longest common subsequence

مسئله: دو دنباله داده شده است.

$X = \langle x_1, \ldots, x_m \rangle$

$Y = \langle y_1, \ldots, y_n \rangle$

یک زیردنباله مشترک با بیشترین پیدایی کنیم.

- زیردنباله لزومی ندارد که پی در پی باشد.
- زیردنباله باید در ترتیب خودش باشد.

Examples:

s p r i n g t i me
p i o n e e r

h o r s e b a c k
s n o w f l a k e

m a e l s t r o m
b e c a l m

h e r o i c a l l y
s c h o l a r l y

یک الگوریتم brute-force:

برای هر زیردنباله از X، بررسی کن که آیا زیردنباله Y هست یا نه؟

Time: $\Theta(n2^m)$

- 2^m زیردنباله از X وجود دارد که باید بررسی شود.
- هر زیردنباله زمان $\Theta(n)$ نیاز دارد که بررسی شود:
  - Y را بررسی کن که اولین حرف را بیابی.
  - از آنجا به دنبال حرف دوم بگرد.
  - ادامه همین روند.

Optimal Substructure

Notation:

$X_i = prefix(x_1, \ldots, x_i)$

$Y_i = prefix(y_1, \ldots, y_i)$

قضیه: اگر $Z = (z_1, \ldots, z_k)$ یک LCS از X و Y باشد.

۱- اگر $x_m = y_n$ پس $z_k = x_m = y_n$، Z_{k-1} یک LCS از X_{m-1} و Y_{n-1} است.

۲- اگر $x_m \neq y_n$ آن‌گاه $z_k \neq x_m$، درنتیجه Z یک LCS از X_{m-1} و Y است.

۳- اگر $x_m \neq y_n$ آن‌گاه $z_k \neq y_n$، درنتیجه Z یک LCS از X و Y_{n-1} است.

بنابراین، LCS برای دو دنباله شامل یک prefix است که یک LCS برای prefix های آن دو دنباله است.

Recursive formulation

تعریف می‌کنیم

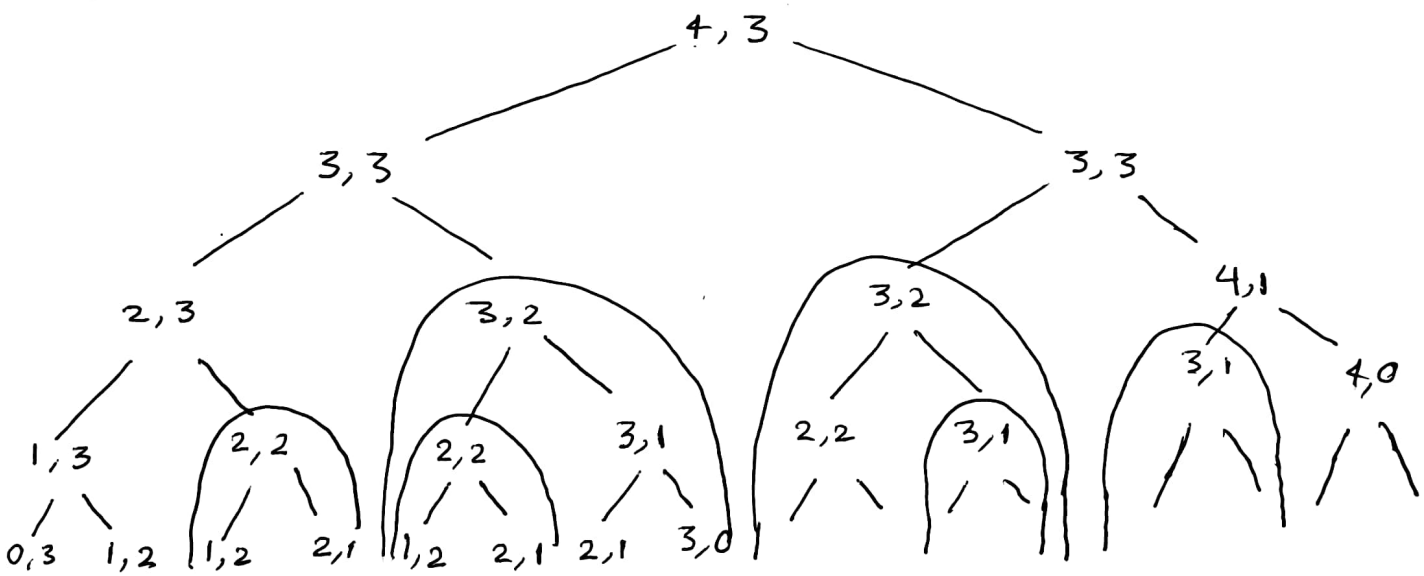
$$c[i,j] = \text{length of LCS of } X_i \text{ and } Y_j$$

به دنبال $c[m,n]$ هستیم.

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{if } i,j>0 \text{ and } x_i \neq y_i \end{cases}$$

دوباره، براساس این فرمول بایستی یک الگوریتم بازگشتی بنویسیم.

bozo
bat



- تعداد بسیار زیادی زیرمسئله تکراری وجود دارد.
- به جای محاسبه مجدد، آنها را در یک جدول ذخیره می‌کنیم.

compute length of optimal solution

```
LCS-LENGTH (X, Y, m, n)
 for i ← 1  to m
    do  c [i,0] ← 0
 for j ← 0  to  n
    do   c [0,j] ← 0
 for  i ← 1  To  m
     do for  j ← 1  To  n
          do  if  x_i = y_j
                then  c [i,j] ← c [i-1, j-1] + 1
                      b [i,j] ← "↖"
                else  if  c [i-1,j] ≥ c [i,j-1]
                        then c[i,j] ← c [i-1,j]
                             b [i,j] ← "↑"
                        else c [i,j] ← c [i,j-1]
                             b [i,j] ← "←"

return  c and b
```

```
PRINT-LCS ( b, X, i, j )
if i=0 or j=0
   then return
if b[i,j] = "↖"
   then PRINT-LCS (b, X, i-1, j)
        print x_i
else if b[i,j] = "↑"
   then PRINT-LCS( b, X, i-1, j)
else PRINT-LCS (b, X , i, j-1)
```

فراخوانی اولیه به صورت :

```
PRINT-LCS( b, X , m , n)
```

b[i,j] به مدخلی از جدول اشاره می کند که زیرمسئله آن در حل استفاده می شود.

در این حالت b[i,j] = ↖ رابطه LCS را توسعه می دهیم character.

Demonstration: show only $c[i, j]$:

زمان : $\Theta(mn)$

		a	**m**	**p**	**u**	**t**	**a**	**t**	**i**	**o**	**n**
	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	1	1	1	1	1	1	1	1
a	0	1	1	1	1	1	2	2	2	2	2
n	0	1	1	1	1	1	2	2	2	2	3
k	0	1	1	1	1	1	2	2	2	2	3
i	0	1	1	1	1	1	2	2	3	3	3
n	0	1	1	1	1	1	2	2	3	3	4
g	0	1	1	1	1	1	2	2	3	3	4
				p			**a**		**i**		**n**

Optimal binary search trees

- یک دنباله $k = (k_1, k_2, \ldots, k_n)$ دارای n key متمایز و مرتب شده داده شده است

$$k_1 < k_2 < \ldots < k_n$$

- می خواهیم یک درخت جستجوی دودویی (BST) از این key ها بسازیم.
- برای k_i با احتمال P_i جستجو انجام می شود.
- یک BST با هزینه جستجوی expected مینیمم می خواهیم.
- Actual Cost = # of items examined

برای k_i $\quad cost = depth_T(k_i) + 1$

عمق k_i در درخت جستجوی T

$$E[\text{search cost in } T] = \sum_{i=1}^{n} (depth_T(k_i) + 1) \cdot P_i$$

$$= \sum_{i=1}^{n} depth_T(k_i) \cdot P_i + \sum_{i=1}^{n} P_i$$

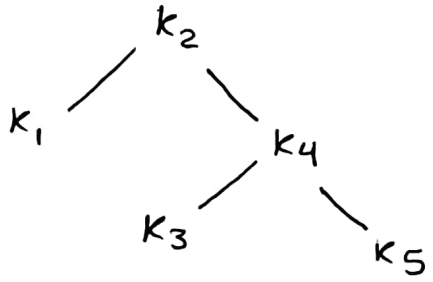
$$= 1 + \sum_{i=1}^{n} depth_T(k_i) \cdot P_i$$

مجموع تمام احتمالات برابر 1 است.

فرض می کنیم که همه جستجوها موفق هستند

i	1	2	3	4	5
P_i	.25	.2	.05	.2	.3

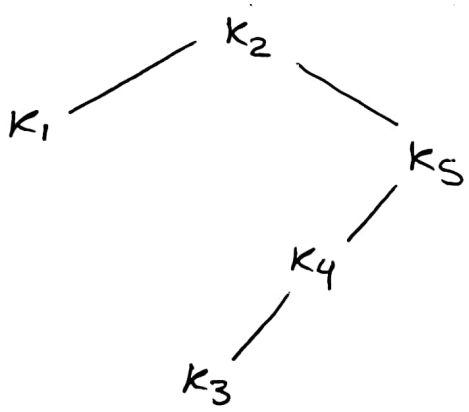
Example:



i	$depth_T(k_i)$	$depth_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

بنابراین:

$E[\text{search cost}] = 2.15$



i	$depth_T(k_i)$	$depth_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	3	.15
4	2	.4
5	1	.3
		1.10

بنابراین:

$E[\text{search cost}] = 2.10$

کمینه است

Observations :

- optimal BST ممکن است که کمترین ارتفاع را نداشته باشد.
- optimal BST ممکن است که key با احتمال بیشترین را در root نداشته باشد.

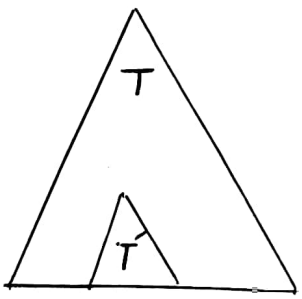
یک روش ؟

- همه BST های دارای n تا گره را می سازیم.
- key ها را به آنها اختصاص می دهیم.
- expected search cost را محاسبه می کنیم.
- $\Omega(4^n/n^{3/2})$ تا BST مختلف وجود دارد که n تا گره دارند.

Optimal substructure

تمام زیردرخت های یک BST را در نظر بگیرید که شامل key های $k_i, \ldots, k_j$ است

$1 \le i \le j \le n$



اگر T یک optimal BST باشد و T شامل زیردرخت T' با key های $k_i, \ldots, k_j$ باشد آن گاه T' باید یک optimal BST بروی $k_i, \ldots, k_j$ باشد.

با استفاده از optimal substructure یک راه حل بهینه می سازیم