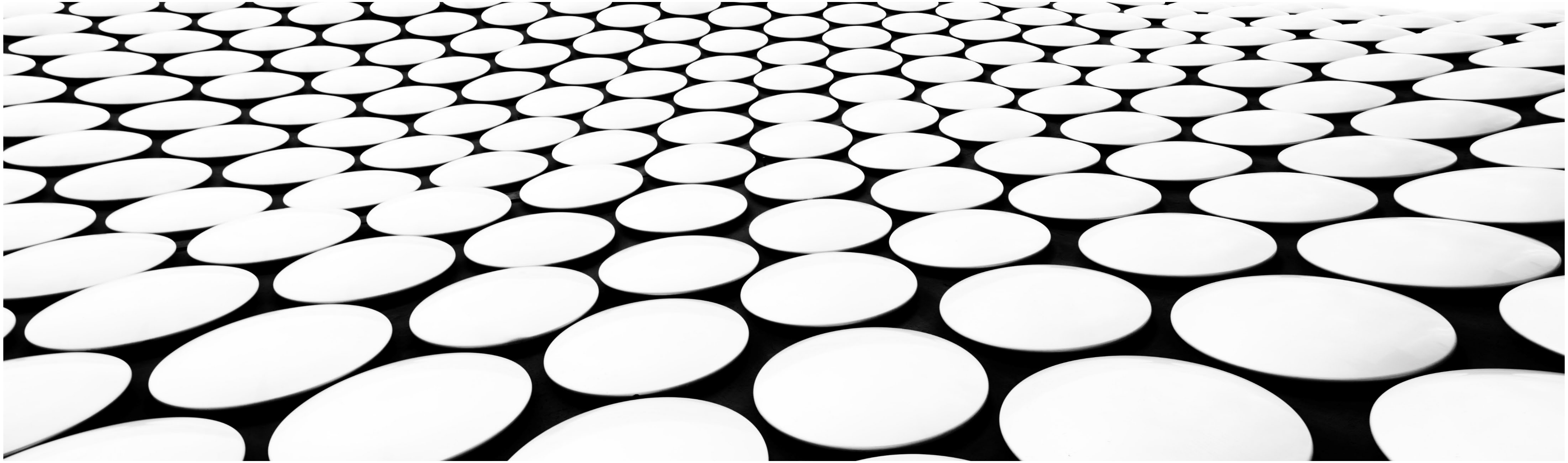
الگوریتم درخت جست و جوی دودویی بهینه

با استفاده روش برنامه نویسی پویا



✓ فهرستی ازعنوان هایی که به آنها خواهیم پرداخت...

✓ تعریف درخت جست و جوی دودویی (BST)

درختی دودویی از کلید ها، را با شرایط زیر یک درخت جست و جوی دودویی می نامیم که:

- هر گره حاوی یک کلید است.
- کلید های موجود در زیر درخت چپ یک گره مفروض از این درخت، کوچک تر از کلید آن گره هستند.
- کلید های موجود در زیر درخت راست یک گره مفروض از این درخت، بزرگ تر از کلید آن گره هستند.

ما به دنبال درخت جست و جویی هستیم که بتوانیم با حداقل مقایسه، کلید مدنظر را پیدا کنیم.

✓ بهینه سازی درخت جست و جوی دودویی

برای یافتن درخت جست و جوی بهینه با n گره، می توانیم تمامی درخت های جست و جوی که می توان با n گره ساخت را در نظر بگیریم، سپس با محاسبه زمان میانگین برای هر درخت و مقایسه آنها باهم، درخت بهینه را از میان آنها پیدا کنیم.

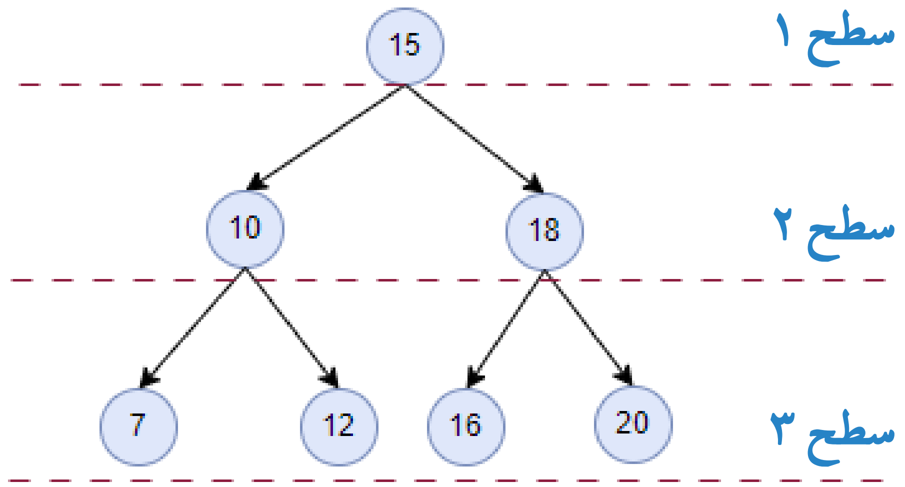
تعداد گره ها n, احتمال گره p_i, سطح گره c_i

رابطه محاسبه زمان میانگین مقایسه

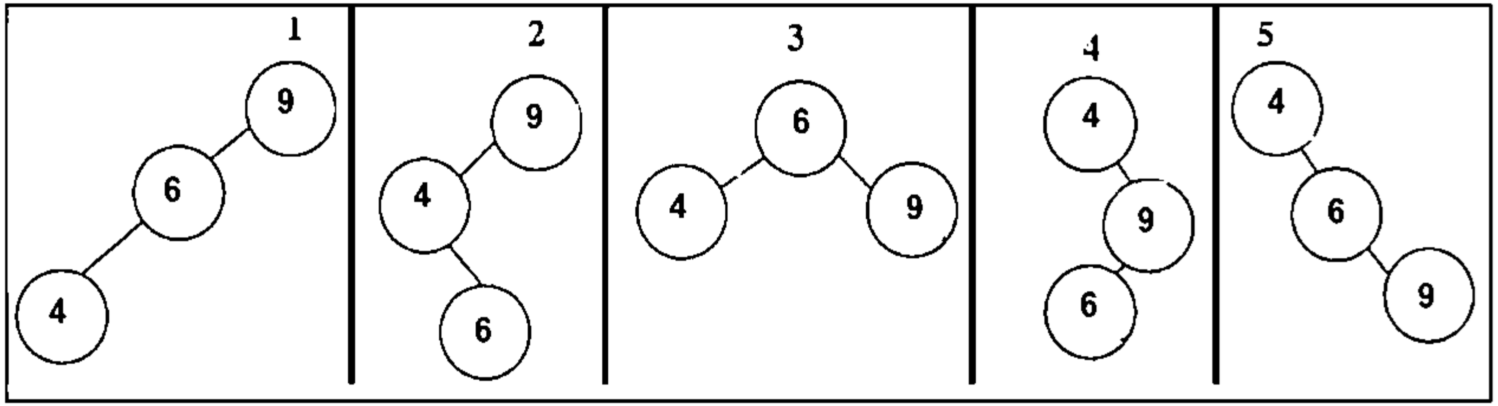
$$\sum_{i=1}^{n} c_i p_i$$

رابطه محاسبه تعداد کل درخت ها

$$\frac{\binom{2n}{n}}{n+1}$$



مثال) درخت جست و جوی بهینه با گره های ۹ و ٦ و ۲ به ترتیب با احتمال های ۰.۷ و ۰.۲ و ۰.۱ را پیدا کنید؟



$$\frac{\binom{2n}{n}}{n+1} = \frac{\binom{6}{3}}{4} = 5$$

$$\sum_{i=1}^{n} c_i p_i$$

$P_1 = 0.7$

$P_2 = 0.2$

$P_3 = 0.1$

1. $3(0.7) + 2(0.2) + 1(0.1) = 2.6$
2. $2(0.7) + 3(0.2) + 1(0.1) = 2.1$
3. $2(0.7) + 1(0.2) + 2(0.1) = 1.8$
4. $1(0.7) + 3(0.2) + 2(0.1) = 1.5$
5. $1(0.7) + 2(0.2) + 3(0.1) = 1.4$ ✓

با روش ارائه شده می توانیم درخت بهینه مطلوب را پیدا کنیم اما با بررسی پیچدگی زمانی آن (تعداد درخت ها) متوجه می شویم، این روش اصلاً بهینه نیست.

$$\begin{cases} T(n) = T(0)\times T(n-1)+T(1)\times T(n-2)+\ldots+T(n-1)\times T(0) = \sum_{i=0}^{n-1} T(i)\times T(n-i-1) \\ T(0) = 1 \end{cases}$$

از آنجایی که در این روش بازگشتی، مقادیر $T(i)$ برای i از 0 تا n-1 ، بارها محاسبه می‌شوند و این محاسبات به صورت بازگشتی انجام می‌شوند، پیچیدگی زمانی آن به طور نمایی است؛ لذا برای مقادیر بزرگ n نامناسب خواهد بود.

بنابرین به طور کلی، نمی توان درخت جست و جوی بهینه را با در نظر گرفتن همه درخت ها به دست آورد اما با توجه به اصل بهینگی و به کار بردن برنامه نویسی پویا می توان الگوریتمی بهینه ارائه کرد.

✓ الگوریتم درخت جست و جوی دودویی بهینه

اصل بهینگی درخت جست و جو

هر زیردرخت مفروض از یک درخت جست و جو بهینه، بهینه خواهد بود.

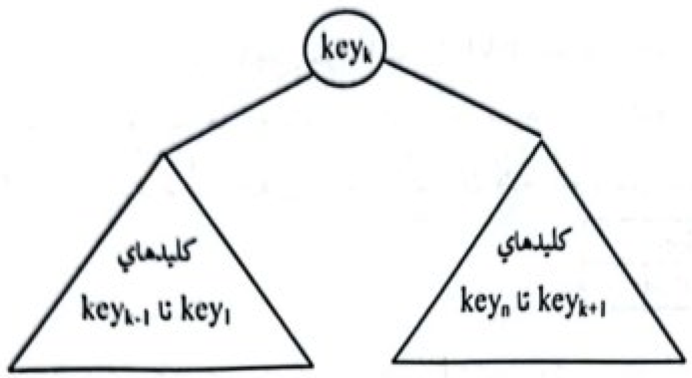
درختی بهینه ای با ریشه key_k در نظر می گیریم، انگاه بر اساس اصل بالا زیر درخت های چپ و راست نیز بهینه هستند.

برنامه نویسی پویا

برای ذخیره زمان میانگین مقایسه و کلید ها به ترتیب به دو ماتریس $[1,\ldots,n+1][0,\ldots,n]_{(n+1*n+1)}$ A,R نیاز خواهیم داشت

به صورتی که:

$$\begin{cases} A[i][j] = \text{مینیمم میانگین زمان جستجو} \\ 1 \le i \le j \le n \,, key_i \le key_j \end{cases}$$



اگر این یک درخت جست و جوی بهینه باشد آنگاه A[1][n] برابر خواهد بود با:

$$A[1][K-1]+A[K+1][n]+(p_1+\ldots+p_{k-1})+p_k+(p_{k+1}+\ldots+p_n) =$$
$$A[1][K-1]+A[K+1][n]+\sum_{m=1}^{n} p_m$$

رابطه فوق، وقتی است که فرض کرده ایم key_k ریشه باشد ولی هر یک کلید های می توانند ریشه باشند که جواب نهایی، مینیمم آنها خواهد بود:

$$\begin{cases} A[i][j] = Min(A[i][k-1] + A[K+1][j]) + \sum_{m=i}^{j} p_m \quad i \le k \le j \;\; ; \;\; (i<j) \\ A[i][j] = P_i \quad (i=j) \end{cases}$$

ما رابطه زیر برای ساختن ماتریس A به کار می بریم که قطر اصلی آن را صفر در نظر گرفته و فقط بالای قطر اصلی را مقدار دهی میکنیم

همگام با ساخت A، ماتریس R نیز ظاهر می شود به طوری که R[i][j] نمایانگر کلید است که به عنوان ریشه انتخاب شده برای مثال

R[3][5] اندیس ریشه درخت بهینه ای است که از کلید های key_5 key_4 key_3 تشکیل شده است.

مثال) درخت جست و جوی بهینه با گره های ۹ و ٦ و ۲ به ترتیب با احتمال های ۰.۷ و ۰.۲ و ۰.۱ را پیدا کنید؟

key1=4
key2=6
key3= 9

طبق رابطه که برای کامل کردن ماتریس ها داریم
قطر اول از ۱ تا n را با p_i گره ها پر می کنیم

$P_1 = 0.7$
$P_2 = 0.2$
$P_3 = 0.1$

قطر اول

	0	1	2	3
1	0	0.7		
2		0	0.2	
3			0	0.1
4				0

$$A[i][j] = \underset{i \le k \le j}{\text{minimum}}(A[i][k-1] + A[k+1][j]) + \sum_{m=i}^{j} p_m \qquad i < j$$

قطر دوم

	0	1	2	3
1	0	0.7	1.1	
2		0	0.2	
3			0	0.1
4				0

$$k = 1 \Rightarrow A[1][2] = A[1][0] + A[2][2] + P_1 + P_2 = 0 + 0.2 + 0.7 + 0.2 = 1.1 \quad \checkmark$$

$$k = 2 \Rightarrow A[1][2] = A[1][1] + A[3][2] + P_1 + P_2 = 0.7 + 0 + 0.7 + 0.2 = 1.6$$

$$A[i][j] = \min_{i \le k \le j} imum(A[i][k-1] + A[k+1][j]) + \sum_{m=i}^{j} p_m \qquad i < j$$

	0	1	2	3
1	0	0.7	1.1	
2		0	0.2	0.4
3			0	0.1
4				0

قطر دوم

$$k = 2 \Rightarrow A[2][3] = A[2][1] + A[3][3] + P_2 + P_3 = 0 + 0.1 + 0.2 + 0.1 = 0.4 \quad \checkmark$$

$$k = 3 \Rightarrow A[2][3] = A[2][2] + A[4][3] + P_2 + P_3 = 0.2 + 0 + 0.2 + 0.1 = 0.5$$

$$k=1 \Rightarrow A[1][3]=A[1][0]+A[2][3]+P_1+P_2+P_3=0+0.4+0.7+0.2+0.1=1.4$$

$$k=2 \Rightarrow A[1][3]=A[1][1]+A[3][3]+P_1+P_2+P_3=0.7+0.1+0.7+0.2+0.1=1.8$$

$$k=3 \Rightarrow A[1][3]=A[1][2]+A[4][3]+P_1+P_2+P_3=1.1+0+0.7+0.2+0.1=2.1$$

قطر سوم

	0	1	2	3
1	0	0.7	1.1	1.4
2		0	0.2	0.4
3			0	0.1
4				0

	0	1	2	3
1	0	1	1	1
2		0	2	2
3			0	3
4				0

ماتریس R

✓ شبه کد الگوریتم

```
optsearchtree(n, p[], minavg, R[]) {
    float A[1...n + 1][0...n];
    for (i = 1; i <= n; i++) {
        A[i][i - 1] = 0;
        A[i][i] = p[i];
        R[i][i] = i;
        R[i][i - 1] = 0;
    }
    A[n + 1][n] = 0;
    R[n + 1][n] = 0;
    for (t = 1; t <= n - 1; t++) {
        for (i = 1; i <= n - t; i++) {
            j = i + t;
            A[i][j] = minimum(i <= k <= j)(A[i][k - 1] + A[k + 1][j]) + Σm=i,j p(mi);
            R[i][j] = a value of k that gave the minimum;
        }
    }
    minavg = A[1][n];
}
```

✓ پیچدگی زمانی الگوریتم

با دقت در شبه کد الگوریتم میبینیم رابطه بازگشتی برای پر کردن دو ماتریس A,R درون سه حلقه تودرتو می باشد، بنابرین داریم:

$$\sum_{t=1}^{n-1}\sum_{i=1}^{n-t}\sum_{k=i}^{j-1}1=\sum_{t=1}^{n-1}\sum_{i=1}^{n-t}(j-1+1)=\sum_{t=1}^{n-1}\sum_{i=1}^{n-t}(i+t-i+1)=\sum_{t=1}^{n-1}\sum_{i=1}^{n-t}(t+1)=\sum_{t=1}^{n-1}(t+1)(n-t)=\frac{n(n-1)(n+4)}{6}$$

$$\in\theta(n^3)$$

همانطور که مشاهد می کنید پیچدگی $\theta(n^3)$ خیلی بهینه تر از حالت پیشین است که دارای پیچدگی نمایی بود.

✓ منابع و لینک ها

- https://www.youtube.com/watch?v=vLS-zRCHo-Y
- https://www.aparat.com/v/8HU9K
- کتاب درس و کنکور طراحی الگوریتم‌ها صفحات ۱۹۵ تا ۲۰۱

لینک مخزن گیت هاب برای درس ساختمان داده و طراحی الگوریتم

https://github.com/amircodel/Algorithms.git

باتشکر امیرعلی محمدی، درس طراحی و تحلیل الگوریتم ها، دکتر فرشید نوریان