登录 | 注册

狼牙灬月
欢迎来访：http://leihuang.org/

个人资料

SpeedMe

访问： 152587次
积分： 3822
等级： BLOG 5
排名： 第3248名

原创： 113篇 转载： 0篇
译文： 0篇 评论： 1054条

个人简介

黄磊
重庆大学2011级在读本科
豆瓣：Kevin
微博：I_like丶it
独立博客：leihuang.org

博客专栏

Thinking in java
文章：19篇
阅读：27365

Thinking In Algorithm
文章：17篇
阅读：22543

文章分类

----------基础---------- (0)
java (17)
Algorithm (17)
Design Pattern (19)
jvm (0)

CSDN学院讲师招募，诚邀您加入！ 博客Markdown编辑器上线啦 那些年我们追过的Wrox精品红皮计算机图书 PMBOK第五版精讲视频教程 火星人敏捷开发1001问

《Thinking In Algorithm》13.详解动态规划问题

分类： Algorithm 2014-04-23 16:51 852人阅读 评论(0) 收藏 举报

动态规划 算法导论

目录(?) [-]

1. 什么是动态规划

dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. (通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法)
咦，这不就是分而治之法吗？不，虽然它们的目的是一样的，但它们分解的子问题属性不同。

- 分而治之将问题划分为互不相交的子问题。
- 动态规划不同，动态规划应用于子问题重叠情况，即不同的子问题具有公共的子子问题。

动态规划解决的问题必须具备三个属性：最优子结构性质，无后效性，子问题重叠性质。

- 最优子结构：如果问题的最优解所包含的子问题的解也是最优的，就称该问题具有最优子结构，即满足最优化原理。
- 无后效性：即某阶段状态一旦确定，就不受这个状态以后决策的影响。也就是说，某状态以后的过程不会影响以前的状态，只与当前状态有关。
- 重叠子问题：即子问题之间是不独立的，一个子问题在下一阶段决策中可能被多次使用到。（该性质并不是动态规划适用的必要条件，但是如果没有这条性质，动态规划算法同其他算法相比就不具备优势）

动态规划背基本思想： 对一个给定问题，我们需要解其不同部分（即子问题），再合并子问题的解以得出原问题的解。 通常许多子问题非常相似，为此动态规划法试图仅仅解决每个子问题一次，从而减少计算量： 一旦某个给定子问题的解已经出，则将其记忆化存储，以便下次需要同一个子问题解之时直接查表。 这种做法在重复子问题的数目关于输入的规模呈指数增长时特别有用。

2. 流水线调度问题

1. Characterize the structure of an optimal solution. (分析最优解的性质，并刻画其结构特征)
2. Recursively define the value of an optimal solution. (递归的定义最优解)
3. Compute the value of an optimal solution in a bottom-up fashion. (以自底向上或自顶向下的记忆化方式（备忘录法）计算出最优值)
4. Construct an optimal solution from computed information. (根据计算最优值时得到的信息，构造问题的最优解)

下面就根据算法导论上的流水线调度的例子来详解。

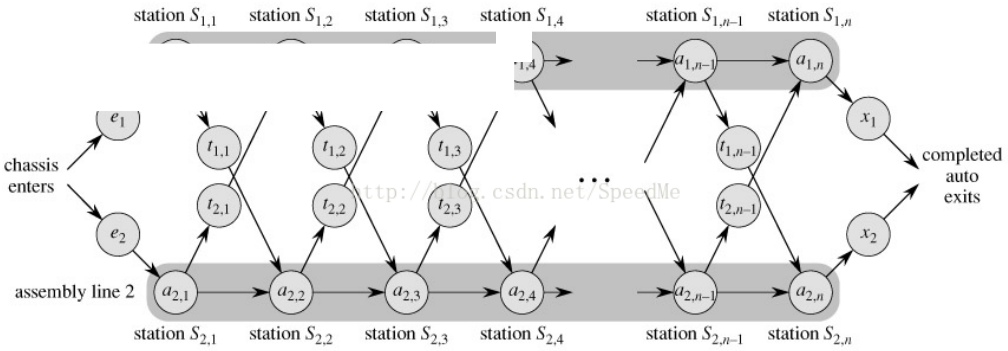
Operating System (6)
Database (1)
C/C++ (3)
C/C++ Vs Java (3)
Network (0)
----------兴趣---------- (0)
Search Engine (1)
源： 动态规划 poj
Android (11)
----------瞎搞---------- (0)
OpenCv/JavaCv (4)
Life (7)
User Experience (2)
windows (4)
Long long ago (5)

阅读排行
java中Map,List与Set的区 (8121)
《Thinking in Algorithm》 (7310)
android错误：The meth (7269)
android设置Spinner字体 (6016)
进程同步之信号量机制（ (4833)
java中的HashTable,Has (3621)
java中的static和final (3533)
三系统安装win7+ubuntu (3304)
ubuntu下配置MyEclipse: (3049)
版本问题android1.5以后 (2846)

评论排行
java中的static和final (928)
《Thinking in Algorithm》 (75)
android错误：The meth (6)
【android课表】一个简单 (5)
在csdn博客上添加qq聊天 (5)
java中Map,List与Set的区 (4)
版本问题android1.5以后 (4)
OpenCv将图片写入到视频 (4)
《Thinking In Algorithm》 (4)
《Thinking In Algorithm》 (2)

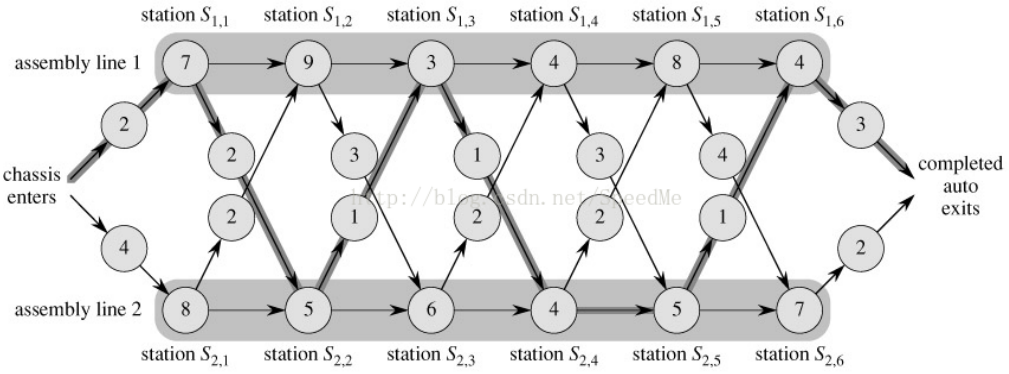
最新评论
android.view.InflateException: B
qq_18271659: 谢谢，也帮到了我
java中Map,List与Set的区别
xiaoxiashou: 写的真好
android错误：The method mak
L1002: 一语中的
android java.net.UnknownHostE
liangliangVAZ: 犯了同样的错误....
进程同步之信号量机制（pv操作）
Jessie__Zhu: 赞！
《心灵捕手》有感、
SpeedMe: @u010850027:好早的文章了..不谢.
《心灵捕手》有感、
丁国华: 谢谢分享 学习了`(*∩_∩*)'
java内部类引用外部类的变量时必
丁国华: 谢谢分享 学习了`(*∩_∩*)'

问题：有两条流水线，每条流水线分别有多个工作站，两个流水线的对应的同一个工作站之间可以传输要加工的产品，但需要一定时间记为$t_{i,j}$，但单条流水线站之间传输要加工的产品时间可以忽略不计。现在要求工作一个产品所需要的最短时间。流程如下图。



e代表产品进入不同流水线的时间，a代表不同站加工产品所需时间，t代表不同流水线的不同站传递产品的时间。

分析：看题目中的图可能不够清楚，我们拿出一个有具体时间的图来分析。如下图：



图中带有阴影部分的箭头代表最优解。那么他们是怎样得出来的呢。下面我们就利用动态规划的步骤来解决这个问题。

2.1 构造最优解结构

我们从起点开始分析，首先 我们会选择哪条流水线开始呢，有人会说那肯定是第一条流水线开始啊，因为2+7<4+8.但其实并不是这样，算法导论这个例子举的不好，很容易产生这样的误解。其实不一定选择这条路径，因为你要保证下一条也时间最短，假设S11-S22的时间不是2是4的话，那么选择从第一条流水线开始就是错误的，因为2+7+4>4+8,这样就是第二条流水线到达S22的时间更快。(如果你觉得我这里说的混乱的话可以跳过这一段，没太大影响)。

知道这一点之后，我们继续。我们分别记录下达到S11和S21的时间，然后选择下一站，到达S12的路径有两条，到达S22的路径也有两条。我们就是分别从两条中选出最快的哪条作为到达S12和S22的最短时间存入表中。。。。这样一直走到末尾。

那么就可以把问题简化为计算通过$S_{1,j}$ 和$S_{2,j}$ 的最短时间路径：

我们知道到达$S_{1,j}$ 站的的路径只有两条：

- the fastest way through station $S_{1,j-1}$ and then directly through station$S_{1,j}$, or
- the fastest way through station $S_{2,j-1}$, a transfer from line 2 to line 1, and then through station$S_{1,j}$.

同理到达$S_{2,j}$ 的路径也只有两条：

- the fastest way through station $S_{2,j-1}$ and then directly through station$S_{2,j}$, or
- the fastest way through station $S_{1,j-1}$, a transfer from line 1 to line 2, and then through station$S_{2,j}$.

我们要做的工作就是分别得出两个站的最短时间，然后存入表中。

《程序员面试题精选》07.后序遍
SpeedMe: @hzw05103020:不好意思,没注意你的回复,更改删除这些博客的时候,才发现.这个程序有问题,...

《程序员面试题精选》07.后序遍
hzw05103020: 你好，递归 searchTree(a,i); searchTree(Arrays.copyOfRa...

2.2 递归定义最优解

上一步中我们分析出了，最优解结构，下面就用递归式来表示得出最优解的过程。

我们用$f_i[j]$ 来表示到达$S_{i,j}$ 的最短时间。

首先我们计算出到达第一站的时间：

$$f_1[1] = e_1 + a_{1,1}$$

$$f_2[1] = e_2 + a_{2,1}.$$

根据第一步骤中构造的最优解结构我们可以计算出达到S1,j和S2,j的时间：

$$f_1[j] = \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j})$$

http://blog.csdn.net/SpeedMe

$$f_2[j] = \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j})$$

总上所属我们可以写出最有解的递归式如下

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1, \\ \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1, \\ \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

2.3 以自底向上或自顶向下的记忆化方式（备忘录法）计算出最优值

这一步骤就是上一步中递归式的实现步骤了，下面我就不多说了，直接上伪代码，大家最好自己去实现一遍。

```
FASTEST-WAY(a, t, e, x, n)
 1  f1[1] ← e1 + a1,1
 2  f2[1] ←e2 + a2,1
 3  for j ← 2 to n
 4       do if f1[j - 1] + a1,j ≤ f2[j - 1] + t2,j-1 + a1,j
 5             then f1[j] ← f1[j - 1] + a1, j
 6                  l1[j] ← 1
 7             else f1[j] ← f2[j - 1] + t2,j-1 + a1,j
 8                  l1[j] ← 2
 9          if f2[j - 1] + a2,j ≤ f1[j - 1] + t1,j-1 + a2,j
10             then f2[j] ← f2[j - 1] + a2, j
11                  l2[j] ← 2
12             else f2[j] ∞ f1[j - 1] + t1,j-1 + a2,j
13                  l2[j] ← 1
14  if f1[n] + x1 ≤ f2[n] + x2
15      then f* = f1[n] + x1
16          l* = 1
17      else f* = f2[n] + x2
18             l* = 2
```

2.4 根据计算得到的结果，构造出最优解(即输出最佳路径)

伪代码：

```
PRINT-STATIONS(l, n)
1  i ← l*
2  print "line " i ", station " n
3  for j ← n downto 2
4       do i ← li[j]
5          print "line " i ", station " j - 1
```

如例子中输出的结果为下：

line 1, station 6

line 2, station 5

line 2, station 4

line 1, station 3

line 2, station 2

line 1, station 1

3 矩阵链乘法

可能大家现在还是有点模糊，下面我们再举一个例子来详细说明动态规划算法流程。
想必大家都学过线性代数，也熟悉矩阵，如果忘了的话也不要紧，下面的问题只涉及到矩阵的乘法。先看题。

问题：给定一个矩阵序列〈$A_1, A_2, ..., A_n$〉做乘法，求如何给他们添加上括号，使得操作数最少？

分析：可能你还有点看不懂题目，我们下面来解释下，先看看两矩阵A,B相乘的代码。

```
MATRIX-MULTIPLY(A, B)
1 if columns[A] ≠ rows[B]
2    then error "incompatible dimensions"
3    else for i ← 1 to rows[A]                  //   rows[A]
4            do for j ← 1 to columns[B]         //   col[B]
5                  do C[i, j] ← 0
6                    for k ← 1 to columns[A]   //   col[A]
7                        do C[i, j] ← C[i, j] + A[i, k] · B[k, j]
8        return C
```

从上面的代码中，我们可以看出两个矩阵AB相乘，他们的操作次数为rows[A]*col[B]*col[A].知道这一点之后，我们看下面：
给出三个矩阵：A是10*30，B是30*5，C是5*60，求(AB)C和A(BC)的操作次数。

(AB)C = (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations
A(BC) = (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations.

这下你应该清楚了题意吧。好，那我们下面想想怎么来减少它的次数。依然按照动态规划的算法流程来解决问题：

(下面，我不会讲的太细，只说些帮助你理解的东西，最好跟着算法导论一起来看，我看了两个多小时才看懂了，囧了。)

3.1 构造最优结构

算法导论中提到一个临界值K(就是这个弄瞎我的眼)，如何来理解这个K值是什么呢？举个例子。A1*A2*A3三个矩阵相乘，

- 当(A1*A2)*A3时，此时K=2，即表示分界点为A2.
- 当A1*(A2*A3)时，此时K=1，即表示分界点为A1.

对于通用式，Ai*.....*Aj(i<j) ，他们之间存在一个临界值K把他们分开，使得他们计算出来的结果最小。
即：(Ai*.....*Ak)*(Ak+1*.....*Aj)
知道K值是什么之后，我们就进入下一环节，递归求出最优解。

3.2 递归求最优解

上面我们知道了临界值K实质就是给一个方程式加括号，当方程式有三个一下值时，很好找到K值，但当方程含有三个以上值的时候该怎么办呢？那就是要利用递归的方法了。

对于Ai*.....*Aj 我们记m(i,j) $1 \le i \le j \le n$ 为方程的最小值。Px为行数或列数($0 \le x \le n$)

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

对于上面的方程你可能不理解，你随便取i和j的值代入进去试一下就懂了。如下：

m[1,2]=m[1,1]+m[2,2]+P0*P1*P2 = P0*P1*P2

上面的例子只要运算一次，下面的就要多次了。

m[1,3]=min{m[1,1]+m[2,3]+P0*P1*P3 , m[1,2]+m[2,3]+P0*P2*P3}

求的之前要先求m[2,3]和m[1,2]的最小值，求出来了之后，再求上面的最小值。

这下应该清楚了吧？如果还是晕晕乎乎的话，那就看下面的实现部分吧，跟着代码走一遍(我就是这样才理解的).

3.3 计算出最优解

上面我们得出了他的递归式，下面我们就来实现它。

伪代码：

```
MATRIX-CHAIN-ORDER(p)
 1 n ← length[p] - 1
 2 for i ← 1 to n
 3     do m[i, i] ← 0
 4 for l ← 2 to n      ▷l is the chain length.
 5     do for i ← 1 to n - l + 1
 6           do j ← i + l - 1
 7              m[i, j] ← ∞
 8              for k ← i to j - 1
 9                  do q ← m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j
10                     if q < m[i, j]
11                        then m[i, j] ← q
12                             s[i, j] ← k
13 return m and s
```

解释一下：

2-3：将m[i,j]，i=j时的值初始化为0.

4： l其实就是j-i+1，即m[i,j]中元素的个数

5-7：同时移动i和j,保持i，j的距离

8-12：执行的就是递归式中的min()操作，寻找最小的K值，并保存下使得方程最小时，K的值

下面给出java代码，运行下。

[java] view plain copy print ?

```
// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
MatrixChainOrder(int p[])
{
    // length[p] = n + 1
    n = p.length - 1;
    // m[i,j] = Minimum number of scalar multiplications (i.e., cost)
    // needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]
    // cost is zero when multiplying one matrix
    for (i = 1; i <= n; i++)
       m[i,i] = 0;

    for (L=2; L<=n; L++) { // L is chain length
        for (i=1; i<=n-L+1; i++) {
            j = i+L-1;
            m[i,j] = MAXINT;
            for (k=i; k<=j-1; k++) {
                // q = cost/scalar multiplications
                q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
                if (q < m[i,j]) {
                    m[i,j] = q;
                    s[i,j]=k// s[i,j] = Second auxiliary table that stores k
                              // k      = Index that achieved optimal cost

                }
            }
        }
```

```
    }
}
```

算法导论中还有个例子：

matrix	dimension
A_1	30 × 35
A_2	35 × 15
A_3	15 × 5
A_4	5 × 10
A_5	10 × 20
A_6	20 × 25

此时m[2,5]的计算过程如下。

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases} = 7125.$$

3.4 得出最佳方案

得出最终的结果，无疑就是把方程式输出来，直接看伪代码：

```
PRINT-OPTIMAL-PARENS(s, i, j)
1 if i = j
2     then print "A"_i
3     else print "("
4          PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5          PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6          print ")"
```

3.5 java实现

[java] view plain copy print ?

```
public class MatrixOrderOptimization {
    protected int[][]m;
    protected int[][]s;
    public void matrixChainOrder(int[] p) {
        int n = p.length - 1;
        m = new int[n][n];
        s = new int[n][n];
        for (int i = 0; i < n; i++) {
            m[i] = new int[n];
            m[i][i] = 0;
            s[i] = new int[n];
        }
        for (int ii = 1; ii < n; ii++) {
            for (int i = 0; i < n - ii; i++) {
                int j = i + ii;
                m[i][j] = Integer.MAX_VALUE;
                for (int k = i; k < j; k++) {
                    int q = m[i][k] + m[k+1][j] + p[i]*p[k+1]*p[j+1];
                    if (q < m[i][j]) {
                        m[i][j] = q;
                        s[i][j] = k;
                    }
                }
            }
        }
    }
    public void printOptimalParenthesizations() {
        boolean[] inAResult = new boolean[s.length];
        printOptimalParenthesizations(s, 0, s.length - 1, inAResult);
    }
```

```
    void printOptimalParenthesizations(int[]
[]s, int i, int j,  /* for pretty printing: */ boolean[] inAResult) {
        if (i != j) {
            printOptimalParenthesizations(s, i, s[i][j], inAResult);
            printOptimalParenthesizations(s, s[i][j] + 1, j, inAResult);
            String istr = inAResult[i] ? "_result " : " ";
            String jstr = inAResult[j] ? "_result " : " ";
            System.out.println(" A_" + i + istr + "* A_" + j + jstr);
            inAResult[i] = true;
            inAResult[j] = true;
        }
    }
}
```

4. 动态规划原理

通过前面的两个例子，大家对动态规划有了初步的理解。下面我们从原理的解读来分析动态规划，标题1中我们讲到动态规划需要维持三种特性最优子结构，子问题重叠，无后效性。

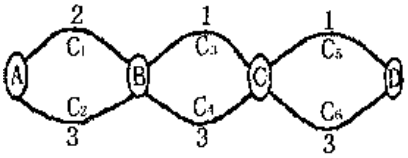
4.1 最优子结构

如果一个问题的最优解包含其子问题的最优解，我们就称此问题具有最优子结构性质。

【例题1】余数最少的路径。

如图所示，有4个点，分别是A、B、C、D，相邻两点用两条连线C2k，C2k-1(1≤k≤3)表示两条通行的道路。连线上的数字表示道路的长度。定义从A到D的所有路径中，长度除以4所得余数最小的路径为最优路径。

求一条最优路径。



【分析】在这个问题中，如果A的最优取值可以由B的最优取值来确定，而B的最优取值为(1+3) mod 4 = 0，所以A的最优值应为2，而实际上，路径C1 - C3 - C5可得最优值为(2+1+1) mod 4 = 0，所以，B的最优路径并不是A的最优路径的子路径，也就是说，A的最优取值不是由B的最优取值决定的，即其不满足最优化原理，问题不具有最优子结构的性质。

由此可见，并不是所有的“决策问题”都可以用“动态规划”来解决，运用“动态规划”来处理问题必须满足最优化原理。

4.2 重叠子问题

问题利用递归方法会反复地求解相同的子问题，而不是一直生成新的子问题，我们就成该问题是重叠子问题。在分治方法中通常就每一步都生成全新的子问题，动态规划就是利用重叠子问题性质：对每个子问题球结一次，将解存入一个表中，当再次需要这个子问题时直接查表，每次查表的代价为常数时间。

4.3 无后效性

它是这样一种性质：某阶段的状态一旦确定，则此后过程的演变不再受此前各种状态及决策的影响，简单的说，就是“未来与过去无关”，当前的状态是此前历史的一个完整总结，此前的历史只能通过当前的状态去影响过程未来的演变。具体地说，如果一个问题被划分各个阶段之后，阶段I中的状态只能由阶段I-1中的状态通过状态转移方程得来，与其它状态没有关系，特别是与未发生的状态没有关系。从图论的角度去考虑，如果把这个问题中的状态定义成图中的顶点，两个状态之间的转移定义为边，转移过程中的权值增量定义为边的权值，则构成一个有向无环加权图，因此，这个图可以进行“拓扑排序”，至少可以按它们拓扑排序的顺序去划分阶段。

简单来说就是在状态i求解时用到状态j而状态j求解时用到状态k.....状态N。

而如果求状态N时有用到了状态i这样求解状态的过程形成了环就没法用动态规划解答了，这也是上面用图论理解动态规划中形成的图无环的原因。

也就是说当前状态是前面状态的完美总结，现在与过去无关。

当然，要是换一个划分状态或阶段的方法就满足无后效性了，这样的问题仍然可以用动态规划解。

动态规划问题还有还多，这里就不一一举例子了，如最长公子序列，背包问题，最有二叉搜索树。之后再添加上去，先把上面两个例子理解了。

不得不说动态规划问题好难啊，我ri。

看下这篇博客吧：最大子数组和

上一篇 求解一道腾讯笔试题(帮帮忙)

下一篇 《Thinking In Algorithm》14.由背包问题了解动态规划和贪心

顶 3 踩 0

主题推荐 动态规划 algorithm 算法导论 structure 产品

猜你在找

《Thinking In Algorithm》11堆结构之二叉堆
简单对象访问协议SOAP及其应用
多重集合的排列和组合问题
尝试用android-logging-log4j去实现log输出内容到
用C#将数据写入到Excel文件的方法
二叉搜索树的建立 查找 删除操作
每天学习一算法系列23写一个程序要求功能求出用
使用Corosync+Pacemaker为Icehouse L3-agent提供
PAT 1034 Head of a Gang 图论DFS
K-D tree 数据结构

准备好了么？跳吧 ！ 更多职位尽在 CSDN JOB

Senior IPTV Software Development Eng 我要跳槽
爱立信(中国)通信有限公司 | 20-30K/月
资深开发工程师-HTML5动态页面（深圳） 我要跳槽
平安科技(深圳)有限公司 | 10-15K/月
产品规划师 我要跳槽
广州华微明天软件技术有限公司 | 8-15K/月
宝智软件诚聘PB编程工程师，年龄不是问 我要跳槽
南昌宝智软件技术有限公司 | 4-8K/月

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide
Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase
Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved