01-F2

绪论

动态规划：最长公共子序列

Make it work, make it right, make it fast.

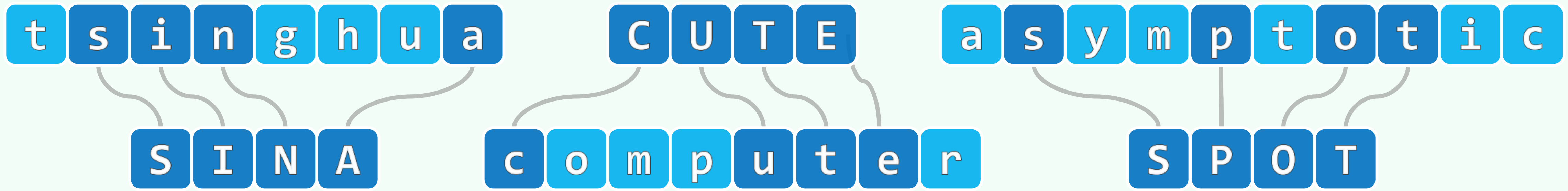
世上一切都无独有偶，为什么你与我却否？

邓俊辉

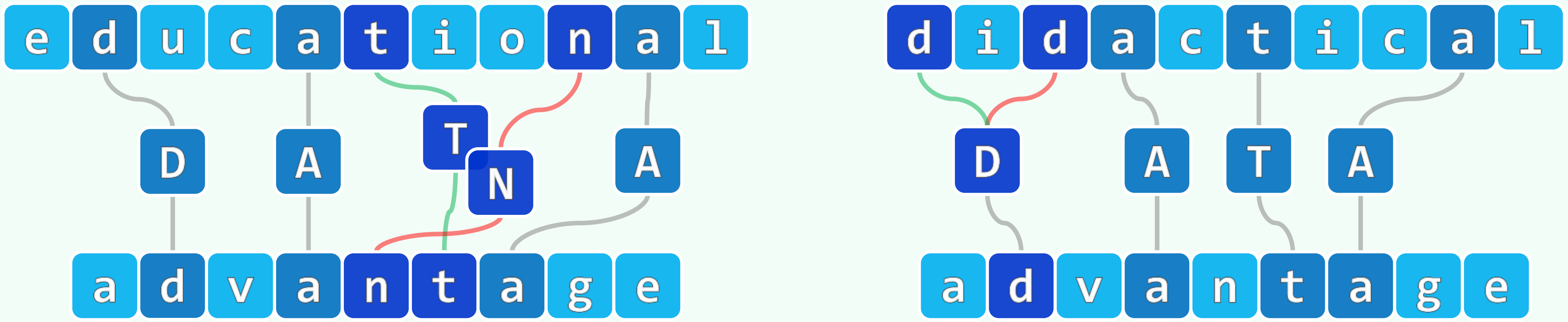
deng@tsinghua.edu.cn

Longest Common Subsequence

❖ 子序列 (Subsequence) : 由序列中若干字符，按原相对次序构成

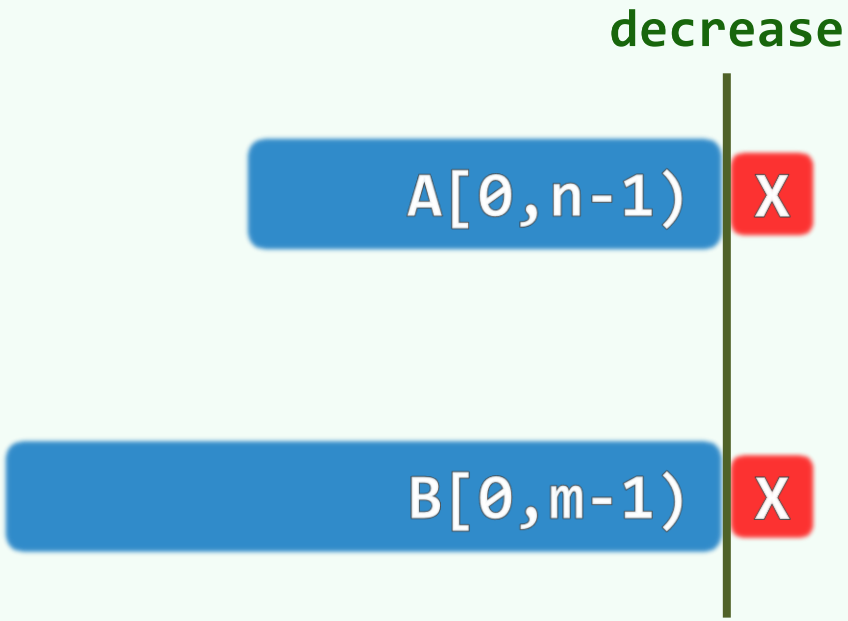


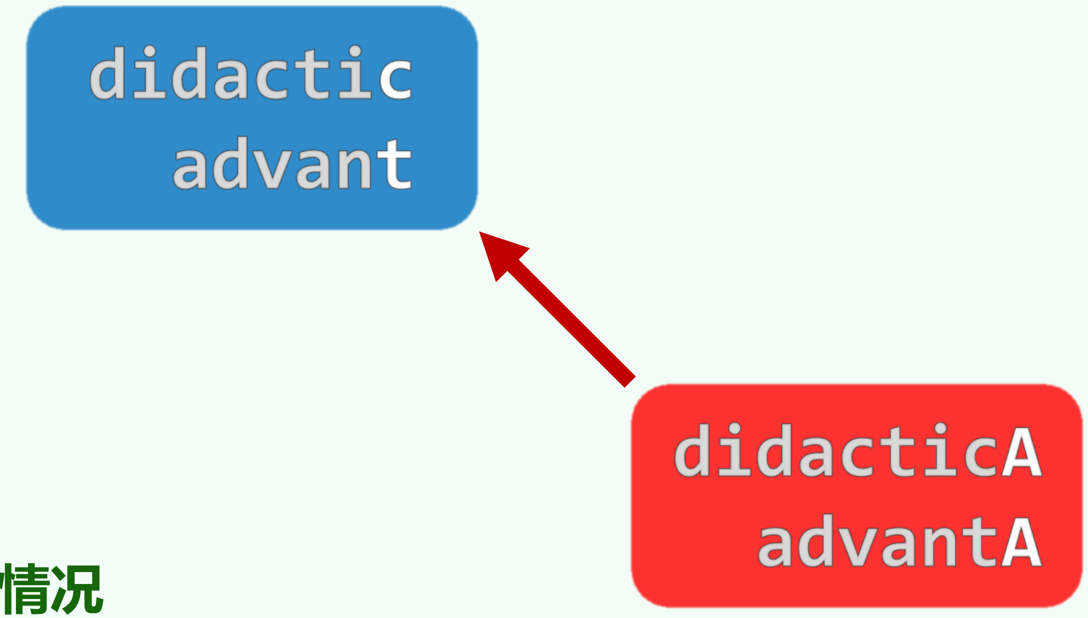
❖ 最长公共子序列：两个序列之间公共子序列中的最长者



可能有多个　　　　　　　　可能有歧义

减治递归



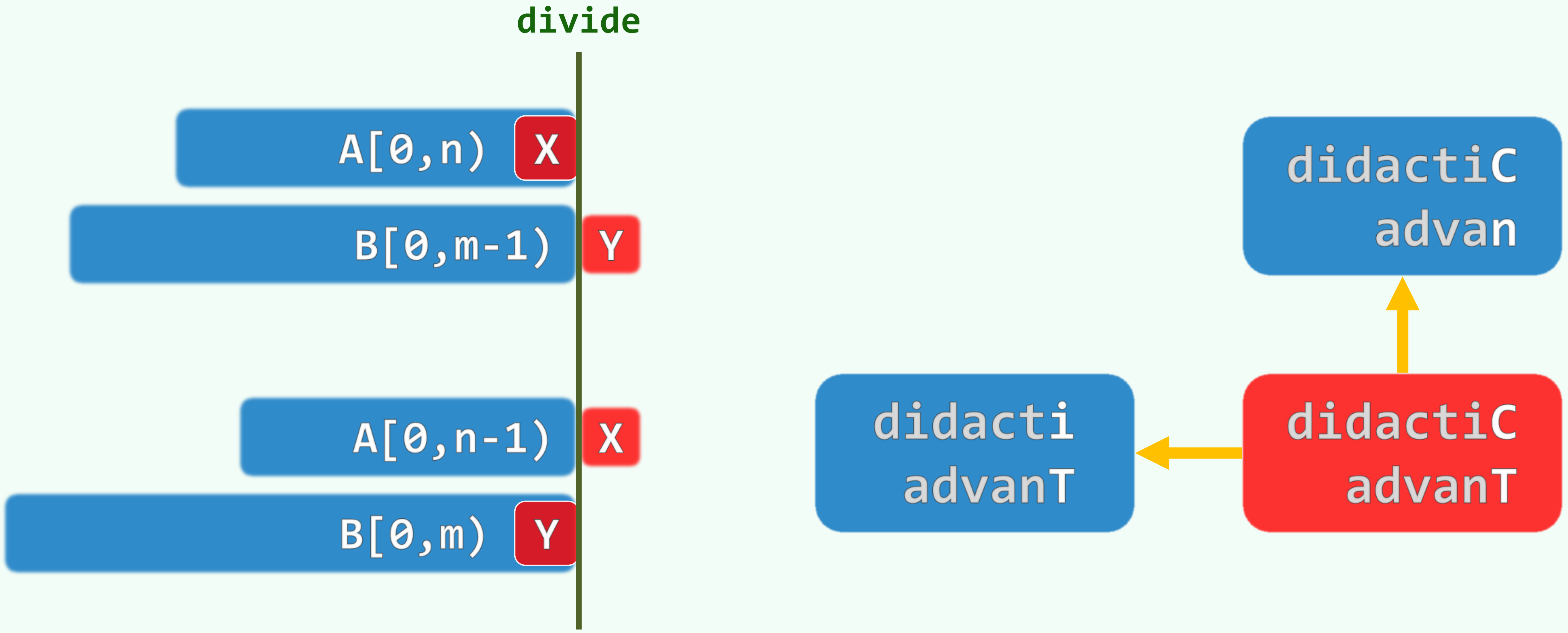


对于序列A[0,n)和B[0,m)，LCS(n,m)无非三种情况

0）若 n = 0 或 m = 0，则取作空序列（长度为零）　//递归基：必然总能抵达

1）若A[n-1] = 'X' = B[m-1]，则取作：LCS(n-1,m-1) + 'X'

分治递归



2) A[n-1] ≠ B[m-1]，则在 LCS(n,m-1) 与 LCS(n-1,m) 中取更长者

描述：伪代码

```
Input: two strings A and B of length n and m resp.,

Output: (the length of) the longest common subsequence of A and B

lcs(A[], n, B[], m)

   Compare the last characters of A and B, i.e., A[n-1] and B[m-1]

   If A[n-1] = B[m-1]

      Compute x = lcs(A, n-1, B, m-1) recursively and return 1 + x

   Else

      Compute x = lcs(A, n-1, B, m) & y = lcs(A, n, B, m-1) and return max(x, y)

   As the recursion base, return 0 when either n or m is 0
```

实现：递归版

```
unsigned int lcs( char const * A, int n, char const * B, int m ) {
   if (n < 1 || m < 1) //trivial cases
      return 0;
   if ( A[n-1] == B[m-1] ) //decrease & conquer
      return 1 + lcs( A, n-1, B, m-1 );
   else //divide & conquer
      return max( lcs( A, n-1, B, m ), lcs( A, n, B, m-1 ) );
}
```

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

复杂度

- 每经一次比对，至少一个序列的长度缩短一个单位
- 最好情况，只需 $\mathcal{O}(n+m)$ 时间　//比如...
- 然而最坏情况下，不仅子问题**数量**巨大，而且大量**重复**

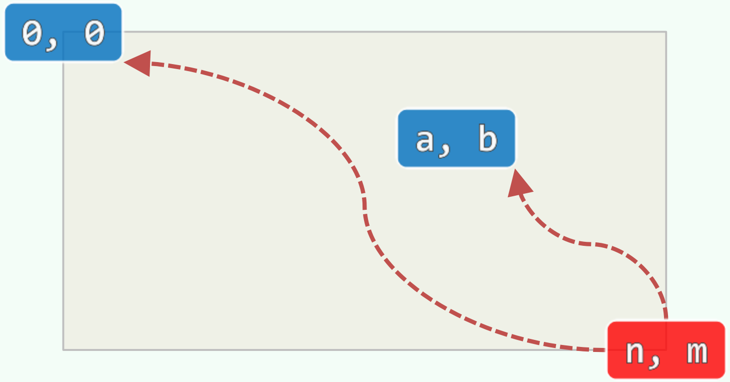
子任务LCS(A[a],B[b])重复的次数，可能多达为

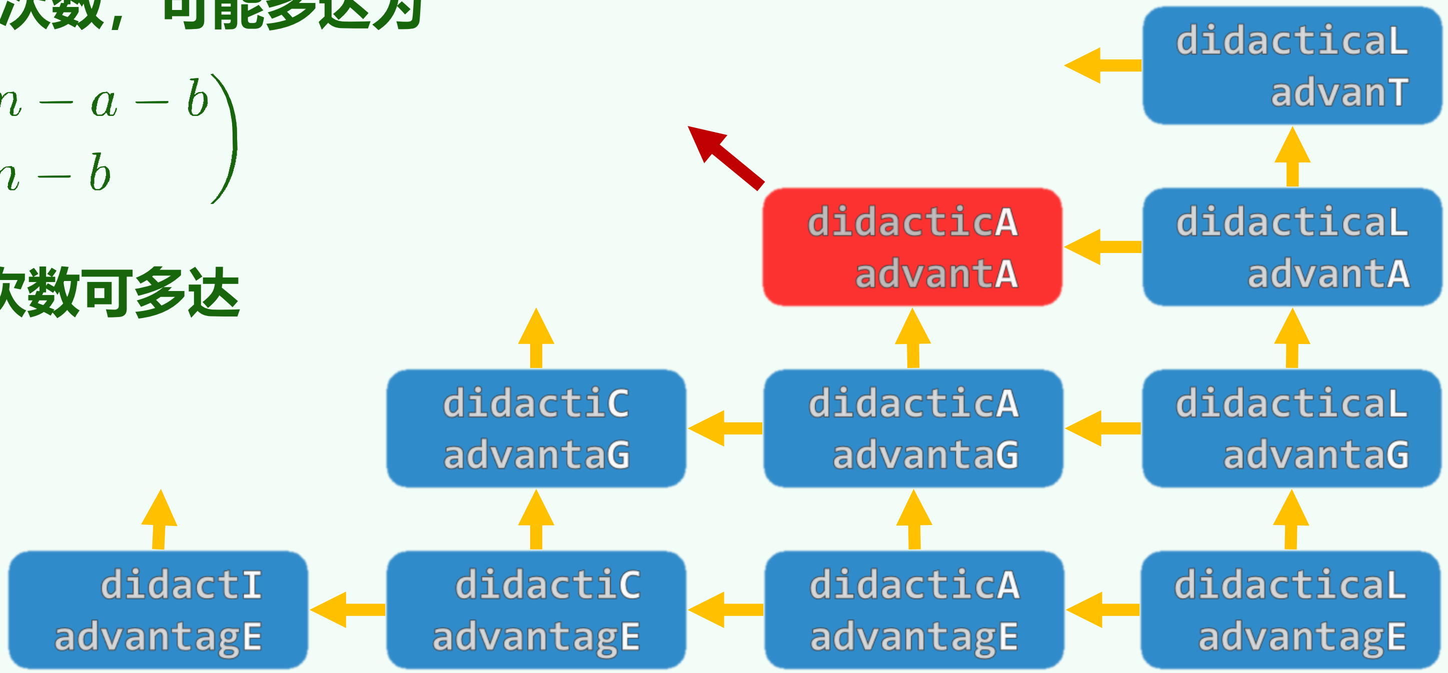
$$\binom{n+m-a-b}{n-a} = \binom{n+m-a-b}{m-b}$$

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 $\Omega(2^n)$





实现：记忆化版

```
unsigned int lcsMemo( char const* A, int n, char const* B, int m ) {
   unsigned int * lcs = new unsigned int[n*m]; memset(lcs, 0xFF, sizeof(unsigned int)*n*m);
   unsigned int solu = lcsM( A, n, B, m, lcs, m ); delete[] lcs; return solu;
}
unsigned int lcsM( char const * A, int n, char const * B, int m,
                   unsigned int * const lcs, int const M ) {
   if ( n < 1 || m < 1 ) return 0; //trivial cases
   if ( UINT_MAX != lcs[(n-1)*M + m-1] ) return lcs[(n-1)*M + m-1]; //recursion stops
   return lcs[(n-1)*M + m-1]  =
            ( A[n-1] == B[m-1] ) ?
               1 + lcsM( A, n-1, B, m-1, lcs, M )
               max( lcsM( A, n-1, B, m, lcs, M ), lcsM( A, n, B, m-1, lcs, M ) );
}
```

动态规划

❖ 与fib()类似，这里也有大量重复的递归实例（子问题）

各子问题，分别对应于A和B的某个前缀组合

因此实际上，总共不过 $\mathcal{O}(n \cdot m)$ 种

❖ 采用动态规划的策略

只需 $\mathcal{O}(n \cdot m)$ 时间即可计算出所有子问题

❖ 为此，只需

- 将所有子问题（假想地）列成一张表
- 颠倒计算方向：从LCS(0,0)出发，依次计算出所有项——直至LCS(n,m)

		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

实现：迭代（动态规划)版

```
unsigned int lcs(char const * A, int n, char const * B, int m) {
   if (n < m) { swap(A, B); swap(n, m); } //make sure m <= n
   unsigned int* lcs1 = new unsigned int[m+1]; //the current two rows are
   unsigned int* lcs2 = new unsigned int[m+1]; //buffered alternatively
   memset( lcs1, 0x00, sizeof(unsigned int) * (m+1) ); lcs2[0] = 0; //sentinels
   for ( int i = 0; i < n; swap( lcs1, lcs2 ), i++ )
      for ( int j = 0; j < m; j++ )
         lcs2[j+1] = ( A[i] == B[j] ) ? 1 + lcs1[j] : max( lcs2[j], lcs1[j+1] );
   unsigned int solu = lcs1[m]; delete[] lcs1; delete[] lcs2; return solu;
}
```