Algorithm Design
project #4

Hana Esfandiar
Mohammad Afshari

ایده کلی الگوریتم:

می‌خواهیم بزرگ‌ترین زیررشته‌ای که k بار در رشته‌ی s تکرار شده‌است را بیابیم.
رشته‌ی s و عدد طبیعی k، به عنوان ورودی به ما داده می‌شود. خواسته‌ی سوال این است که بزرگ‌ترین زیررشته‌ای که در رشته‌ی s موجود است را بیابیم به صورتی که این زیر رشته، k بار در رشته‌ی s تکرار شده باشد.
برای مثال اگر رشته‌ی s برابر با "abaaab" باشد، زیررشته‌ی ab از رشته‌ی s، دو بار تکرار شده: abaaab
که ab و abab، هر دو جزو زیر رشته های رشته‌ی s هستند.
برای حل این سوال، باید ابتدا تمام زیر رشته های رشته‌ی داده شده را پیدا کنیم. سپس بررسی می‌کنیم که کدامیک از زیر رشته ها، اگر k بار در کنار هم قرار گیرند، باز هم زیر رشته ای از رشته‌ی داده شده خواهند بود و این زیر رشته باید بیشترین طول را داشته باشد.
باید به این نکته نیز توجه کرد که تمام کاراکتر های موجود در رشته‌ی داده شده، در زیر رشته‌ی خواسته شده قرار نمی گیرند؛ بلکه کاراکتر هایی در رشته‌ی نهایی قرار دارند که بیشتر از k بار (حداقل k بار) تکرار شده اند.
همچنین نکته‌ی دیگری که لازم به توجه اس، ترتیب حروف است. یعنی از نظر lexicographically نیز باید زیر رشته ها را مقایسه و بررسی کنیم و اگر دو زیر رشته با شرایط لازم موجود بود، زیر رشته ای را انتخاب میکنیم که از نظر ترتیب الفبایی انگلیسی، در انتهای جدول الفبا است.

بررسی کد:

در خط ۴-۸ ،تمام حروف الفبا (حروف بزرگ و کوچک) و کاراکتر space را در متغیر **letters** ذخیره می کنیم.

سپس در خط ۴-۱۸ ، متغیر **letter_locations** را تعریف می‌کنیم که دیکشنری‌ای است که برای هر کاراکتر که در رشته‌ی ورودی قرار دارد، مشخص می‌کند که در کدام خانه‌ی آرایه قرار دارد. (اگر فرض کنیم رشته‌ی ورودی را در آرایه‌ای ذخیره کردیم و هر حرف، در یک خانه‌ی آرایه قرار دارد.)

برای مثال:

S = “letter”

Letter_locations = {‘l’ : [0] , ‘e’ : [1,4] , ‘t’ : [2,3] , ‘r’ : [5]}

همانطور که توضیح داده شد، تمام کاراکتر های موجود در رشته‌ی ورودی، در رشته‌ی نهایی مورد نظر ما قرار ندارند؛ کاراکتر هایی قرار دارند که حداقل k بار تکرار شده باشند. لذا کاراکتر های با این ویژگی را در متغیر letters_repeated_at_least_k ذخیره کردیم.

اگر کاراکتری موجود نباشد که حداقل k بار در رشته‌ی ورودی تکرار شده باشد، پس منطقا نمیتوان زیر رشته ای یافت که k بار تکرار شده باشد و زیر رشته‌ی رشته‌ی ورودی باشد پس طبق خواسته‌ی سوال، در خروجی عبارت Not Exist را چاپ می‌کنیم. (خط ۲۵-۲۷)

زیر رشته‌ی نهایی و خواسته شده‌ی سوال را در متغیر longest_sub_rep_k_times ذخیره می‌کنیم.

پس از پیدا کردن کاراکتر هایی که میتوانند در زیر رشته ی longest_sub_rep_k_times قرار داشته باشند، به خط ۷۳ می‌رویم و در آن بیرونی ترین حلقه‌ی for ، تک تک کاراکترهایی که در letters_repeated_at_least_k قرار دارد را در نظر می‌گیرد و در لیست ending ، اندیس مکان هایی را که این کاراکتر در رشته‌ی داده شده در آن جایگاه قرار دارد را ذخیره می‌کند. سپس به کمک الگوریتم DFS که از رویکرد backtracking نیز استفاده می‌کند، تمام زیر رشته هایی را که شامل این کاراکتر می‌شوند را پیدا می‌کند که البته باید شرط اینکه k بار تکرار شود و زیر رشته‌ی حاصل از k بار تکرار نیز، جزو زیر رشته های رشته‌ی ورودی باشد را داشته باشد و همچنین lexicographically نیز بررسی می‌شود. (این بررسی ها در تابع DFS انجام می‌شود.)
از رویکرد backtracking آنجایی استفاده شده که تمام حالت های پیش آمده (تمام زیر رشته ها) را در تابع DFS می‌یابیم و بررسی می‌کنیم.

پیچیدگی زمانی الگوریتم:

در خط ۷۳ تا ۷۹ ، سه حلقه‌ی for تو در تو داریم که بیرونی ترین حلقه به اندازه‌ی اندازه‌ی لیست letters_repeated_at_least_k اجرا می‌شود که در بدترین حالت می توان فرض کرد که به اندازه‌ی طول رشته‌ی ورودی یعنی n بار اجرا می‌شود. حلقه‌ی دورنی در خط ۷۶ نیز در بدترین حالت به اندازه‌ی رشته‌ی ورودی

اجرا می‌شود و داخلی‌ترین حلقه نیز به اندازه‌ی n بار که طول لیست ending نیز هست اجرا می‌شود. همراه با حلقه‌ی دوم(۷۶)، تابع DFS نیز اجرا می‌شود که در این تابع نیز در خط های ۴۰، ۴۶ و ۴۸، سه حلقه داریم که تو در تو هستند و هرکدام در بدترین حالت، n بار اجرا میشوند و در خط های ۶۰ و ۶۴ نیز دو حلقه‌ی تو در تو داریم. در نهایت نتیجه می‌شود که پیچیدگی زمانی کد $O(n^3)$ است.

مثال:

برای رشته‌ی s = 'letsleetcode ' الگوریتم را مرحله به مرحله اجرا می‌کنیم.

```
Letters = [‘a’, ‘b’, … , ‘A’, ‘B’, ‘ ‘]

Letters_repeated_at_least_k = (‘e’, ‘l’ , ‘t’)

Letters_location = {'a': [], 'A': [], 'b': [], 'B': [], 'c': [8], 'C':
[], 'd': [10], 'D': [], 'e': [1, 5, 6, 11], … ,' ': []}
```

در تابع DFS تمام زیر رشته هایی که کاراکتر های موجود در `letters_repeated_at_least_k` را شامل می‌شود، پیدا میکند.