دانشکده مهندسی کامپیوتر

طراحی و تحلیل الگوریتم‌ها

تمرین ۴*

اساتید حل تمرین: یاسمین لطف اللهی، سهند نظرزاده
تهیه و تنظیم مستند: مریم سادات هاشمی

استاد درس: سید صالح اعتمادی

نیم‌سال دوم ۱۴۰۰-۱۳۹۹

تلگرام	@Ysmn_ltf @SahandNZ
نام شاخه	fb_A4
نام پروژه/پوشه/پول ریکوست	A4
مهلت تحویل	۱۴۰۰/۰۱/۲۸

*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 تمرین‌های قبل، یک پروژه به نام A4 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 پیاده‌سازی کنید.
- متد دوم: تابع Process است که مانند تمرین‌های قبلی در TestCommon پیاده‌سازی شده است. بنابراین با خیال راحت سوال را حل کنید و نگران تابع Process نباشید! زیرا تمامی پیاده‌سازی‌ها برای شما انجام شده است و نیازی نیست که شما کدی برای آن بزنید.

۳. اگر برای حل سوالی نیاز به تابع‌های کمکی دارید؛ می‌توانید در کلاس مربوط به همان سوال تابع‌تان را اضافه کنید.

اکنون که پیاده‌سازی شما به پایان رسیده است، نوبت به تست برنامه می‌رسد. 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 ساخته‌اید اضافه کنید.

توجه:
برای این‌که تست شما از بهینه‌سازی کامپایلر دات نت حداکثر بهره را ببرد زمان تست‌ها را روی بیلد Release امتحان کنید، درغیر این‌صورت ممکن است تست‌های شما در زمان داده شده پاس نشوند.

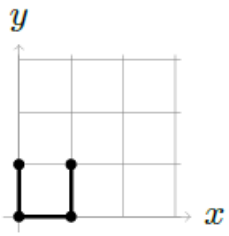
```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using A4;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestCommon;

namespace A4.Tests
{
    [DeploymentItem("TestData", "A4_TestData")]
    [TestClass()]
    public class GradedTests
    {
        [TestMethod(), Timeout(2000)]
        public void SolveTest_Q1BuildingRoads()
        {
            RunTest(new Q1BuildingRoads("TD1"));
        }

        [TestMethod(), Timeout(4000)]
        public void SolveTest_Q2Clustering()
        {
            RunTest(new Q2Clustering("TD2"));
        }


        [TestMethod(), Timeout(30000)]
        public void SolveTest_Q3ComputeDistance()
        {
            RunTest(new Q3ComputeDistance("TD3"));
        }

        public static void RunTest(Processor p)
        {
            TestTools.RunLocalTest("A4", p.Process, p.TestDataName, p.Verifier,
                VerifyResultWithoutOrder: p.VerifyResultWithoutOrder,
                excludedTestCases: p.ExcludedTestCases);
        }
    }
}
```

۱ ساخت جاده‌ها برای اتصال شهرها [۱]

در این مسئله، هدف ساختن جاده بین شهرهاست به طوری که بین هر دو شهر مسیری وجود داشته باشد و طول کل مسیرها کمینه باشد. به یاد داشته باشید که طول پاره‌خط بین دو نقطه‌ی $(x_1، y_1)$ و $(x_2، y_2)$ برابر است با $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

خط اول فایل ورودی شامل n - تعداد نقاط - است.هر کدام از n خط بعدی نقطه‌ی $(x_i، y_i)$ را نشان می‌دهد. در خروجی باید کمینه طول کل مسیرها را چاپ کنید. جواب نهایی خود را به ۶ رقم اعشار گرد کنید.

ورودی نمونه	خروجی نمونه
4 0 0 0 1 1 0 1 1	3.000000000



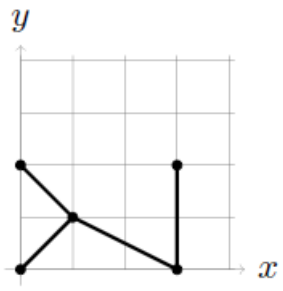
شکل ۱: نمونه اول

یک راه بهینه برای اتصال این چهار نقطه را در شکل ۱ مشاهده می‌کنید. دقت کنید که راه‌های دیگری هم برای اتصال این نقاط با پاره‌خط‌هایی که در مجموع طول ۳ داشته باشند، وجود دارد.

ورودی نمونه	خروجی نمونه
5 0 0 0 2 1 1 3 0 3 2	7.064495102

یک راه بهینه برای اتصال این پنج نقطه را در شکل ۲ مشاهده می‌کنید. در اینجا طول کل برابر است با $2\sqrt{2} + \sqrt{5} + 2$

[۱] Building Roads to Connect Cities



شکل ۲: نمونه دوم

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

namespace A4
{
    public class Q1BuildingRoads : Processor
    {
        public Q1BuildingRoads(string testDataName) : base(testDataName) { }

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

        public double Solve(long pointCount, long[][] points)
        {
            throw new NotImplementedException();
        }
    }
}
```

راهنمایی: برای حل این سوال میتوانید الگوریتم های Kruskal یا Prim را پیاده سازی کنید.

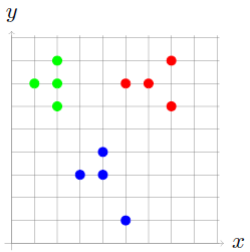
۲ خوشه‌بندی [۲]

خوشه‌بندی یک مسئله‌ی بنیادی در داده‌کاوی است. هدف خوشه‌بندی تقسیم یک مجموعه از اشیا به زیرمجموعه‌هایی (یا خوشه‌هایی) است که اعضای هر زیرمجموعه نزدیک (شبیه) یکدیگر باشند؛ در حالی که اعضای زیرمجموعه‌های متفاوت از یکدیگر دور باشند. حال با دریافت n نقطه در صفحه و عدد صحیح k بزرگ‌ترین مقدار d را حساب کنید به طوری که نقاط داده شده بتوانند به k زیرمجموعه‌ی ناتهی تقسیم شوند، به طوری که فاصله‌ی بین هر دو نقطه از زیرمجموعه‌های متفاوت حداقل d باشند.

در خط اول n که تعداد نقاط است داده می‌شود. هرکدام از n خط بعد نقطه‌ی (y_i, x_i) را نشان می‌دهد. خط آخر شامل k – تعداد خوشه‌ها – است. در خروجی، بزرگ‌ترین مقدار ممکن d را چاپ کنید. جواب نهایی خود را به ۶ رقم اعشار گرد کنید.

[۲] Clustering

ورودی نمونه	خروجی نمونه
12 7 6 4 3 5 1 1 7 2 7 5 7 3 3 7 8 2 8 4 4 6 7 2 6 3	2.828427124746



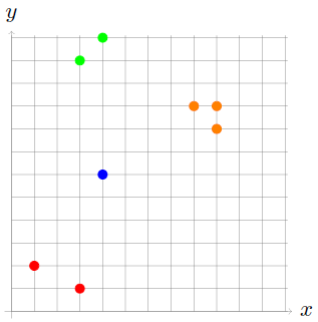
شکل ۳: نمونه اول

جواب $\sqrt{8}$ است. دسته‌بندی مجموعه نقاط داده شده در سه خوشه، در شکل ۳ نشان داده شده است.

ورودی نمونه	خروجی نمونه
8 3 1 1 2 4 6 9 8 9 9 8 9 3 11 4 12 4	5.000000000

جواب ۵ است. دسته‌بندی مجموعه نقاط داده شده در چهار خوشه، در شکل ۴ نشان داده شده است.

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



شکل ۴: نمونه دوم

```
using System.Text;
using System.Threading.Tasks;
using TestCommon;
using static A4.Q1BuildingRoads;

namespace A4
{
    public class Q2Clustering : Processor
    {
        public Q2Clustering(string testDataName) : base(testDataName) { }

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

        public double Solve(long pointCount, long[][] points, long clusterCount)
        {
            throw new NotImplementedException();
        }
    }
}
```

راهنمایی: از راه حل خود در سوال اول برای حل این سوال کمک بگیرید.

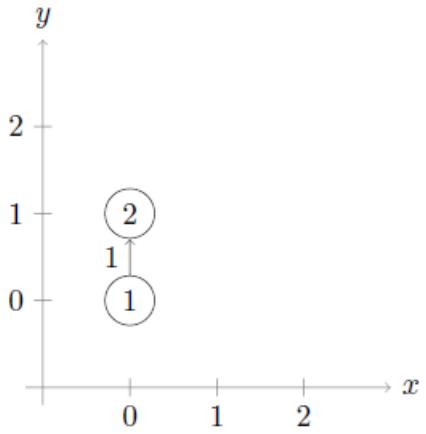
۳ محاسبه سریع تر فاصله با استفاده از مختصات [۳]

در این مسئله به شما توصیفی از یک شبکه‌ی واقعی از جاده‌ها، نه تنها با یال‌ها و طولشان بلکه همراه با مختصات راس‌ها داده می‌شود. شما همچنان باید فاصله‌ی بین جفت‌هایی از راس‌ها را پیدا کنید اما همزمان باید از اطلاعات اضافه راجع به مختصات راس‌ها استفاده کنید و سرعت جستجوی خود را بالا ببرید.

در خط اول ورودی دو عدد صحیح n و m – تعداد راس‌ها و یال‌های موجود در شبکه – داده می‌شود. راس‌ها از ۱ تا n شماره‌گذاری می‌شوند. در هر کدام از n خط بعد مختصات x و y راس متناظر داده شده است. در هر کدام از m خط بعدی شامل سه عدد u و v و l است که نشان می‌دهد یالی جهت‌دار با طول l از راس u به راس v وجود دارد. تضمین می‌شود که طول l بزرگتر مساوی فاصله‌ی دکارتی بین دو راس u و v است. در خط بعدی عدد صحیح q داده می‌شود که تعداد سوال‌هاست. هر کدام از q خط بعد شامل دو عدد صحیح u و v است که شماره‌ی دو راسی است که باید فاصله‌ی u به v را حساب کنید.

[۳] Compute Distance Faster Using Coordinates

ورودی نمونه	خروجی نمونه
2 1 0 0 0 1 1 2 1 4 1 1 2 2 1 2 2 1	0 0 1 -1

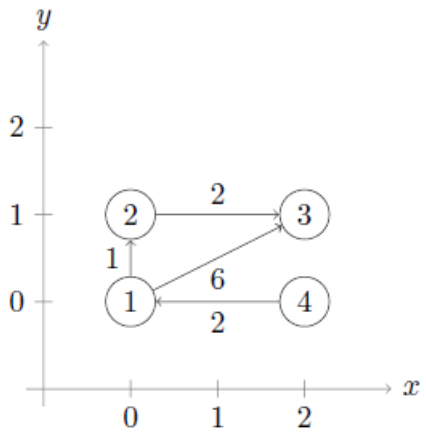


شکل ۵: نمونه اول

فاصله‌ی یک راس تا خودش همواره صفر است. با توجه به شکل ۵ فاصله‌ی ۱ به ۲ برابر با یک است. مسیری از ۲ به ۱ وجود ندارد.

ورودی نمونه	خروجی نمونه
4 4 0 0 0 1 2 1 2 0 1 2 1 4 1 2 2 3 2 1 3 6 1 1 3	3

با توجه به شکل ۶ یالی مستقیم از راس ۱ به راس ۳ با طول ۶ وجود دارد. اما مسیر کوتاه تر ۱ ← ۲ ← ۳ با طول $1 + 2 = 3$ وجود دارد.



شکل ۶: نمونه دوم

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

namespace A4
{
    public class Q3ComputeDistance : Processor
    {
        public Q3ComputeDistance(string testDataName) : base(testDataName) { }

        public static readonly char[] IgnoreChars = new char[] { '\n', '\r', ' ' };
        public static readonly char[] NewLineChars = new char[] { '\n', '\r' };
        private static double[][] ReadTree(IEnumerable<string> lines)
        {
            return lines.Select(line =>
                line.Split(IgnoreChars, StringSplitOptions.RemoveEmptyEntries)
                                    .Select(n => double.Parse(n)).ToArray()
                            ).ToArray();
        }
        public override string Process(string inStr)
        {
            return Process(inStr, (Func<long, long, double[][], double[][], long,
                                    long[][], double[]>)Solve);
        }
        public static string Process(string inStr, Func<long, long, double[][]
                                    ,double[][], long, long[][], double[]> processor)
        {
           var lines = inStr.Split(NewLineChars, StringSplitOptions.RemoveEmptyEntries);
           long[] count = lines.First().Split(IgnoreChars,
                                                StringSplitOptions.RemoveEmptyEntries)
                                            .Select(n => long.Parse(n))
                                            .ToArray();
            double[][] points = ReadTree(lines.Skip(1).Take((int)count[0]));
            double[][] edges = ReadTree(lines.Skip(1 + (int)count[0]).Take((int)count[1]));
```

```
            long queryCount = long.Parse(lines.Skip(1 + (int)count[0] + (int)count[1])
                                             .Take(1).FirstOrDefault());
            long[][] queries = ReadTree(lines.Skip(2 + (int)count[0] + (int)count[1]))
                                           .Select(x => x.Select(z => (long)z).ToArray())
                                           .ToArray();

            return string.Join("\n", processor(count[0], count[1], points, edges,
                                queryCount, queries));
        }
        public double[] Solve(long nodeCount,
                              long edgeCount,
                              double[][] points,
                              double[][] edges,
                              long queriesCount,
                              long[][] queries)
        {
            throw new NotImplementedException();
        }
    }
}
```

راهنمایی: الگوریتم*A را برای حل این سوال پیاده سازی کنید.

یک نکته ی مهم!

سه تست‌کیس آخر این سوال با استفاده از داده‌های واقعی بخشی از جاده‌های NewYork و SanFrancisco و Colorado تهیه شده‌اند. به همین دلیل از مختصات جغرافیایی برای مشخص کردن هر گره استفاده شده است، در نتیجه برای پیاده سازی تابع پتانسیل باید ابتدا پکیج GeoCoordinate.NetCore را از طریق Nuget نصب و طبق راهنمای ارائه شده از آن استفاده کنید. توجه کنید به دلیل تعداد زیاد گره‌ها و یال‌ها در سه تست کیس آخر، درصورتی که کد شما بهینه نباشد ممکن است به خطای SystemOutOfMemoryException برخورد کنید. دقت کنید که در تست کیس‌ها چنانچه از مختصات جغرافیایی استفاده شده باشد (۳ تست کیس آخر) مختصات حتما یک عدد اعشاری خواهد بود و در غیر اینصورت مقدار اعشار آن صفر خواهد بود. باید از این نکته برای تشخیص نحوه محاسبه فاصله استفاده کنید.

شکل ۷: راهنما