جزوه جلسه بیست و چهارم داده ساختارها و الگوریتم

۲۳ آذر ۱۴۰۰

فهرست مطالب

۱ مجموعه های مجزا

مجموعه های مجزا یک interface با عملیات تعریف شده زیر می باشد:

- ایجاد یک مجموعه جدید
- ادغام دو مجموعه
- پیدا کردن مجموعه مربوط به عنصر داده شده

برای مثال در یک گراف، هر راس را در ابتدا می توان درون یک مجموعه مجزا قرار داد که هیچ یالی به هیچ راسی نیز ندارد. در ادامه با اضافه شدن یالها، می توان مولفه های همبندی (ادغام دو مجموعه) ایجاد کرد. به طور کلی یالها یا بین مولفه های همبندی هستند یا درون رئوس یک مولفه همبند. همچنین می توان پرسید هر راس در کدام مجموعه همبندی است (پیدا کردن مجموعه مربوط به عنصر داده شده)
همچنین فرض های ساده سازی زیر را نیز انجام می دهیم:

- به جای نام مجموعه، برای هر مجموعه یکی از عناصرش را به عنوان نماینده در نظر می گیریم.
- عناصر را اعداد 1 تا n فرض می کنیم. (به طبع، ایجاد مجموعه جدید نداریم و همه اعضا را از قبل داریم) پس هر کدام از اعداد 1 تا n مجموعه اختصاصی خود را دارند.

۱.۱ اردر زمانی عملیات تعریف شده با پیاده سازی های مختلف

۱.۱.۱ ذخیره مجموعه مربوط به عناصر در یک لیست

در یک آرایه به طول n (A) ذخیره کنیم که عنصر iام در کدام مجموعه قرار دارد. پس:

- قرار دادن عناصر در مجموعه های اختصاصی خود در $\Theta(n)$ انجام می شود.
- پیدا کردن مجموعه مربوط به یک عنصر در $\Theta(1)$ انجام می شود.
- ادغام دو مجموعه نیز با تغییر A[i] عناصر موجود در مجموعه مبدا در $\Theta(n)$ صورت می گیرد.

۲.۱.۱ نگه داری عناصر هر مجموعه در کنار آرایه قسمت قبل

در این بخش، در کنار آرایه A قسمت قبل، برای هر مجموعه لیست عناصر موجود در آن را ذخیره می کنیم. در این حالت نیز:

- ساختن لیست های اولیه در $\Theta(n)$ انجام می گیرد.
- پیدا کردن مجموعه مربوط به یک عنصر در زمان ثابت $\Theta(n)$ صورت می گیرد.
- برای ادغام ۲ مجموعه α و β، با فرض کمتر بودن تعداد اعضای مجموعه β، به جای for زدن روی کل آرایه A، $\mathrm{A[k]}$ را برای عناصر موجود در β تغییر می دهیم. پس این عملیات در $\|\beta\|$ انجام می شود. (در حالت کلی برای ادغام دو مجموعه α و β زمان $min(\|\alpha\|, \|\beta\|)$ انجام می شود). البته واضح است برای مجموعه های بزرگ، به دلیل allocate کردن حافظه جدید، این عملیات در $O(n)$ انجام می شود.

حال اگر بخواهیم مرتبه زمانی ادغام را دقیق تر تحلیل کنیم، نیاز است مجموع همه ادغام ها را محاسبه کنیم. ادعا می کنیم همه ادغام ها در زمان $O(nlogn)$ انجام می گیرد. زیرا در ابتدا همه عناصر در مجموعه های تک عضوی بودند. در مرحله بعد مجموعه هایی به طول حداقل ۲، بعد از آن مجموعه هایی به حداقل طول ۴ و ... داریم. پس هر عنصر به اندازه $logn$ بار مجموعه مربوط به خودش عوض می شود و برای همه عناصر، زمان کل ادغام ها برابر با $O(nlogn)$ می شود.
پس به صورت سرشکن زمان ادغام ۲ مجموعه برابر با $O(logn)$ می شود.

۳.۱.۱ پیاده سازی با اشاره غیر مستقیم هر عنصر به نماینده مجموعه

برای کاهش زمان ادغام، مجبوریم زمان پیدا کردن مجموعه یک عنصر خاص را افزایش دهیم.
در حالت قبلی دسترسی به مجموعه یک عنصر در $\Theta(1)$ انجام می شد. زیرا هر عنصر مستقیما به نماینده خود اشاره می کرد. اما در این حالت، هر عنصر لزوما به نماینده اشاره نمی کند و به یک عنصر دیگر در مجموعه اشاره می کند. همچنین نماینده نیز به خودش اشاره می کند.
با این فرض، برای ادغام دو مجموعه α و β، کافی است پوینتر α را به β تغییر دهیم که در زمان $\Theta(1)$ انجام می شود.
همچنین با این رویکرد، پیدا کردن مجموعه یک عنصر در بدترین حالت در (n) انجام می شود. برای کوتاه کردن این مسیر می توان از ۲ ایده زیر بهره برد:

۱. فشرده سازی مسیر: هنگامی که از یک مسیر به نماینده رسیدیم، همه عناصر مسیر را به صورت مستقیم به نماینده متصل کنیم.

۲. وصل کردن درخت با مرتبه کمتر به درخت با مرتبه بیشتر: منظور از مرتبه، ارتفاع درخت قبل فشرده سازی است. ریشه درخت نماینده مجموعه و رئوس آن بقیه عناصر مجموعه هستند.

شبه کد مجموعه های مجزا با فشرده سازی مسیر و در نظر گرفتن مرتبه در نکه کد زیر آمده است:

```
1.  def find_set(parent, x):
2.      if x != parent[x]:
3.          parent[x] = find_set(parent, parent[x])
4.      return parent[x]

5.  def union(parent, rank, x, y):
6.      x, y = find_set(parent, x), find_set(parent, y)
7.      if rank[x] < rank[y]:
8.          parent[x] = y
9.      else:
10.         parent[y] = x
11.         if rank[x] == rank[y]:
12.             rank[x] += 1
```

۴ خط اول عملیات فشرده سازی و برگرداندن نماینده و ۸ خط بعد نیز عملیات ادغام را نشان می دهد.
در کد بالا، منظور از parent[x]، عنصری است که x به آن اشاره می کند. همچنین در ابتدا و برای initialize، برای همه عناصر داریم: parent[x] = x , rank[x] = 0
الگوریتم فوق در 1964 معرفی شد و در سال 1973 اثبات شد که زمان اجرای الگوریتم برابر با $O(log^*(n))$ می باشد. برای درک تفاوت log^* و log به مثال زیر توجه کنید:

$$D = 2^{2^{2^{2}}} \implies log(D) = 2^{2^{2}} = 64536 \ , \ log^*(D) = 5$$

در سال 1975 ثابت شد که زمان اجرا از log^* نیز کمتر است و برابر است با $O(m\alpha(n))$ می باشد که تابع α معکوس تابع Ackermann است. برای اطلاعات بیشتر به لینک زیر مراجعه کنید.

https://en.wikipedia.org/wiki/Ackermann_function

در نهایت در سال 1989 اثبات شد که زمان اجرا برابر است با $\Omega(\alpha(n))$

۲ درخت عبارت

عبارات ریاضی به صورت یکتا نمایش داده نمی شوند و نیاز به پرانتز گذاری دارند.
به دلیل خوش ترتیب نبودن عبارت هایی مانند $2 + 3 * 4$، اولویت عملگرها تعریف شد

و تا حدودی مشکل را حل کرد.
برای نمایش یکتای یک عبارت ریاضی، هم میتوان آنرا پرانتر گذاری کرد و هم می توان آنرا در یک درخت نشان داد. راس های درخت می تواند اعداد، متغییرها و عملگرهایی مانند جمع، ضرب، تقسیم، تفریق (برای تفریق دو عدد) و منفی (برای قرینه کردن یک عدد) باشد.
نمایش in-order این درخت ها بدون پرانترگذاری یکسان است و باعث ایجاد ابهام می شود. اما نمایش pre/post-order آنها متفاوت است.

۱.۲ قضیه برای درخت مرتب

درخت مرتب درختی است که فرزندان هر راس ترتیب خاصی دارند که ترتیب ذکر شده نیز مهم است.
طبق این قضیه، اگر نمایش pre/post-order یک درخت را به همراه تعداد فرزندان هر راس داشته باشیم، درخت مدنظر به صورت یکتا مشخص می شود.
همانگونه که ذکر شد هر راس یا یک عملگر است و یا یک متغیر یا عدد. با توجه به اینکه تعداد فرزندان بر اساس نوع راس مشخص می شود، نمایش pre/post-order درخت عبارت ریاضی را بدون نیاز به پرانتربندی مشخص می شود.

۲.۲ نکات تکمیلی درخت عبارت

با استفاده از یک stack machine می توان از روی نمایش pre-order درخت، عبارت را ساخت.
همچنین با استفاده از این درخت می توان عملیات پیچیده مانند مشتق گیری از رابطه مشخص شده راانجام داد.