedException();
17         }
18     }
19 }

```

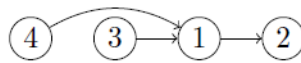
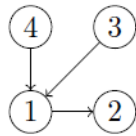
۴ تعیین یک برنامه درسی معتبر^۴

حال که مطمئن شدید که در برنامه‌ی درسی علوم کامپیوتر، وابستگی‌های چرخه‌ای وجود ندارد، وظیفه‌ی شما این است که ترتیبی را برای اخذ درس‌ها، با رعایت پیش‌نیازها پیشنهاد دهید که تمامی درس‌ها در این ترتیب وجود داشته باشد. برای این کار کافی است همان گراف جهت‌دار سوال قبل را در نظر بگیرید و یکی از توپولوژی‌های این گراف را پیدا کنید.

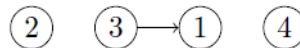
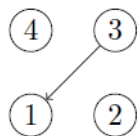
خط اول ورودی، تعداد گره‌های گراف را مشخص می‌کند و در هر یک از خط‌های بعدی، دو راس وجود دارد که نشان‌دهنده‌ی وجود یک یال جهت‌دار از راس اول به راس دوم می‌باشد. تمامی گراف‌های تست کیس‌ها یک گراف جهت‌دار بدون دور هستند (DAG). در فایل خروجی هم، توپولوژی گراف داده شده، می‌باشد.

برای این سوال جواب ارائه شده فقط یک راه حل معتبر است. کد شما با متدی که داخل کلاس Q4 پیاده سازی شده راست آزمایی می‌شود. لطفاً این کد را تغییر ندهید

ورودی نمونه	خروجی نمونه
4 1 2 4 1 3 1	4 3 1 2



ورودی نمونه	خروجی نمونه
4 3 1	2 3 1 4



```

1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Text;
6 using TestCommon;
7
8 namespace A12
9 {
10     public class Q4OrderOfCourse: Processor
11     {
12         public Q4OrderOfCourse(string testDataName) : base(testDataName) { }
13
14         public override string Process(string inStr) =>
15             TestTools.Process(inStr, (Func<long, long[][], long[]>)Solve);
16
17         public long[] Solve(long nodeCount, long[][] edges)
18         {
19             throw new NotImplementedException();
20         }
21
22         public override Action<string, string> Verifier { get; set; } = TopSortVerifier;
23
24         public static void TopSortVerifier(string inFileName, string strResult)
25         {
26             long[] topOrder = strResult.Split(TestTools.IgnoreChars)
27                 .Select(x => long.Parse(x)).ToArray();
28
29             long count;
30             long[][] edges;
31             TestTools.ParseGraph(File.ReadAllText(inFileName), out count, out edges);
32
33             // Build an array for looking up the position of each node in topological order
34             // for example if topological order is 2 3 4 1, topOrderPositions[2] = 0,
35             // because 2 is first in topological order.
36             long[] topOrderPositions = new long[count];
37             for (int i = 0; i < topOrder.Length; i++)
38                 topOrderPositions[topOrder[i] - 1] = i;
39             // Top Order nodes is 1 based (not zero based).
40
41             // Make sure all direct dependencies (edges) of the graph are met:
42             // For all directed edges u -> v, u appears before v in the list
43             foreach (var edge in edges)
44                 if (topOrderPositions[edge[0] - 1] >= topOrderPositions[edge[1] - 1])
45                     throw new InvalidDataException(
46                         $"{Path.GetFileName(inFileName)}: " +
47                         $"{edge[0]}->{edge[1]}");
48
49         }
50     }
51 }

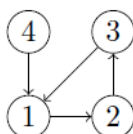
```


۵ بررسی دسترسی به تمام چهارراه‌های یک شهر^۵

فرض کنید که اداره پلیس یک شهر همه خیابان‌ها را یک طرفه کرده است. شما باید بررسی کنید که آیا هنوز هم می‌توان به صورت قانونی از هر تقاطع به هر تقاطع دیگر رفت و رانندگی کرد. بدین منظور، یک گراف جهت دار ایجاد کنید که رأسها تقاطع باشند و هر زمان که یک خیابان (یک طرفه) از u تا v در شهر وجود داشته باشد یک یال بین رأس‌های u و v باشد. سپس، کافی است که بررسی کنید که آیا تمام رأس‌های گراف در یک مولفه قویا همبند^۶ هستند یا خیر.

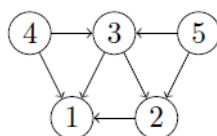
خط اول ورودی، تعداد گره‌های گراف را مشخص می‌کند و در هر یک از خط‌های بعدی، دو رأس وجود دارد که نشان دهنده وجود یک یال جهت‌دار از رأس اول به رأس دوم می‌باشد. در فایل خروجی هم تعداد مولفه‌های قویا همبند در گراف داده شده است.

ورودی نمونه	خروجی نمونه
4 1 2 4 1 2 3 3 1	2



همانطور که در شکل بالا مشاهده می‌کنید، این گراف دارای دو مولفه قویا همبند می‌باشد: (۱ و ۲ و ۳) - (۴)

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



همانطور که در شکل بالا مشاهده می کنید، این گراف دارای ۵ مولفه قویا همبند می باشد: (۱) - (۲) - (۳) - (۴) - (۵)

```
1 using System;
2 using System.Collections.Generic;
3 using TestCommon;
4
5 namespace A12
6 {
7     public class Q5StronglyConnected: Processor
8     {
9         public Q5StronglyConnected(string testDataName) : base(testDataName) { }
10
11         public override string Process(string inStr) =>
12             TestTools.Process(inStr, (Func<long, long[][], long>)Solve);
13
14         public long Solve(long nodeCount, long[][] edges)
15         {
16             throw new NotImplementedException();
17         }
18     }
```