بسمه تعالی

دانشگاه فردوسی مشهد

مساله HashTable نسخه java:

در این مساله، از شما می‌خواهیم تا ساختمان داده HashTable را مطابق روش Open Addressing با الگوریتم Double Hashing برای حل مشکل تداخل پیاده‌سازی کنید. البته ما از نسخه‌ی خاصی از الگوریتم Double Hashing استفاده می‌کنیم که در آن تابع درهم‌ریزی دوم همواره یک عدد ثابت بر می‌گرداند. در این مورد در ادامه واضح‌تر توضیح خواهیم داد.

در الگوریتم Double Hashing از دو کلید درهم‌ریزی $h1$ و $h2$ استفاده می‌شود. برای درج داده‌ای با کلید x ابتدا خانه‌ی $h1(x)$ به عنوان محل اصلی (home bucket) در نظر گرفته می‌شود و اگر این خانه خالی باشد، داده در آنجا درج می‌شود. اما اگر خانه‌ی $h1(x)$ پر بود، با گام $h2(x)$ در پیمانه‌ی تعداد خانه‌های جدول درهم‌ریزی، خانه‌های بعدی را بررسی می‌کنیم تا اینکه نهایتا یک خانه‌ی خالی پیدا شود. اگر تعداد خانه‌های جدول درهم‌ریزی عددی اول باشد و حداقل یک خانه از جدول خالی باشد، این تضمین وجود دارد که بالاخره یک خانه‌ی خالی ملاقات خواهد شد. بنابراین می‌توان گفت که، اگر اندازه‌ی جدول درهم‌ریزی n باشد، در روش double hashing به ترتیب خانه‌های زیر بررسی می‌شوند تا نهایتا به یک خانه‌ی خالی برسیم.

$$h_1(x)\ mod\ n,\ \ \big(h_1(x)+h_2(x)\big)mod\ n,\ \ \ \big(h_1(x)+2h_2(x)\big)mod\ n,\ \dots,\big(h_1(x)+(n-1)h_2(x)\big)mod\ n$$

اما ما در این تمرین نسخه‌ی خاص‌تری از الگوریتم double hashing را در نظر می‌گیریم که در آن h_2 یک تابع ثابت است و مقدار آن به x مربوط نیست.

توجه کنید که یک جدول درهم‌ریزی یک ساختمان داده برای پیاده‌سازی نوع داده‌ی مجرد نگاشت (map) است. بنابراین ما جدول در هم‌ریزی را به عنوان پیاده‌سازی‌ای از رابطه Map می‌بینیم.

```
interface Map<K extends Comparable<K> ,V> {
   void assign(K key, V value);
   void remove(K key);
   V get(K key);
   boolean hasKey( K key);
   int size();
}
```

برای تعریف توابع درهم‌ریزی رابط HashFunction به صورت زیر تعریف شده است:

```
public interface HashFunction<K> {
        int hash(K key);
}
```

در پیاده‌سازی‌های این رابط متد hash یک کلید را دریافت می‌کند و متناظر با آن یک عدد صحیح نامنفی بر می‌گرداند. نیازی نیست این عدد صحیح در بازه‌ی خانه‌های جدول درهم‌ریزی باشد. بعدا در کد جدول درهم‌ریزی باقی‌مانده‌ی مقدار خروجی تابع hash را بر تعداد خانه‌های جدول محاسبه می‌کنیم.

می‌رسیم به کلاس اصلی که قرار است پیاده‌سازی شود و آن کلاس HashTable است. درون این کلاس، کلاس KeyValuePair به صورت خصوصی چنین تعریف شده است:

```
private class KeyValuePair<K, V>{
        public K key;
        public V value;
}
```

ما از این کلاس برای ساخت آرایه‌ی ذخیره‌سازی داده‌ها (یا همان جدول) استفاده می‌کنیم. در هر خانه کلید key و مقدار value که متناظر با آن کلید است ذخیره می‌شود.

Attribute های کلاس HashTable به شرح زیر است:

```
private int                capacity;
private int size;
private HashFunction h1;
private HashFunction h2;
private boolean[] stateTable;      // 0 => empty ,  1 => full
private KeyValuePair<K, V> [] table;
private float      maxLoadingFactor;
```

capacity تعداد خانه‌های جدول درهم‌ریزی را نشان می‌دهد.

size تعداد داده‌های ذخیره شده را نشان می‌دهد.

$h1$ و $h2$ به اشیائی از یک پیاده‌سازی از رابط HashFunction اشاره می‌کنند و توابع در هم‌ریزی اول و دوم را در اختیار ما می‌گذارند. برای درک بهتر، می‌توانید فایل تست و نحوه عملکرد آن را بررسی کنید تا با نحوه کارکرد آن، بهتر آشنا شوید.

stateTable یک آرایه منطقی به طول capacity است که اگر خانه‌ای از جدول خالی باشد آن را با مقدار صفر و اگر پر باشد با مقدار یک علامت‌گذاری می‌کند.

table یک آرایه به طول capacity است که در خانه‌های پر آن شیئی از نوع KeyValuePair<K, V> اطلاعات یک زوج (کلید، مقدار) را نگهداری می‌کند. اینکه کدام خانه‌ها خالی و کدام پر هستند را stateTable مشخص می‌کند.

maxLoadingFactor یک عدد بین ۰ تا ۱ است که تعیین می‌کند چه زمانی اندازه‌ی جدول درهم‌ریزی افزایش پیدا کند. به صورت مشخص ما زمانی اندازه‌ی جدول را افزایش می‌دهیم که شرط زیر برقرار شده باشد:

```
if (size > capacity * maxLoadingFactor){
        …
}
```

در این صورت capacity جدید برابر اولین عدد اولی است که از دو برابر capacity فعلی بزرگ‌تر باشد. مثلا اگر ظرفیت اولیه 7 باشد، ظرفیت جدید باید 17 شود. پس از تغییر اندازه، مقادیر قبلی دوباره باید در آرایه بزرگتر درج شوند. در پیاده‌سازی صحیح ما انتظار داریم که عناصر جدول قبلی با توجه به مکان آنها در آرایه یکی یکی در جدول جدید درج شوند (ابتدا عنصر خانه‌ی صفر جدول ، سپس عنصر خانه‌ی یک جدول ، ... و به همین ترتیب در هر خانه ای که عنصری وجود داشته باشد، باید به جدول جدید منتقل شود)

توجه: در طول پیاده‌سازی حتما به این جزئیات دقت کنید. تست‌ها مبتنی بر این مطالب نوشته شده اند و اگر کوچک‌ترین تفاوتی در الگوریتم افزایش طول جدول درهم‌ریزی با مطالب گفته شده در بالا باشد، تست‌ها با شکست مواجه خواهند شد.

حالا به شرح متدهای کلاس HashTable می‌پردازیم. اولین متد سازنده است

```
public HashTable(
        HashFunction h1,
        HashFunction h2,
        float maxLoadingFactor,
        int initCapacity) {
        // Write your code here
}
```

ورودی‌های متد سازنده عبارتند از :

یک شیء $h1$ که تابع درهم‌ریزی اول (تابع hash کردن) را مشخص می‌کند.

یک شیء $h2$ که تابع درهم‌ریزی دوم (تابع محاسبه کننده گام) را مشخص می‌کند.

mMaxLoadingFactor مقدار ویژگی maxLoadingFactor که در بالا توضیح داده شد.

initCapacity: مقدار اولیه‌ی ویژگی capacity.

متد بعدی assign است که برای درج یک مقدار جدید در جدول درهم‌ریزی (و یا جایگزینی مقدار قبلی) استفاده می‌شود.

```
public void assign(K key, V value) {
        // Write your code here
}
```

تابع assign با دریافت کلید و داده، آن را در مکان مناسب خود در table قرار می‌دهد. از طرفی مقدار متناظر با آن خانه در stateTable نیز باید true شود، چرا که آن خانه پر شده است. توجه کنید که در جدول درهم‌ریزی عناصر با کلید مساوی نداریم و در صورتی که کلید قبلا در جدول وجود داشته باشد، لازم است که value جدید به آن کلید منتسب شود و انتساب قبلی از بین برود.

متد بعدی remove است که زوج $(key, value)$ با کلید key را در صورت وجود پیدا کرده و از جدول در هم‌ریزی حذف می‌کند. اگر داده‌ای با این کلید وجود نداشته باشد، نیاز به انجام عملی اضافی نیست و در صورتی که کد شما تغییری در آرایه ایجاد نکند، خطایی هم گرفته نمی‌شود.

```
public void remove(K key) {
        // Write your code here
}
```

متد بعدی hasKey است که یک کلید را دریافت می‌کند و بررسی می‌کند که آیا داده‌ای با این کلید در جدول در هم‌ریزی وجود دارد یا نه؟ اگر key در جدول درهم‌ریزی پیدا شود مقدار true و در غیر این صورت مقدار false برگردانده می‌شود.

```
public boolean hasKey( K key) {
        // Write your code here
}
```

آخرین متدی که لازم است پیاده‌سازی شود، متد get است. این متد یک کلید دریافت می‌کند و در صورتی که قبلا آن کلید به یک مقدار value منتسب شده باشد، مقدار value را برمی‌گرداند. در غیر این صورت یک exception پرتاب می‌شود.

```
@Override
public V get(K key){
        // Write your code here
}
```

در این تمرین نحوه ارزیابی به شرح زیر است:

1. testAssign: در این تست نحوه صحیح اضافه کردن داده به HashTable و همچنین نحوه صحیح تغییر اندازه و استفاده از mMaxLoadingFactor تست می‌شود. این تست، 10٪ از نمره نهایی را به خود اختصاص می‌دهد. به دلیل اهمیت و کاربرد عمل assign، این تست از اهمیت بیشتری برخوردار است.
2. testAssignOrder: این تست، مرتبه زمانی عمل assign پیاده‌سازی شده توسط شما را بررسی می‌کند. عمل درج پیاده‌سازی شده توسط شما باید از مرتبه زمانی O(1) باشد. نمره این تست، 10٪ از نمره کل است.
3. testRemoving: در این تست ابتدا مقادیری به HashTable اضافه شده و پس از آن تعدادی از داده های موجود در آرایه از آن حذف می‌شوند و در هربار حذف بررسی می‌شود که وضعیت HashTable صحیح بوده و داده ها به درستی جابجا شده اند. این کار بر روی چند HashTable انجام می‌شود. شروطی که باید جهت جابجایی استفاده شود مهم هستند. به این تست نیز 10٪ از نمره را اختصاص داده ایم.
4. testRemovingOrder: این تست، مرتبه زمانی عمل remove پیاده‌سازی شده توسط شما را بررسی می‌کند. عمل حذف پیاده‌سازی شده توسط شما باید از مرتبه زمانی O(1) باشد. نمره این تست، 10٪ از نمره کل است.
5. testWithHashFunc: در این تست بررسی می‌کنیم که به درستی از تابع hashFunc استفاده شود. این تست 10٪ از نمره را به خود اختصاص می‌دهد.
6. testWithHashFunc2: در این تست بررسی می‌کنیم که به درستی از تابع hashFunc2 استفاده شود. این تست 10٪ از نمره را به خود اختصاص می‌دهد.
7. testMap: در این تست، توابع hasKey و []operator پیاده‌سازی شده توسط شما، مورد ارزیابی قرار می‌گیرد. به دلیل اهمیت این تست، 30٪ نمره کل به این تست اختصاص پیدا می‌کند.
8. testTemplate: این تست بررسی می‌کند که آیا کد نوشته شده توسط شما می‌تواند داده هایی با نوع داده ای (type data) به غیر از اعداد (int) را نیز مرتب سازی کند یا خیر. این تست شامل 10٪ از نمره ی کل است.

* پیشنیاز قبولی در تست testOrderAssign، قبولی در تست testAssign است.

* پیشنیاز قبولی در تست testOrderRemoving، قبولی در تست testRemoving است.

* پیشنیاز قبولی در تست testMap، قبولی در تمام تست های دیگر است.

* توجه داشته باشید که در صورت استفاده از نوع متغییر هایی به جز template، با خطای کامپایل مواجه می‌شوید.

* تمام کتابخانه های مورد نیاز برا ی حل مسأله در اختیار شما قرار گرفته است. افزودن کتابخانه جدید موجب رخ دادن Error compile در مرحله تصحیح کد می‌شود. بنابراین مجاز به استفاده از کتابخانه های متفرقه نیستید.

برای بارگذاری این تمرین گام‌های زیر را دنبال کنید :

1- ابتدا فایل info.txt را با مشخصات خود پر کنید.
2- پس از حل تمرین، از پوشه src همه فایل های اضافی که به دلیل کامپایل برنامه بوجود آمده اند را پاک نمایید. (ممکن است IDE شما به طور خودکار، فایل‌هایی را اضافه کند). در نهایت فقط فایل‌هایی که از ابتدا در پوشه src وجود داشته‌اند، همچنان باقی می‌مانند.
3- پوشه src و فایل info.txt را در کنار این پوشه، زیپ کنید. مطمئن شوید که وقتی فایل zip را باز می کنید پوشه src و همچنین فایل info.txt را می بینید.
4- دقت کنید که پسوند فایل شما حتما zip باشد و حجم فایل بالای یک مگابایت نباشد.
5- فایل را در سامانه بارگذاری کنید.
6- اشکالاتی را که سامانه مشخص کرده است برطرف نمایید و مجددا تمرین را در سامانه بارگذاری کنید.
7- مرحله قبل را آن قدر ادامه دهید که از صحت عملکرد برنامه خود اطمینان حاصل نمایید.

با آرزوی موفقیت