# 最长d维箱嵌套问题的贪心算法*)

宋传鸣　王相海
(辽宁师范大学计算机与信息技术学院　大连116029)
(南京大学计算机软件新技术国家重点实验室　南京210093)

## The Greedy Algorithm of the Longest Nesting of d-Dimension Boxes

SONG Chuan-Ming　WANG Xiang-Hai
(College of Computer and Information Technology, Liaoning Normal University, Dalian 116029)
(National Laboratory for Novel Software, Nanjing University, Nanjing 210093)

**Abstract** The Greedy algorithm is a simple, direct and efficient method to many problems. In this paper, the longest nesting problem of d-dimension boxes is brought up firstly. And then a novel algorithm for this problem based on greedy strategy is proposed. Finally, the time complexity of the proposed algorithm is analyzed. Simulation results show it is effective.

**Keywords** Greedy algorithm, Llongest nesting of d-dimension boxes, Depth degree, Time complexity

## 1. 引言

在众多的算法设计策略中，贪心算法以其简单、直接和高效而受到重视[1]。尽管贪心算法并不从整体最优方面考虑问题，而是从某种意义上的局部最优的角度作出选择，但对范围相当广泛的许多实际问题它通常能产生整体最优解[2,3]。对一些问题，即使采用贪心算法不能得到整体最优解，但其最终结果也可以是最优解的很好的近似[4]。

本文首先对d维箱嵌套问题[5]进行了分析和讨论，证明了该问题的嵌套关系所具有的可传递性质，同时给出了两个d维箱嵌套关系的判定策略，在此基础上对最长d维箱嵌套序列问题提出了一种基于贪心策略的解决算法，最后对所提出算法的时间复杂度进行了分析和讨论。

## 2. 问题的提出

一个d维箱$(x_1,x_2,\cdots,x_d)$嵌入另一个$d$维箱$(y_1,y_2,\cdots,y_d)$是指存在$1,2,\cdots,d$的一个排列$\pi$，使得$x_{\pi(1)}<y_1, x_{\pi(2)}<y_2,\cdots,x_{\pi(d)}<y_d$。给定由$n$个$d$维箱组成的集合$\{B_1,B_2,\cdots,B_n\}$，设计一个有效算法，找出这n个d维箱中的一个最长嵌套序列。我们将此问题称为最长d维箱嵌套问题。

## 3. d维箱嵌套关系的可传递性

**命题**：设有三个任意d维箱$B_1$、$B_2$和$B_3$，并且$B_1(x_1,x_2,\cdots,x_d)$能嵌入$B_2(y_1,y_2,\cdots,y_d)$，$B_2$能嵌入$B_3(z_1,z_2,\cdots,z_d)$，则$B_1$一定能嵌入$B_3$。

证明：已知$B_1(x_1,x_2,\cdots,x_d)$能嵌入$B_2(y_1,y_2,\cdots,y_d)$，则存在排列$\pi'$，使得：

$$x_{\pi'(1)}<y_1, x_{\pi'(2)}<y_2,\cdots,x_{\pi'(d)}<y_d \quad (1)$$

又$B_2$能嵌入$B_3(z_1,z_2,\cdots,z_d)$，则存在排列$\pi''$，使得：

$$y_{\pi''(1)}<z_1, y_{\pi''(2)}<z_2,\cdots,y_{\pi''(d)}<z_d \quad (2)$$

由(2)可知，对$[1,d]$区间中的任意整数$i$，均有$y_{\pi''(i)}<z_i$，又由(1)可知，对于$\pi''(i)$存在排列$\delta$，使得$x_{\delta(\pi''(i))}<y_{\pi''(i)}<z_i$，即存在一排列$\pi'''=\delta\pi''$，使得$x_{\pi'''(1)}<z_1, x_{\pi'''(2)}<y_2,\cdots,x_{\pi'''(d)}<y_d$。故由$d$维箱嵌套的定义可知，$B_1$亦能嵌入$B_3$中。证毕。

## 4. 两个d维箱嵌套关系的判定

设有两个d维箱$B_1(x_1,x_2,\cdots,x_d)$和$B_2(y_1,y_2,\cdots,y_d)$，分别将它们各分量$x_i$和$y_i(1\leqslant i\leqslant d)$按从大到小的顺序进行排列，即存在$1,2,\cdots,d$的两个排列$\rho$和$\tau$，使得$x_{\rho(1)}>x_{\rho(2)}>\cdots>x_{\rho(d)}$和$y_{\tau(1)}>y_{\tau(2)}>\cdots>y_{\tau(d)}$。由两个d维箱的嵌套定义及排列变换的特性，容易证明下列命题成立：

**命题**：d维箱$B_1$嵌入$B_2\Leftrightarrow$对区间$[1,d]$的任意整数$i$均有$x_{\rho(i)}<y_{\tau(i)}$。

## 5. 最长d维箱嵌套问题的贪心解法

### 5.1 d维箱的深度概念

d维箱之间的关系通常有三种，即嵌套、被嵌套和无嵌套关系。所以，我们不能利用简单的排序方法查找d维箱嵌套序列，也不能每次查找最大的d维箱作为最长嵌套序列的首箱，因为并不是每种情况下都存在能嵌套其它所有箱子的箱子。为此，我们引进d维箱的"深度"概念来定量地表示一个箱子在嵌套序列中的位置。

**定义**　将n个d维箱之间的关系用一棵树来表示，其中具有嵌套关系的箱子为父亲节点，具有被嵌套关系的箱子为孩子节点，不具有嵌套关系的两个箱子为兄弟节点，所谓一个d维箱的"深度"是指该箱子在这棵树中的深度。

规定，不嵌套其它任何箱子的箱子深度为0，一个箱子的深度值等于其嵌套箱子中的最大深度值加1。这样，d维箱$i$的深度$Depth(i)$的计算方法为：

$$Depth(i)=\begin{cases}0, \text{当箱子 } i \text{ 不嵌套其它任何箱子时}\\ Max\{Depth(j)\mid \text{箱子 } j \text{ 嵌套在 } i \text{ 中}\}+1,\\ \quad \text{当箱子 } i \text{ 嵌套其它箱子时}\end{cases}$$

### 5.2 算法的基本思想

*)南京大学计算机软件新技术国家重点实验室开放课题基金项目、大连市科技基金计划项目资助。宋传鸣　硕士生，研究方向为算法分析与设计、图像处理。王相海　博士，教授，主要研究领域为计算机图形学、图像及多媒体信息处理，算法分析与设计。

由d为箱嵌套关系所具有的传递性以及d维箱的深度定义可以得出如下结论:对于同一个嵌套序列,深度值大的箱子必定嵌套深度值小的箱子,一个箱子的深度值由它在序列中所嵌套的箱子的深度值决定。

约定,规模为n×(n+1)的二维数组A[i][j]用来记录箱子$i$与箱子$j$的关系,其分量A[i][n]中记载着第$i$个箱子的深度值。查找的开始,首先找出深度值最大的箱子,从而确定嵌套序列的首箱,然后,从首箱$i$所对应的一维数组A[i]中可以找出被$i$嵌套的深度值A[k][n]最大的箱子$k$,再在箱子$k$所对应的一维数组A[k]中找出深度值最大的箱子,按此贪心选择策略如此继续下去,直到找到深度值为0的箱子为止。下面说明上述思想的有效性。

(1)*贪心选择性质* 假定最长d维箱嵌套问题的一个最优解为$B_1,B_2,\cdots,B_k(k>1)$,其对应的深度值分别为$H_1,H_2,\cdots,H_k$。

a. 若$H_1$为最大的深度值,则说明问题的最优解以一个贪心选择开始;

b. 若$H_1$不是最大的深度值,不妨假设有$H_1<H_j(1<j\leqslant K)$,即有$B_1$嵌套$B_j$,同时有$H_1<H_j$,这与深度值的定义发生矛盾。因此,假设不成立,即$H_1$为最大的深度值。这说明问题的最优解以一个贪心选择开始。

(2)*最优子结构性质* 设嵌套序列$B_1,B_2,\cdots,B_k(k>1)$为问题的一个最优解,各个箱子的深度值分别为$H_1,H_2,\cdots,H_k$。由上面的证明可知,$H_1$为最大深度值,而其余箱子组成的序列$B_2,B_3,\cdots,B_k(k>1)$即是在所有的箱子中去掉$B_1$及与其具有相同深度值的箱子后,在剩下的箱子中查找最长嵌套箱序列的一个最优解。因此,最长嵌套箱序列问题具有最优子结构性质。

### 5.3 算法的具体实现过程

整个算法包括主调过程Longest、判断箱子$i$和箱子$j$之间嵌套关系的过程Include、查找深度值最大的箱子的过程LookMax、确定箱子深度值的函数BackPatchHeight以及生成嵌套箱子序列的函数TraceBack。下面分别给以介绍。

(1)Longest *过程* 该过程为主调过程,其功能为在n个箱子中查找最长的嵌套序列。

```
void Longest(int * * B,int n,int d)
                //B为存放n个d维箱的数组
{ int * * A; //用来存放各箱子间嵌套关系的数组
  for (int i=0;i<n;i++)
     A[i]=new int[n+1];
          //其中A[i][n]用来存放箱子i的深度值
  for(i=0;i<n;i++)
    A[i][n]=0; //深度值初值均为0
  for(i=0;i<n;i++)
    for(int j=i+1;j<n;j++)
     { A[i][j]=Include(B[i],B[j],d);
          //利用Include()函数判断箱子i与j之间的
          //嵌套关系。
       A[j][i]=-A[i][j];
     }
  for(i=0;i<n;i++)
    for(j=0;j<n;j++) //生成各个箱子的深度值
      if( A[i][j]==-1) //若箱子i被嵌套,则回填j
                       //的深度值,其值为Hj=Hi+1
          BackPatchHeight(A,n,i,j);
  i=LookMax(A,n);//查找深度值最大的箱子作
                 //为首嵌套箱
  TraceBack(A,i,n);//利用A数组构造首箱之后
                   //的嵌套箱序列
}
```

(2)Include *函数* 该函数的功能为判断箱子$i$和箱子$j$之间的嵌套关系,返回0表示无嵌套关系,返回1表示$i$嵌套$j$,返回-1表示$j$嵌套$i$。

```
int Include(int * B1,int * B2,int d)
{ sort(B1,d);//用任意一种排序方法对d维箱的分
             //量进行排序
  sort(B2,d);
  if (B1[0]>=B2[0])
    { for(int i=0;i<d;i++)
        if(B1[i]<B2[i] return 0;//无嵌套关系
      return 1; //B1嵌套B2
    }
  else
  { for(int i=0;i<d;i++)
    if(B1[i]>B2[i]) return 0;//无嵌套关系
    return-1; //B2嵌套B1
  }
}
```

(3)BackPatchHeight *函数* 该函数的功能是当箱子$i$与箱子$j$具有嵌套关系时,修改箱子$i$或箱子$j$的深度值,并相应改变与箱子$i$或$j$有嵌套关系或被嵌套关系的箱子的深度值。

```
BackPatchHeight(int * * A,int n,int i,int j)
{if(A[i][n]==A[j][n]) //Bj嵌套Bi,因此Hj>Hi。
//若A[j][n]≥A[i][n],则不必将Hj加1
{A[j][n]++; //若A[j][n]=A[i][n],
            //则将Hj的值加1
for(int k=0;k<n;k++) //递归地修改嵌套箱
                     //子j的所有箱子的深度值
If(A[j][k]==-1)
BackPatchHeight(A,n,j,k);
}
```

(4)LookMax *函数* 该函数的功能为查找具有最大深度值的箱子。

```
int LookMax(int * * B,int k)
{ int max=B[0][k-1],temp=0;
  for (int i=0;i<k-1;i++)
   { if (B[i][k-1]>max)
       { max=B[i][k-1];
         temp=i;
       }
   }
  return(temp);
}
```

(5)TraceBack *函数* 其功能是生成以箱子max为首箱的嵌套箱序列,max为函数LookMax返回的具有最大深度值的箱子号。

```
void TraceBack(int * * B,int max,int n)
{
  int temp;
  cout<<"Box Serial:";
  while(B[max][n]>0)
      //若深度值不为0,则继续查找内层的嵌套箱
  { cout<<max+1<<">";
        //显示的效果为1>2>…
    int m=0;//m存放当前最大深度值,
        //temp存放其对应下标
    for (int i=0;i<n;i++)//在深度值最大的箱
        //子所嵌套的所有箱子中查找深度值
        //最大的箱子号,并且将其作为后继
       //序列的首箱
       { if (B[max][i]==1)
           if (B[i][n]>=m)
             { m=B[i][n];
               temp=i;
             }
         }
      max=temp;
  }
  cout<<max+1;//输出深度值为0的箱子的标号
}
```

## 6. 算法的时间复杂度分析

(1)判断两个嵌套关系的算法Include()

该算法的运算主要集中在对两个箱子内部d维容量的排序上,因此该算法的时间复杂度为$O(d\cdot\log_2 d)$。

(2)主调算法Longest()

算法耗费的时间主要集中在两处,一处是比较任意两个箱子的嵌套关系,花费的时间复杂度为$O(n^2\cdot d\cdot\log_2 d)$;另一处是回填各箱子的深度值,耗费的时间复杂度为$O(n^3)$。因此

# 内存异常数据的检测与恢复*)

朱智林[1,2] 李 航[1] 刘西洋[1] 陈 平[1] 张其林[2]
(西安电子科技大学软件工程研究所 西安710071)[1] (中国煤炭经济学院计算机系 烟台264005)[2]

## Data Corruption in Memory Detection and Recovery

ZHU Zhi-Lin LI Hang LIU Xi-Yang CHEN Ping ZHANG Qi-Lin
(Institute of Software Engineering, Xidian University, Xi'an 710071)

**Abstract** For some certain applications, they require massive application data to be resident in main memory. We categorize the exceptional data resided in memory based on the analysis of corresponding reasons. Exploiting mature CRC technologies to build a redundancy code for each data unit in memory to achieve integrity and combining traditional log and checkpoint technologies, we present the basic methods of prevention for corrupted data propagation and directly static detection and recover of corrupted data.

**Keywords** Redundancy code, Data corruption, Detect and recovery, Main-memory data management, Data consistence

## 1. 引言

在计算机应用领域,存在一类应用,要求系统满足一定的实时性要求,较高的性能,较好的容错能力。这类应用的一个重要特征就是业务关键(mission critical),所涉及的数据量大,比如电信计费、军事指挥控制、银行业务处理等等。在集中式省级移动计费系统中,用户群庞大,话费数据不能及时反映用户帐户当前的实际情况,数据一般滞后数小时乃至二十四小时。随着硬件技术的发展,RAM 价格的下降,计算机配置数 GB 内存已经成为现实的情况下,用户的相关数据常驻内存,已经成为可能,从而可以提高系统的实时性。然而数据常驻在易失性介质中,虽然能够提高系统工作效率,但也容易造成数据丢失。

本文以移动记费为背景,针对大量数据常驻内存的情况下的内存数据管理中数据异常产生的原因进行了简单的分析和分类,主要讨论了基于校验码的技术在内存数据管理[1,6]中的应用,探讨了静态检测与预防数据异常发生以及进行简单的数据异常恢复的基本思想。

## 2. 数据异常产生的原因与简单分类

移动计费系统体系结构如图1所示,其中 BOOM (Business Object Organization and Management)[8]提供对完整的 ACID 语义事务模型的支持。

在应用系统中由于进程的异常中止等软件错误往往会引起数据状态的不一致,异常数据(data corruption)[3]就是其中的一类。事实上在软件错误中约有25%～30%属于地址错误[2],诸如数组下标越界、指针走飞、程序栈错误等等。对于这类数据状态的不一致情况仅仅依靠 ACID 无法解决,需要采用新的措施加以处理。

在 BOOM 数据管理中,约定包含在 BeginUpdate 和 EndUpdate 之间的数据更新是合法数据更新,其它一律为非法数据更新。非法的数据更新所产生的数据不一致状态称之为直接物理异常数据;而由用户错误输入或应用逻辑错误所导致的数据状态的不一致称之为直接逻辑异常数据;直接异常数据被其它进程作为操作依据时,所产生的新的数据状态的不一致则称之为间接异常数据。

异常数据会缓慢地导致数据状态的不一致进一步扩散,而这类数据状态不一致的发现比较滞后,进行修复代价也较高。预防异常数据产生,阻止其传播,及时发现并修复,可提高系统的性能和容错能力。

BOOM 的事务可恢复框架[4,8]如图2所示,系统数据区禁止应用程序访问,其中有系统的内核,日志,ATT 表等,应用

*)基金项目:可信软件工程(413150501)。朱智林 博士生,副教授,主要研究方向为软件工程,面向对象技术,数据库技术;李 航 博士生;刘西洋 博士,副教授;陈 平 教授,博导。

整个算法的时间复杂度为 $O(\max\{n^2 \cdot d \cdot \log_2 d, n^3\})$。

**结束语** 本文提出了一种基于贪心策略的最长 d 维箱嵌套问题的解决方法,并对算法的时间复杂度进行了分析。实验证明了方法的有效性,实际上,如果算法的输入为内部已经排好序的 $n$ 个 d 维箱,那么算法的时间复杂度可降至 $O(n^3)$,这个复杂度是比较令人满意的。

## 参 考 文 献

1 Weisstein E W. Greedy algorithm from mathworld. http://mathworld.wolfram.com/GreedyAlgorithm.html 1999
2 Black P E. Greedy algorithm. http://www.nist.gov/dads/HTML/greedyalgo.html 2003
3 Yu Jiansheng. Greedy algorthm. http://icl.pku.edu.cn/yujs/papers/pdf/ComAlg10.pdf 2002
4 Sahni S. Data Structures, Algorithms, and Applictions in C++, McGraw-Hill, 1998
5 王晓东. 计算机算法设计与分析. 北京:电子工业出版社,2001