دانشکده مهندسی کامپیوتر

جزوه درس
ساختمان‌های داده

استاد درس: سید صالح اعتمادی

پاییز ۱۳۹۸

جلسه ۷

Divide and Conquer

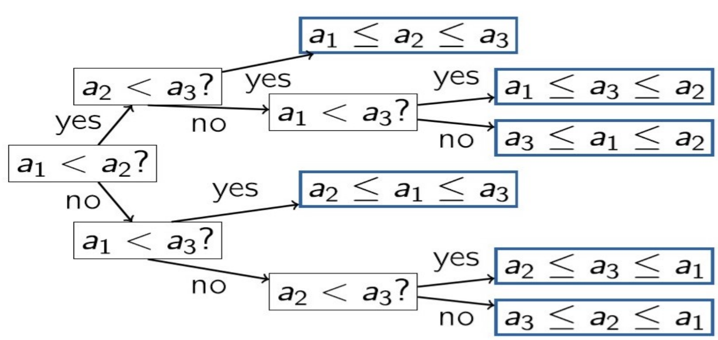
پریسا علائی - ۱۳۹۸/۷/۲۰

جزوه جلسه ۷ام مورخ ۱۳۹۸/۷/۲۰درس ساختمان‌های داده تهیه شده توسط پریسا علائی. در جهت مستند کردن مطالب درس ساختمان های داده، بر آن شدیم که از دانشجویان جهت مکتوب کردن مطالب کمک بگیریم. هر دانشجو می‌تواند برای مکتوب کردن یک جلسه داوطلب شده و با توجه به کیفیت جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهش‌مند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید.

۱.۷ مرتب سازی

برای مرتب سازی مقایسه ای ، حداقل زمان لازم $n \log n$ است. در واقع برای این نوع مرتب سازی ، یک درخت تصمیم گیری داریم .

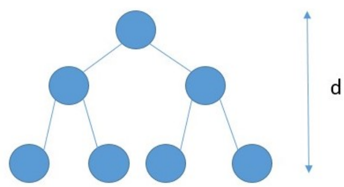
`Example)` اگر $a1, a2, a3, , an$ را داشته باشیم :



شکل ۱.۷: Decision Tree

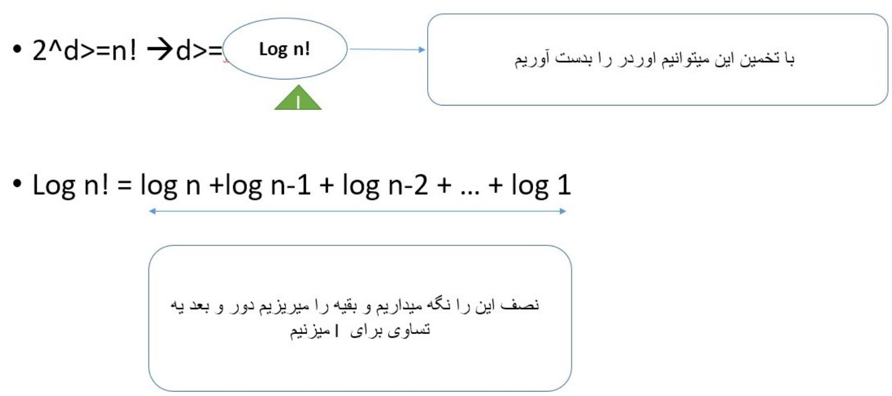
۲.۷ نکات مثال

- در آخر این درخت به یک برگ میرسیم که مرتب شده ی a1 تا an وجود دارد.
- در هر نود این درخت یا شاخه های آن حالت هایی است که مثلا an ها می توانند نسبت به هم داشته باشند !
- در واقع حالت های متفاوت چیدن مثل جایگشت بدون تکرار است که کل حالت ها $n!$ است.
- تعداد مقایسه ها برابر است با عمق درخت. پس بیشترین عمق برابر است با حالت max در درخت.
- در حالت max تعداد برگ ها برابر است با 2^d البته درخت باید متوازن باشد .

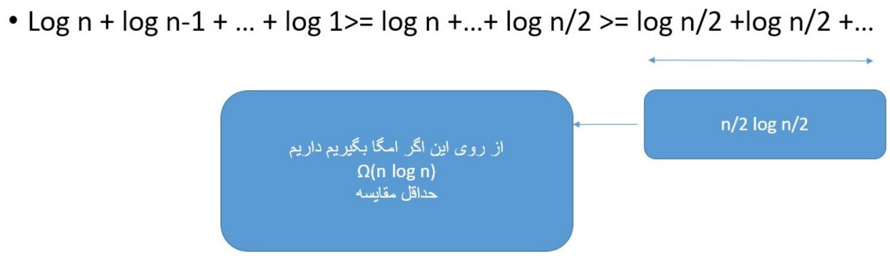


شکل ۲.۷: Tree

۳.۷ اثبات اینکه n log n بهترین حالت است



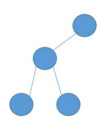
شکل ۳.۷: proof ۱



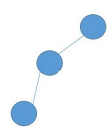
شکل ۴.۷: ۲ proof

۴.۷ Example

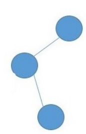
یک درخت به عمق ۲ چند حالت برگ دارد؟
پاسخ : میتواند حالت max باشد یا میتواند برگ هایش کمتر باشد.



شکل ۵.۷: ۱ Tree



شکل ۶.۷: ۲ Tree

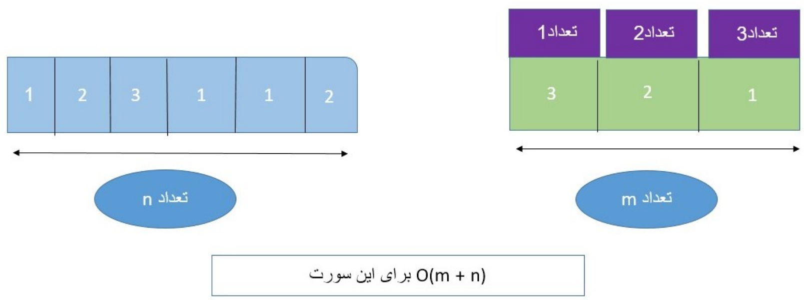


شکل ۷.۷: ۳ Tree

عکس وسط و آخر یک حالت به حساب می آیند و حالت تکراری اند چون فقط تعداد برگ ها اینجا مهم است.
چند حالت دیگر هم برای این مثال هست و محدود به این مواردی که در بالا نشان داده شد نمی شود.

۵.۷ Sorting Small Integers

برای استفاده از این روش باید تعداد متغییر ها محدود باشد و باید متغییر ها را بشناسیم.
تعداد را با یک دور چک کردن می توان فهمید ، بعد به تعداد متغییر هایی که داریم از کوچک به بزرگ آن ها را مینویسیم مثلا : از کوچک ترین متغییر ۲ تا داریم اول دو تا از آن متغییر مینویسیم و بعد به ترتیب بقیه ی متغییر ها را مینویسیم.
برای استفاده از این روش حتما اعداد گسسته اند مانند : اعداد صحیح!! اگر پیوسته باشند جواب نمیدهد.



شکل ۸.۷: مثال

$\texttt{CountSort}(A[1\ldots n])$

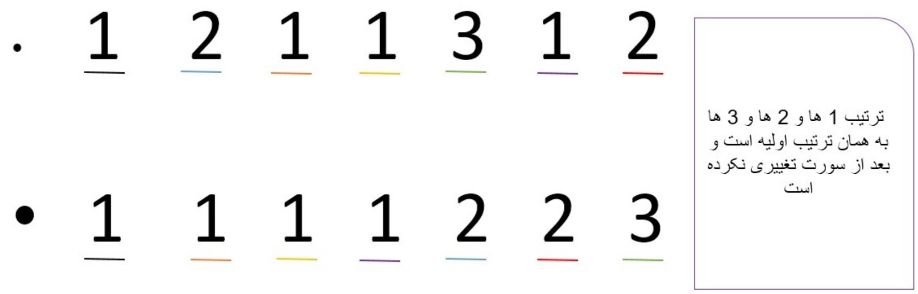
$Count[1\ldots M] \leftarrow [0,\ldots,0]$
for i from 1 to n:
  $Count[A[i]] \leftarrow Count[A[i]] + 1$
{k appears $Count[k]$ times in A}
$Pos[1\ldots M] \leftarrow [0,\ldots,0]$
$Pos[1] \leftarrow 1$
for j from 2 to M:
  $Pos[j] \leftarrow Pos[j-1] + Count[j-1]$
{k will occupy range $[Pos[k]\ldots Pos[k+1]-1]$}
for i from 1 to n:
  $A'[Pos[A[i]]] \leftarrow A[i]$
  $Pos[A[i]] \leftarrow Pos[A[i]] + 1$

شکل ۹.۷: شبه کد

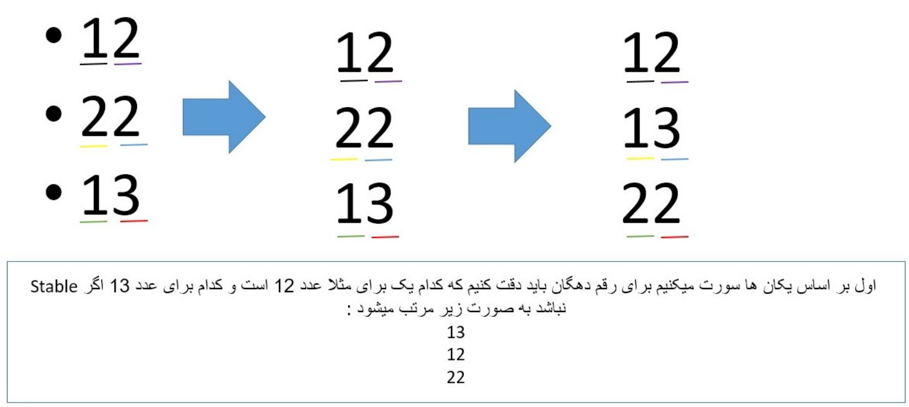
منابع دیگر : [۱] [۲]

۶.۷ Stable Sort

سورتی است که اگر دو تا عنصر مقدارشان باهم برابر باشد اگر یکی قبل دیگری باشد ، ترتیبشان در بعد از سورت نیز رعایت میشود.



شکل ۱۰.۷: ۱ مثال



شکل ۱۱.۷: مثال ۲

منابع دیگر : [۳] [۴]

۷.۷ Quick Sort

برای آرایه بهترین سورت است و حتی بهتر از مرج سورت ! در واقع روش عملکرد آن این گونه است که : یک خانه را آرایه انتخاب میکنیم و بعد جوری میچینیم که خانه هایی که دارای مقدار کوچک تر هستند سمت چپ و خانه هایی که دارای مقدار بزرگ تر هستند سمت راست قرار گیرند.
این روش به صورت بازگشتی است ، یعنی برای آن دو قسمت جدید باز باید این کار انجام شود تا تمام خانه ها مرتب شوند .

```
QuickSort(A, ℓ, r)

if ℓ ≥ r:
  return
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```

شکل ۱۲.۷: Quick Sort

```
Partition(A, ℓ, r)

x ← A[ℓ]    {pivot}
j ← ℓ
for i from ℓ + 1 to r:
  if A[i] ≤ x:
    j ← j + 1
    swap A[j] and A[i]
  {A[ℓ + 1 . . . j] ≤ x,  A[j + 1 . . . i] > x}
swap A[ℓ] and A[j]
return j
```

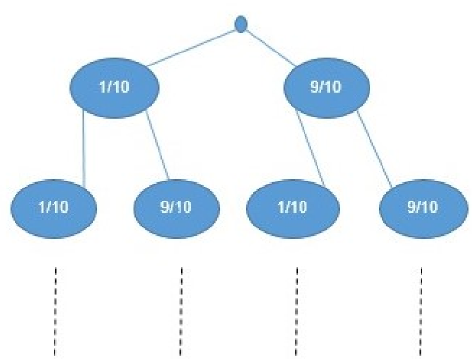
شکل ۱۳.۷: Partition

۸.۷ چند نکته :

- در quick sort (شکل ۱۲.۷) خود m را بررسی نمیکنیم ، چون در جایگاه نهایش قرار گرفته است.
- در (شکل ۱۳.۷) pivot را خانه ای را در نظر میگیریم که جایگاهش تغییر نکند پس نمیشود در حالیکه داریم با اولین pivot مقایسه میکنیم برای دو تا بازه ی بعدی که هنوز همه ی خانه هایشان مشخص نشده است pivot در نظر بگیریم و آن ها را هم سورت کنیم .
- در بهترین حالت O(n log n) و در بدترین حالت $O(n^2)$ است .

$$T(n) = T(n/10) + T(9n/10) + O(n)$$

شکل ۱۴.۷: Balanced Partitions



شکل ۱۵.۷: Balanced Partitions Tree

منابع دیگر : [۵] [۶]

۹.۷ Random Pivot

در این حالت با Quick Sort (شکل ۱۲.۷) عادی یک تفاوت وجود دارد آن هم این است که در این حالت خانه ی اول را انتخاب نمیکنیم و یکی از خانه های آرایه را به صورت رندوم انتخاب و بعد نسبت به آن جابه جا میکنیم .
با این کار احتمال $O(n^2)$ را کم میکنیم چون داده های متفاوتی را هر دفعه به آن میدهیم.
به طور متوسط داریم : O(n log n)
منبع دیگر : [۷]

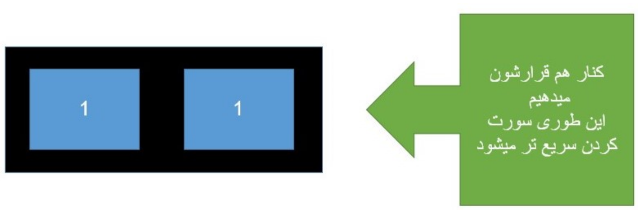
```
RandomizedQuickSort(A, ℓ, r)

if ℓ ≥ r:
  return
k ← random number between ℓ and r
swap A[ℓ] and A[k]
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
RandomizedQuickSort(A, ℓ, m − 1)
RandomizedQuickSort(A, m + 1, r)
```

شکل ۱۶.۷: Balanced Partitions Tree

۱۰.۷ Equal Elements

اگر آرایه دارای خانه هایی با متغییر های یکسان باشد ، آن خانه ها را کنار هم قرار میدهیم و اگر لازم شد باهم جابه جایشان میکنیم .
با این کار سورت کردن سریع تر صورت میگیرد.
تئوری ساده و خودمونی : اگر یک بازه باشد که دارای عناصر برابر باشد مثل $p, p-1, n, n, n, k, k+1$ در Partition (شکل ۱۳.۷) که در قسمت quick sort (شکل ۱۲.۷) گفتیم به جای اینکه عدد وسط را بازگرداند ، باید یک بازه از جایگاه هایی که دارای عدد مساوی است را به ما بدهد ، و بعد فورها را برای دوتا بازه ی سمت راست و چپ باید جوری بنویسیم که بازه ی مساوی را دست نزند و از بعد آخرین خانه و از قبل اولین خانه ی بازه ی برگردانده شده شروع کند .
منبع دیگر : [۸]



شکل ۱۷.۷: نمونه

```
RandomizedQuickSort(A, ℓ, r)

if ℓ ≥ r:
  return
k ← random number between ℓ and r
swap A[ℓ] and A[k]
(m1, m2) ← Partition3(A, ℓ, r)
{A[m1 ... m2] is in final position}
RandomizedQuickSort(A, ℓ, m1 − 1)
RandomizedQuickSort(A, m2 + 1, r)
```

شکل ۱۸.۷: شبه کد

۱۱.۷ Sort Visualization

- این لینک انیمیشن سورت ها با داده هایی که از پیش تعیین شده است را نشان میدهد. لینک اول
- در این لینک میتوانید یکی از انواع سورت ها را انتخاب کنید و نحوه ی عملکرد آن ها را به صورت درخت ببینید.(داده ها را به صورت دستی وارد میکنید.) لینک دوم
- این لینک دارای کد سورت ها به زبان جاوا است و میتوانید به صورت انیمیشن نحوه ی عملکرد سورت ها را ببینید.(هم به صورت دستی داده وارد میکنید و هم دارای داده های از پیش تعیین شده است.) لینک سوم

Bibliography

[1] https://www.programiz.com/dsa/counting-sort.

[2] https://www.geeksforgeeks.org/counting-sort/.

[3] https://www.geeksforgeeks.org/stability-in-sorting-algorithms/.

[4] https://www.geeksforgeeks.org/stable-selection-sort/.

[5] https://www.geeksforgeeks.org/quick-sort/.

[6] https://en.wikipedia.org/wiki/Quicksort.

[7] https://www.geeksforgeeks.org/quicksort-using-random-pivoting/.

[8] https://www.geeksforgeeks.org/3-way-quicksort-dutch-national-flag/.