1.绪论

(f) 动态规划

Make it work,

make it right,

make it fast.

\- Kent Beck

邓俊辉

deng@tsinghua.edu.cn

fib()：递归

- fib(n) = fib(n-1) + fib(n-2) : {0, 1, 1, 2, 3, 5, 8, ...}
- `int fib(n) { return (2 > n) ? n : fib(n - 1) + fib(n - 2); }` //为何这么慢？
- 复杂度： $T(0) = T(1) = 1; \quad T(n) = T(n-1) + T(n-2) + 1, \quad n > 1$

  令 $S(n) = [T(n) + 1] / 2$

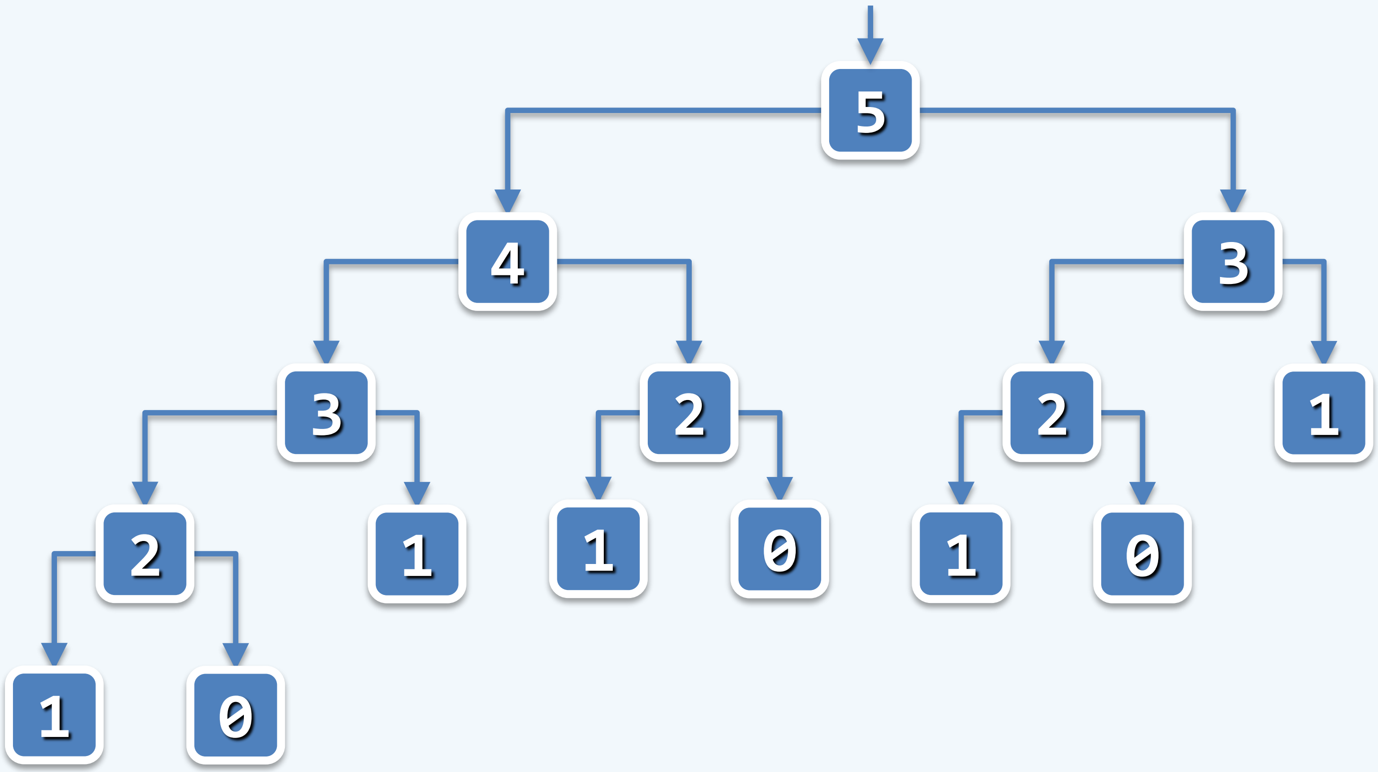
  则 $S(0) = 1 = fib(1), \; S(1) = 1 = fib(2)$

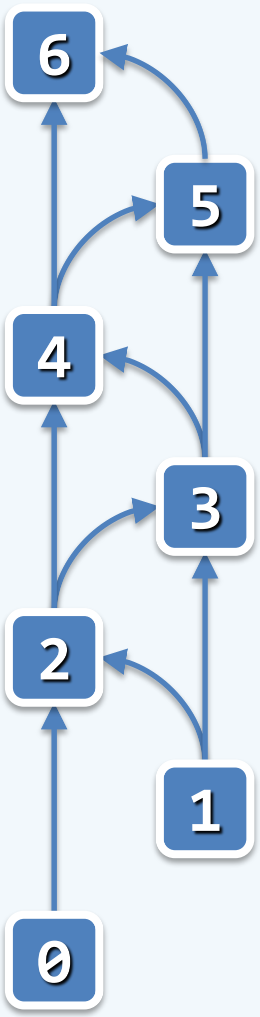
  故 $S(n) = S(n-1) + S(n-2) = fib(n+1)$ $\quad //\Phi = \frac{1+\sqrt{5}}{2} = 1.61803...$

  $T(n) = 2*S(n) - 1 = 2*fib(n+1) - 1 = \mathcal{O}(fib(n+1)) = \mathcal{O}(\Phi^n) = \mathcal{O}(2^n)$
- $\Phi^{36} = 2^{25}, \quad \Phi^{43} = 2^{30} = 10^9$ flo = 1 sec
- $\Phi^{5} = 10, \quad \Phi^{67} = 10^{14}$ flo = 10^5 sec = 1 day
- $\Phi^{92} = 10^{19}$ flo = 10^{10} sec = 10^5 day = 3 century

fib()：递归

❖ 递归版fib()低效的根源在于，各递归实例均被大量重复地调用





❖ 先后出现的递归实例，共计$\mathcal{O}(\Phi^n)$个；而去除重复之后，总共不过$\mathcal{O}(n)$种

fib()：迭代

❖ 解决方法A（记忆：memoization）

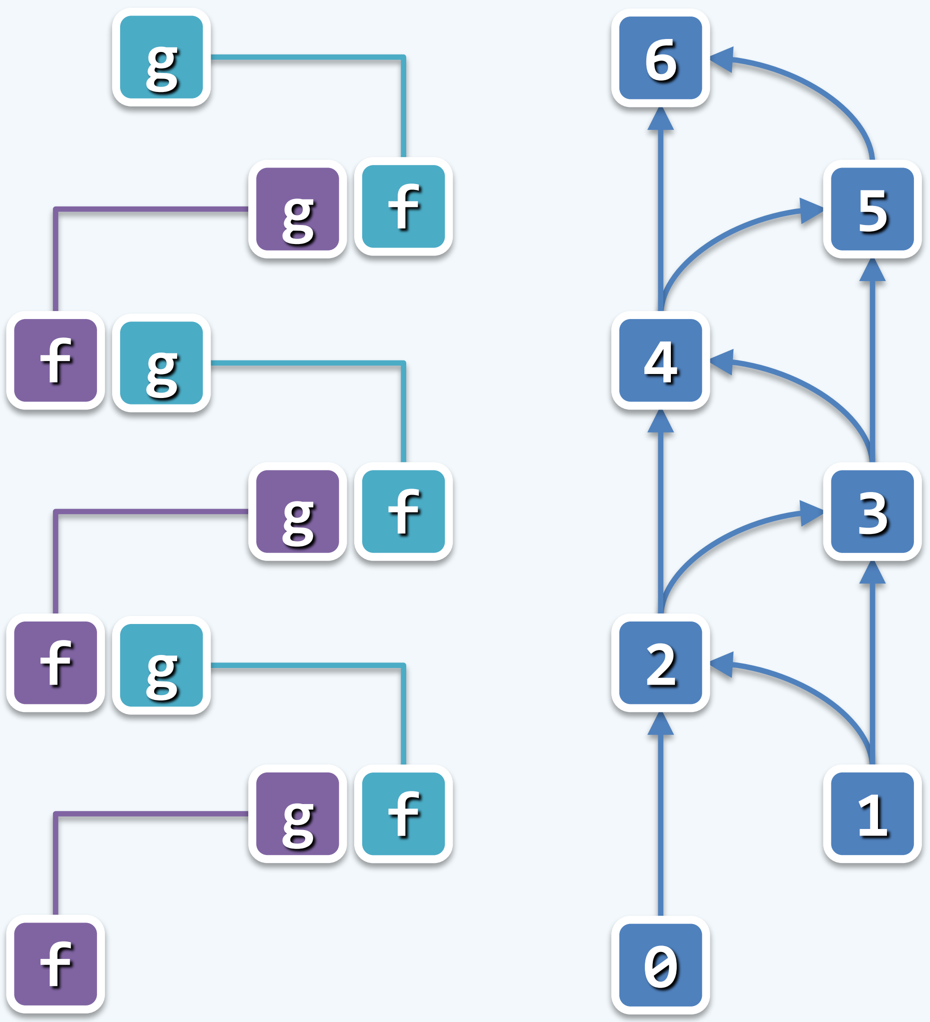
将已计算过实例的结果制表备查

❖ 解决方法B（动态规划：dynamic programming）

颠倒计算方向：由自顶而下递归，为自底而上迭代

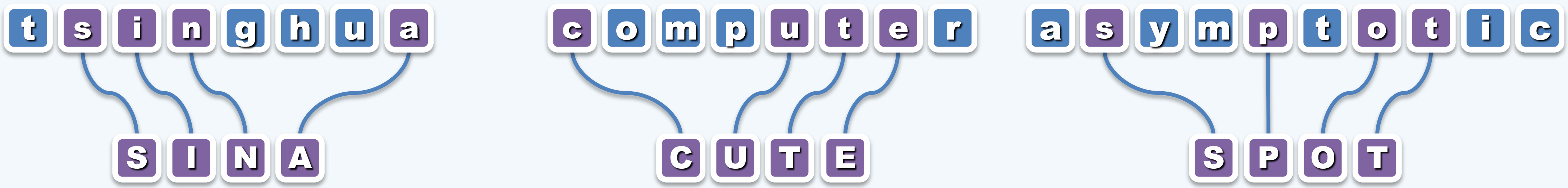
```
f = 0; g = 1; //fib(0), fib(1)
while (0 < n--) {
   g = g + f;
   f = g - f;
}
return g;
```

❖ T(n) = $\mathcal{O}(n)$，而且仅需$\mathcal{O}(1)$空间！



LCS：递归

❖ 子序列（Subsequence）：由序列中若干字符，按原相对次序构成



❖ 最长公共子序列（Longest Common Subsequence）：两个序列公共子序列中的最长者

可能有多个

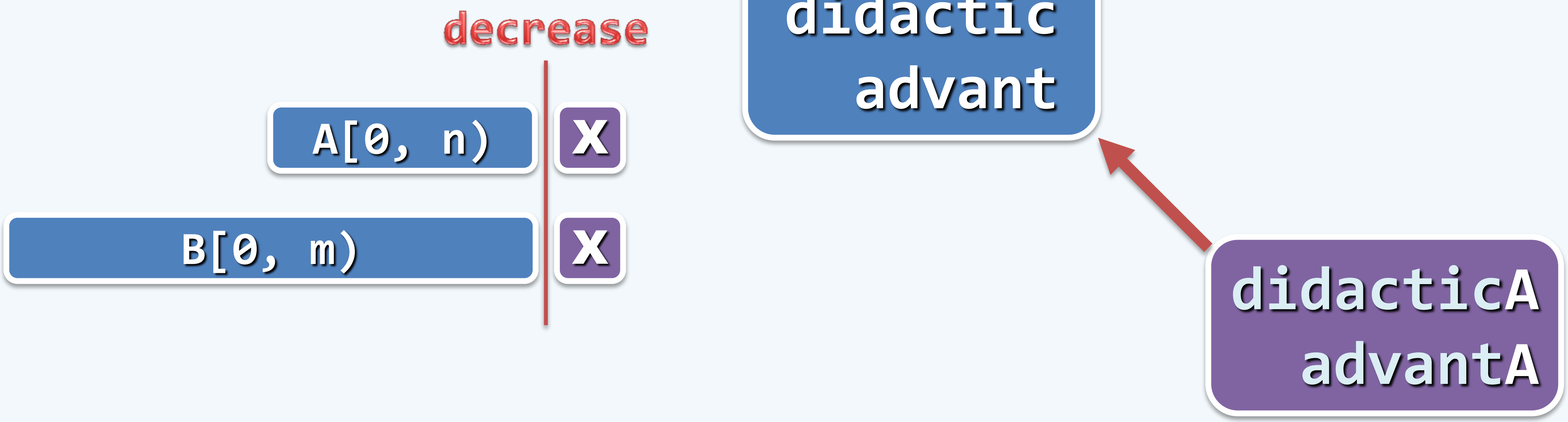
可能有歧义

LCS：递归

❖ 对于序列A[0, n]和B[0, m]，LCS(A, B)无非三种情况

0) 若n = -1或m = -1，则取作空序列("")　　　　//递归基

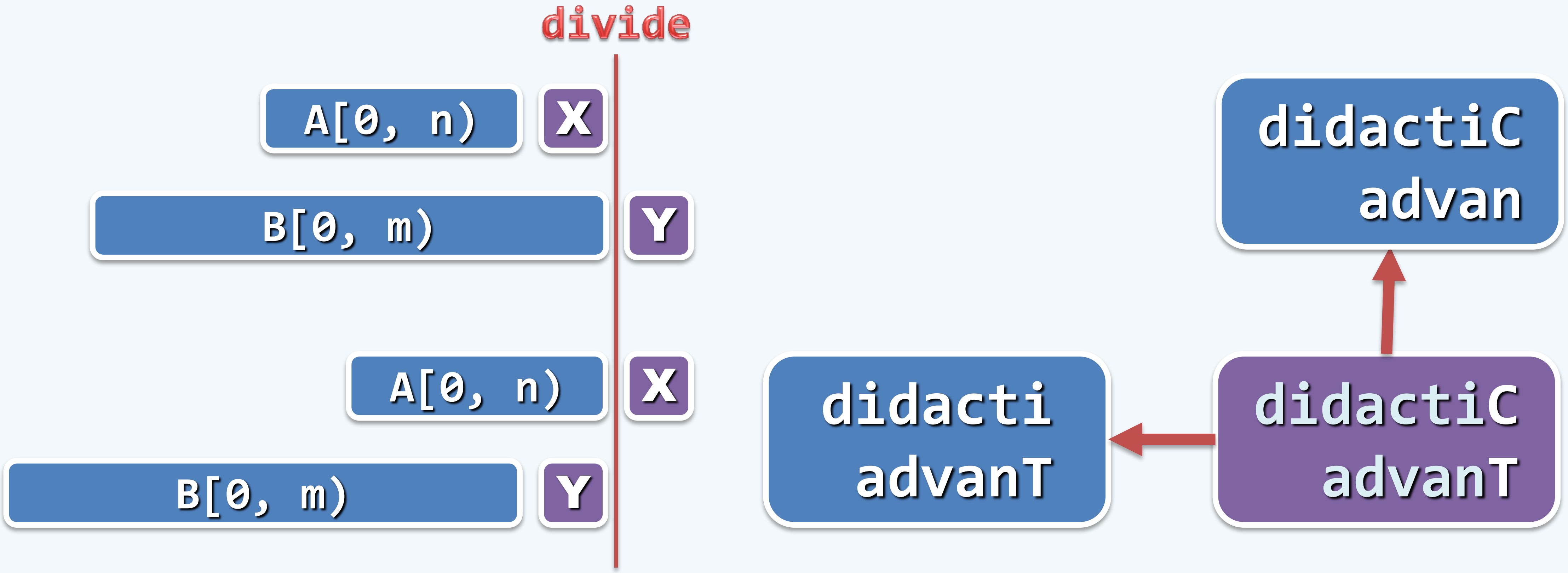
1) 若A[n] = 'X' = B[m]，则取作：LCS(A[0, n), B[0, m)) + 'X'　　　　//减而治之



LCS：递归

2) A[n] ≠ B[m]，则在 LCS(A[0, n], B[0, m)) 与

LCS(A[0, n), B[0, m]) 中取更长者 //分而治之



LCS：理解

❖ LCS的每一个解，对应于(0, 0)与(n, m)之间的一条单调通路；反之亦然

		E	D	U	C	A	T	I	O	N	A	L
	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1	1	1	1	1
D	0	0	1	1	1	1	1	1	1	1	1	1
V	0	0	1	1	1	1	1	1	1	1	1	1
A	0	0	1	1	1	2	2	2	2	2	2	2
N	0	0	1	1	1	2	2	2	2	3	3	3
T	0	0	1	1	1	2	3	3	3	3	3	3
A	0	0	1	1	1	2	3	3	3	3	4	4
G	0	0	1	1	1	2	3	3	3	3	4	4
E	0	1	1	1	1	2	3	3	3	3	4	4

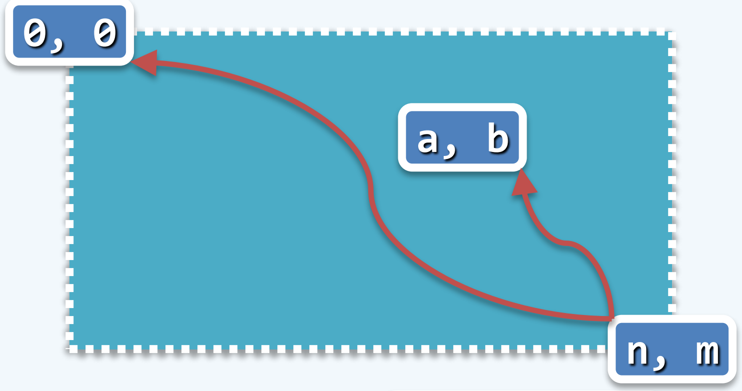
多解

		D	I	D	A	C	T	I	C	A	L
	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1	1	1	1	1
D	0	1	1	1	1	1	1	1	1	1	1
V	0	1	1	1	1	1	1	1	1	1	1
A	0	1	1	1	2	2	2	2	2	2	2
N	0	1	1	1	2	2	2	2	2	2	2
T	0	1	1	1	2	2	3	3	3	3	3
A	0	1	1	1	2	2	3	3	3	4	4
G	0	1	1	1	2	2	3	3	3	4	4
E	0	1	1	1	2	2	3	3	3	4	4

❖

歧义

LCS：递归



❖ 单调性：无论如何，每经过一次比对，原问题的规模必可减小

具体地，作为输入的两个序列，至少其一的长度缩短一个单位

❖ 最好情况（不出现第2种情况）下，只需$\mathcal{O}$(n + m)时间

❖ 但问题在于，（在第2种情况）原问题将分解为两个子问题

更糟糕的是，它们在随后进一步导出的子问题，可能雷同

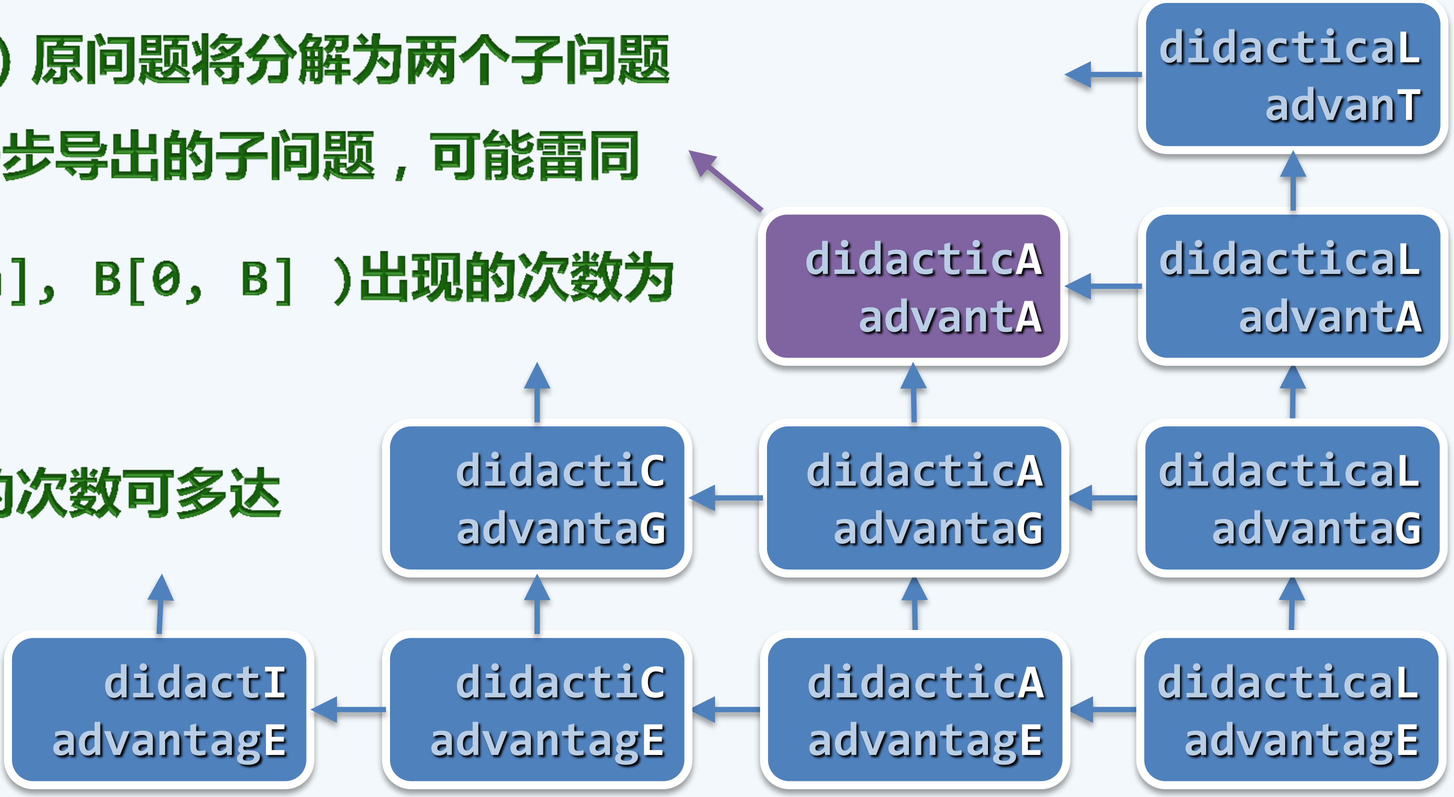
❖ 在最坏情况下，LCS(A[0, a], B[0, B])出现的次数为

$$\binom{n+m-a-b}{n-a} = \binom{n+m-a-b}{m-b}$$

特别地，LCS(A[0], B[0])的次数可多达

$$\binom{n+m}{n} = \binom{n+m}{m}$$

当n = m时，为$\mathcal{O}(2^n)$



LCS：迭代

- 与fib()类似
  这里也有大量重复的递归实例（子问题）
  （最坏情况下）先后共计出现$\mathcal{O}(2^n)$个
- 各子问题，分别对应于A和B的某个前缀组合
  因此总共不过$\mathcal{O}(n * m)$种
- 采用动态规划的策略
  只需$\mathcal{O}(n * m)$时间即可计算出所有子问题
- 为此，只需
  0）将所有子问题（假想地）列成一张表
  1）颠倒计算方向，从LCS(A[0], B[0])出发
  　　依次计算出所有项

		d	i	d	a	c	t	i	c	a	l
	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	1	1	1	1	1	1	1
d	0	1	1	1	1	1	1	1	1	1	1
v	0	1	1	1	1	1	1	1	1	1	1
a	0	1	1	1	2	2	2	2	2	2	2
n	0	1	1	1	2	2	2	2	2	2	2
t	0	1	1	1	2	2	3	3	3	3	3
a	0	1	1	1	2	2	3	3	3	4	4
g	0	1	1	1	2	2	3	3	3	4	4
e	0	1	1	1	2	2	3	3	3	4	4

课后

- 温习：程序设计基础（第3版）之第11章（动态规划）
- 自学：Introduction to Algorithms, §15.1, §15.3, §15.4
- 本节所介绍的迭代式LCS算法，似乎需要记录每个子问题的局部解，从而导致空间复杂度激增

  实际上，这既不现实，亦无必要

  试改进该算法，使得每个子问题只需常数空间，即可保证最终得到LCS的组成（而非仅仅长度）
- 考查序列 A = "immaculate" 和 B = "computer"

  1）它们的LCS是什么？

  2）这里的解是否唯一？是否有歧义性？

  3）按照本节所给的算法，找出的是其中哪一个解?
- 实现LCS算法的递归版和迭代版，并通过实测比较其运行时间
- 采用memoization策略，改进fib()与LCS算法的递归版