Department of
Computer Engineering

دانشگاه صنعتی امیرکبیر
( پلی تکنیک تهران )

# به نام خدا

امیرفاضل کوزه گر کالجی

9931099

تمرین تحویلی شماره 3

درس طراحی الگوریتم

# تمرین اول:

فاصله اقلیدسی هر نقطه از مبدا با روش فیثاغورث برابر است با $\sqrt{x^2 + y^2}$

در پیمایش اولیه، با O(n)، تمام n نقطه داده شده را می پیماییم و فاصله شان تا مقصد را محاسبه کرده و ذخیره میکنیم.

خواسته سوال از ما m تا کمترین مقدار از مقادیر ذخیره شده است.

روش اول این است که آرایه ای که مقادیر داخل آن ذخیره شده است را سورت کنیم که میتوانیم از روش های دستی یا سورت های built-in خود زبان برنامه نویسی استفاده کنیم.



روش دیگر استفاده از priority queue هنگام اضافه کردن مقادیر است.
Priority queue را با استفاده از maxheap، implement میکنیم و مقادیر فاصله ها را در آن اضافه میکنیم. وقتی سایز هیپ بزرگتر از m میشود عضو ماکس و ریشه را extract میکنیم تا اندازه ماکس-هیپ همیشه m بماند. به همین دلیل ماکس-هیپ مان همیشه m مقدار کوچکتر را در خود ذخیره میکند.

مرتبه زمانی به دست آوردن و اضافه کردن فاصله O(n) است و مرتبه زمانی extract max value برابر با O(Log n) میباشد که در نهایت مرتبه زمانی کلی برابر با O(n Log n) میشود.

# تمرین دوم:

الف)

```
Parking_space(n):
    If n==1:
        Return n
    If n==2:
        Return n
Retrun parking_space(n-1) + parking_space(n-2)
```



به ازای n = 10 مقدار 89 را دریافت میکنیم.

ب) با استفاده از روش برنامه نویسی پویا و ذخیره مقادیر در یک جدول میتوان از چندین دفعه صدا زده شدن تابع روی یک مقدار جلو گیری کرد و مرتبه زمانی اجرا را از $O(2^n)$ به O(n) رساند. داریم:

```
Dp = []
Parking_sapace(n)
If dp[n] != null:
    Return dp[n]
If n==1:
    Return 1
If n==2:
    Return 2
Dp[n] = parking_space[n-1]+parking_space[2]
Return dp[n]
```

# تمرین سوم:

میدانیم با پرانتز گذاری های متفاوت تعداد عملیات ضرب کردنمان نیز متفاوت خواهد بود. و همچنین برای تعداد کل روش هایی که میتوانیم پرانتز قرار دهیم داریم:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$$

فرض کنید یک دنباله از ماتریس‌ها به صورت $\langle A1,A2,...,An\rangle$ $\langle A1,A2,...,An\rangle$به ما داده شده است و می‌خواهیم مناسب‌ترین راه را برای ضرب این ماتریس‌ها پیدا کنیم.

باید به دنبال اندیس $k$ ای باشیم که بتوانیم دنباله را به $A_{i...k}$ و $A_{k+1..j}$ تقسیم کنیم به گونه ای که $i<j$ باشد. در این صورت هزینه نهایی مان برابر است با هزینه ضرب ماتریس های $i$ تا $k$ به علاوه هزینه ضرب ماتریس های $k+1$ تا $j$ به علاوه هزینه ضرب این دو در هم پس باید به ازای تمام $k$ های میان $j$و$i$ پیمایش کنیم تا بتوانیم جواب بهینه را به دست اوریم:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

از آنجایی که زیر مسئله های مشابه تکرار میشوند میتوانیم از برنامه نویسی پویا بهره ببریم.

در این مسئله dp را به شکل زیر تعریف میکنیم:

Dp[i][j] برابر است با تعداد ضرب های مورد نیاز برای محاسبه ماتریس $A_{i..j}$

و اگر i = j در آن صورت dp[i][j] برابر با 0 میشود.

راه حل جدید به شکل زیر میشود:

$$dp[i][j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \{dp[i][k] + dp[k+1][j] + p[i-1].p[k].p[j]\} & i < j \end{cases}$$

پاسخ ما در خانه dp[1][n] وجود دارد که قطر اصلی آن برابر 0 است و قسمت پایین مثلثی آن مقداری نمیگیرد .

با اجرای الگوریتم بالا بر روی نمونه داده شده مینیمم تعداد ضرب ها برابر میشود با 132. این مقدار در صورتی قابل به دست آمدن است که k برابر با 1 یا همان اندیس اولین ماتریس باشد. پس پرانتز گذاری به شکل زیر میشود.

$A_1(A_2A_3A_4)$

# سوال 4:

برای حل این سوال به روش پویا، جدول زیر را تشکیل میدهیم و به ازای هر سطر i ، بر تمام اعضای ستون های j پیمایش میکنیم خانه هایی که i یا j برابر با 0 دارند را برابر با 0 قرار میدهیم تا مقدار اولیه ای برای تولید مقادیر باقی جایگاه های جدول داشته باشیم. الگوریتم مان برای به دست اوردن اعداد داخل در هر پیمایش جدول به شکل زیر است:

```
If (A[i] == B[j])
        LCS [i ,j] = 1+(LCS[i-1, j-1])
Else
        LCS [i ,j] = max(LCS[i-1, j], LCS[I , j-1])
```



| | | j | T | G | C | A | T | T | A |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| i | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| G | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 3 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| T | 4 | 0 | 1 | 1 | 1 | 1 | 2 | 3 | 3 |
| C | 5 | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 3 |
| G | 6 | 0 | 1 | 2 | 2 | 2 | 2 | 3 | 3 |

طول طولانی ترین رشته مشترک برابر است با 3 (پایین-راست جدول). برای به دست اوردن رشته مشترک از پایین-راست ترین مقدار شروع به پیمایش میکنیم. اگر مقدار ستون یا سطر قبلیش با آن برابر بود، یعنی این عدد جدید تولید نشده پس کاراکتری که آن را نشان میدهد عضو زیر رشته مشترکمان نیست

به عبارت دیگر هرگاه که یک پیکان اریب مشاهده میشود یعنی کاراکتر مربوط به مبدا آن پیکان، عضو زیر رشته مشترکمان میباشد.

"GTT" زیر رشته مشترک مورد نظرمان می باشد.



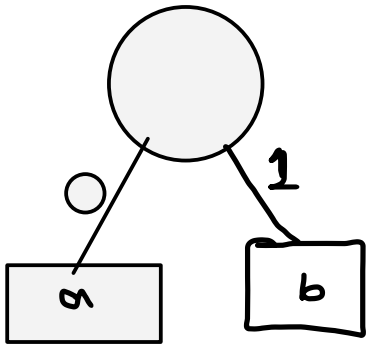
## سوال پنجم:

## سوال ششم:

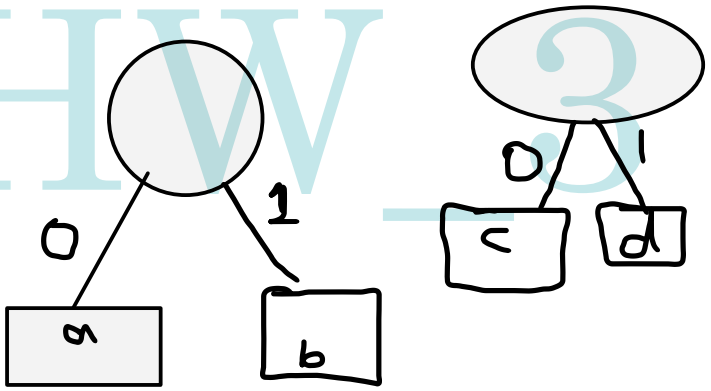
کاراکتر ها درون صف اولویت را به صورت زیر در نظر میگیریم(چپ به راست فراوانی بیشتر میشود):

a , b, c, d, e, f, g, h

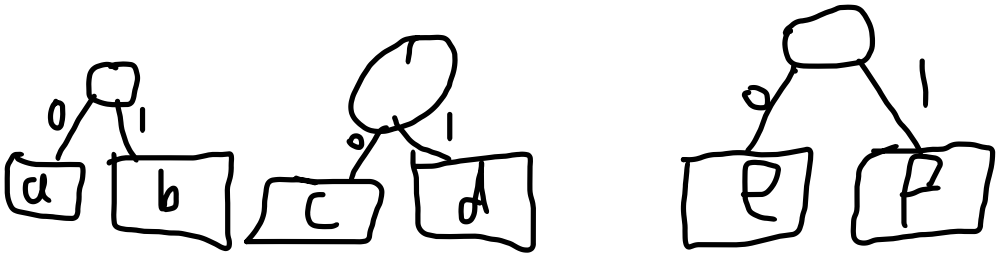
1) c, d, e, f, g, h



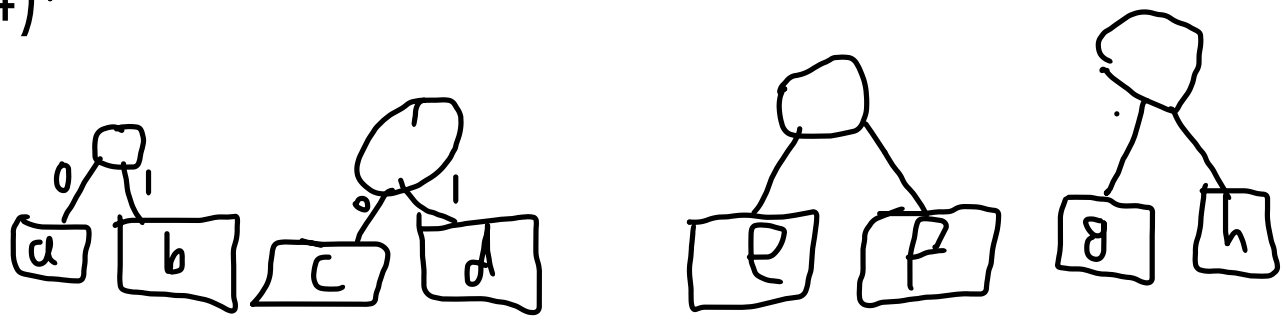

2) e , f , g, h



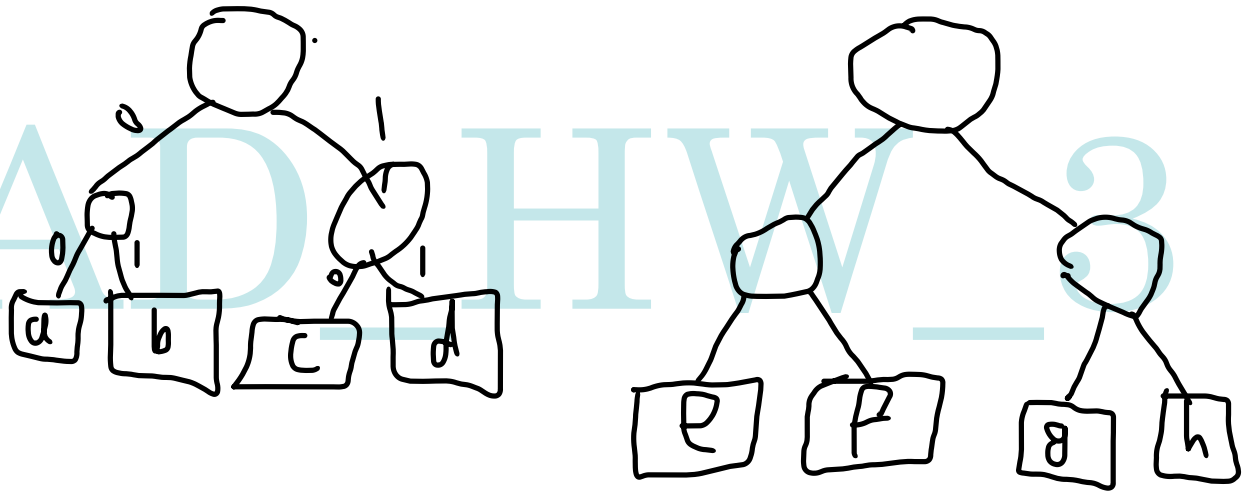

3) g , h



4)

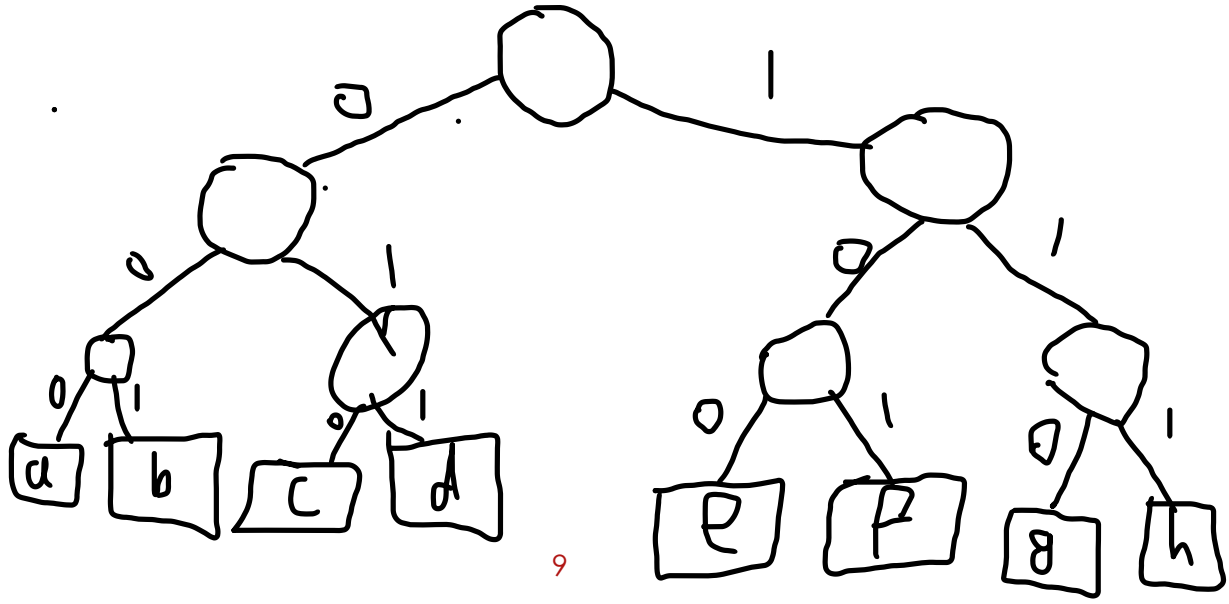



5)






6)



AD_HW_3