基于链表的择换排序新算法

——“数据结构”与“程序设计”课程的融合创新案例研究

黄 涛

（西南财经大学信息技术应用研究所 成都 610074）（西南财经大学经济信息工程学院 成都 610074）

摘 要 依据同构化基本原理，研究和发现了基于传统内部择换排序算法的同构化特点与本质；进而，利用其同构化特点与本质，提出了几种基于链表的择换排序新算法；从而，进一步深化和推广了择换排序算法的应用方式与实用范围；同时，也为“程序设计”、“数据结构”的课程融合、教学改革、教育创新提供了重要研究案例。

关键词 同构化，链表，择换排序，算法，课程融合

New Sorting Algorithms with Choice and Swap Based on Chain Lists

——Fused and Innovated Case Study of “Data Structure” and “Program Design” Subjects

HUANG Tao

(Research Institute of Information Technology Application, Southwestern University of Finance and Economics, Chengdu 610074, China)

(School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu 610074, China)

Abstract According to isomorphic fundamental principles, the isomorphic characters and natures of choice and swap sorting algorithms based on traditional internal sorting were studied and found. Further, several new sorting algorithms with choice and swap based on chain lists were advanced by using their isomorphic characters and natures. Therefore, the application forms and applied ranges of choice and swap sorting algorithms were deepened and expended. Provided an important studying case for fusing subjects, teaching reform and innovating education of “Data Structure” and “Program Design”.

Keywords Isomorphic, Chain list, Sorting with choice and swap, Algorithm, Fusing subjects

1 引言

众所周知，排序问题是计算机科学中基本而重要的基础问题，是教学研究（例如：它可作“程序设计”、“数据结构”的课程融合、教学改革、教育创新的重要研究内容之一）、算法设计、系统开发与应用实践既密切相关、又屡见不鲜的重要基本内容[1-4]。排序，分为基于内存的内（部）排序与面向外存的外（部）排序。一般说来，内排序是外排序的基础（因外排序往往离不开内排序）。长期以来，数组作为人们最常见、最常用、最习惯的内排序工具，使许多人习以为常地误认为它似乎就是唯一好用的内排序工具。但事实并非如此，因为对那些采用链表作为主体数据结构的理论研究课题与实际应用问题，所论课题性质与问题场景涉及到其所论数据集的频繁增删、动态排序、顺序查找与随机检索等基本运算时，如果再机械地沿袭通常基于数组的内排序方法，势必会因其“作为中间数据处理场的数组与作为目标数据处理场的链表之间，必定存在不可避免的数据对倒”而付出大量的时间开销，严重降低解决所论课题与问题相关处理的算法效率。事实上，此时别开生面、新颖明智地采用基于链表的排序算法，是有效提高解决所论课题与问题相关处理算法效率的可行途径之一。限于篇幅，本文在此仅以按递增为序输出任意 n（<100）个自然数为例，给出几个基于链表的择换排序新算法，可深化和推广择换排序算法的应用方式与实用范围，还可作“数据结构”与“程序设计”课程的课程融合、教学改革、教育创新案例。

2 基于数组的择换排序算法

把通常找最大（或最小）数的算法思想应用于排序，就产生了一种思想简明、效率较高的择换排序算法。其算法描述，可同构化表征如下：

算法 Eg01 {基于数组的择换法排序算法}

```
{{i,j,n,p,x:整型; a:整型 数组 [1..100]}}; {全局变量定义}
>>> {算法开始}
@:\\输出 “输入数据个数:”; 输入 n // {提示输入数据个数}
  直到 0<n 且 n<100; {容错输入}
对 i←1,n @:{数据个数控制}
  \\输出 “a[“,i,”]=”; 输入 a[i] // {提示输入第 i 个数据}
对 i←1,n-1 @:\\ {排序遍数控制}
  p←i; {当前第 i 小者初始处}
  对 j←i+1,n @:{待择选数据个数控制}
    如果 a[p]>a[j] {当前数据较小吗?}
    T:p←j; {标记当前最小者位置}
  如果 p<>i {第 i 小者不在其位吗?}
  T:\\x←a[p]; a[p]←a[i]; a[i]←x // {第 i 小者交换到位}
//;
对 i←1,n-1 @:{待输出数据个数控制}
  输出 a[i]; {递增输出各有序化数据}
行输出 a[n];
!!! {算法结束}
```

黄 涛 讲师，主要研究方向为计算机应用。

3 择换排序算法的根本特点与同构本质

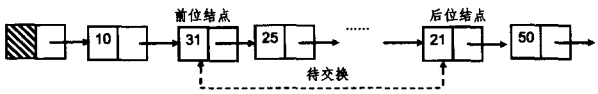
依据我国西南财经大学周启海教授提出的同构化基本原理[3]，我们研究并发现了择换排序算法的根本特点与同构本质如下：

①所论数据最小(或最大)者的比较择选，只利用了存储这些数据的变量本身(此处为下标变量)，而与其所在位置无关(即下标)；

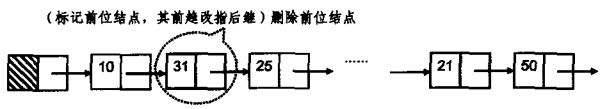
②当前最小者应在位置，则只利用了标记这些最小者原位置的变量本身(此处为标记其下标位置的标记变量)，而与其所具数值无关；

③把当前最小者交换到其应在位置，可不必打乱原变量序列的顺序。

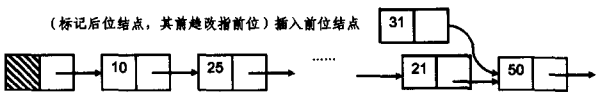
据此，可得出结论：只要能分别同构化地正确处理上述3个基本操作(即：当前最小者挑选，当前最小者位置标记，当前最小者交换到位)，就未必一定要数组来实现择换排序算法。事实上，与数组一样同为线性表的链表，不仅应该而且可以用来同样解决排序问题。



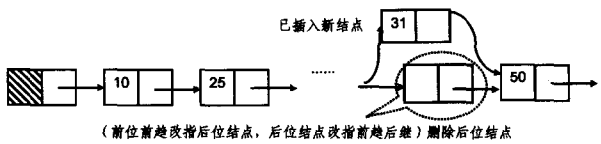
(a) 单向链表的两结点交换前状况



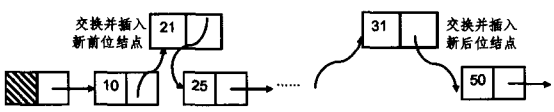
(b) 欲删除单向链表的前位结点



(c) 应先把单向链表的前位结点删除并插入到其后位结点后



(d) 欲删除单向链表的后位结点，应先把它的后位结点删除并插入到其前位结点前



(e) 经结点插删的单向链表两结点交换后状况

图1 基于单向链表结点插删的择换排序处理示意图

4 基于结点插删的择换法单向链表排序处理

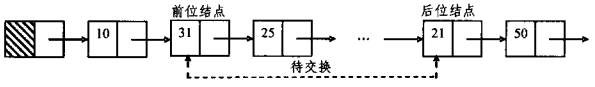
基于单向链表结点插删基本操作的择换法处理过程，如图1所示。据此，可得基于结点插删的择换法单向链表排序算法 Eg02 如下。

算法 Eg02 {基于单向链表结点插删的择换排序算法}

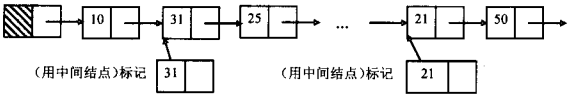
```
[[Link(==)指针 ^Nodes]]; {单向链表的指针类型定义}
[[Nodes(==)记录 [ Data:整型; {元素体定义:自身数据字段}
                  Next:Link ] {邻元体定义:直接后继指针字段}
                ]]; {单向链表的指针元素类型定义}
{{ i,j,n:整型; }}; {整形变量定义}
{{ Head, Rear, Visit1, Visit2, Pre1, Pre2, Place, PlacePre: Link; }}
{定义链表头,尾,访问,位置等}
))) {算法开始}
@:\\
  输出 "请输入数据个数:n=?"; 输入 n {提示下输入数据个数}
// 直到 n<=2; {直到个数 n 已合理且中意为止}
{"用单向链表存放初始数据序列"处理}
生成结点 Head; {生成表头结点,以存放第1个数据}
输入 Head. Data {存放头结点数据}
Head. Next←Null {使头结点向后指针为空指针}
Rear←Head; {头结点同时也是尾指针}
Visit1←Head; {标记初始链表头}
i←2; {所选数据个数计数器初值}
当 i<=n @:\\ {当尚有未生成结点时(生成链表其它结点,并存放各自数据)}
  生成结点 Visit1; {生成当前新尾结点}
  输出"请输入第 i 个数据"; 输入 Visit1. Data; {新尾结点加载数据}
  Visit1. Next←Null {使新尾结点向后指针为空指针}
  Rear. Next←Visit1; {单向连通原尾结点与新尾结点}
  Rear←Visit1; {标记当前新尾结点}
  i←i+1 {生成下一结点序号}
//;
{"基于单向链表结点插删对初始数据序列进行排序"处理}
Visit1←Head; Pre1←Visit1; {外层指针控制应插入处:从头结点起}
当 Visit1. Next<>Null @:\\ {当应插入处非空指针时}
  Place←Visit1; {标记当前范围最小者初始应插入处}
  PlacePre←Pre2; {标记应插入处直接前趋}
  Pre2←Visit1; {标记待插入者直接前趋}
  Visit2←Visit1. Next; {内层指针控制待插入者:从下一结点起}
  当 Visit2<>Null @:\\ {当待插入者非空指针时}
    如果 Place. Data>Visit2. Data {当待插入者非最小者时}
    T:\\ Place←Visit2; PlacePre←Pre2 // {标记当前最小者位置及其直接前趋}
  Pre2←Visit2; {标记下一待访结点直接前趋}
  Visit2←Visit2. Next; {指向下一待访结点}
//;
如果 Place<>Visit1 {当前待插入者不在其位(应插入处)吗?}
  T:\\
    如果 Visit1=Head {当前应插入处 Visit1 为头结点吗?}
    T:\\
        如果 Place. Next<>NULL {当前待插入者 Place 为非尾结点吗?}
        T: {直接后继跨越它而指向直接前趋处理}
          PlacePre. Next←Place. Next {直接前趋跨越它而指向直接后继}
        F:\\ {当前应插入者为非尾结点处}
            Rear←PlacePre; {当前待插入者直接前趋作尾结点}
            PlacePre. Next←NULL {直接前趋向后指针为空指针}
            //
          Place. Next←Visit1; {待插入者直接后继指向应插入处}
          Head←Place {标记新链头}
          //
        F: {当前应插入者为非头结点处}
        如果 Place. Next<>NULL {当前待插入者为非尾结点吗?}
```

T:PlacePre. Next←Place. Next {直接后继跨越它而指向直接前趋处理}
F:\\ {当前应插入处 Visit1 为非头结点处理}
Rear←PlacePre; {当前待插入者直接前趋作尾结点}
PlacePre. Next←NULL {直接前趋向后指针为空指针}
//
Pre1. Next←Place; {应插入处直接前趋向后指针指向应插入者}
Place. Next←Visit1 {应插入者直接后继指向当前应插入处}
//
Visit1←Place; {标记当前最小者已插入处}
//
Pre1←Visit1; {标记应插入处直接前趋}
Visit1←Visit1. Next {指向下一最小者应插入处}
//
{"输出单向链表中递增有序化各数据"处理}
Visit1←Head; {指针 Visit1 标记双向链表头结点}
当 Visit1<>Null @:\\ {当指针 Visit1 非空指针时}
输出 Visit1. Data; {输出当前结点数据字段}
Visit1←Visit1. Next {指向下一结点位置}
//
行输出;
!!! {算法结束}

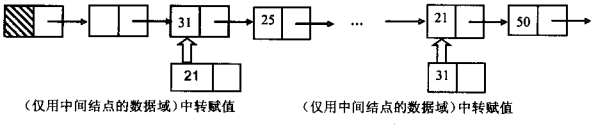
5 基于结点标记的择换法单向链表排序处理



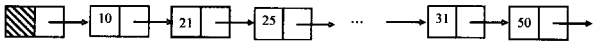
(a) 单向链表的两结点交换前状况



(b) 用中间结点分别标记单向链表的前位、后位结点



(c) 仅用中间结点数据域分别对单向链表的前位、后位结点数据域中转赋值



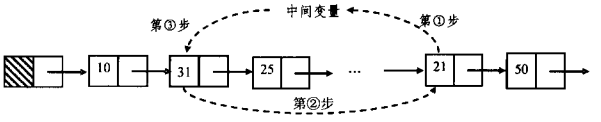
(d) 经结点标记与数据域赋值的单向链表两结点交换后状况

图 2 基于单向链表结点标记的择换排序处理示意图

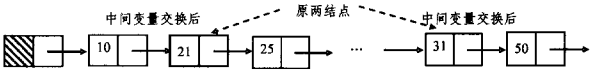
显然,基于单向链表结点插删的择换排序算法 Eg02,存在处理复杂、效率较低的弱点,但它可用如图 2 所示的基于单向链表结点标记的择换法单向链表排序处理来改进。据此,已不难得其基于单向链表结点标记的择换排序算法 Eg02(限于篇幅,本文从略)。

6 基于字段变量标记的择换法单向链表排序处理

如图 2 所示,基于单向链表结点标记的择换排序算法处理中的两结点的标记处理,其目的并不是非得要标记这两个结点本身,而是着眼于一定要标记这两个结点中所存放的数据。既然结点所存放的数据实际上只与该结点的数据字段(或称数据域)有关,故本质上只需用与该数据字段同型的中间变量来存放所需数据即可,而完全无需用与存放所论数据结点同型的中间结点来存放所需数据。据此,已可得基于字段(而非结点)变量标记的单向链表择换排序算法 Eg03,而其基于变量标记的择换法单向链表排序处理如图 3 所示。



(a) 只用中间变量交换数据



(b) 用中间变量的链表两结点交换后状况

图 3 基于变量标记的择换法单向链表排序处理示意图

算法 Eg03 {基于变量标记的单向链表择换排序算法}
[[Link(==)指针 ^Nodes]]; {单项链表的指针类型定义}
[[Nodes(==)记录 [Data:整型; {元素体定义:自身数据字段}
Next:Link] {邻元体定义:直接后继指针字段}
]]:
{{i,j,n:整型; }}; {整形变量定义}
{{ Head,Rear,Visit1,Visit2,Place:Link }}; {定义链表头、尾、访问、位置等}
))) {算法开始}
@:\\ 输出 "请输入数据个数:n=?"; 输入 n // {提示下输入数据个数}
直到 n>=2; {直到个数 n 已合理且中意为止}
{"用单向链表存放初始数据序列"处理}
生成结点 Head; {生成表头结点,以存放第 1 个数据}
输入 Head. Data {存放头结点数据}
Head. Next←Null {使头结点向后指针为空指针}
Rear←Head; {头结点同时也是尾指针}
Visit1←Head; {标记初始链表头}
i←2; {所选数据个数计数器初值}
当 i<=n @:\\ {当尚有未生成结点时(生成链表其它结点,并存放各自数据)}
生成结点 Visit1; {生成当前新尾结点}
输出 "请输入第 i 个数据"; 输入 Visit1. Data; {新尾结点加载数据}
Visit1. Next←Null {使新尾结点向后指针为空指针}
Rear. Next←Visit1; {单向连通原尾结点与新尾结点}
Rear←Visit1; {标记当前新尾结点}
i←i+1 {生成下一结点序号}
//;
{"基于(字段)变量标记的单向链表排序"处理}
Visit1←Head; {外层指针控制:从头结点起排序}
当 Visit1. Next<>Null @:\\ {当还有应排序处理时}
Place←Visit1; {标记当前范围最小者初始处}
Visit2←Visit1. Next; {内层指针控制:从下一结点起选择当前最小者}
当 Visit2<>Null @:\\ {当还有可选结点时}
如果 Place. Data>Visit2. Data {当前结点是最小者吗?}
T:Place←Visit2; {标记当前最小者处}
Visit2←Visit2. Next {指向下一待选结点}
//
如果 Place<>Visit1 {当前最小者不在其位吗?}

```
    T:\\ i←Place. Data; Place. Data←Visit1. Data; Visit1. Data←i
//{其数据字段直接交换到位}
    Visit1←Visit1. Succeed {指向下一遍排序范围始端结点}
  //
  {"输出单向链表中递增有序化各数据"处理}
  Visit1←Head; {指针 Visit1 标记双向链表头结点}
  当 Visit1<>Null @:\\ {当指针 Visit1 非空指针时}
    输出 Visit1. Data; {输出当前结点数据字段}
    Visit1←Visit1. Next {指向下一结点位置}
  //;
  行输出;
  !!!
```

结束语 依据同构化基本定理，本文研究和发现了基于传统内部择换排序算法的同构化特点与本质；利用其同构化特点与本质，逐个更优地构造了基于链表的结点插删、结点标记、(结点的数据字段)变量标记的3种择换排序新算法，有利于进一步深化和推广择换排序算法的应用方式与实用范围。自然，只需另用双向链表的向前指针对直接前趋、向后指针对直接后继的位置标记能力，即可把基于单向链表结点插删的择换法排序算法改造成更为简明、更加高效的基于双向链表结点插删的择换法排序算法。并且，这些基于链表的择换排序算法及其编程实现，完全可作"程序设计"与"数据结构"的课程融合、教学改革、教育创新的一个有效案例，而这已被由作者成功主研2006年四川省级精品课程"数据结构"的教学研究与教改实践所证实。

最后，还应当指出：基于链表的同构化择换排序算法，还可进一步推广到基于链表的同构化首尾排序新算法[5]。

参 考 文 献

[1] Knuth D E. The Art of Computer Programming, Volume 1, Fundamental Algorithms. Addison-Wesley Publishing Company, Inc., 1973

[2] Knuth D E. The Art of Computer Programming, Volume 3, Sorting and Searching. Addison-Wesley Publishing Company, Inc., 1973

[3] 周启海. C++同构化对象程序设计原理. 北京：清华大学出版社，北方交通大学出版社，2004

[4] 严蔚敏，等. 数据结构(C语言版). 北京：清华大学出版社，2002

[5] 周启海. 基于链表的首尾排序新算法——"程序设计"与"数据结构"课程的融合创新案例研究[J]. 计算机科学，2008，35(11)

[6] 周启海. C语言程序设计教程. 北京：机械工业出版社，2004

[7] 周启海，李朔枫，杨祥茂，等. 论程序设计课程教学中的同构化创新思想教育——"对→好→巧→妙→绝"的算法案例[J]. 计算机科学，2007，34(5)

(上接第294页)

(1)ID域包含一个IPv4或IPv6地址，用ID类型和长度域标识，其值与在ANEP源地址可选项的源地址域中的值相同；

(2)签名域是一个数字签名，用数字签名类型和长度域标识，这个智能包的数字签名算法的有效类型可以是哈希算法或是MD5(第5类报文摘要算法)；

(3)认证域遵循X.509公开密钥认证，用标识类型和长度域标识，该认证域包含有IPv4或IPv6地址的值。

封装体是网络中实现分布式管理的主要元素，它的报文是相当灵活的，可以在传输过程中根据某个节点的状态信息进行计算，决定报文的转发方向、向管理节点返回封装体报文的类型及其携带的数据信息。这种灵活性是通过转发例程来实现的，例程存在于各个节点中，目前将转发例程分为4类。

(1)One-One模式：One-One的转发模式是一种最简单的转发模式，它又根据包是否在所经过节点分成两种结构。一是所经中间节点不执行，其方式与目前的端到端通信方式类似；二是沿传输路径计算的转发模式，在这种转发模式中，封装体报文按照指定的每到一个中间节点时，就要在该节点执行其携带的程序。通过这种执行方式，管理节点可以把集中的任务分发到沿整个传输路径上去执行。

(2)One-Multi模式：在这种转发模式中，多个同样的封装体报文同时从一个节点发出，这些封装体报文发向不同的目的节点并且在该节点执行。这种转发模式可以用于信息的广播(如拥塞位置的检测)、子网的控制。

(3)BFST(Breadth First Search Traversing)转发模式：这是一种并行控制模式。当封装体报文到达一个主动节点后，它被转发到与当前节点直接相连的邻居节点。到达下一个节点时，同样按照将该报文直接转发给相邻的节点。显然，经过一次转发后，网络中将出现很多该封装体报文的副本。当这些副本到达下一个节点后，它们又被复制，并转发到它们的邻居节点。报文副本依次转发下去，直到遍历完整个网络。

(4)DFST(Depth First Search Traversing)转发模式：这是一种串行控制模式。在该模式中，封装体报文到达一个主动节点后，被转发到与当前节点直接相连的一个邻居节点。当它到达这个邻居节点后，又被转发到该邻居节点的一个邻居节点。依次转发下去，直至遍历完整个网络。

结束语 主动网络管理体现了主动网络的思想，将一部分网络管理功能动态地分布在主动节点上，充分利用了主动节点的计算能力，使节点能够自动发现、解决问题，从而极大地优化了网络管理。本文讨论了基于主动节点的层次管理模式，该模式中各个模块相互独立、任务明确，而且在每个层都可以动态更新以适应主动网络中主动节点的易变性和主动应用的扩展性，因此网络管理的稳定性和扩展性都大为提高，适应了现代网络管理的需要。

参 考 文 献

[1] Kawamura R, Stadler R. Active Distributed Management for IP Networks [J]. IEEE Communications Magazine, 2000, 38(4): 114-121

[2] Al Shaer E. Active Management Framework for Distributed Multimedia Systems [J]. Journal of Networks and Systems Management, 2000, 8: 49-72

[3] Brunner M, Stadler R. Service Management in Multi-party Active Networks [J]. IEEE Communications Magazine, 2000, 38(3): 281-286

[4] Kiwior D, Zabele S. Active Resource Allocation in Active Networks [J]. IEEE JSAC, 2000, 19(3): 452-459

[5] Hicks M, Kakkar P, Moore J T, et al. PLAN: A programming language for active networks [C]. ACM SIGPLAN Notices, 1999, 34 (1): 86-93

[6] Fatta G D, Re G L. Active Networks: An Evolution of the Internet [C]// Proc. of AICA2001-39th Annual Conference. Cernobbio, Italy, Sept. 2001: 19-22