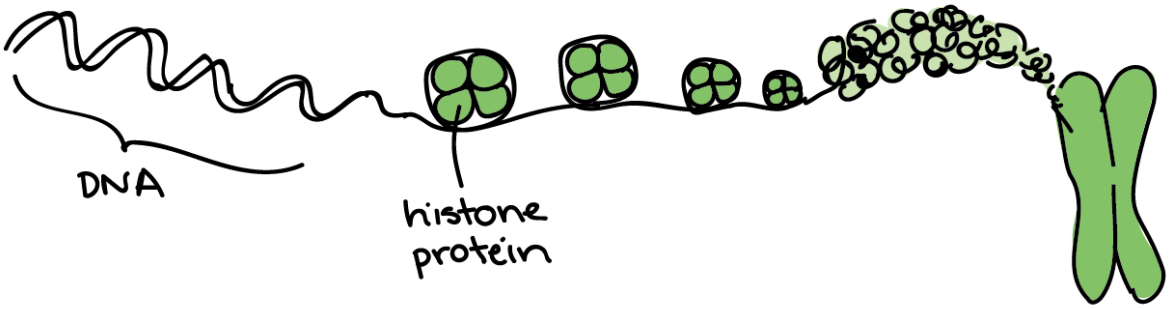
پروفایل توالی‌ها

در هسته هر سلول ما ماده ژنتیکی وجود داره که اطلاعات ساخت تمامی بدن در آن قرار داره. ماده ژنتیکی ما به تعدادی کروموزوم تقسیم میشه.



هر کروموزوم خودش شامل یک مولکول طویل DNA است که کدهای ژنتیکی روی اون قرار دارن. برای اینکه سلول بخواد کاری انجام بده باید پروتئین بسازه مثلا فرض کنید سلول می‌خواد به یه جا وصل بشه. کاری که می‌کنه اینه که یه دست پروتئینی میسازه و اونو روی غشائش قرار میده بعد با استفاده از اون به جایی که می‌خواد میچسبه. حالا سوال اینه که پروتئین رو چجوری میسازه؟

پروتئین ساختن برای سلول کاری نداره چون نقشه‌شو داره نقشه هم چیزی نیست جز ماده ژنتیکی یا همون DNA. خوب حالا چجوری از روی DNA پروتئین ساخته میشه؟ خیلی وارد جزئیاتش نمیشم اما همین قدر کافیه که بدونیم روی DNA کلی ژن هست هر کدوم از این ژن‌ها برای ساخت یه پروتئین خاصه حالا هر موقع سلول به یه پروتئین نیاز داشته باشه یه نفرو می‌فرسته تا از روی ژن مربوطش بسازتش.

چیزی که ما الان باهاش سرو کار داریم اینه که خوب سلول چجوری یه ژن رو شناسایی می‌کنه؟ واقعیت اینه که هر ژن یه توالی از چهار نوع نوکلئوتید هست حالا تو بالادست یه ژن خاص یه توالی خاص قرار داره. سلول این توالی رو میشناسه و یه نفرو می‌فرسته تا به اون بچسبه و ... . به این توالی **راه انداز** میگن.

یکی از زیست‌شناسا می‌خواد راه‌انداز چند تا ژن رو پیدا کنه سر همین میره اون ژن رو تو موجودات مختلف پیدا می‌کنه بعد بالادست ژنشون رو بررسی می‌کنه و با توالی‌های زیر مواجه میشه:

```
ATCCCGGGG
ATCCCGGGG
AACCCGCGG
AACCCGAGG
AACCCGAGG
```

همونطوری که می‌بینید همه توالی‌ها مثل هم نیستن و بعضی از نوکلئوتیدها جهش پیدا کردن. حالا دانشمند ما می‌خواد یه تعداد آزمایش روی این توالی‌ها انجام بده اما مشکلش اینه که سختشه روی کاغذ محاسبات رو انجام بده و درنتیجه از ما می‌خواد یک **class** براش طراحی کنیم که اولا همه این توالی‌ها رو ذخیره کنه و با چند تا تابع خوب اطلاعاتی که می‌خواد رو در اختیارش بزاریم.

کلاس Profile

اول میریم سراغ fieldهایی که این کلاس نیاز داره. خب اول از همه باید توالی‌ها رو بگیریم پس به یک فیلد برای نگه‌داری **آرایه** دنباله‌ها نیاز داریم می‌تونید اسم این field رو sequences بزارید.

برای نگه‌داری اطلاعات آماری باید از یک آرایه دو بعدی استفاده کنیم که احتمال هر نوکلئوتید توی هر موقعیت رو مشخص می‌کنه. مثلا جدول مربوط به دنباله‌های خودمون به صورت زیر میشه:

موقعیت	1	2	3	4	5	6	7	8	9
A	1	$\frac{3}{5}$	0	0	0	0	$\frac{2}{5}$	0	0
C	0	0	1	1	1	0	$\frac{1}{5}$	0	0
G	0	0	0	0	0	1	$\frac{2}{5}$	1	1
T	0	$\frac{2}{5}$	0	0	0	0	0	0	0

حالا بعد از این که **سازنده** کلاس صدا زده بشه باید این ماتریس تشکیل بشه. برای تشکیل دادن اون از یه آرایه دو بعدی به اسم profile استفاده می‌کنیم.

```
1  public class Profile {
2
```

```
    String[] sequences;
    double[][] profile;

    public Profile(String[] sequences) {
        this.sequences = sequences;
        // TODO initial and fill profile matrix
    }
}
```

توابع

در ادامه یسری تابع ایجاد می‌کنیم که بررسی profile رو برامون راحت تر کنه.

تابع printProfile

این تابع پروفایل رو برامون چاپ میکنه. مثلا من با کد خودم یه پروفایل خاص رو چاپ کردم و خروجی به صورت زیر شده:

```
A : 0.20 0.20 0.00 0.00 0.00 0.00 0.90 0.10 0.10 0.10 0.30 0.00
C : 0.10 0.60 0.00 0.00 0.00 0.00 0.00 0.40 0.10 0.20 0.40 0.60
G : 0.00 0.00 1.00 1.00 0.90 0.90 0.10 0.00 0.00 0.00 0.00 0.00
T : 0.70 0.20 0.00 0.00 0.10 0.10 0.00 0.50 0.80 0.70 0.30 0.40
```

تابع getConsensus

به توالی‌های قبلیمون دقت کنید:

```
ATCCCGGGG
ATCCCGGGG
AACCCGCGG
AACCCGAGG
AACCCGAGG
```

توی هر موقعیت یه نوع نوکلئوتید هست که فراوانیش از بقیه بیشتره. مثلا تو موقعیت اول A از همه بیشتره همینطور تو موقعیت دوم. حالا اگه یه دنباله درست کنیم که تو هر موقعیتش همین نوکلئوتیدها رو چیده باشیم میشه **Consensus** یا به عبارت دیگه نماینده یا اجماع. اجماع توالی‌های بالا میشه توالی پایین:

```
AACCCGAGG
```

اگه تو یه موقعیت فراوانی چند تا نوکلئوتید برابر شد یکی رو به **تصادف** انتخاب می‌کنیم. تابع getConsensuus رشته اجماع رو برمی‌گردونه.

بررسی خطاها

در سازنده مجموعه توالی‌ها به عنوان ورودی گرفته میشه اما ما همواره فرض کردیم که طول این توالی‌ها یکسانه حالا ممکنه کسی که از کلاس شیئ می‌سازه حواسش نباشه و طول‌ها رو متفاوت بده. تو این حالت تابع‌های ما درست کار نمی‌کنن و در نتیجه به خطا می‌خوریم اما درست متوجه نمی‌شیم این خطا از کجا ناشی میشه. برای همین همون اول بررسی می‌کنیم که طول همه توالی‌ها یکسان باشه و در غیر این صورت یک **Exception** تولید می‌کنیم. نوع این خطا باید IllegalArgumentException باشه و تو پیامش باید توضیح بدیم که چرا این خطا رخ داده.

کامنت گذاری

کد تمرین قبلی خودتون رو کامنت گذاری کنید. به این صورت که قبل از هر تابع به شکل زیر کامنت بنویسید:

```
/**
 * this method adds two numbers
 * @param numberOne first number
 * @param numberTwo second number
 * @return sum of two numbers
 */
public int add(int numberOne, int numberTwo) {
    return numberOne + numberTwo;
}
```

همونطور که می‌بینید به ازای هر پارامتر یک خط با عنوان param داریم و باید توضیح بدیم که این ورودی چی هست. به علاوه اگه تابع خروجی داشته باشه باید توضیح بدیم که این خروجی چی هست.

توضیحات

همون کد قبلیتون رو اصلاح کنید و دوباره بفرستید.

توضیح خطا

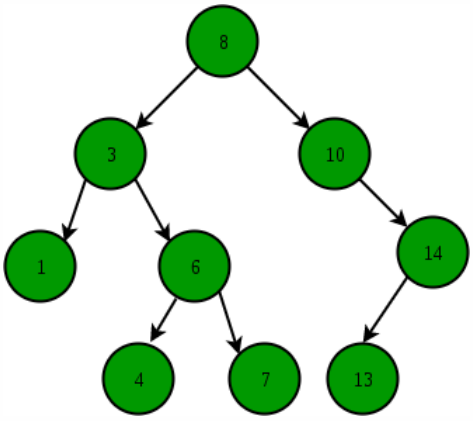
کد زیر رو اجرا کنید و خطای مربوطش رو بررسی کنید. توضیح بدید که این چه نوع خطایی هست و چرا این اتفاق می‌افته؟

```
public class MyClass {
    private MyClass myClass;
    public MyClass() {
        myClass = new MyClass();
    }

    public static void main(String[] args) {
        MyClass myClass = new MyClass();
    }
}
```

درخت جستجوی دودویی

در این تمرین قصد داریم که درخت جستجوی دودویی را به وسیله کلاس‌های جاوا مدل کنیم. درخت جستجوی دودویی برای ذخیره کردن داده‌هایی استفاده میشه که ترتیب دارن مثلا اعداد. تو این تمرین ما از درخت برای ذخیره عددهای صحیح استفاده می‌کنیم.



همونطور که تو شکل بالا می‌بینیم درخت جستجوی دودویی سه تا ویژگی داره:

- اول اینکه هر **راس** این درخت مقداری رو نگه میداره که از مقدار تمام فرزندهای سمت چپش بیشتره.
- هر **راس** این درخت مقداری نگه میداره که از مقدار تمام فرزندهای سمت راست کمتر مساوی هست.
- زیردرخت هر **راس** خودش یه درخت جستجوی دودویی هست.

به طور خلاصه برای اینکه بتونیم این درخت رو مدل کنیم به دو تا کلاس نیاز داریم:

- کلاس **BinarySearchTree**
- کلاس **Node**

رابطه این دو تا کلاس به صورت زیره:

اسم نمودار بالا UML هست و رابطه بین کلاس‌ها رو نشون میده. علاوه بر اون نشون میده که هر کلاس چه fieldها و توابعی داره. مثلا کلاس BinarySearchTree ما یک field به اسم root داره که در واقع راسی که ریشه هست رو نگه میداره. در رابطه با UML می‌تونید اینجا بیشتر بخونید.

کلاس Node

این کلاس دو تا field داره:

- فرزند سمت چپ
- فرزند سمت راست

هر دو تا field از نوع Node هستن. همونطور که تو تمرین قبل دیدیم اگه این دو تا field رو داخل **سازنده** مقدار دهی کنیم با خطا مواجه میشیم در نتیجه مقدار دهی به این دو تا field باید از خارج کلاس و با استفاده از توابع **public** صورت بگیره مثلا می‌تونیم از توابع **setter** استفاده کنیم.

کلاس BinarySearchTree

این کلاس تنها fieldای که داره root هست این field از نوع Node هست و اول کار مقدار null داره. وقتی اولین داده **insert** میشه باید این root رو مقدار دهی کنیم.

تابع insert

هر بار که می‌خوایم یه مقدار جدید رو وارد درخت کنیم باید اول جایگاه مناسبش رو پیدا کنیم. برای این کار از root شروع می‌کنیم و هر بار اگه راس مقدار بیشتری از مقدار جدید داشته باشه میریم سمت چپش و اگه مقدار کمتری داشته باشه میریم سمت راستش. این کار رو انقدر تکرار می‌کنیم تا به یک Node برسیم که مقدار هر دو فرزندش null باشه. وقتی به این راس رسیدیم یک Node جدید تولید می‌کنیم و اون رو تو سمت مناسب این راس قرار میدیم.

تابع search

این تابع درخت رو جستجو میکنه و اگه مقدار مورد نظر تو درخت وجود داشت true رو برمیگردونه در غیر اینصورت false رو برمی‌گردونه.

توضیحات

کد شما شامل این دو تا کلاس هست به علاوه یک کلاس Main که کد خودتون رو داخلش امتحان می‌کنید. برای امتحان کردن کدتون ابتدا ۱۰ عدد تصادفی تولید کنید و به درخت اضافه کنید به علاوه هر ۱۰ عدد رو در خروجی استاندارد چاپ کنید. بعد از این ۵ عدد تصادفی دیگه تولید کنید و اون‌ها رو داخل درخت جستجو کنید. این ۵ عدد به علاوه نتیجه جستجو رو داخل خروجی استاندارد چاپ کنید.

ماز

فردی دقیقا در ابتدای یک ماز قرار دارد و تنها راه نجاتش گذر و عبور از این جدول ماز است. پس دست به کار می شود تا راه نجات را پیدا کند. او پس از چندین روز تلاش نا امید می شود اما در می یابد که در مازی با طول m (تعداد ردیف های جدول ماز) و عرض n (تعداد ستون های جدول ماز) گیر افتاده است.در همین حین یادش می افتد که لپ تاپ را با خود آورده است! پس به سرعت سراغ کیفش می رود و با هیجان در میابد که لپ تاپ هنوز سالم هست و کار می کند، اما لپتاپ تنها 15درصد شارژ دارد. پس توماس از شما می خواهد تا هر چه سریعتر به او کمک کنید تا برنامه ای بنویسد که به او برای یافتن نقشه ی احتمالی این ماز کمک کند. او می داند در هر روز توانایی t بار بررسی کل ماز را دارد پس باید برنامه ای بنویسد که با گرفتن m (طول ماز) و n (عرض ماز)، t حالت و نقشه متفاوت رندوم از ماز را ترسیم کند تا او بتواند با آن نقشه ها به بررسی ماز بپردازد. نکته ای که باید به آن دقت کنید این است که اصلا فرصت برای بررسی نقشه های تکراری را ندارد و باید هر t نقشه خروجی برنامه شما حداقل در یک خانه تفاوت داشته باشند!

این ماز ویژگی های زیر را باید داشته باشد:

- تمامی دیوارهای داخل ماز، حداقل به یک دیوار متصل هستند و دیواری وجود ندارد که به هیچ دیواری متصل نباشد. همچنین تمامی دیوارها به صورت مستقیم یا غیرمستقیم به دیواره اطراف ماز وصل هستند.
- بین هر دو خانه مربعی ماز، تنها یک مسیر یکتا وجود دارد.

دیواره ی دور ماز در تصویر بالا قابل مشاهده است.

در تصویر بالا مشاهده می شود که بین هر دو خانه در ماز دقیقا یک مسیر یکتا وجود دارد.

این ماز قابل مدل کردن با گراف ها است. یکی از الگوریتم های پیمایش گراف که در حل این مسئله به شما کمک میکند، الگوریتم dfs است که برای فهم آن میتوانید از این لینک و یا سایر منابع موجود در اینترنت استفاده کنید.

ورودی

ورودی به ترتیب شامل سه عدد m و n و t است. m عرض ماز و n طول آن است. همچنین t تعداد مازهای متفاوتی است که باید در خروجی چاپ شود.

$$1 \leq n, m, t \leq 100$$

خروجی

خروجی به تعداد t جدول به عرض 2*m+1 و طول 2*n+1خواهد بود که هریک به فرمت زیر است. تمامی خانه های ماز (نقاط قرمز رنگ نشان داده شده) می بایست با * در خروجی نشان داده شوند. تمام دیواره های اطراف هر خانه از جدول ماز بدین صورت نشان داده می شوند که در صورت وجود دیوار آن را با `1` و در غیر این صورت (در صورتی که دیواری بین دو خانه وجود نداشت و امکان عبور وجود داشت) آن را با `0` نشان می دهیم.همچنین برای ماز یک ورودی و یک خروجی در نظر گرفته شده است که جای ثابتی دارند که در شکل نیز مشخص است. این ورودی و خروجی ماز با کاراکتر `e` نمایش داده میشوند. در تصویر زیر یک ماز 7*7 و نحوه خروجی دادن آن مشخص شده است:

مثال

در اینجا چند نمونه برای فهم بهتر صورت سوال و قالب ورودی و خروجی تست‌ها داده می‌شود.

ورودی نمونه ۱

```
3 6 3
```

خروجی نمونه ۱

```
1e11111111111
1*0*0*0*0*0*1
1110111111101
1*1*0*1*0*1*1
1011111010101
1*0*0*0*1*0*1
11111111111e1

1e11111111111
1*1*0*0*0*0*1
1011111110101
1*0*0*0*0*1*1
1111111111101
1*0*0*0*0*0*1
11111111111e1

1e11111111111
1*0*0*1*0*1*1
1011101010101
1*0*1*0*1*0*1
1110111111101
1*0*1*0*0*0*1
11111111111e1
```

توجه داشته باشید که با توجه به اینکه ماز ها به طور رندوم ساخته میشوند، این تنها یک خروجی درست برای این ورودی است.

ورودی نمونه ۲

```
4 11 3
```

خروجی نمونه ۲

```
1e1111111111111111111111
1*0*1*0*0*1*1*0*0*1*0*1
1110101110101010111101
1*0*1*1*1*1*0*1*1*0*1*1
1011101010111111011101101
1*1*0*1*1*0*1*0*1*0*1*1
1011101011101011101101
1*0*0*1*0*0*0*1*0*0*0*1
1111111111111111111111e1

1e1111111111111111111111
1*0*0*0*0*0*0*0*0*0*0*1
1111101111101111111101
1*0*1*1*0*0*1*0*0*1*0*1
1010111011111010111101111
1*1*0*1*0*1*0*1*1*0*1*1
1111101110101110101101
1*0*0*0*0*1*0*1*0*0*0*1
1111111111111111111111e1

1e1111111111111111111111
1*0*1*0*1*0*0*1*0*0*0*1
1010101011101010111101
1*1*0*1*1*0*1*0*1*0*1*1
1111111010111111010101
1*0*1*0*1*1*0*0*0*1*0*1
1010101110111111111111
1*1*0*0*0*1*0*0*0*0*0*1
1111111111111111111111e1
```