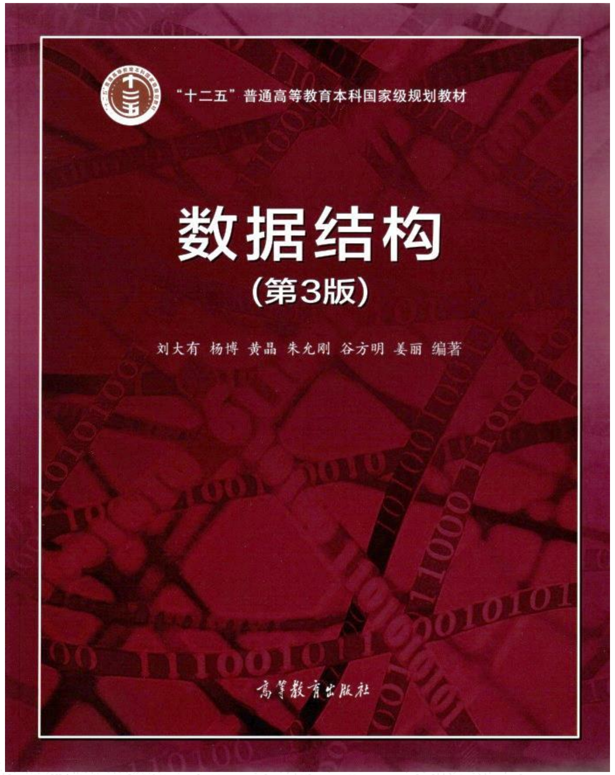


“十二五”普通高等教育本科国家级规划教材

数据结构

(第3版)

刘大有 杨博 黄晶 朱允刚 谷方明 姜丽 编著

高等教育出版社

字符串的模糊匹配

- 基本概念
- 字符串的编辑距离
- 最长公共子序列

数据之法

结构之美

算法之道

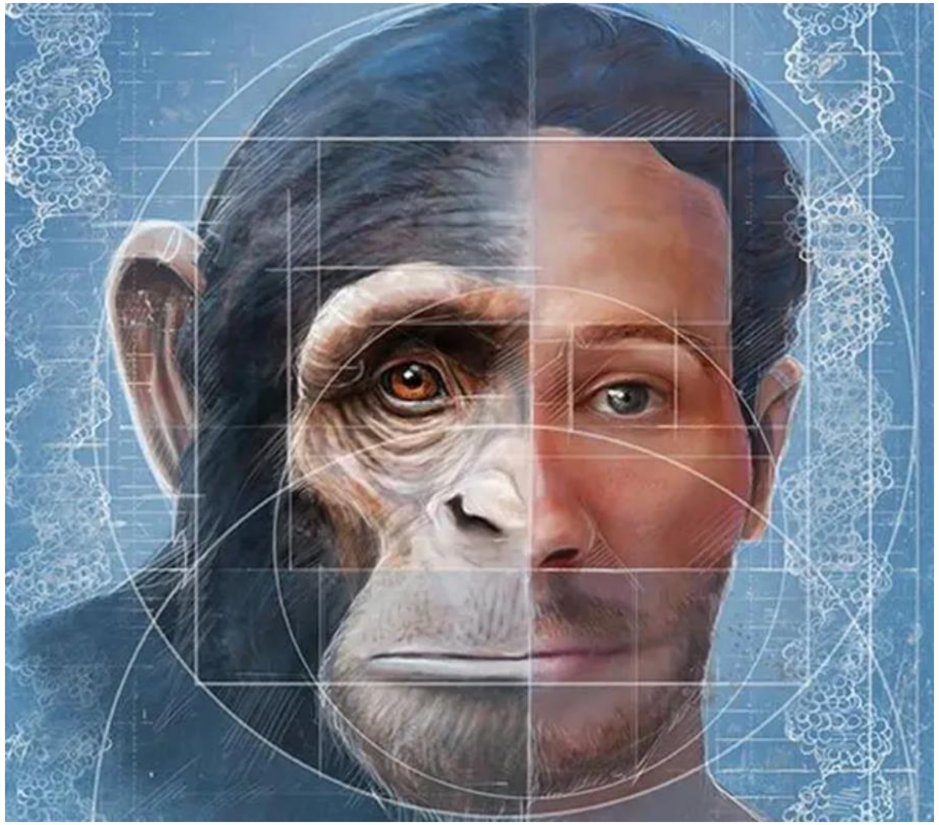
Last updated on 2023.12

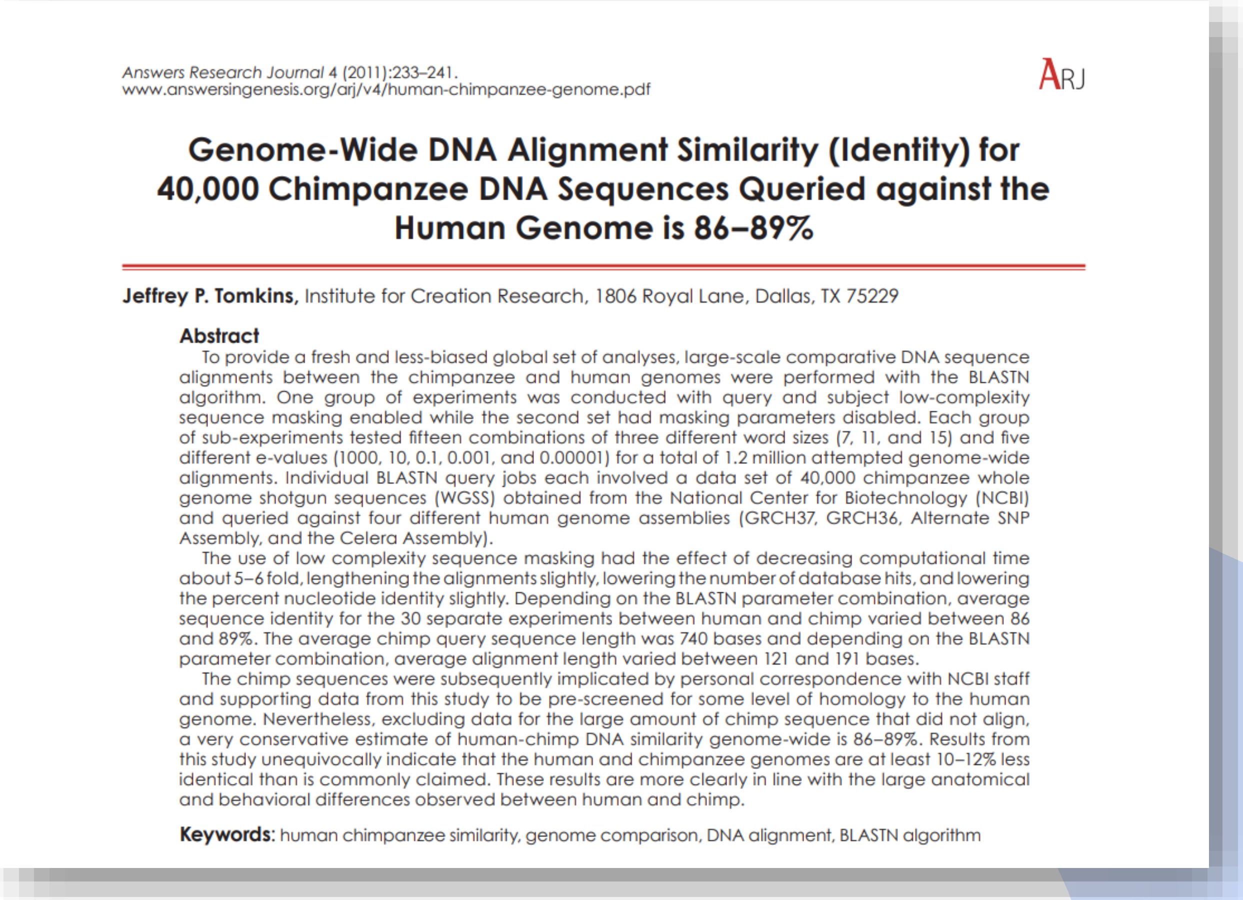
zhuyungang@jlu.edu.cn

字符串的模糊匹配

- 模糊匹配：降低精确匹配的要求：*S*和*P*之间只要有一定程度的相似，就认为匹配成功。
- 用于衡量两个字符串之间的相似程度。

应用举例：基因序列比对





Answers Research Journal 4 (2011):233–241.
www.answersingenesis.org/arj/v4/human-chimpanzee-genome.pdf

ARJ

Genome-Wide DNA Alignment Similarity (Identity) for 40,000 Chimpanzee DNA Sequences Queried against the Human Genome is 86–89%

Jeffrey P. Tomkins, Institute for Creation Research, 1806 Royal Lane, Dallas, TX 75229

Abstract

To provide a fresh and less-biased global set of analyses, large-scale comparative DNA sequence alignments between the chimpanzee and human genomes were performed with the BLASTN algorithm. One group of experiments was conducted with query and subject low-complexity sequence masking enabled while the second set had masking parameters disabled. Each group of sub-experiments tested fifteen combinations of three different word sizes (7, 11, and 15) and five different e-values (1000, 10, 0.1, 0.001, and 0.00001) for a total of 1.2 million attempted genome-wide alignments. Individual BLASTN query jobs each involved a data set of 40,000 chimpanzee whole genome shotgun sequences (WGSS) obtained from the National Center for Biotechnology (NCBI) and queried against four different human genome assemblies (GRCH37, GRCH36, Alternate SNP Assembly, and the Celera Assembly).

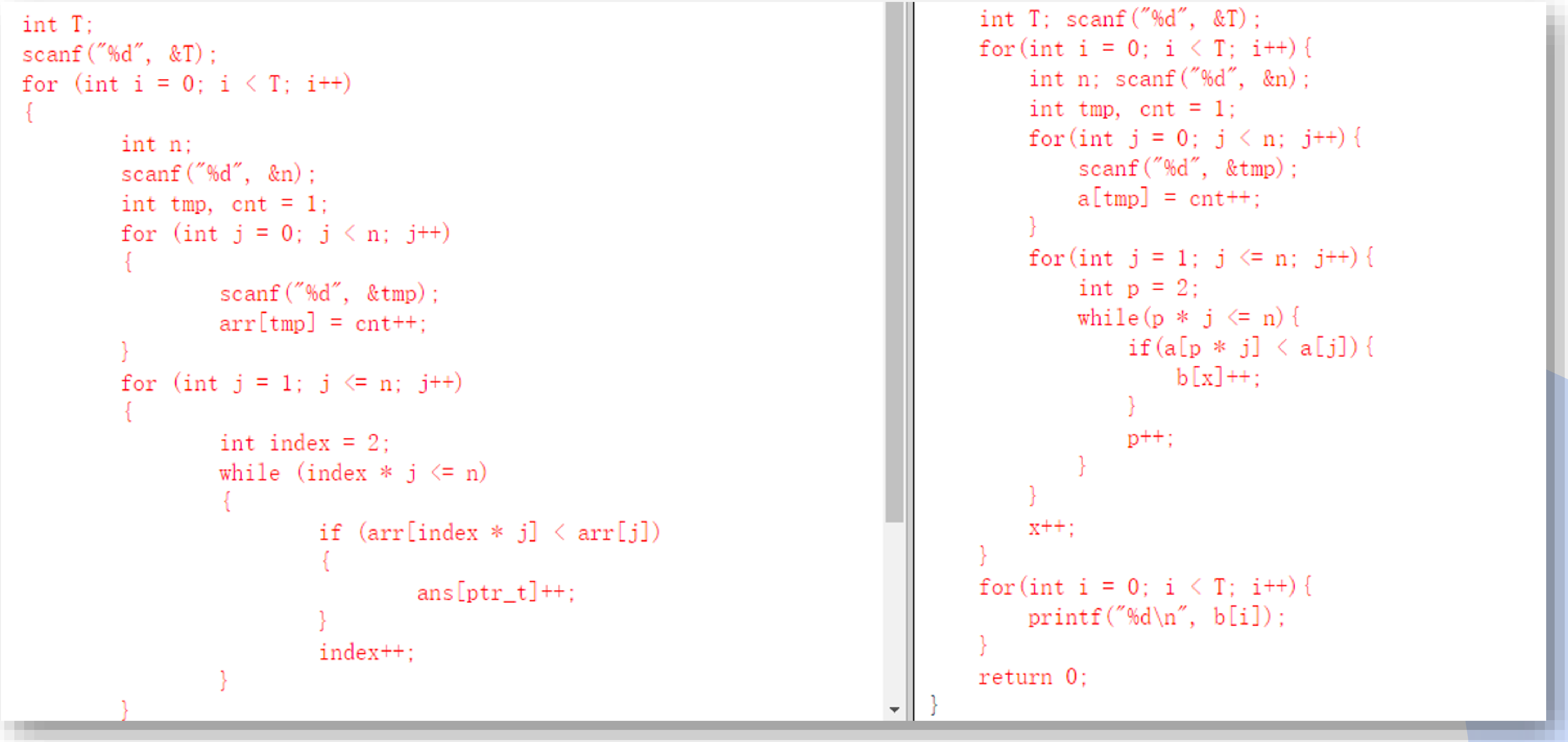
The use of low complexity sequence masking had the effect of decreasing computational time about 5–6 fold, lengthening the alignments slightly, lowering the number of database hits, and lowering the percent nucleotide identity slightly. Depending on the BLASTN parameter combination, average sequence identity for the 30 separate experiments between human and chimp varied between 86 and 89%. The average chimp query sequence length was 740 bases and depending on the BLASTN parameter combination, average alignment length varied between 121 and 191 bases.

The chimp sequences were subsequently implicated by personal correspondence with NCBI staff and supporting data from this study to be pre-screened for some level of homology to the human genome. Nevertheless, excluding data for the large amount of chimp sequence that did not align, a very conservative estimate of human-chimp DNA similarity genome-wide is 86–89%. Results from this study unequivocally indicate that the human and chimpanzee genomes are at least 10–12% less identical than is commonly claimed. These results are more clearly in line with the large anatomical and behavioral differences observed between human and chimp.

Keywords: human chimpanzee similarity, genome comparison, DNA alignment, BLASTN algorithm

Tomkins J. Genome-Wide DNA Alignment Similarity (Identity) for 40000 Chimpanzee DNA Sequences Queried against the Human Genome is 86-89%. Answers Research Journal, 4(2011): 233-241.

应用举例：论文/代码查重



```
int T;
scanf("%d", &T);
for (int i = 0; i < T; i++)
{
        int n;
        scanf("%d", &n);
        int tmp, cnt = 1;
        for (int j = 0; j < n; j++)
        {
                scanf("%d", &tmp);
                arr[tmp] = cnt++;
        }
        for (int j = 1; j <= n; j++)
        {
                int index = 2;
                while (index * j <= n)
                {
                        if (arr[index * j] < arr[j])
                        {
                                ans[ptr_t]++;
                        }
                        index++;
                }
        }
```

```
    int T; scanf("%d", &T);
    for(int i = 0; i < T; i++){
        int n; scanf("%d", &n);
        int tmp, cnt = 1;
        for(int j = 0; j < n; j++){
            scanf("%d", &tmp);
            a[tmp] = cnt++;
        }
        for(int j = 1; j <= n; j++){
            int p = 2;
            while(p * j <= n){
                if(a[p * j] < a[j]){
                    b[x]++;
                }
                p++;
            }
        }
        x++;
    }
    for(int i = 0; i < T; i++){
        printf("%d\n", b[i]);
    }
    return 0;
}
```

字符串的编辑距离

- 如何量化两个字符串之间的相似程度？
- 字符串编辑距离：将一个字符串转化成另一个字符串，需要的最少编辑操作次数。
- 编辑操作：增加一个字符、删除一个字符、替换一个字符。
- 编辑距离越大，说明两个字符串的相似程度越小；编辑距离越小，说明两个字符串的相似程度越大。对于两个完全相同的字符串，编辑距离为0。

字符串的编辑距离

- FOOD转换成MONEY：
- 编辑距离4：第1位F替换为M，第3位O替换为N，第4位插入E，第5位D替换为Y。

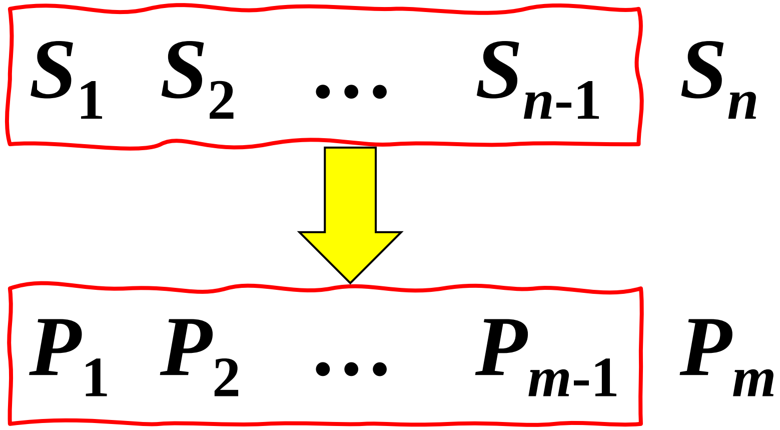
F	O	O		D
M	O	N	E	Y

- 给定两个字符串$S[1..n]$和字符串$P[1..m]$，计算S和P的编辑距离。【华为、字节跳动、腾讯、百度、微软、苹果、谷歌、网易、滴滴、快手面试题LeetCode72】

$$\text{Dist}(n, m)$$

*S*和*P*的编辑距离Dist(*n*, *m*)

- 将*S*变换为*P*：若$S_n = P_m$



$S_1 \quad S_2 \quad \dots \quad S_{n-1} \quad S_n$

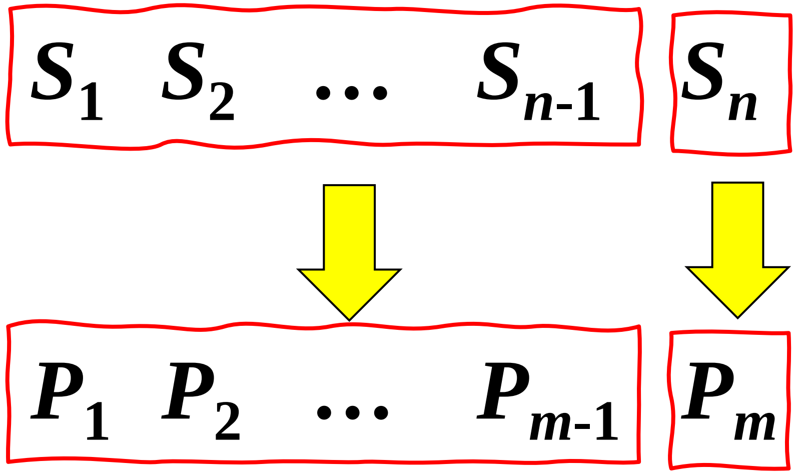
$P_1 \quad P_2 \quad \dots \quad P_{m-1} \quad P_m$

- 将*S*[1…*n*-1]变换为*P*[1…*m*-1]. 即

$$\text{Dist}(n, m) = \text{Dist}(n-1, m-1)$$

S和P的编辑距离Dist(n, m)

- 将S变换为P：若$S_n \neq P_m$



S_1 S_2 … $S_{n\text{-}1}$ S_n

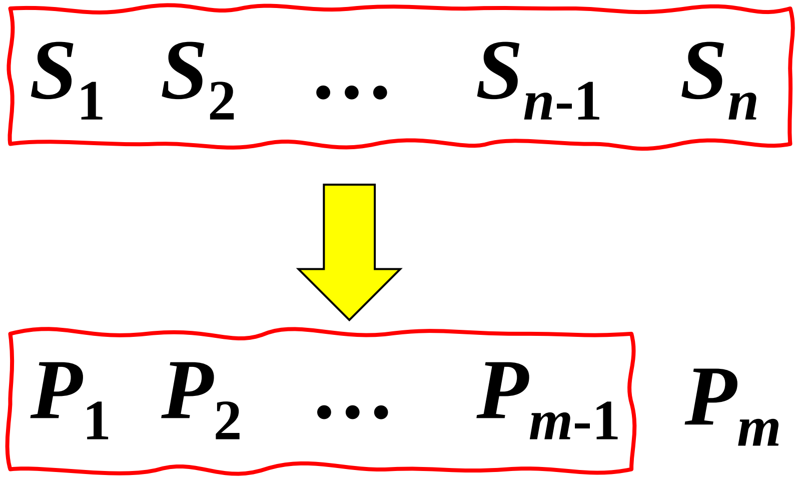
P_1 P_2 … $P_{m\text{-}1}$ P_m

- 方案1：S[1…n-1]变换为P[1...m-1]，将S[n]替换为P[m].

$$\text{Dist}(n, m) = \text{Dist}(n\text{-}1, m\text{-}1)+1$$

*S*和*P*的编辑距离Dist(*n*, *m*)

- 将*S*变换为*P*：若$S_n \neq P_m$



$S_1 \quad S_2 \quad \dots \quad S_{n-1} \quad S_n$

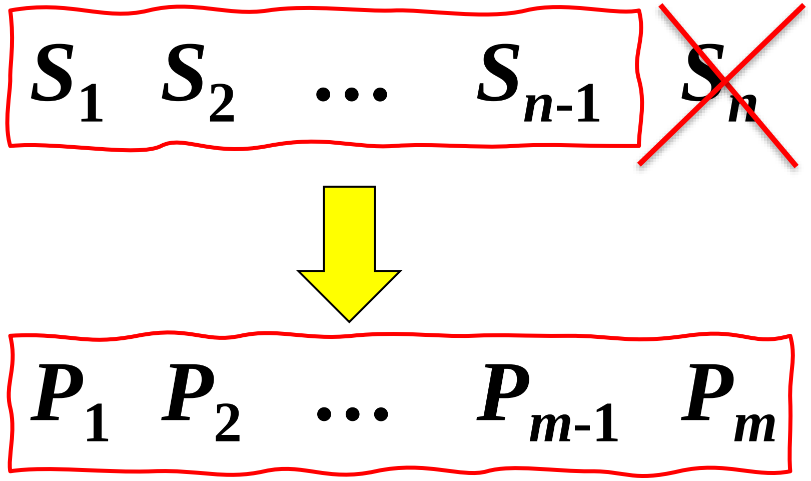
$P_1 \quad P_2 \quad \dots \quad P_{m-1} \quad P_m$

- 方案2：*S*[1…*n*]变换为*P*[1…*m*-1]，并插入*P*[*m*].

$$\text{Dist}(n, m) = \text{Dist}(n, m-1) + 1$$

S和P的编辑距离Dist(n, m)

- 将S变换为P：若$S_n \neq P_m$



$S_1 \quad S_2 \quad \dots \quad S_{n-1} \quad \cancel{S_n}$

$P_1 \quad P_2 \quad \dots \quad P_{m-1} \quad P_m$

- 方案3：S[1...n-1]变换为P[1...m]，并删去S[n].

$$\text{Dist}(n, m) = \text{Dist}(n-1, m)+1$$

*S*和*P*的编辑距离Dist(*n*, *m*)

$$Dist(n,m)=\begin{cases} Dist(n-1,m-1), & S_n = P_m \\ \min\{Dist(n-1,m-1), Dist(n,m-1), Dist(n-1,m)\}+1, & S_n \neq P_m \end{cases}$$

				m
			Dist[*n*-1][*m*-1]	**Dist[*n*-1][*m*]**
n			**Dist[*n*][*m*-1]**	**Dist[*n*][*m*]**

*S*和*P*的编辑距离Dist(*n*, *m*)

$$Dist[i][j]=\begin{cases} Dist[i-1][j-1], & S_i = P_j \\ \min\{Dist[i-1][j-1], Dist[i][j-1], Dist[i-1][j]\}+1, & S_i \neq P_j \end{cases}$$

Dist[i][j]: $S_1 \ldots S_i$和$P_1 \ldots P_j$的编辑距离

			j	
		Dist[*i*-1][*j*-1]	**Dist[*i*-1][*j*]**	
i		**Dist[*i*][*j*-1]**	**Dist[*i*][*j*]**	

字符串S和P的编辑距离

$$Dist[i][j]=\begin{cases} Dist[i-1][j-1], & S_i = P_j \\ \min\{Dist[i-1][j-1], Dist[i][j-1], Dist[i-1][j]\}+1, & S_i \neq P_j \end{cases}$$

	“ ”	B	C	D
“ ”	0	1	2	3
A	1	1	2	3
B	2	1	2	3
C	3	2	1	2

字符串*S*和*P*的编辑距离

$$Dist[i][j]=\begin{cases}Dist[i-1][j-1], & S_i=P_j\\ \min\{Dist[i-1][j-1],Dist[i][j-1],Dist[i-1][j]\}+1, & S_i\neq P_j\end{cases}$$

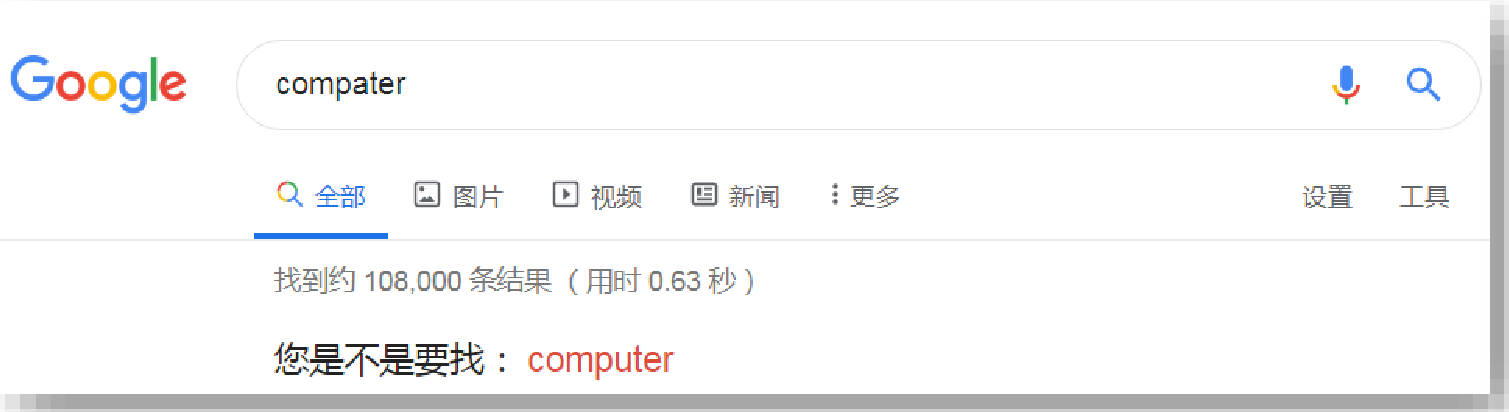
```
const int N=110;
int EditDistance(char *s, char *p, int n, int m) {
   int i, j, D[N][N];
   for(i=0; i<=n; i++) D[i][0]=i; //填第0列
   for(j=0; j<=m; j++) D[0][j]=j; //填第0行
   for(i=1; i<=n; i++)
     for(j=1; j<=m; j++)
       if(s[i-1]==p[j-1]) D[i][j]=D[i-1][j-1];
       else D[i][j]=1+min(D[i-1][j-1],D[i][j-1],D[i-1][j]);
   return D[n][m];
}
```

时间复杂度
O(*nm*)

0	1	2	3
1			
2	D[*i*-1][*j*-1]	D[*i*-1][*j*]	
3	D[*i*][*j*-1]	D[*i*][*j*]	

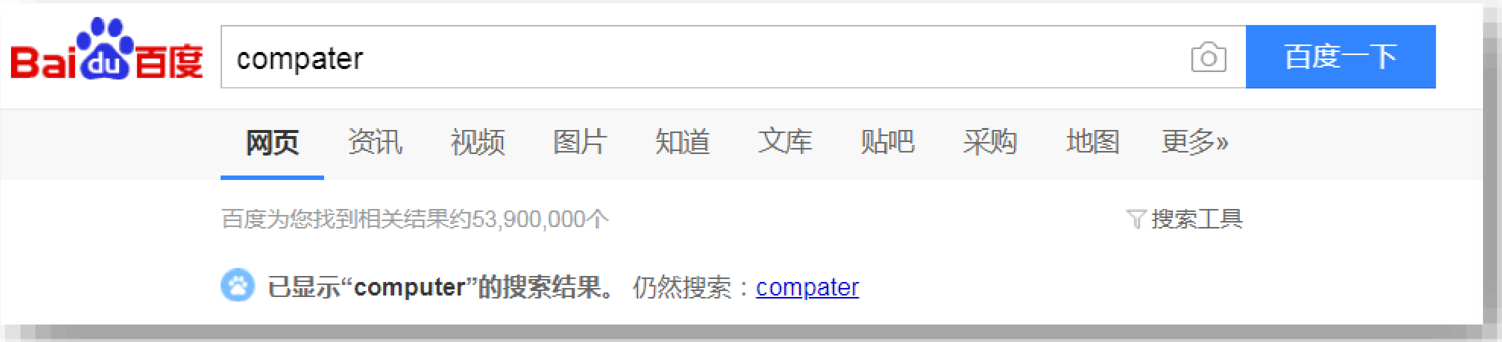
***Dist*[*n*][*m*]**表示$S_1 \ldots S_n$和$P_1 \ldots P_m$的编辑距离

字符串*S*和*P*的编辑距离



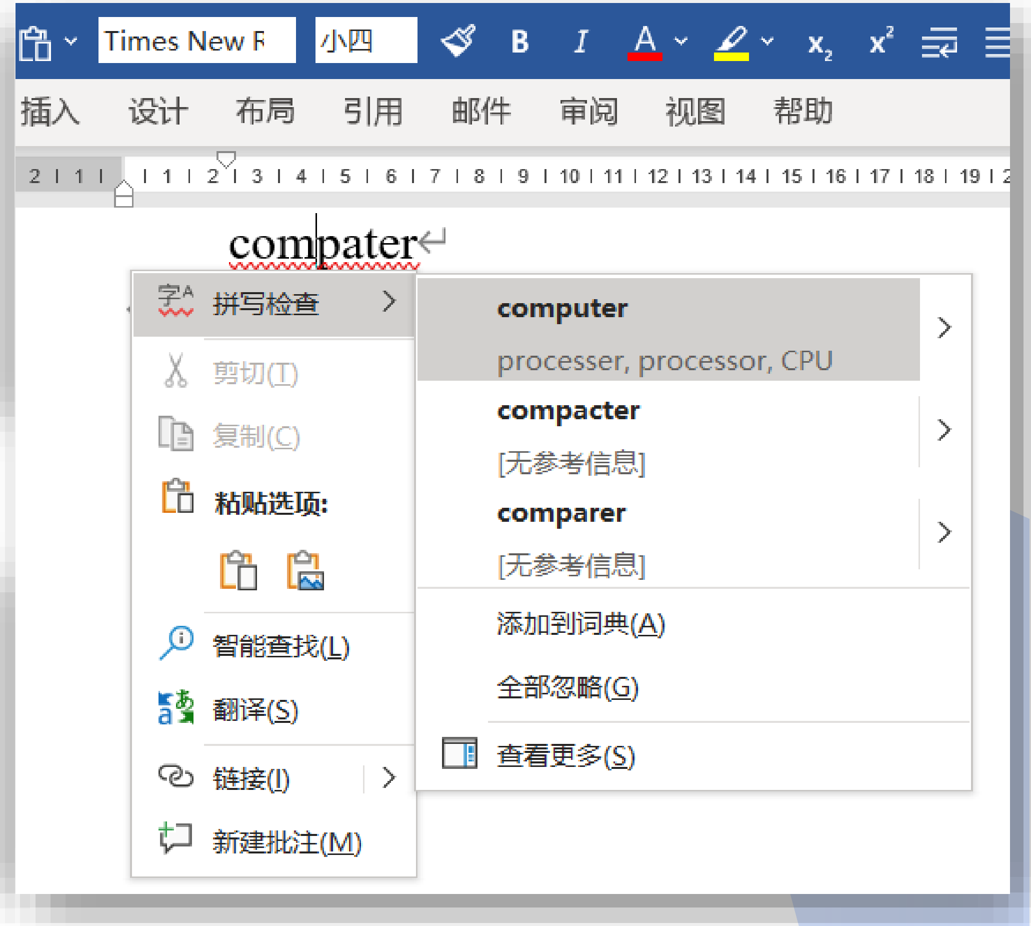
找到约 108,000 条结果（用时 0.63 秒）

您是不是要找： computer



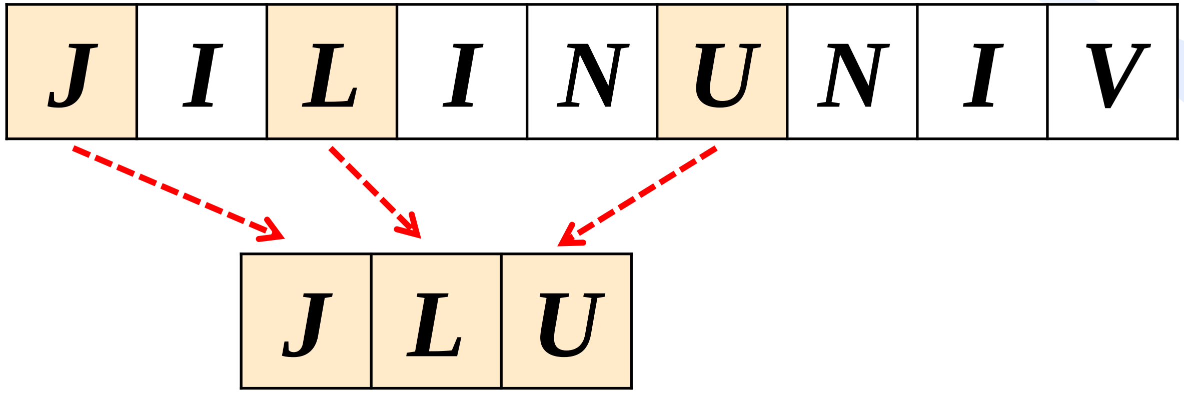
百度为您找到相关结果约53,900,000个

已显示“computer”的搜索结果。 仍然搜索：compater



最长公共子序列（LCS, Longest Common Sequence）

- 可用两个字符串的最长公共子序列长度衡量其相似程度。
- 子序列：由字符串中若干字符按原相对次序构成



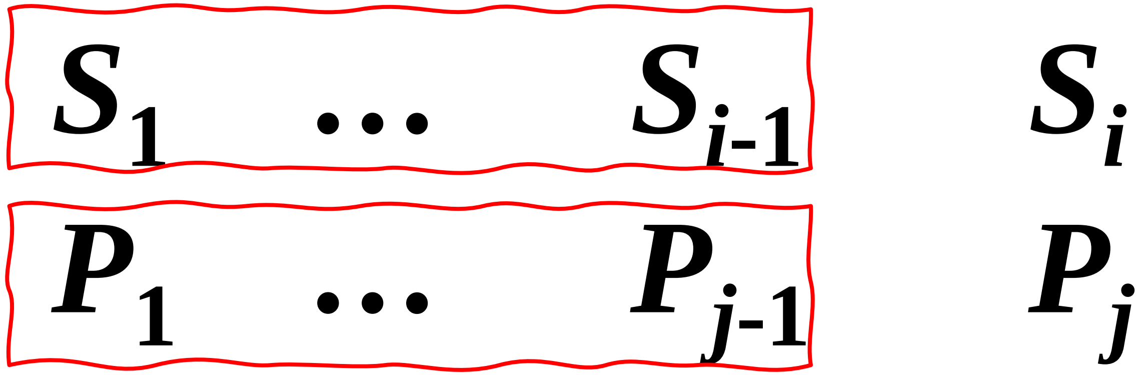
J	*I*	*L*	*I*	*N*	*U*	*N*	*I*	*V*

J	*L*	*U*

- 最长公共子序列：两个字符串的公共子序列中的最长者。
- 例：*S* = "HelloWorld" ， *P* = "loop"
- *S*和*P*的最长公共子序列为loo

最长公共子序列（LCS, Longest Common Sequence）

- 给定两个字符串*S*和*P*，长度分别为*n*和*m*，要求找出它们最长的公共子序列长度。【百度、阿里、华为、字节跳动、腾讯、小米、美团、京东面试题LeetCode1143】
- 考察$S_1 \ldots S_i$和$P_1 \ldots P_j$
- 令LCS(*i*, *j*)表示$S_1 \ldots S_i$和$P_1 \ldots P_j$的最长公共子序列长度。
- 若$S_i = P_j$，则LCS(*i*, *j*)=LCS(*i*-1, *j*-1)+1



$$S_1 \quad \cdots \quad S_{i-1} \qquad S_i$$

$$P_1 \quad \cdots \quad P_{j-1} \qquad P_j$$

最长公共子序列（LCS, Longest Common Sequence）

- 若$S_i \neq P_j$， 则LCS(i, j)=max{$\boxed{\text{LCS}(i\text{-}1, j)}$, LCS($i$, j-1)}

$$\boxed{S_1 \quad \cdots \quad S_{i\text{-}1}} \quad S_i$$

$$\boxed{P_1 \quad \cdots \quad P_{j\text{-}1} \quad P_j}$$

最长公共子序列（LCS, Longest Common Sequence）

- 若$S_i \neq P_j$， 则$\mathrm{LCS}(i, j)=\max\{\mathrm{LCS}(i-1, j), \boxed{\mathrm{LCS}(i, j-1)}\}$

$$\boxed{S_1 \quad \cdots \quad S_{i-1} \quad S_i}$$

$$\boxed{P_1 \quad \cdots \quad P_{j-1}} \quad P_j$$

最长公共子序列（LCS, Longest Common Sequence）

- 若$S_i \neq P_j$， 则LCS(i, j)=**max**{LCS(i-1, j), LCS(i, j-1)}

$$\begin{array}{cccc} S_1 & \cdots & S_{i\text{-}1} & S_i \\ P_1 & \cdots & P_{j\text{-}1} & P_j \end{array}$$

最长公共子序列（LCS, Longest Common Sequence）

$$LCS[i][j]=\begin{cases}LCS[i-1][j-1]+1, & S_i=P_j\\ \max\{LCS[i][j-1],LCS[i-1][j]\}, & S_i\neq P_j\end{cases}$$

***LCS*[*i*][*j*]:S$_1$…S$_i$和P$_1$…P$_j$的最长公共子序列长度**

			j	
		LCS[*i*-1][*j*-1]	LCS[*i*-1][*j*]	
i		LCS[*i*][*j*-1]	LCS[*i*][*j*]	

最长公共子序列（LCS, Longest Common Sequence）

$$LCS[i][j]=\begin{cases} LCS[i-1][j-1]+1, & S_i = P_j \\ \max\{LCS[i][j-1], LCS[i-1][j]\}, & S_i \neq P_j \end{cases}$$

$LCS[i][j]$:$S_1 \ldots S_i$和$P_1 \ldots P_j$的最长公共子序列长度

	" "	B	C	D
" "	0	0	0	0
A	0	0	0	0
B	0	1	1	1
C	0	1	2	2

最长公共子序列（LCS, Longest Common Sequence）

$$LCS[i][j]=\begin{cases}LCS[i-1][j-1]+1, & S_i=P_j\\ \max\{LCS[i][j-1],LCS[i-1][j]\}, & S_i\neq P_j\end{cases}$$

***LCS*[*n*][*m*]表示$S_1\ldots S_n$和$P_1\ldots P_m$的最长公共子序列长度**

```
const int N=1010;
int longestCommonSubsequence(char *s,char *p,int n,int m){
   int i, j, LCS[N][N];
   for(i=0; i<=n; i++) LCS[i][0]=0; //填第0列
   for(j=0; j<=m; j++) LCS[0][j]=0; //填第0行
   for(i=1; i<=n; i++)
     for(j=1; j<=m; j++)
       if(s[i-1]==p[j-1]) LCS[i][j]=LCS[i-1][j-1]+1;
       else LCS[i][j]=max(LCS[i][j-1], LCS[i-1][j]);
   return LCS[n][m];
}
```

时间复杂度
O(*nm*)

0	0	0	0
0			
0			
0			

最长公共子序列——变形问题

给定两个字符串*S*和*P*，长度分别为*n*和*m*，编写程序计算最少去掉几个字符可使二者相等。【字节跳动、华为、美团、微软、谷歌面试题LeetCode583】

例：$S = a\ x\ b\ c\ e$，$P = a\ b\ c\ f\ e$. *S*应去掉1个字符*x*, *P*应去掉1个字符*f*，合计2个字符

S	***a***	***x***	***b***	***c***	***e***
P	***a***	***b***	***c***	***f***	***e***

- ➢ *S*和*P* 去掉最少的字符后，若剩下的字符串相等，则剩下的字符串一定是*S*和*P* 的最长公共子序列；
- ➢ 若最长公共子序列长度为*k*，则*S*应删去$n - k$个字符， *P*应删去$m - k$个字符。

最长公共子序列——变形问题

给定一个字符串*S*，计算其最长回文子序列长度。【百度、字节跳动、微软、谷歌面试题LeetCode516】

- *S*和其逆序S^{-1}的最长公共子序列，即为最长回文子序列

S	*x*	*a*	*b*	*y*	*c*	*k*	*b*	*a*
S^{-1}	*a*	*b*	*k*	*c*	*y*	*b*	*a*	*x*

最长公共子序列——变形问题

给定一个长度为*n*的字符串*S*，计算最少删除几个字符，能使剩下的子序列成为回文。

- *n* - 最长回文子序列长度

S	*x*	*a*	*b*	*y*	*c*	*k*	*b*	*a*