هفت: درخت‌ها

ساختمان داده ها و الگوریتم

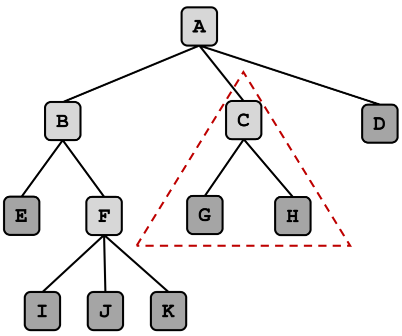
مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

۱. درخت چیست؟

درخت (tree) یک ساختمان داده پر استفاده است که شبیه به یک ساختار درختی با مجموعه‌ای از گره‌های متصل به هم است؛ درحقیقت درخت یک گراف همبند بدون دور است. اکثر نویسندگان این قید را نیز اضافه می‌کنند که گراف باید بدون جهت باشد، به علاوه بعضی قید بدون وزن بودن یال‌ها را نیز اضافه می‌کنند؛ اما لزومی به وجود این دو شرط برای تعریف درخت‌ها نیست.

۱.۱ تعاریف و انواع گره در درخت



هر گره در درخت تعدادی (صفر یا بیشتر) گره **فرزند** دارد، که در زیر آن در درخت قرار دارند (به طور قراردادی، درخت به سمت پایین رشد می‌کند، برخلاف آنچه در طبیعت می بینیم).

یک گره که فرزند دارد گره **پدر** آن فرزند گفته می‌شود؛ یک گره حداکثر ۱ پدر دارد.

ارتفاع یک گره طول طولانی‌ترین مسیر پایین رو از آن گره به یک برگ است.

مسیری که از گره به ریشه وصل می‌شود **مسیر ریشه** نام دارد و طول این مسیر **عمق** آن گره است.

گره‌های ریشه

بالاترین گره درخت گره ریشه نام دارد؛ پس گره ریشه پدر ندارد. این گره گرهی است که عملیات روی درخت معمولاً از آن شروع می‌شود.(هر چند بعضی الگوریتم‌ها از برگ شروع شده و به ریشه ختم می‌شوند). بقیه گره‌ها با دنبال کردن یالها از گره ریشه قابل دسترسی اند. در بعضی درخت ها، گره ریشه ویژگی‌های خاصی دارند. هر گره در یک درخت را می‌توان ریشه یک **زیر درخت** در نظر گرفت.

گره‌های برگ

پایین‌ترین گره‌های یک درخت گره‌های برگ نام دارند؛ چون این گره‌ها زیرترین گره هستند هیچ فرزندی ندارند.

گره‌های داخلی

یک گره داخلی، هر گرهی است که فرزند داشته باشد، پس تنها برگها گره داخلی نیستند.

۱.۲ درخت‌های منظم

به درختی گفته می‌شود که هر گره داخلی آن دقیقا k فرزند داشته باشد؛ برخلاف تعریف عمومی درختان، در درختان kتایی ترتیب قرار گرفتن این فرزندان مهم است.

۲. نمایش درخت‌ها

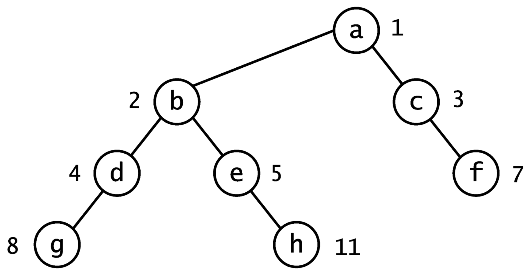
۲.۱ لیست مجاورت

همانطور که دیدیم، به طور کلی برای ذخیره سازی گراف ها از دو روش «ماتریس مجاورت» و «لیست مجاوت» استفاده می‌کردیم، هرچند که روش های دیگری نیز موجود بود، اما برای نمایش گراف های درختی، با توجه به این که تعداد یال ها از $O(n)$ است، برای استفاده از ماتریس مجاورت به یک ماتریس اسپارس می‌رسیم و این یعنی برای ذخیره کردن $O(n)$ داده باید $O(n^2)$ هزینه کنیم که اصلا به‌صرفه نیست(کافیست تا n را ۱۰۰۰ در نظر بگیرید!).

۲.۲ آرایه ها

اما نگاهی دقیق‌تر داشته باشیم، دسته مهم و پر کاربردیی از درختان را درختان منظم تشکیل میدهند، که برای آنها یک فرض مازاد بر درختان داریم، یعنی برای هر راس دقیقا k فرزند داریم که این مقدار ثابت است، پس به شکل زیر میتوانیم عمل کنیم:

- به گره ریشه شماره(اندیس) 0 را می‌دهیم.
- بقیه گره‌های درخت، برای هر گره با شماره v :
- به فرزند iام این راس (در صورت وجود) شماره kv+i را اختصاص می‌دهیم.

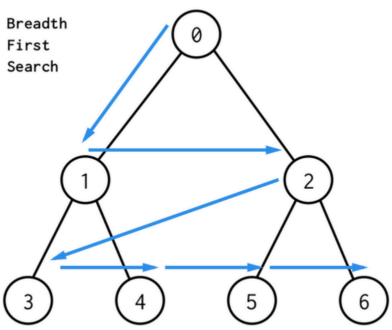


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	d	e	-	f	g	-	-	h	-	-	-	-

۳. پیمایش درخت‌ها

۳.۱ پیمایش سطح اول

دوباره، البته که درخت ها هم یک گراف هستند و روش های بیان شده در باره آن ها هم صادق است؛ پیمایش اول سطح در درخت‌ها همانند گراف‌ها انجام می‌شود و هیچ تفاوتی ندارد،چیز متفاوت یا بیشتری را هم نشان نمی‌دهد!



با توجه به این که مرتبه زمانی این الگوریتم برابر با $O(|V|+|E|)$ است و در درخت ها رابطه $|E|=O(|V|)$ بنابراین این پیمایش در درخت ها مرتبه زمانی برابر با $O(|V|)$ دارد.

۳.۲ پیمایش عمق اول

پیمایش عمق اول نیز در درخت ها همانند گراف ها انجام می‌شود، اما ترتیب پیمایش راس های فرزند و ریشه برای هر زیر درخت، نکته ای است که در پیاده‌سازی و طراحی الگوریتم ها مورد توجه است. به‌طور کلی برای درختان دودویی روش های زیر توسعه داده شده اند، که به راحتی(همانگونه که در پیاده‌سازی ها میبینید) این روش‌ها به سادگی قابل تعمیم به تمامی درختان هستند.

پیمایش پیش‌ترتیب:

۱. ریشه را ملاقات کن.

۲. زیر درخت چپ را پیمایش کن.

۳. زیر درخت راست را پیمایش کن.

```
void preOrder(int x){
	queue.push_back(x);
	for(int i=0;i< children[x].size(); i++){
		preOrder(children[x][i]);
	}
	return;
}
```

پیمایش میان‌ترتیب:

۱. زیردرخت چپ را پیمایش کن.

۲. ریشه را ملاقات کن.

۳. زیردرخت راست را پیمایش کن.

```
void inOrder(int x){
	inOrder(leftChild[x]);
	queue.push_back(x);
	inOrder(rightChild[x]);
	return;
}
```

پیمایش پس‌ترتیب:

۱. زیر درخت چپ را پیمایش کن.

۲. زیر درخت راست را پیمایش کن.

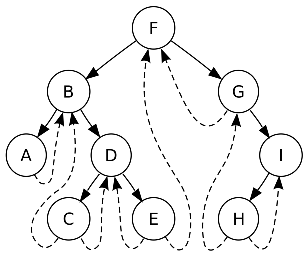
۳. ریشه را ملاقات کن.

```
void postOrder(int x){
	for(int i=0;i< children[x].size(); i++){
		postOrder(children[x][i]);
	}
	queue.push_back(x);
}
```

۳.۳ درخت نخ‌کشی شده (پیمایش موریس)

در یک درخت دودویی از کل اتصالات یعنی 2n، تعداد n+1 اتصال تهی است. از این اتصالات تهی می‌توان برای ارتباط با دیگر گره‌های درخت استفاده کرد. در درختی که اتصالات تهی آن به این صورت استفاده شده‌است، **درخت دودویی نخ‌کشی شده** نامیده می‌شود. یک درخت دودویی را می‌توان به چند روش نخ‌کشی کرد. این نخ‌کشی بسته به روش پیمایش درخت دارد. برای مثال، درخت دودویی نخ‌کشی شده به روش میانوندی به شکل زیر تعریف می‌شود:

«برای هر گره، اتصال تهی سمت راست، به گره بعدی در پیمایش میانوندی اشاره کنند و اتصال تهی سمت چپ، به گره قبلی در پیمایش میانوندی اشاره کنند.»

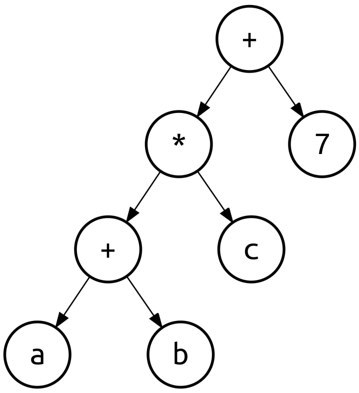


یک درخت دودویی نخ‌کشی شده.

درخت دودویی نخ‌کشی شده این امکان را فراهم می‌سازد که پیمایش مقادیر در درخت دودویی به روش پیمایش خطی، سریعتر از پیمایش بازگشتی درخت به صورت میانوندی صورت گیرد. همچنین پیدا کردن والد یک گره در یک درخت دودویی نخی، بدون استفاده از اشاره‌گرهای والد یا بدون استفاده از پشته هم امکان‌پذیر است، هر چند که این کار کمی آهسته است. این قابلیت وقتی مفید است که فضای پشته اندک باشد. (برای پیدا کردن گره والد به روش الگوریتم جستجوی عمق اول)

۴. کاربردهای درخت

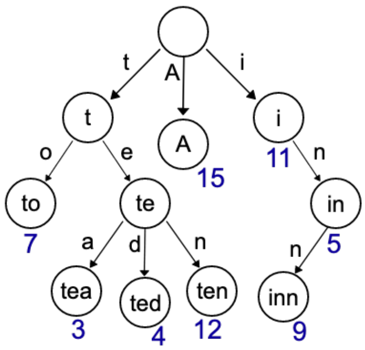
۴.۱. درخت عبارت



برای تجزیه، محاسبه و نمایش یک عبارت ریاضی (جبری، بولی، جبر مجموعه‌ها و ...) شامل ثابت‌ها، متغیرها و عملگرهای یک‌عملونده (قرینه، قدر مطلق، نقیض و ...) و دو عملونده (اشتراک، تقسیم و ...) می توان از یک درخت دودویی بهره برد که به آن درخت عبارت می‌گویند. البته برای عبارت‌هایی که عملگرهایی با بیش از دو عملوند دارند هم می‌توان درخت عبارت تشکیل داد؛ اما به ندرت چنین عملوندهایی استفاده می‌شوند و درخت حاصل دیگر دودویی نخواهد بود.برگ های درخت، بیانگر عملوندها (ثابت‌ها و متغیرها) هستند و هر رأس غیر برگ، یک عملگر را نشان می‌دهد.

درخت عبارت را می‌توان به سه شکل پیش‌ترتیب، میان‌ترتیب و پس‌ترتیب پیمایش کرد که به ترتیب نمایش پیشوندی، میانوندی و پسوندی یک عبارت را تولید می‌کنند.

۴.۲. درخت پیشوندی



ترای یا درخت پیشوندی یک داده‌ساختار درختی است که برای نگاشت‌ها استفاده می‌شود (نگاشت مجموعه‌ای از زوج مرتب‌های (کلید، مقدار) است که هر کلید حداکثر یک‌بار می آید). وقتی از ترای استفاده می کنیم، کلیدها معمولاً رشته هستند. البته اگر نگاشتی در کار نباشد و صرفاً تعدادی رشته داشته باشیم، باز هم می‌توان برای پردازش آن‌ها از ترای استفاده کرد.گره‌ی ریشه نیز یک رشته‌ی خالی است. معمولاً همه گره‌ها مشخص‌کننده‌ی کلیدها نیستند. فقط برگ‌ها و بعضی از گره‌های داخلی با کلیدها مرتبط می‌شوند.

البته یک ترای الزاماً شامل رشته‌های کاراکتری نمی‌باشد؛ بلکه حتی برای جایگشتهای عددی و مواردی از این قبیل هم می‌توان از ترای استفاده کرد.

۴.۳. کدگذاری هافمن

کدهای پیشوندی نوعی از کدها (توالی بیت‌ها) هستند که در آن‌ها کد اختصاص داده شده به یک کاراکتر پیشوند کد تخصیص داده شده به هیچ کاراکتر دیگری نیست. این، روشی است که کدگذاری هافمن با استفاده از آن اطمینان حاصل می‌کند که هیچ ابهامی هنگام رمزگشایی توالی بیت‌های (جریان بیت) تولید شده وجود نخواهد داشت.

فرض می‌شود که چهار کاراکتر a، b، c و d موجود هستند و کدهای طول متغیر متناظر با آن‌ها به ترتیب ۰۰، ۰۱، ۰ و ۱ است. این کدگذاری موجب ابهام می‌شود زیرا کد تخصیص یافته به c، پیشوند کدهای تخصیص یافته به a و b است. اگر جریان رشته فشرده شده ۰۰۰۱ است، خروجی که از حالت فشرده خارج شود امکان دارد cccd یا ccb یا acd یا ab باشد.

مراحل کدگذاری برای این الگوریتم به شرح زیر است:

I. چگالی هر کاراکتر را محاسبه میکنیم (تعداد دفعات حضور کاراکتر در متن مورد نظر).

II. دو کاراکتر با کمترین میزان تکرار (چگالی) را انتخاب میکنیم.

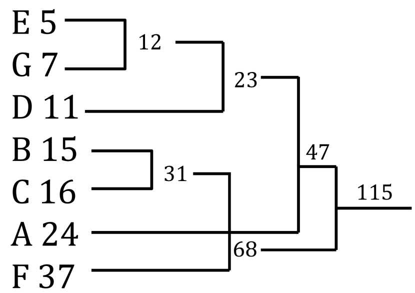
III. کاراکتر های مرحله ۲ را با کاراکتر جدیدی که دارای چگالی برابر با مجموع چگالی دو کاراکتر فوق است جایگزین میکنیم.

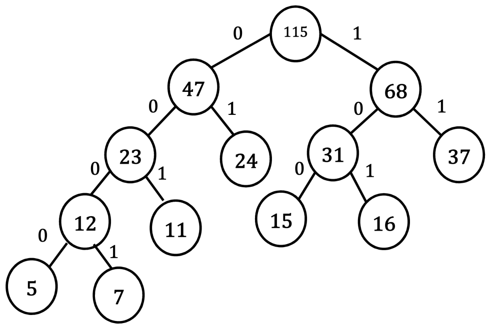
IV. زمانی که فقط یک کاراکتر باقی مانده باشد، به مرحله ۲ میرویم.

V. از عملیات فوق یک درخت حاصل می شود، بر روی این درخت هر مسیر به سمت چپ با 0 و هر مسیر به سمت راست با 1 وزن دهی میشود.

VI. کد هر کاراکتر با کنار هم گذاشتن وزن ها از ریشه تا آن کاراکتر به دست می آید.

حرف	تعداد تکرار
A	24
B	15
C	16
D	11
E	5
F	37
G	7





حرف	تعداد تکرار	مقدار عددی
A	24	01
B	15	100
C	16	101
D	11	001
E	5	0000
F	37	11
G	7	0001

نکته قابل توجه برای کدگذاری هافمن این است که سعی می‌کند با بهینه کردن میانگین طول رشته های کد شده، کمینه طول متن را برای متن کد شده داشته باشد؛ به طور دقیقتر در این الگوریتم مجموعه‌ای از نمادها (حروف و کاراکترها) به همراه وزن هر کدام را به ما ورودی می‌دهند و یک کد دودویی بدون پیشوند با کمترین امید ریاضی برای طول کد را محاسبه کند.

باید دقت کرد با توجه به کارآیی بالای این الگوریتم، کدگذاری هافمن همواره بهینه نیست و در مواری که قصد فشرده‌سازی بهینه‌تری داشته باشیم، می‌توان از الگوریتم‌های کدگذاری حسابی و یا سیستم‌های عددی نامتقارن استفاده کرد.

انواع مختلفی از کد گذاری هافمن وجود دارد، که بعضی از آن‌ها از الگوریتم‌هایی شبیه الگوریتم هافمن و بعضی دیگر از کدهای بهینه پیشوندی (با محدودیت‌های خاص برای خروجی)استفاده می‌کنند. برای مثال می‌توان از کد هافمن n تایی، کد هافمن انطباقی، کد هافمن با طول محدود، کد هافمن با ارزش حرفی متفاوت و ... نام برد.

برای مطالعه بیشتر می‌توانید به اینجا و برای مطالعه صحت درستی این الگوریتم می‌توانید به اینجا مراجعه کنید.

۵. سوالات برنامه‌نویسی

۱. دو درخت، کوئرا

۲. تایپ خسته، کوئرا

۳. ارتباطات فامیلی، کوئرا

۴. چارلی در کازابلانکا، کوئرا

۶. برای جستجو

```
1. Barnes–Hut simulation
2. JSON and YAML
3. Abstract syntax tree
4. Quad Tree and Octree
5. Aho-Corasick algorithm
6. Alpha-beta pruning
7. File system
8. Lowest Commen Ancestor
```