دانشکده ریاضی و علوم کامپیوتر

برنامه‌سازی پیشرفته و کارگاه

سرچ

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

علی توفیقی، صالح ملازاده، محمدرضا شیخ الاسلامی

بهار ۱۴۰۳

فهرست

سرچ یعنی چی ؟

جستجو یا سرچ یکی از کارهای پایه‌ای توی علوم کامپیوتره که هدفش پیدا کردن یه مقدار خاص بین کلی داده‌ست. این داده‌ها می‌تونن توی آرایه، لیست، درخت یا هر ساختار داده‌ی دیگه‌ای باشن.
جستجو همه جا استفاده می‌شه؛ از پیدا کردن شماره تلفن توی دفترچه مخاطبین گرفته تا سرچ کردن یه محصول خاص توی فروشگاه‌های آنلاین. به زبان ساده، یه الگوریتم جستجو یه لیست یا آرایه رو به عنوان ورودی می‌گیره و همین طور یه مقدار خاص (همون چیزی که دنبالشیم) رو می‌گیره. بعد، شروع می‌کنه به بررسی عناصر یکی‌یکی یا با یه روش خاص، تا ببینه اون مقدار توی مجموعه هست یا نه و اگه هست، دقیقاً کجاست. حالا الگوریتم‌های جستجو انواع مختلفی دارن که هرکدوم برای ساختارهای داده‌ای و کاربردهای متفاوتی مناسب هستن و برای اینکه الگوریتم‌های جستجو رو درست طراحی کنیم و بهترین روش رو انتخاب کنیم، باید ویژگی‌های جستجو رو بدونیم. حالا می‌خوایم چندتا از ویژگی های اصلی الگوریتم های جستجو رو با هم ببینیم :

مقدار هدف (Target Element):

تو جستجو همیشه یه مقدار مشخصی داریم که دنبال اون هستیم. این مقدار می‌تونه هر چیزی باشه، مثلاً یه عدد، یه رکورد یا حتی یه کلید خاص (داخل دیتابیس با این موضوعات بیشتر آشنا می‌شید)

فضای جستجو (Search Space):

فضای جستجو یعنی تمام داده‌هایی که می‌خواهیم توش دنبال مقدار هدف بگردیم. بسته به ساختار داده‌ای که داریم، اندازه و نوع سازماندهی این فضای جستجو می‌تونه متفاوت باشه.

پیچیدگی (Complexity):

جستجو می‌تونه بسته به نوع الگوریتم و داده‌ها زمان و حافظه مختلفی بخواد،. یعنی یه جستجو ممکنه خیلی سریع باشه و یکی دیگه زمان بیشتری بگیره.

حالا تو این داک ما ۲ تا از معروف ترین الگوریتم های سرچ رو با هم بررسی می کنیم که اولیش **جستجوی خطی** یا همون **Linear Search** هستش :

جستجوی خطی

جستجوی خطی یک الگوریتم ساده و پایه‌ای برای پیدا کردن یه عنصر خاص توی یه مجموعه داده‌ست. توی جستجوی خطی، به ترتیب همه‌ی عناصر رو یکی یکی چک می‌کنیم. یعنی از اول آرایه شروع می‌کنیم و میریم جلو تا زمانی که عنصر مورد نظر رو پیدا کنیم. هر عنصر رو بررسی می‌کنیم و اگر پیدا شد، همون رو برمی‌گردونیم. اگر هم پیدا نشد، ادامه می‌دیم تا انتهای آرایه.

ایده‌ی اصلی جستجوی خطی:

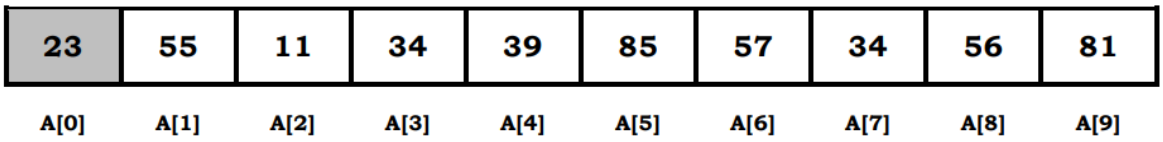
- اول، باید بگیم دنبال چی می‌گردیم.
- بعد شروع می‌کنیم به مقایسه عنصر جستجو با اولین عنصر لیست. اگه پیدا کردیم، می‌گیم "عنصر پیدا شد" و تموم می‌کنیم.
- اگه پیدا نکردیم، می‌ریم سراغ عنصر بعدی و همینطور ادامه می‌دیم.
- این کار رو تا آخر لیست تکرار می‌کنیم.
- اگه به آخر لیست رسیدیم و هنوز پیدا نکردیم، می‌گیم "عنصر پیدا نشد".

حالا بیاید نحوه ی کار این الگوریتم رو توی یک مثال با هم ببینیم :

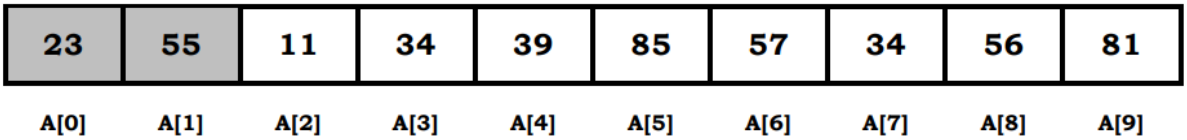
فرض کنید یک آرایه ای از ۱۰ تا عدد داریم و می خوایم عدد **۳۴** رو داخلش پیدا کنیم :

23	55	11	34	39	85	57	34	56	81
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

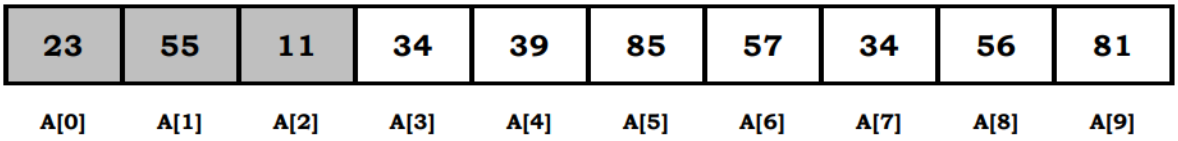
مرحله ۱ : عدد **۳۴** رو با اولین عضو آرایه که در اندیس **۰** قرار داره مقایسه می کنیم . چون ۲۳ با ۳۴ برابر **نیست** سراغ عدد بعدی میریم:



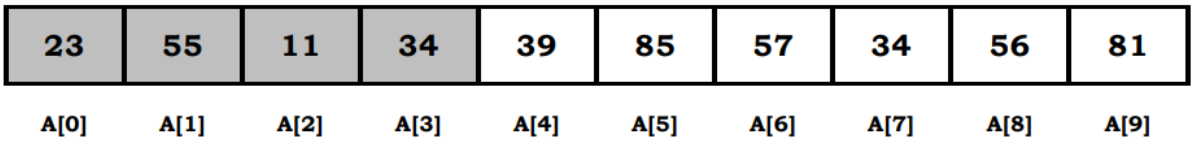
مرحله ۲ : عدد **۳۴** رو با دومین عضو آرایه که در اندیس **۱** قرار داره مقایسه می کنیم و چون ۵۵ با ۳۴ برابر نیست سراغ عدد بعدی میریم :



مرحله ۳ : عدد **۳۴** رو با سومین عضو آرایه که در اندیس **۲** قرار داره مقایسه می کنیم و چون ۱۱ با ۳۴ برابر نیست سراغ عدد بعدی میریم :



مرحله ۴ : در نهایت عدد **۳۴** رو با چهارمین عضو آرایه که در اندیس ۳ قرار داره مقایسه می کنیم و چون ۳۴ با ۳۴ برابره در همین مرحله الگوریتم متوقف میشه و اندیس **۳** رو به عنوان خروجی برمی گردونه



حالا فرض کنید می‌خواستیم عدد **۸۱** رو داخل همین آرایه پیدا کنیم . به نظرتون باید تا کجا پیش می رفتیم ؟ یا اینکه فرض کنید می‌خواستیم عدد **۹۰** رو داخل این آرایه پیدا کنیم . به نظرتون الگوریتم تا کجا پیش می‌ره و چه خروجی باید بده ؟

حالا بیاید با هم **پیاده سازی** این الگوریتم رو داخل جاوا ببینیم :

```
public class LinearSearch {
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] numbers = {23,55,11,34,39,85,57,34,56,81};
        int[] targets = {34,81,90};

        for (int target : targets) {

            int result = linearSearch(numbers, target);
            if (result != -1)
                System.out.println("Value " + target + " found at index " +
result);
            else
                System.out.println("Value " + target + " not found ");

        }

    }
}
```

تو این کد تابع **LinearSearch** یه آرایه و یه عدد (مقدار مورد نظر) رو می‌گیره و با یه for ساده، خونه‌های آرایه رو یکی‌یکی چک می‌کنه. حالا اگه اون عددی که دنبالشیم پیدا شد، ایندکسش رو برمی‌گردونیم (یعنی موقعیتش توی آرایه) ولی اگه کل آرایه رو گشتیم و چیزی پیدا نکردیم، **۱-** برمی‌گردونیم که یعنی «پیدا نشد».

تحلیل پیچیدگی

پیچیدگی زمانی الگوریتم جستجوی خطی در بهترین حالت $O(1)$ هستش یعنی اگر عضو مورد نظر تو اولین مرحله پیدا بشه (یعنی این عضو در اندیس ۰ قرار داشته باشه) ، زمان جستجو ثابت هست. در حالت میانگین و بدترین حالت، پیچیدگی زمانی $O(n)$ میشه، چون ممکنه بخوایم تمام عناصر لیست رو بررسی کنیم و اون عضو مورد نظر آخرین عضو آرایه باشه یا اینکه اصلا داخل آرایه وجود نداشته باشه.

کی از جستجوی خطی استفاده کنیم ؟

1. **وقتی که داده‌ها کم باشند:** جستجوی خطی برای مجموعه‌های داده **کوچیک** مناسبه چون در این حالت سرعت و کارایی مسئله‌ساز نمی‌شود و جستجو سریع‌تر انجام می‌شود.
2. **وقتی که داده‌ها نامرتب باشند:** جستجوی خطی زمانی مفیده که داده‌ها به ترتیب خاصی مرتب نباشند. تو این حالت، هیچ الگوریتم دیگه ای که نیاز به داده‌های مرتب داشته باشه، کاربردی نداره، بنابراین جستجوی خطی بهترین انتخابه.

جستجوی دودویی

یکی از روش‌های معروف و پرکاربرد دیگه سرچ، جستجوی دودویی یا همون Binary Searchـه. این الگوریتم مخصوص وقتیه که آرایه‌مون مرتب باشه، یعنی مثلا از کوچیک به بزرگ یا از بزرگ به کوچیک چیده شده باشه.

فلسفه‌ی کارش اینه که به جای اینکه دونه‌دونه بره سراغ همه‌ی عناصر (مثل جستجوی خطی)، از وسط آرایه شروع می‌کنه و هر بار محدوده‌ی جستجو رو نصف می‌کنه. یعنی مثلا نگاه می‌کنه وسط آرایه چیه، اگه همونی بود که می‌خواستیم که تمومه! ولی اگه نبود، چون آرایه مرتبه، راحت می‌فهمیم که باید سمت چپ بگردیم یا سمت راست. این کار رو انقدر ادامه می‌ده تا یا عنصر مورد نظر پیدا بشه یا بفهمیم که اصلا توی آرایه وجود نداره؛ نکته جالب قضیه اینجاست که اگه اون عنصر رو پیدا نکرد، علاوه بر اعلام عدم موجودی، می‌تونه جای قرارگیری اون عنصر رو بهت اعلام کنه!

برتری‌ای که این الگوریتم نسبت به الگوریتم جستجوی خطی، سرعت اونه که در ادامه بیشتر راجع بهش صحبت می‌کنیم.

چرا مرتب بودن لازمه؟

حالا شاید برات سوال بشه که اصلاً چرا باید آرایه مرتب باشه؟ چرا جستجوی دودویی بدون مرتب بودن جواب نمی‌ده؟

داستان اینه که جستجوی دودویی قراره بر اساس ترتیب تصمیم بگیره که سمت چپ رو بگرده یا سمت راست رو. مثلا وقتی وسط آرایه رو نگاه می‌کنیم و می‌بینیم عددش از چیزی که دنبالشیم کوچیک‌تره، سریع می‌فهمیم که عدد مورد نظر اگه وجود داشته باشه حتماً باید توی نیمه‌ی راست باشه. چون عددها مرتب هستن و سمت چپ همشون از عدد وسط کوچیک‌ترن.

ولی اگه آرایه مرتب نباشه دیگه این منطق بهم می‌ریزه و هیچ تضمین و قطعیتی درباره مقادیر سمت چپ و راست هر درایه وجود نداره، یعنی ممکنه عددی که دنبالشیم توی سمت چپ باشه ولی بزرگ‌تر از عدد وسط باشه و ما اشتباهی سمت راست بگردیم و هیچ‌وقت پیداش نکنیم.

با این توضیحات می‌تونیم الگوریتم جستجوی دودویی رو به صورت زیر خلاصه کنیم:

اول از همه میایم و عنصر وسط آرایه رو پیدا می‌کنیم و با مقداری که دنبالشیم مقایسه می‌کنیم؛ اگه دقیقا همون عدد بود، کارمون تمومه و پیداش کردیم. اگه نه، چک می‌کنیم ببینیم عدد وسط بزرگ‌تره یا کوچیک‌تر از چیزی که می‌خوایم؛ اگه کوچیک‌تر بود یعنی باید بریم توی نیمه دوم آرایه دنبالش بگردیم و اگه بزرگ‌تر بود یعنی باید بریم سراغ نیمه اول.

بعدش دقیقا همین کار رو روی اون نیمه‌ای که انتخاب کردیم تکرار می‌کنیم؛ یعنی دوباره عنصر وسط اون بخش جدید رو پیدا می‌کنیم و همون مقایسه‌ها رو انجام می‌دیم.

این روند همین‌طوری ادامه پیدا می‌کنه، هی آرایه نصف می‌شه و نصف می‌شه، تا یا بالاخره عدد مورد نظرمون رو پیدا کنیم و یا اینکه محدوده جستجو اونقدر کوچیک شه که دیگه چیزی برای گشتن باقی نمونه و بفهمیم که اصلاً عدد مورد نظر توی آرایه نیست.

خب حالا بیاید با یه مثال این الگوریتم رو دنبال کنیم:

فرض کنید یه آرایه مرتب داریم مثل این:

2	5	8	12	16	23	38	56	72	91

حالا فرض کنید که دنبال مقدار 38 هستیم. ابتدا وسط آرایه رو پیدا می‌کنیم و با 38 مقایسه می‌کنیم:

2	5	8	12	16	23	38	56	72	91

می‌بینیم که 16 کوچکتر از 38 عه، پس باید نیمه دوم آرایه رو بررسی کنیم:

2	5	8	12	16	23	38	56	72	91

می‌بینیم که وسط ناحیه جدید برابر 56 عه و از 38 بزرگتره، پس باید نیمه اول ناحیه جدید رو بررسی کنیم:

2	5	8	12	16	23	38	56	72	91

23 کوچکتر از 38 عه، پس باید بریم نیمه دوم ناحیه فعلی رو بررسی کنیم:

2	5	8	12	16	23	38	56	72	91

و به همین ترتیب به عدد 38 رسیدیم و اندیس شماره 6 رو اعلام می‌کنیم.

حالا فرض کنید دنبال مقدار 40 بودیم، اونوقت به نظرتون الگوریتم چجوری پیش می‌رفت؟ مشخصا مقدار توی آرایه نیست، به نظرتون الگوریتم جستجوی دودویی چطور می‌تونه به ما بگه که این مقدار کجا باید قرار بگیره؟

یکم با این مثال بیشتر کار کنین و سعی کنین که یه شهود کامل از نحوه اجرای این الگوریتم داشته باشین؛ اگه بخواین این الگوریتم رو با جاوا پیاده‌سازی کنین، چجوری پیاده‌سازیش می‌کنین؟

پیاده‌سازی

این الگوریتم را می‌تونین به دو روش بازگشتی و غیربازگشتی پیاده‌سازی کنین؛ اما هدف ما اینه که کدی بنویسیم که برای هر نوع داده‌ای (اعداد، رشته‌ها، یا حتی کلاس‌های سفارشی) کار کنه. برای پیاده‌سازی عمومی این الگوریتم، از دو مفهوم کلیدی در جاوا استفاده می‌کنیم: Generics و اینترفیس Comparable.[1]

با Generics که قبلا آشنا شدین؛ حالا شاید بپرسین که Comparable چی هست و چجوری این اینترفیس بهمون توی پیاده‌سازی کمک می‌کنه. خب وقتی توی جاوا بخوایم دو تا آبجکت رو با هم مقایسه کنیم، چجوری این کار رو انجام می‌دیم؟ مثلا دو عدد رو با > و < مقایسه می‌کنیم، ولی این برای آبجکت‌ها جواب نمی‌ده! برای اینکه کلاس‌های خودمون رو هم بتونیم قابل مقایسه کنیم، از اینترفیس Comparable استفاده می‌کنیم.

Comparable یه اینترفیس هست که فقط یک متد داره:

```
public interface Comparable<T> {

    public int compareTo(T o);
}
```

این متد یه مقدار اینتجر برمی‌گردونه که نشون می‌ده این آبجکت نسبت به آبجکت دیگه بزرگ‌تره، کوچیک‌تره یا مساویه (عدد مثبت اگه این آبجکت بزرگ‌تر باشه، عدد منفی اگه این آبجکت کوچیک‌تر باشه و صفر اگه دو آبجکت برابر باشن).

خب حالا وقتشه که دست به کد بشین و سعی کنین این الگوریتم رو خودتون پیاده‌سازی کنین!

[1] https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

برای اینکه کدتون مرتب و تمیز باشه و از یه ساختار منسجم پیروی کنه، یه کلاس به اسم BinarySearch بسازین که نشه ازش آبجکت ساخت (متدهاش باید استاتیک باشن). این کلاس دو تا متد Generic داره، یکی برای پیاده‌سازی جستجوی دودویی به روش بازگشتی و یکی هم برای روش غیر بازگشتی. متدها در صورت یافتن مقدار خواسته شده، باید مقدار ایندکس و اگر مقدار موجود نبود، عدد 1- رو برگردونن.

برای پیاده‌سازی جستجوی دودویی به شکل کلی اول از همه، باید چک کنین که آیا نوع داده‌ای که قراره روی اون جستجو انجام بشه، Comparable رو پیاده‌سازی کرده یا نه (این کار مهمه، چون فقط در این صورته که می‌تونین از متد compareTo استفاده کنین). موقع پیاده‌سازی، حواستون باشه که آرایه‌ای که ورودی می‌گیرین، باید حتما مرتب شده باشه، وگرنه الگوریتم جواب درستی نمی‌ده.

در نهایت کدتون باید تا حد خوبی مشکلات احتمالی و اکسپشن‌های مربوطه رو هندل کنه تا برنامه موقع اجرا به مشکل نخوره.

کدتون رو با ساختار توضیح داده شده داخل کوئرا آپلود کنین.

تحلیل پیچیدگی

این الگوریتم خیلی هوشمندانه عمل می‌کنه چون هر بار که یه مقدار رو با وسط آرایه مقایسه می‌کنه، نصف آرایه رو کنار می‌گذاره. پس به جای اینکه همه رو چک کنه، جستجو رو فقط توی نیمی از آرایه ادامه می‌ده. این یعنی تعداد دفعاتی که باید بررسی کنه خیلی کمتر میشه. همونطور که دیدین هم مقدار داده‌ها هر بار نصف می‌شه؛ یعنی مقدار داده‌هایی که داریم (n) هر بار داره به طور نمایی کاهش پیدا می‌کنه.

حالا با این توضیحات، بهترین حالت وقتیه که مقدار مورد نظر ما همون درایه وسط آرایه باشه $(O(1))$ و بدترین حالت $(O(\log(n)))$. شاید بپرسید که $O(\log(n))$ از کجا اومد؟

خب بیاید فرض کنید که داده‌ای که دنبالشیم پس از k بار نصف شدن بدست میاد، حالا از اونجایی که بدترین حالت رو بررسی می‌کنیم، پس در واقع پس از k بار نصف کردن به آرایه‌ای به طول یک می‌رسیم:

$$1 = \frac{n}{2^k}$$

و به همین ترتیب داریم:

$$n = 2^k \rightarrow k = log_2^n = O(\log(n))$$

بررسی دقیق‌تر این اثبات رو توی درس ساختمان‌داده و الگوریتم‌ها خواهید خوند؛ اگه دوست داشتید هم بیشتر درموردش بخونید، می‌تونید از این منبع هم استفاده کنید.

تفاوت جستجوی دودویی با جستجوی خطی

خب، حالا می‌خوایم تفاوت جستجوی دودویی و جستجوی خطی رو بررسی کنیم. جستجوی خطی ساده عمل می‌کنه؛ یعنی شما از اول آرایه شروع می‌کنید و تک تک عناصر رو بررسی می‌کنید تا ببینید آیا مقدار مورد نظر رو پیدا می‌کنید یا نه. از طرفی جستجوی دودویی یه الگوریتم خیلی هوشمندانه‌تره که **از اون فقط وقتی می‌تونیم استفاده کنیم که آرایه مرتب باشه**. این الگوریتم به جای اینکه همه‌ی عناصر رو یکی یکی بررسی کنه، هر بار آرایه رو نصف می‌کنه و با مقدار وسط هر ناحیه مقایسه رو انجام می‌ده.

با این توضیحات، جستجوی خطی ساده است و همه‌ی عناصر آرایه رو یکی یکی بررسی می‌کنه، بنابراین پیچیدگی زمانی‌اش O(n) هست. اما جستجوی دودویی تنها برای آرایه‌های مرتب استفاده میشه و با نصف کردن آرایه در هر مرحله، زمان کمتری می‌گیره و پیچیدگی زمانی‌اش O(log n) هست. در نتیجه، اگر داده‌ها مرتب نباشن، از جستجوی خطی استفاده می‌کنیم، اما اگر مرتب باشن، جستجوی دودویی سریع‌تر عمل می‌کنه.

چه چیزی یادگرفتیم؟

توی این داک فهمیدیم که:

- جستجو یا سرچ یه مقدار خاص توی مجموعه داده‌ها چیه و چرا این کار خیلی مهمه.
- جستجوی خطی (Linear Search) چیه، چطور کار می‌کنه و کی باید ازش استفاده کنیم.
- جستجوی دودویی (Binary Search) چیه، چطور کار می‌کنه و فقط روی آرایه‌های مرتب میشه استفاده کرد.
- مزیت‌ها و محدودیت‌های جستجوی دودویی نسبت به خطی و در چه شرایطی جستجوی دودویی بهتره.
- پیچیدگی زمانی هر دو الگوریتم جستجو (O(n) برای خطی و O(log n) برای دودویی).