

## هوالحق

### مستند پروژه پایانترم ساختمان داده

استاد فاطمی

ارائه دهنده : فاطمه جانثاری و ریحانه خرمیان

#### ● بخش آشپزخانه

ابتدا ورودی ها را مطابق نمونه ای که در تست کیس داده شده از کاربر دریافت می کنیم. و هر غذا در کلاس Food و پیش نیازهای آن در یک گراف که داخل کلاس آن غذا است ذخیره می شود. و همه غذاها نیز در آرایه ای در کلاس kitchen ذخیره می شوند.

✓ حذف یک غذا و تمام نیازمندی های آن: یک غذا از لیست غذاهای داخل آشپزخانه حذف می شود. و از  $O(1)$  انجام می شود.

✓ دریافت غذا با بیشترین نیازمندی: در تابع `getFoodWithMaxRequisites` بین غذاها جستجو می کند و غذا با بیشترین نیازمندی را برمی گرداند .  
مرتبه زمانی  $O(n)$ : که  $n$  تعداد غذاهاست.

✓ اضافه کردن یک رابطه دوتایی: بعد از دریافت رابطه دوتایی از کاربر، آنها را به تابع `addARelationship` می فرستیم و از آنجا هم به تابع `setEdgeWithoutCycle`. (در حالت عادی که کاربر به ما ورودی می دهد فرض بر این است که ورودی ها درستند و دور ایجاد نمی کنند اما اینجا برای اطمینان از این تابع استفاده می کنیم).  
در این تابع، ابتدا ما این رابطه دوتایی داده شده را به گراف اضافه می کنیم اما بعد چک می کنیم که دور به وجود نیامده باشد. اگر دور به وجود آمد، این یال را حذف می کنیم.  
برای همه گره های گراف یکبار چک می کنیم؛ یعنی هر گره اگر ویزیت نشده بود به تابع `checkCycle` می رود.

تابع `checkCycle` از الگوریتم `dfs` استفاده می کند. به این صورت که هر نودی که به آن داده می شود را ویزیت می کند بعد در یک حلقه، همه همسایه های آن گره را چک می کند اگر ویزیت نشده بودند، دوباره به صورت بازگشتی وجود یا عدم وجود دور در آنها را چک می کند. اما اگر ویزیت شده بودند و در همین `dfs` ویزیت شده بودند، این یعنی که ما دور داشته ایم پس تابع مقدار `true` برمی گرداند .

مرتبه زمانی این قسمت  $O(n + E)$  که  $n$  تعداد گره ها و  $E$  تعداد یال هاست. چون برای هر گره یکبار  $dfs$  می زنیم در واقع مثل این است که برای کل گراف الگوریتم  $dfs$  را انجام دهیم پس درنهایت مرتبه زمانی همان مرتبه زمانی الگوریتم  $dfs$  می شود .

✓ دریافت غذا با بیشترین زمان و کمترین زمان: برای هر کدام یکبار بین غذاهای آشپزخانه جستجو می کنیم و غذا با بیشترین/کمترین زمان را برمی گردانیم. برای پیدا کردن زمان نهایی هر غذا هم از تابع  $getTime$  استفاده می کنیم که با یک حلقه روی نیازمندی ها زمان کل را پیدا می کند. مرتبه زمانی  $O(n * m)$  که  $n$  تعداد غذاها و  $m$  تعداد گره های گراف غذاست.

✓ پیدا کردن دستور غذای مرتب شده: برای این کار به مرتب سازی توپولوژیکی نیاز داریم .  
○ اولین سورت  $topologicalSort$  بر پایه الگوریتم  $kahn$  است که اول گره هایی که درجه ورودی شان صفر است را ویزیت می کند و به همین ترتیب ادامه می دهد و از گره های ویزیت نشده گره های با درجه ورودی صفر را ویزیت می کند و ...  
مرتبه زمانی آن  $O(n + E)$  است که  $n$  تعداد گره ها و  $E$  تعداد یال هاست. چون برای هر گره که در صف هست چک می کند و در چک کردن هر بار همسایه های آن گره را بررسی می کند پس حلقه داخلی به تعداد یال ها انجام می شود.  
○ دومین سورت  $topologicalSort2$  بر پایه الگوریتم  $dfs$  است. به این صورت که برای هر گره  $dfs$  را ادامه می دهد تا به پایین ترین گره برسد و آن را به استک پوش می کند و به همین ترتیب بالا می آید. بعد از اتمام وقتی یکی یکی از استک پاپ کنیم لیست مرتب شده را به دست می آوریم .  
مرتبه زمانی آن همان مرتبه زمانی الگوریتم  $dfs$  است یعنی  $O(n + E)$  که  $n$  تعداد گره ها و  $E$  تعداد یال هاست .

ما این دو سورت را نوشته و امتحان کردیم اما برای تعداد زیاد داده جواب نمی داد و مرتبه زمانی زیادی داشت برای همین الگوریتم دیگری با مرتبه زمانی کمتر پیدا کردیم :

○ سومین سورت :  $topologicalSort3$  در این روش ما همه گره های گراف را در یک لیست می ریزیم. بعد روی یال ها یک حلقه می زنیم و برای هر یال مثلاً از  $A$  به  $B$  چک می کنیم که آیا در لیست گره ها،  $A$  قبل از  $B$  قرار دارد یا نه) چون  $A$  قبل از  $B$  باید اجرا شود (اگر اینطور نبود جای  $A$  و  $B$  را با هم عوض می کنیم. بعد از اتمام حلقه بازهم از اول لیست را چک می کنیم تا اگر با این جابه جایی ها ترتیب بعضی گره ها به هم خورده درست شود. و در پایان لیست ما به ترتیب مرتب می شود. مرتبه زمانی این الگوریتم  $O(E)$  است که  $E$  تعداد یال هاست. چون حلقه ما فقط به تعداد یال های گراف اجرا می شود.

## ● بخش مهمانی

ابتدا در ورودی دستور لازم گرفته می‌شود .

### ✓ : Search

برای این دستور ریشه و کلید مورد نظر داده می‌شود، سپس درواقع جستجوی باینری انجام می‌شود و کلید با ریشه مقایسه می‌شود اگر کوچکتر بود سرچ برای سمت چپ انجام می‌شود و اگر بزرگتر بود سرچ برای سمت راست انجام می‌شود تا کلید یافت شود اگر به انتها رسیدیم و پیدا نشد null بر می‌گردانیم و اگر پیدا شد خود نود را .  
این بخش از مرتبه ی زمانی  $O(\text{Logn})$  است

### ✓ : Insert

در ابتدای این بخش ابتدا شبیه به سرچ در درخت جستجو می‌شود که کجا مناسب قرارگیری عنصر جدید است زمانی که به null رسید همانجا یک نود جدید اضافه می‌شود . سپس برای ریشه عدد توازن مشخص می‌شود . سپس این عدد توازن برای هر گره به همراه نسبت آن با گره های فرزند سنجیده می‌شود و اگر نیاز به تغیر و جابه جایی بود اعلام می‌کند و چرخش متناسب با آن انجام می‌شود.  
دو نوع چرخش داریم:  
چرخش راست و چپ  
چرخش راست نود با عدد توازن نامناسب را به عنوان y می‌گیرد فرزند سمت چپ ک قرار است به عنوان ریشه در نهایت داده شود به عنوان x مشخص می‌شود . فرزند سمت راست x به نام T2 ذخیره می‌شود . سپس y در جایگاه فرزند سمت راست x قرار گرفته و T2 می‌شود فرزند چپ y ، و چرخش انجام می‌شود . چرخش چپ هم به همین صورت در جهت عکس .  
و در انتهای تابع insert ریشه ی نهایی درخت داده می‌شود .  
این بخش از مرتبه ی زمانی  $O(\text{Logn})$  است.

### ✓ : Delete

در تابع نود ریشه و کلید داده میشود( در ورودی نام مهمان داده می‌شود که توسط پیمایش pre-order نود ها جستجو میشوند و کلید متناسب با آن نام داده می‌شود) سپس در ابتدا باهم روند جستجوی دو دویی نود مورد نظر پیدا می‌شود. بعد از پیدا شدن بررسی می‌شود ک چند فرزند دارد.  
اگر هیچ فرزندی نداشت که به راحتی نود را null میکنیم . اگر یک فرزند داشت آن فرزند را جایگزینش میکنیم . اگر دو فرزند داشت فرزند سمت راست که بزرگتر است را جایگزین آن میکنیم . و اگر فرزندان هم فرزند داشتند به همین صورت برایشان عمل می‌شود.

در نهایت باز عدد توازن ریشه بررسی می‌شود و اگر نیاز به چرخش بود انجام می‌شود.  
این بخش نیز از مرتبه ی زمانی  $O(\text{Log}n)$  است.

### ✓ ShowTree:

در اینجا ابتدا نود ریشه داخل یک لیست ذخیره می‌شود. سپس ارتفاع درخت معلوم شده و متناسب با آن تعدادنود های درخت کامل این ارتفاع ذخیره می‌شود. (برای چاپ زیبا تر)  
سپس برای هر نود که داخل لیست اولیه هست مراحل چاپ انجام می‌شود و فرزندان آن داخل لیست ذخیره میشوند.

برای چاپ:

هر نود، به اندازه ی تقسیم تعداد نود بر دو به دو به دو شماره ی مورد نظر، تب می‌خورد و از ابتدای خط فاصله می‌گیرد. سپس نام مهمان و شماره نوبت آن چاپ می‌شود.  
این بخش از مرتبه ی زمانی  $O(n\text{Log}n)$  است. حلقه ی وایل به اندازه ی ارتفاع درخت طی می‌شود و داخل حلقه از مرتبه ی  $n$  قرار است تب زده شود.

## پایان