

## تمرین ۱۰ درس ساختمان داده

محمد جواد میرشکاری حقیقی

غزاله محمودی

سید صالح اعتمادی

دانشگاه علم و صنعت ۹۸ - ۹۹

لطفاً به نکات زیر توجه کنید:

- مهلت ارسال این تمرین شنبه ۲۳ آذر ماه ساعت ۱۱:۵۹ ب.ظ است.
- این تمرین شامل سوال های برنامه نویسی می باشد، بنابراین توجه کنید که حتماً موارد خواسته شده را رعایت کنید. .
- نام شاخه، پوشه و پول ریکوست همگی دقیقاً "A۱۰" باشد.
- در صورتی که به اطلاعات بیشتری نیاز دارید می توانید با ایدی تلگرام @mj\_haghighi در ارتباط باشید.

موفق باشید.

## توضیحات کلی تمرین

تمرین این هفته ی شما، ۴ سوال دارد که باید به همه ی این سوال ها پاسخ دهید. برای حل این سری از تمرین ها مراحل زیر را انجام دهید:

۱. ابتدا مانند تمرین های قبل، یک پروژه به نام A10 بسازید.

۲. کلاس هر سوال را به پروژه ی 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.

**توجه کنید که سوال ۱ و ۲ دارای متد های دیگری غیر از دو متد بالاست که حتما شما باید بدنه ی این متد ها را 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 ی خود بسازید.

۲. فولدر TestData که در ضمیمه همین فایل قرار دارد را به پروژه ی تست خود اضافه کنید.

۳. فایل GradedTests.cs را به پروژه ی تستی که ساخته اید اضافه کنید. توجه کنید که مانند تمرین های قبل، لازم نیست که برای هر سوال TestMethod بنویسید. تمامی آنچه که برای تست هر سوالتان نیاز دارید از قبل در این فایل برای شما پیاده سازی شده است.

```
[TestMethod(), Timeout(1000)]
0 references
public void SolveTest_Q1PhoneBook()
{
    RunTest(new Q1PhoneBook("TD1"));
}

[TestMethod(), Timeout(1000)]
0 references
public void SolveTest_Q2HashingWithChain()
{
    RunTest(new Q2HashingWithChain("TD2"));
}

[TestMethod(), Timeout(1000)]
0 references
public void SolveTest_Q3RabinKarp()
{
    RunTest(new Q3RabinKarp("TD3"));
}
```

## Phone book ۱

در این سوال شما باید یک دفترچه تلفن ساده را پیاده سازی کنید. برنامه ی شما باید بتواند درخواست های کاربر را پردازش کند. این درخواست ها به صورت زیر است:

- add number name  
این بدان معنی است که کاربر یک فرد را با نام و شماره تلفنی که وارد کرده است را به دفترچه اضافه می کند. اگر فردی با چنین شماره ای در دفترچه تلفن وجود دارد، شما باید نام متناظر را بازنویسی کند.
- del number  
این بدان معنی است که باید یک فرد با شماره تلفن ورودی را از دفترچه تلفن پاک کنید. اگر چنین شخصی وجود نداشته باشد، این درخواست را نادیده بگیرید.
- find number  
این بدان معنی است که کاربر به دنبال فردی با شماره تلفن ورودی است. شما باید نام شخص متناظر با شماره ی تلفن را برگردانید. اگر چنین شخصی در دفترچه تلفن وجود نداشت، عبارت not found را برگردانید.  
در هر خط از فایل ورودی یکی از درخواست های بالا وجود دارد. در هر خط از خروجی هم پاسخ به درخواست find قرار دارد.  
**لطفا نمونه های ورودی و خروجی سوال را از داخل داکيومنت اصلی مطالعه فرمایید.**

توجه کنید که برای راحتی شما کلاس contact را پیاده سازی کردیم. این کلاس دو ویژگی name و number دارد. شما می توانید در صورت نیاز از این کلاس استفاده کنید و یا آن را تغییر دهید.

```
public class Contact
{
    public string Name;
    public int Number;

    public Contact(string name, int number)
    {
        Name = name;
        Number = number;
    }
}
```

در این سوال شما باید تابع های Add و Delete و Find که در فایل Q1PhoneBook.cs قرار دارد را پیاده سازی و کامل کنید. در این سوال پیاده سازی ساده و ابتدایی برای شما انجام شده است که اگر تست کنید خواهید دید که نسبت به الگوریتم اصلی که شما باید پیاده سازی کنید بسیار کند است و بیش از اندازه طول می کشد.

```
public void Add(string name, int number)
{
    for(int i=0; i<PhoneBookList.Count; i++)
    {
        if (PhoneBookList[i].Number == number)
        {
            PhoneBookList[i].Name = name;
            return;
        }
    }
    PhoneBookList.Add(new Contact(name, number));
}
```

```
public string Find(int number)
{
    for (int i = 0; i < PhoneBookList.Count; i++)
    {
        if (PhoneBookList[i].Number == number)
            return PhoneBookList[i].Name;
    }
    return "not found";
}
```

```
public void Delete(int number)
{
    for (int i = 0; i < PhoneBookList.Count; i++)
    {
        if (PhoneBookList[i].Number == number)
        {
            PhoneBookList.RemoveAt(i);
            return;
        }
    }
}
```

```

public class Contact
{
    public string Name;
    public int Number;

    1 reference
    public Contact(string name, int number)
    {
        Name = name;
        Number = number;
    }
}

1 reference
public class Q1PhoneBook : Processor
{
    0 references
    public Q1PhoneBook(string testDataName) : base(testDataName) { }

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

    protected List<Contact> PhoneBookList;

    1 reference
    public string[] Solve(string [] commands) {...}

    1 reference
    public void Add(string name, int number) {...}

    1 reference
    public string Find(int number) {...}

    1 reference
    public void Delete(int number) {...}
}

```

---

## ۲ Hashing with chains

در این سوال شما باید یک جدول hash را به صورت زنجیره ای پیاده سازی کنید. Chaining یکی از رایج ترین روش های اجرای جدول های hash است. از چنین جدول hash ای می توان برای پیاده سازی یک دفترچه تلفن بر روی تلفن همراه یا ذخیره جدول رمز عبور کامپیوتر و سرویس وب استفاده کرد.

فرض کنید که تعداد bucket ها  $m$  باشد بنابراین برای پیاده سازی جدول hash به صورت زنجیره ای از تابع چند جمله ای زیر به عنوان تابع hash استفاده کنید.

$$h(S) = \left( \sum_{i=0}^{|S|-1} S[i]x^i \bmod p \right) \bmod m$$

که در این تابع  $s[i]$  کد ASCII، المان  $i$  رشته  $s$  است و

$$p = 1\,000\,000\,007, x = 263$$

برنامه شما باید دستور های زیر را پشتیبانی کند:

- add string  
این بدان معنی است که رشته را به جدول وارد کنید. اگر قبلاً چنین رشته ای در جدول hash وجود دارد، این درخواست را نادیده بگیرید.
- del string  
این بدان معنی است که رشته را از جدول حذف کنید. اگر چنین رشته ای در جدول hash وجود ندارد، پس درخواست را نادیده بگیرید.
- find string  
این بدان معنی است که بسته به اینکه آیا جدول شامل رشته هست یا نه، خروجی "yes" یا "no" را برگردانید.
- check i  
این بدان معنی است که محتوای  $i$  امین لیست در جدول را خروجی دهید. با استفاده از space المان ها از هم جدا می شوند. اگر لیست  $i$  ام خالی باشد، یک خط خالی را در خروجی نمایش دهید.

هنگام وارد کردن یک رشته جدید به زنجیره hash باید آن را در ابتدای زنجیره وارد کنید. خط اول ورودی تعداد Bucket هاست و هر یک از خطوط بعدی یکی از درخواست های بالا می باشد. هر یک از خطوط خروجی جواب درخواست های check و find به ترتیبی که این دستور ها در ورودی آمده است، می باشد.

**لطفا نمونه های ورودی و خروجی سوال را از داخل داکيومنت اصلی مطالعه فرمایید.**

در این سوال شما باید تابع های Add و Delete و Find و check که در فایل Hashing-WithChain.cs قرار دارد را پیاده سازی و کامل کنید. دقت کنید برخی از مواردی که برای حل این سوال نیاز دارید برای شما پیاده سازی شده است.

```
public class Q2HashingWithChain : Processor
{
    0 references
    public Q2HashingWithChain(string testDataName) : base(testDataName) { }

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

    1 reference
    public string[] Solve(long bucketCount, string[] commands) {...}

    public const long BigPrimeNumber = 1000000007;
    public const long ChosenX = 263;

    0 references
    public static long PolyHash(
        string str, int start, int count,
        long p = BigPrimeNumber, long x = ChosenX) {...}

    1 reference
    public void Add(string str) {...}

    1 reference
    public string Find(string str) {...}

    1 reference
    public void Delete(string str) {...}

    1 reference
    public string Check(int i) {...}
}
```



## ۳ Find pattern in text

در این سوال شما باید الگوریتم Rabin-Karp را برای جستجوی یک الگو در یک متن پیاده سازی کنید. در فایل ورودی دو رشته وجود دارد که رشته ی اول الگو و رشته ی دوم متن می باشد. در خروجی هم باید مکان هایی از متن که در آن الگو داده شده اتفاق افتاده است را برگردانید. یعنی:

ورودی:

aba

abacaba

همان طور که ملاحظه می کنید الگو ی aba در متن داده شده، یک بار در مکان صفر اتفاق افتاده است و بار دیگر در مکان ۴. بنابراین خروجی به صورت زیر خواهد بود:

خروجی:

0 4

**لطفا ادامه ی نمونه های ورودی و خروجی سوال را از داخل داکيومنت اصلی مطالعه فرمایید.**

همانطور که در شکل زیر مشاهده می کنید، در این سوال پیاده سازی ساده و ابتدایی برای شما انجام شده است. اگر این برنامه را تست کنید خواهید دید که نسبت به الگوریتم اصلی که شما باید پیاده سازی کنید بسیار کند است و بیش از اندازه طول می کشد.

```

public class Q3RabinKarp : Processor
{
    0 references
    public Q3RabinKarp(string testDataName) : base(testDataName) { }

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

    1 reference
    public long[] Solve(string pattern, string text)
    {
        List<long> occurrences = new List<long>();
        int startIdx = 0;
        int foundIdx = 0;
        while ((foundIdx = text.IndexOf(pattern, startIdx)) >= startIdx)
        {
            startIdx = foundIdx + 1;
            occurrences.Add(foundIdx);
        }
        return occurrences.ToArray();
    }

    0 references
    public static long[] PreComputeHashes(
        string T,
        int P,
        long p,
        long x)
    {
        return new long[] { };
    }
}

```

## ۴ Bloom filter ( امتیازی)

BloomFilter یک ساختمان داده فشرده بر اساس Hash Function است. این ساختمان داده شبیه Hash Table است با چند تفاوت: اول اینکه برخلاف HashTable این ساختمان داده به ازای کلید، مقداری بر نمیگرداند. تنها کاری که میکند این است که میگوید آیا این کلید قبلا اضافه شده یا نه. در واقع فقط دو متد دارد  $Add(key)$  و  $Test(key)$ . تفاوت دوم اینکه فقط می شود به این ساختمان داده ای کلید اضافه کرد. حذف نمیتوان کرد. و مهمترین تفاوت اینکه اگر جواب  $Test$  منفی باشد، جواب صد در صد درست است. ولی اگر جواب مثبت، به احتمالی هم وجود دارد که واقعا درست نبوده باشد. به اینکه چند درصد از جواب های مثبت  $Test$  واقعا درست بوده اصطلاحا گفته میشود False Positive Rate. این مقدار بستگی به دو عامل دارد. یکی اندازه این ساختمان داده (اندازه ساختمان داده بر اساس بیت است و بعد از اضافه کردن اولین کلید، دیگر قابل تغییر نیست). دوم تعداد و نوع تابع های hash استفاده شده. هر چه اندازه این ساختمان داده بزرگتر باشد درصد False Positive Rate کمتر میشود. تابع های hash باید از هم مستقل یا مقدارهاشون به هم مربوط نباشد. بطوریکه اگر دو کلید متفاوت با یک تابع hash مقدار یکسان داشتند، با hash دوم حتی الامکان مقدار یک سان نداشته باشند. تعداد توابع هم تقریبا  $0.7n$  نسبت تعداد کل کلید ها به تعداد بیت ها باشد خوبه. مستقل از اندازه کلید، تقریبا ده بیت به ازای هر کلید منجر به False Positive Rate حدود ۱ درصد میشود. مثلا اگر هزار تا کلید داشته باشیم، اگر ده هزار بیت داشته باشیم و هفت تا تابع hash داشته باشیم، خوبه. البته اینها حدودی است. باید بالا/پایین کنید تا مقدار درست در بیاید. متد  $Add$  به این شکل کار میکند که کلید را به تمام hash function ها میدهد و تعداد عدد دریافت میکند بین صفر تا اندازه فیلتر. بیت های متناظر با این اعداد را برابر یک قرار میدهد. متد  $Test$  هم به این شکل کار میکند که مجدد کلید را به تمام hash function ها میدهد و اعداد متناظر را دریافت میکند. اگر بیت های متناظر با تمام این اعداد یک بودند جواب مثبت برمیگرداند وگرنه منفی.

خوب حالا که با Bloom-Filter آشنا شدید میخواهیم با آن یک مساله واقعی حل کنیم. مساله این است آیا پسورد شما لو رفته؟ تا به حال نزدیک پانصد میلیون پسورد واقعی از کاربران لو رفته که میتوانید در این [سایت](#) در آن جستجو کنید. کل پسوردها هم ده گیگا بایت هستند که میتوانید دانلودشون کنید. فرض کنیم شما میخواهید یک اپلیکیشن موبایل بنویسید که بدون دسترسی به شبکه (این سایت) بتواند چک کند که آیا پسورد شما داخل این لیست هست یا نه. برای این کار میتوانید از Bloom Filter استفاده کنید. در اینصورت اگر به کاربر گفتید که پسوردت لو نرفته که حتما نرفته. ولی اگر گفتین لو رفته، خوب با به احتمالی لو رفته (بسته به اندازه فیلتر شما). در این سوال شما باید سازنده و متدهای  $Add$  و  $Test$  را برای کلاس [Q4BloomFilter](#) پیاده سازی کنید بطوری که تست کلاس [Q4BloomFilterTests](#) پاس شود. برای پاس شدن این تست لازم است که False Positive Rate ده در صد کمتر باشد و اندازه فیلتر هم از پنج در صد اندازه کل پسوردها کوچکتر باشد. این مقداری را در یونیت تست با یک میلیون پسورد تصادفی تست

میکنیم. توی کلاس یک خانواده از `hash function` ها را برای کلیدهای از نوع `string` یاد گرفتیم. کار اصلی شما این است که در سازنده از این خانواده به تعداد لازم تابع انتخاب کنید و در متدهای `Test` و `Add` از آنها استفاده کنید. برای فهم بهتر میتوانید از این [سایت](#) استفاده کنید.