حوری دهش 9821413

به نام خدا

تمرین تئوری سری چهارم ساختمان داده

==================================

سوال 1:

الف) غلط است الگوریتم quicksort یک الگوریتم مقایسه ای است و الگوریتم های مقایسه ای نمی توانند کمتر از nlogn باشند.

ب) غلط است چون در روش radix sort ما برای هر رقم در یک ستون این الگوریتم را اجرا می کردیم پس برای هر رقم ما O(n+k) زمان نیاز داریم و چون d رقم هم داریم پس پیچیدگی کل ما O(d(n+k)) می شود.

ج) درست است بخاطر اینکه هر کلید به هر خونه از ارایه که بخواهد مپ شود احتمالش $\frac{1}{k}$ است یعنی مثلا اگر k=11 باشد و یک مجموعه هم داشته باشیم از 1 تا 11 و x هم عددهای این مجموعه باشند زمانی که باقی مانده را بر 11 حساب می کنیم تمامی خونه هایی ارایه را پوشش می دهد حالا اگر x در یک ضریب هم ضرب کنیم بازهم ما یک جدول درهم ساز k یکنوا داریم فقط این سری جای عددها در این مثال پشت سر هم قرا نمی گیرد یعنی خونه های مختلف ارایه را پرمی کنند. اکنون اگر این را به یک مجموعه بزرگتر هم تعمیم بدیم باز هم این موضوع برقرار است پس تقریبا می توانیم یک جدول درهم ساز k یکنوا ایجاد کنیم. چرا تقریبا ؟ چون به مقادیر x و k وابسته است که x و k چه عددهایی باشند ولی بازم تقریبا احتمال کلیدهایی که به هر خونه از ارایه مپ می شوند یکسان است.

د) غلط است بخاطر اینکه ما یک ارایه k تایی داریم در صورتی که باقی مانده x را بر k-1 می خواهیم حساب کنیم که این باعث می شود همیشه یک خونه از ارایه ما تحت هر شرایطی پر نشود پس نمی توانیم بگوییم به همه خونه ها کلید هایی با احتمال یکسان مپ می شود زیرا در هر صورتی هیچ وقت به یک خونه کلیدی مپ نمی شود پس ما جدول درهم ساز k یکنوا نداریم.

سوال 2 :

مراحل الگوریتم:

1- ابتدا ارایه داده شده را مرتب می کنیم.

2- اثبات را از سومین المان از اخر شروع می کنیم و ارایه را برای پیدا کردن دو عدد دیگر که به سومین عنصر خلاصه می شود می پیمایم.

3- دو پوینتر j (که از جلوی ارایه شروع می شود) و k (که مقدار اولیه اش برابر با i-1 است) می گیریم که کوچیکترین دو عدد را پیدا کنیم و از i-1 بزرگترین ان دو عدد باقی مانده را پیدا می کنیم. حالا می گوییم اگر پوینتر j کمتر از k بود یکی از حالت های زیر رخ می دهد:

a) اگر مجموع دو عدد برابر با A[i] باشد انگاه ما مجموعه سه تایی را پیدا کردیم.

b) اگر جمع دو عدد ما هنوز کمتر از A[i] باشد انگاه ما نیاز داریم که مقدار مجموع دو عدد را افزایش دهیم پس پوینتر j را افزایش می دهیم بنابراین مقدار A[j] + A[k] افزایش پیدا می کند.

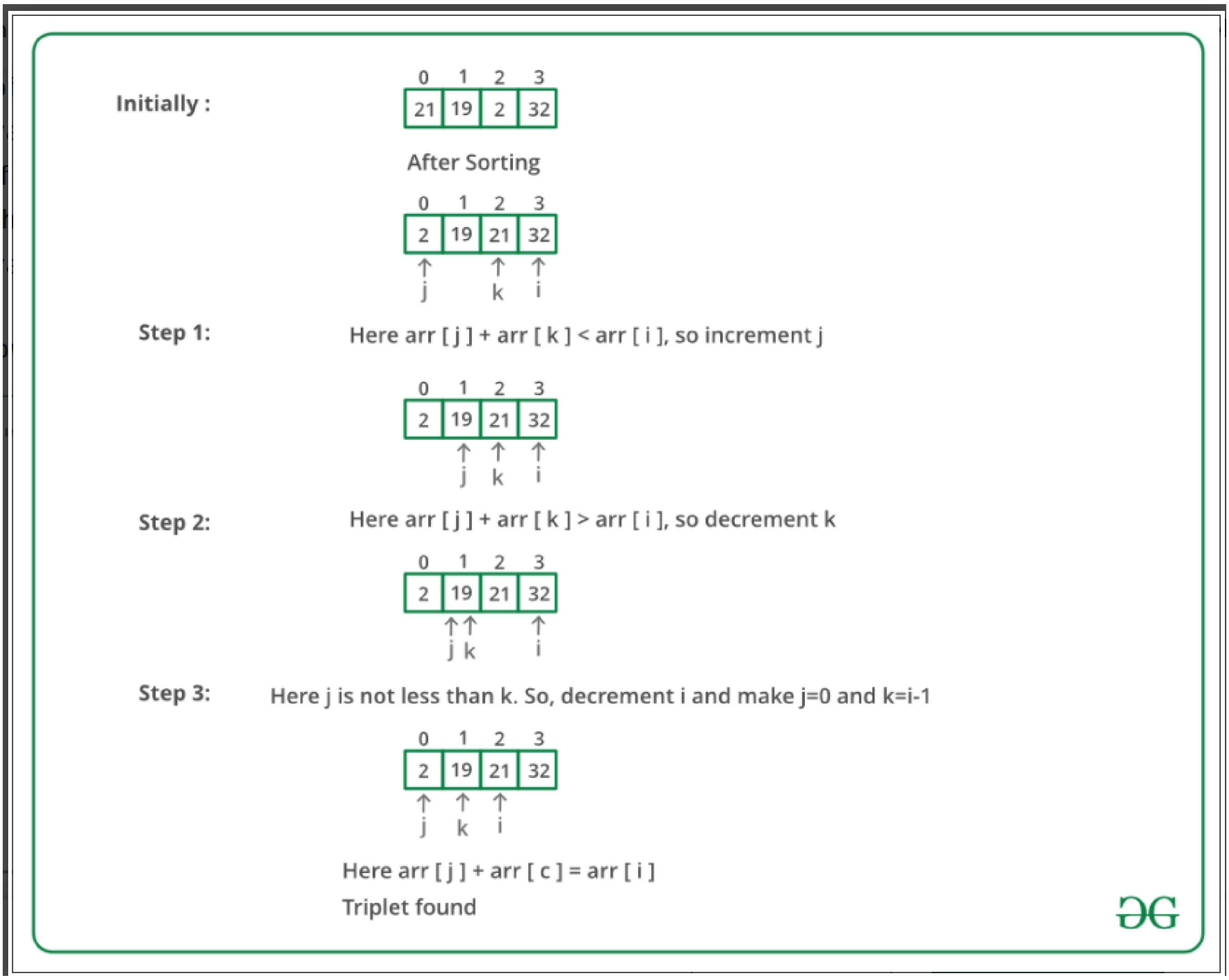
c) اگر مجموع دو عدد بیشتر از A[i] باشد انگاه ما نیاز داریم که مقدار جمع دو عدد را کاهش دهیم پس پوینتر k را کاهش می دهیم بنابراین مقدار A[j] + A[k] کاهش پیدا می کند.

4- اگر پوینتر j کمتر از k نبود ما i را کاهش می دهیم و j=0 و k=i-1 قرار می دهیم.

حوری دهش 9821413

Time complexity: O(n^2)

تصویر زیر برای درک بهتر است:



References:

/https://www.geeksforgeeks.org/find-triplet-sum-two-equals-third-element

سوال 3 :

مراحل الگوریتم:

1- دو متغییر sum=0 و maxLen=0 را در نظر می گیریم.

2- یک hash table که تاپل (sum , index) را دارد می سازیم .

3- برای i=0 تا n-1 مراحل زیر را طی می کنیم (این یک حلقه است) :

a) مقدار sum + arr[i] را در sum می ریزیم.

b) اگر sum==k بود مقدار maxLen را به صورت maxLen= i+1 اپدیت می کنیم. (k همان عددی است که از ورودی می گیریم)

c) بررسی می کنیم که ایا مقدار sum در hash table وجود دارد یا نه اگر وجود نداشت این sum را به hash table به صورت جفت (sum , i) اضافه می کنیم.

d) بررسی می کنیم که ایا مقدار sum – k در hash table وجود دارد یا نه اگر وجود داشت index of (sum - k) را از hash table به عنوان index به دست می اوریم سپس چک می کنیم که اگر maxLen < (i – index) بود انگاه مقدار maxLen = (i - index) را اپدیت می کنیم.

4- و در اخر maxLen را برمی گردانیم.

Time complexity: O(n)

References:

/https://www.geeksforgeeks.org/longest-sub-array-sum-k

سوال 4 :

؟؟؟

سوال 5 :

برای حل این سوال می توانیم 4 کیس را مورد بررسی قرار دهیم که دو مورد از این 4 کیس مقدار وزن یال را کاهش می دهد و دو مودر دیگر افزایش.

پس کیس هایی که مقدار یال را کاهش می دهند :

1- یال ما در MST قرار دارد اکنون وزن این یال را کاهش می دهیم:

درخت (MST) ما همان درخت (MST) قبلی است -->

Time complexity: O(1)

2- یال ما در MST قرار ندارد اکنون وزن این یال را کاهش می دهیم:

این یال را به MST اضافه می کنیم و با این کار یک حلقه در MST درست می شود حالا برای از بین بردن این حلقه باید بیاییم یال بزرگتر درون حلقه را پیدا کنیم و ان را حذفش کنیم که این کاررا می توانیم با استفاده از الگوریتم DFS یا الگوریتم BFS انجام دهیم -->

Time complexity: O(n)

References:

https://stackoverflow.com/questions/9933438/update-minimum-spanning-tree-with-modification-of-edge

سوال 6 :

؟؟؟؟

حوری دهش 9821413

سوال 7 :

یافتن Minimum Spanning Tree (MST) با استفاده از الگوریتم Kruskal یا الگوریتم prim یا الگوریتم sollin

الگوریتم Kruskal :

1- تمام یال ها را براساس وزنشان به صورت صعودی مرتب می کنیم.

2- کمترین وزن یال را در هر مرحله انتخاب می کنیم و بررسی می کنیم که ایا این یال با یالهایی که از قبل به MST اضافه کرده بودیم دور تشکیل می دهد یا نه اگر دور تشکیل نداد ان را هم به MST اضافه می کنیم و اگر دور تشکیل داد ان را در نظر نمی گیریم این کار را انقدر تکرار می کنیم که MST ما کامل شود.

نکته = این الگوریتم می تواند جنگل هم درست کند.

نکته = در اول کار MST نداشتیم --> مرحله دوم را انقدر تکرار کردیم که MST کامل شد.

پیچیدگی زمانی:

$O(ElogE)$ یا $O(ElogV)$ است چون برای مرتب سازی یالها به صورت صعودی $O(ElogE)$ نیاز داریم از طرف دیگر هم می دانیم که اگرگرافمان کامل باشد هر نودی به $V(V-1)$ نود وصل می شود پس $|E| < |V|^2$ در نتیجه $\log|E| < log|V|^2$ --> $\log|E| = O(logV)$ که اگر در فرمول $O(ElogE)$ قرار بدهیم پیچیدگی ما $O(ElogV)$ می شود.

سوال 8 :

اثبات ما دارای دو بخش است:

1- اگر G دور اویلر C داشته باشد، C یا یک دور ساده است (یعنی خود را قطع نمی کند) یا نیست. اگر C یک دور ساده باشد، هر راس در یک دور ساده دارای indeg=outdeg=1 است، بنابراین این ادعا درست است. اگر C یک دور است اما یک دور ساده نیست، پس باید شامل یک دور ساده باشد پس ان را از G و بعد از C حذف می کنیم.C باقی مانده، همچنان یک دور اویلر برای G باقی مانده است. تا زمانی این دورهای ساده را حذف می کنیم که هیچ یالی باقی نماند. هنگام حذف یک دور، یک یال درونی و بیرونی از رئوس روی دور حذف می شود. پس از حذف دور، درجه درون و درجه بیرونی یک گره در دور دقیقاً یکی کاهش می یابد. در پایان، وقتی هیچ یالی باقی نمانده است، تمام درجات داخلی و خارجی 0 هستند. بنابراین همه رئوس v باید با indeg(v) = outdeg(v) شروع شده باشند.

2- اگر هر رأس v دارای indeg(v) = outdeg(v) باشد، در اولین لحظه بنظر می رسد که برای هر رأس v، باید مسیری وجود داشته باشد که از v شروع شود و به v برگردد --> اثبات:

فرض می کنیم ادعای بالا درست است و یک راس v را به صورت تصادفی انتخاب می کنیم و یک دور C را پیدا می کنیم که به v برمی‌گردد. تمام یال‌های روی C را از G حذف می کنیم. هر رأس در G جدید هم، همچنان دارای indeg(v) = outdeg(v) است. یک راس 'v روی C انتخاب می کنیم که دارای یال‌هایی باشد (چنین راسی باید وجود داشته باشد) و تکرار می کنیم. به طور کلی ما یک دور C پیدا می کنیم، سپس یک دور دیگر 'C که (حداقل) یک راس مشترک با C دارد و...

پس می‌توانیم یک دور بزرگ بسازیم که به دور C می‌رود، به C می‌پرد و حول 'C می‌رود، سپس به C برمی‌گردد و C را تمام می‌کند.

اثبات ادعای بالا: برای هر رأس v، باید دوری وجود داشته باشد که شامل v است. از v شروع می کنیم، و هر یال خروجی v را انتخاب می کنیم، مثلاً (v, u). از انجایی که indeg(u) = outdeg(u) می‌توانیم برخی از یالهای خروجی u را انتخاب کنیم و به بازدید از یالها ادامه دهیم. هر بار که یک یال را انتخاب می کنیم، می توانیم ان را از بررسی بیشتر حذف کنیم. در هر رأس غیر از v، در زمانی که از یک یال ورودی بازدید می کنیم، باید یک یال خروجی بدون بازدید باقی بماند، زیرا indeg = outdeg برای همه رئوس است. تنها راسی که ممکن است یک یال خروجی بازدید نشده برای ان وجود نداشته باشد، v است - زیرا ما دور را با بازدید از یکی از یال های خروجی v شروع کردیم. از انجایی که همیشه یک یال خروجی وجود دارد که می‌توانیم برای هر رأسی غیر از v از ان بازدید کنیم، در نهایت دور باید به v برگردد، بنابراین این ادعا اثبات می‌شود.

حوری دهش 9821413

References:

https://tildesites.bowdoin.edu/~ltoma/teaching/cs231/fall09/Homeworks/old/H9-sol%203.pdf

سوال 9 :

مراحل الگوریتم:

1- برای هر یک از رئوس موجود در dag ابتدا درجه ورودی هر راس را حساب می کنیم و تعداد راس های بازدید شده را صفر مقداردهی می کنیم.

2- تمام رئوس با درجه ورودی صفر را انتخاب می کنیم و انها را به یک صف اضافه می کنیم --> عملیات Enqueue

3- یک راس را از صف حذف می کنیم (عملیات Dequeue) سپس :

a) تعداد راس های بازدید شده را یکی افزایش می دهیم.

b) برای تمامی راس های همسایه ان راس، درجه ورودی را یکی کاهش می دهیم.

c) اگر درجه ورودی یک راس همسایه به صفر کاهش پیدا کرد ان را به صف اضافه می کنیم.

4- مرحله 3 را انقدر تکرار می کنیم که صف خالی شود.

5- اگر تعداد راس های بازدید شده با تعداد راس های گراف یکسان نبود ان وقت topological sort برای گراف داده شده امکان پذیر نیست.

اگر گراف دارای دور باشد چه اتفاقی می افتد؟

در topological sort ایده این است که از نود پدر و سپس نود فرزند بازدید کنیم. اگر گراف داده شده دارای دور باشد، حداقل یک نود وجود دارد که هم پدر است و هم فرزند، بنابراین ترتیب topological را به هم می زند و topological sort با شکست مواجه خواهد شد. بنابراین، پس از مرتب سازی توپولوژیکی، هر یال جهت دار را بررسی می کنیم که ایا از ترتیب پیروی می کند یا نه.

References:

/https://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/amp

https://www.geeksforgeeks.org/detect-cycle-in-directed-graph-using-topological-sort/#:~:text=Approach%3A,this%20will%20break%20Topological%20Order