

liufeng_king的专栏

☰ 目录视图

☰ 摘要视图

RSS 订阅

个人资料



风仲达

访问: 195897次

积分: 2834

等级: BLOG > 5

排名: 第4587名

原创: 57篇

转载: 33篇

译文: 0篇

评论: 117条

文章搜索

博客专栏



算法笔记——《算法设计与分析》
文章: 50篇
阅读: 150542

文章分类

java基础 (16)

oracle (8)

工具 (5)

java框架 (6)

linux (1)

操作系统 (1)

观点 (3)

算法 (50)

android (1)

文章存档

2014年12月 (1)

2014年11月 (1)

2014年10月 (3)

2014年04月 (1)

2013年12月 (1)

展开

2015年4月微软MVP申请 《京东技术解密》有奖试读, 礼品大放送 “我的2014”年度征文活动火爆开启 CSDN 2014博客之星

0020算法笔记——【动态规划】最优二叉搜索树问题

分类: 算法2013-03-20 10:373339人阅读评论(4)收藏举报

最优二叉搜索树

算法笔记

最小平均路长

四边形不等式

动态规划

1、问题描述:

设 $S=\{x_1, x_2, \cdots, x_n\}$ 是一个有序集合, 且 x_1, x_2, \cdots, x_n 表示有序集合的二叉搜索树利用二叉树的顶点存储有序集中的元素, 而且具有性质: 存储于每个顶点中的元素 x 大于其左子树中任一顶点中存储的元素, 小于其右子树中任意顶点中存储的元素。二叉树中的叶顶点是形如 (x_i, x_{i+1}) 在表示 S 的二叉搜索树中搜索一个元素 x , 返回的结果有两种情形:
(1) 在二叉树的内部顶点处找到: $x = x_i$
(2) 在二叉树的叶顶点中确定: $x \in (x_i, x_{i+1})$
设在情形(1)中找到元素 $x = x_i$ 的概率为 b_i ; 在情形(2)中确定 $x \in (x_i, x_{i+1})$ 的概率为 a_i 。其中约定 $x_0 = -\infty, x_{n+1} = +\infty$, 有
$$a_i \geq 0, 0 \leq i \leq n; b_j \geq 0, 1 \leq j \leq n; \sum_{i=0}^n a_i + \sum_{j=1}^n b_j = 1$$
集合 $\{a_0, b_1, a_1, \cdots, b_n, a_n\}$ 称为集合 S 的存取概率分布。
最优二叉搜索树: 在一个表示 S 的二叉树 T 中, 设存储元素 x_i 的结点深度为 c_i ; 叶结点 (x_j, x_{j+1}) 的结点深度为 d_j .
$$p = \sum_{i=1}^n b_i(1 + c_i) + \sum_{j=0}^n a_j d_j$$

注: 在检索过程中, 每进行一次比较, 就进入下面一层, 对于成功的检索, 比较的次数就是所在的层数加1。对于不成功的检索, 被检索的关键码属于那个外部结点代表的可能关键码集合, 比较次数就等于此外部结点的层数。对于图的内结点而言, 第0层需要比较操作次数为1, 第1层需要比较2次, 第2层需要3次。

p 表示在二叉搜索树 T 中作一次搜索所需的平均比较次数。 P 又称为二叉搜索树 T 的平均路长, 在一般情况下, 不同的二叉搜索树的平均路长是不同的。对于有序集 S 及其存取概率分布 $(a_0, b_1, a_1, \cdots, b_n, a_n)$, 在所有表示有序集 S 的二叉搜索树中找出一棵具有最小平均路长的二叉搜索树。

设 P_i 是对 a_i 检索的概率。设 q_i 是对满足 $a_i < x < a_{i+1}, 0 \leq i \leq n$ 的标识符 x 检索的概率, (假定 $a_0 = -\infty$ 且 $a_{n+1} = +\infty$)。

http://blog.csdn.net/liufeng_king/article/details/8694652

1/8

阅读排行

0019算法笔记——【动规】(9168)
0010算法笔记——【动规】(6763)
0007算法笔记——【分治】(5879)
0022算法笔记——【贪心】(5453)
0008算法笔记——【分治】(5319)
0018算法笔记——【动规】(5215)
0013算法笔记——【动规】(5126)
<sset> <sif> (4994)
0012算法笔记——【动规】(4863)
0014算法笔记——【动规】(4748)

评论排行

0010算法笔记——【动态规划】	(15)
0006算法笔记——【分治】	(10)
001java面试笔记——【j	(8)
0018算法笔记——【动态规划】	(7)
0008算法笔记——【分治】	(5)
0016算法笔记——【动态规划】	(5)
0035算法笔记——【分治】	(5)
让我们一起成长吧~(2013)	(5)
0025算法笔记——【贪心】	(4)
0020算法笔记——【动态规划】	(4)

推荐文章

- * 分布式系统的设计几个要注意的地方
- * ShadowGun系列之二——雾和体积光
- * Android Xfermode 实战 实现圆形、圆角图片
- * “暗隐间谍”——利用NDK NativeActivity技术实现Android加固
- * 一起来开发Android的天气软件

最新评论

0049算法笔记——【随机化算法秋_daisy】:在主元素问题中,k值为什么取那个值呢

0040算法笔记——【分支限界法qq_17468457:19, 18, 20, 21, 19, 这9数据怎么来的? ? ? ! ? 普及及在线急等

0040算法笔记——【分支限界法qq_17468457:19, 18, 20, 21, 19, 这9个数据是怎么来的啊? 求大神普及

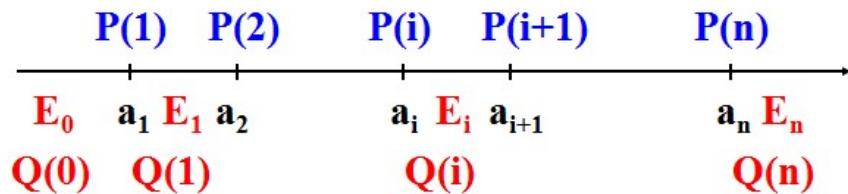
0047算法笔记——【随机化算法fregocco:问问, 这个一直看不懂

0010算法笔记——【动态规划】CodeMonkeyAndroid: 楼主, 第一个递推关系式P的下标怎么不是i,而是i-1?

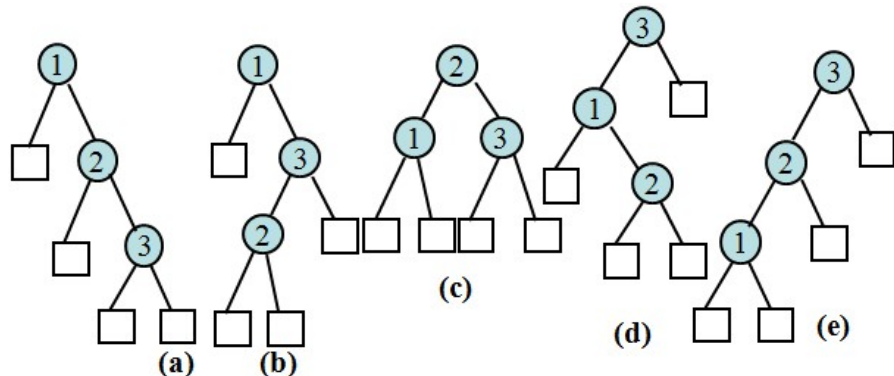
0035算法笔记——【分支限界法liningwoainiya:讲的很好, csdn中缺乏很全面的基础知识的讲解。支持一下。

0030算法笔记——【回溯法】最demo_helloworld: else { for (int i=1;j=m;i++) { ...

0001算法笔记——NP完全理论Boandy: 学习算法内容, 更要学习博士扎实的精神!



对于有n个关键码的集合，其关键码有n!种不同的排列，可构成的不同二叉搜索树有 $\frac{1}{n+1} C_{2n}^n$ 棵。(n个结点的不同二叉树,卡塔兰数)。如何评价这些二叉搜索树，可以用树的搜索效率来衡量。例如：标识符集{1, 2, 3}={do, if, stop}可能的二分检索树为：



若 $P_1=0.5$, $P_2=0.1$, $P_3=0.05$, $q_0=0.15$, $q_1=0.1$, $q_2=0.05$, $q_3=0.05$, 求每棵树的平均比较次数（成本）。

$$P_{a(n)} = 1 \times p_1 + 2 \times p_2 + 3 \times p_3 + 1 \times q_0 + 2 \times q_1 + 3 \times (q_2 + q_3) = 1 \times 0.5 + 2 \times 0.1 + 3 \times 0.05 + 1 \times 0.05 + 2 \times 0.1 + 3 \times (0.05 + 0.05) = 1.5$$

$$P_{b(n)} = 1 \times p_1 + 2 \times p_3 + 3 \times p_2 + 1 \times q_0 + 2 \times q_3 + 3 \times (q_1 + q_2) = 1 \times 0.5 + 2 \times 0.05 + 3 \times 0.1 + 1 \times 0.15 + 2 \times 0.05 + 3 \times (0.1 + 0.05) = 1.6$$

$$P_{c(n)} = 1 \times p_2 + 2 \times (p_1 + p_3) + 2 \times (q_0 + q_1 + q_2 + q_3) = 1 \times 0.1 + 2 \times (0.5 + 0.05) + 2 \times (0.15 + 0.1 + 0.05 + 0.05) = 1.9$$

$$P_{d(n)} = 1 \times p_3 + 2 \times p_1 + 3 \times p_2 + 1 \times q_3 + 2 \times q_0 + 3 \times (q_1 + q_2) = 1 \times 0.05 + 2 \times 0.5 + 3 \times 0.1 + 1 \times 0.05 + 2 \times 0.15 + 3 \times (0.1 + 0.05) = 2.15$$

$$P_{e(n)} = 1 \times p_3 + 2 \times p_2 + 3 \times p_1 + 1 \times q_3 + 2 \times q_2 + 3 \times (q_0 + q_1) = 1 \times 0.05 + 2 \times 0.1 + 3 \times 0.5 + 1 \times 0.05 + 2 \times 0.15 + 3 \times (0.15 + 0.1) = 2.85$$

因此，上例中的最小平均路长为 $P_{a(n)}=1.5$ 。

可以得出结论：结点在二叉搜索树中的层次越深，需要比较的次数就越多，因此要构造一棵最小二叉树，一般尽量把搜索概率较高的结点放在较高的层次。

2、最优子结构性质：

假设选择 k 为树根, 则 $1, 2, \dots, k-1$ 和 a_0, a_1, \dots, a_{k-1} 都将位于左子树 L 上, 其余结点 ($k+1, \dots, n$ 和 a_k, a_{k+1}, \dots, a_n) 位于右子树 R 上。设 $COST(L)$ 和 $COST(R)$ 分别是二分检索树 T 的左子树和右子树的成本。则检索树 T 的成本是: $P(k) + COST(L) + COST(R) + \dots$ 。若 T 是最优的, 则上式及 $COST(L)$ 和 $COST(R)$ 必定都取最小值。

证明：二叉搜索树T 的一棵含有顶点 x_i, \dots, x_j 和叶顶点 $(x_{i-1}, x_i), \dots, (x_j, x_{j+1})$ 的子树可以看作是有序集 $\{x_i, \dots, x_j\}$ 关于全集为 $\{x_{i-1}, x_{j+1}\}$ 的一棵二叉搜索树(T自身可以看作是有序集)。根据S 的存取分布概率，在子树的顶点处被搜索到的概率是：

$$w_{ij} = \sum_{i-1 \leq k \leq j} a_k + \sum_{i \leq k \leq j} b_k$$

。{x_i, ..., x_j}的存储概率分布为{a_{i-1}, b_i, ..., b_j, a_j}，其中， $\bar{b}_k = b_k / w_{ij}$, $i \leq k \leq j$; $\bar{a}_h = a_h / w_{ij}$, $i-1 \leq h \leq j$

赞!!!!

0017算法笔记——【动态规划】
Crazy--journey: 不错。讲得很详细。多谢楼主了

0019算法笔记——【动态规划】
百晓生: 博主第一个程序有点儿问题吧, 结果不完全正确

设 T_{ij} 是有序集 $\{x_i, \dots, x_j\}$ 关于存储概率分布为 $\{a_{i-1}, b_i, \dots, b_j, a_j\}$ 的一棵最优二叉搜索树, 其平均路长为 p_{ij} , T_{ij} 的根节点存储的元素 x_m , 其左子树 T_l 和右子树 T_r 的平均路长分别为 p_l 和 p_r 。由于 T_l 和 T_r 中顶点深度是它们在 T_{ij} 中的深度减1, 所以得到:

$$\begin{aligned}
 w_{ij} p_{ij} &= w_{i,m-1}(p_l + 1) + w_{m,m} + w_{m+1,j}(p_r + 1) \\
 &= w_{ij} + w_{i,m-1}p_l + w_{m+1,j}p_r \quad i \leq j
 \end{aligned}$$

左子树的搜索概率
右子树的搜索概率

由于 T_l 是关于集合 $\{x_i, \dots, x_{m-1}\}$ 的一棵二叉搜索树, 故 $p_l \geq p_{i,m-1}$ 。若 $p_l > p_{i,m-1}$, 则用 $T_{i,m-1}$ 替换 T_l 可得到平均路长比 T_{ij} 更小的二叉搜索树。这与 T_{ij} 是最优二叉搜索树矛盾。故 T_l 是一棵最优二叉搜索树。同理可证 T_r 也是一棵最优二叉搜索树。因此最优二叉搜索树问题具有最优子结构性质。

3、递推关系:

根据最优二叉搜索树问题的最优子结构性质可建立计算 p_{ij} 的递归式如下:

$$\begin{aligned}
 w_{ij} p_{ij} &= w_{ij} + \min_{i \leq k \leq j} \{w_{i,k-1}p_{i,k-1} + w_{k+1,j}p_{k+1,j}\}, \quad i \leq j \\
 p_{i,i-1} &= 0, \quad 1 \leq i \leq n
 \end{aligned}$$

初始时:

记 w_{ij} p_{ij} 为 $m(i,j)$, 则 $m(1,n)=w_{1,n}$ $p_{1,n}=p_{1,n}$ 为所求的最优值。计算 $m(i,j)$ 的递归式为:

$$\begin{aligned}
 m(i,j) &= w_{ij} + \min_{i \leq k \leq j} \{m(i,k-1) + m(k+1,j)\}, \quad i \leq j \\
 m(i,i-1) &= 0, \quad i = 1, 2, \dots, n
 \end{aligned}$$

4、求解过程:

1) 没有内部节点时, 构造 $T[1][0], T[2][1], T[3][2], \dots, T[n+1][n]$

2) 构造只有1个内部结点的最优二叉搜索树 $T[1][1], T[2][2], \dots, T[n][n]$, 可以求得 $m[i][i]$ 同时可以用一个数组存做根结点元素为: $s[1][1]=1, s[2][2]=2 \dots s[n][n]=n$

3) 构造具有2个、3个、.....、n个内部结点的最优二叉搜索树。

.....

r (起止下标的差)

0 $T[1][1], T[2][2], \dots, T[n][n]$,

1 $T[1][2], T[2][3], \dots, T[n-1][n]$,

2 $T[1][3], T[2][4], \dots, T[n-2][n]$,

.....

r $T[1][r+1], T[2][r+2], \dots, T[i][i+r], \dots, T[n-r][n]$

.....

n-1 $T[1][n]$

具体代码如下:

```

[cpp]
01. //3d11-1 最优二叉搜索树 动态规划
02. #include "stdafx.h"
03. #include <iostream>
04. using namespace std;
05.
06. const int N = 3;
07.
08. void OptimalBinarySearchTree(double a[], double b[], int n, double **m, int **s, double **w);
09. void Traceback(int n, int i, int j, int **s, int f, char ch);

```

```

10.
11. int main()
12. {
13.     double a[] = {0.15,0.1,0.05,0.05};
14.     double b[] = {0.00,0.5,0.1,0.05};
15.
16.     cout<<"有序集的概率分布为: "<<endl;
17.     for(int i=0; i<N+1; i++)
18.     {
19.         cout<<"a"<<i<<"="<<a[i]<<" ,b"<<i<<"="<<b[i]<<endl;
20.     }
21.
22.     double **m = new double *[N+2];
23.     int **s = new int *[N+2];
24.     double **w = new double *[N+2];
25.
26.     for(int i=0; i<N+2; i++)
27.     {
28.         m[i] = new double[N+2];
29.         s[i] = new int[N+2];
30.         w[i] = new double[N+2];
31.     }
32.
33.     OptimalBinarySearchTree(a,b,N,m,s,w);
34.     cout<<"二叉搜索树最小平均路长为: "<<m[1][N]<<endl;
35.     cout<<"构造的最优二叉树为:"<<endl;
36.     Traceback(N,1,N,s,0,'0');
37.
38.     for(int i=0; i<N+2; i++)
39.     {
40.         delete m[i];
41.         delete s[i];
42.         delete w[i];
43.     }
44.     delete[] m;
45.     delete[] s;
46.     delete[] w;
47.     return 0;
48. }
49.
50. void OptimalBinarySearchTree(double a[],double b[],int n,double **m,int **s,double **w)
51. {
52.     //初始化构造无内部节点的情况
53.     for(int i=0; i<=n; i++)
54.     {
55.         w[i+1][i] = a[i];
56.         m[i+1][i] = 0;
57.     }
58.
59.     for(int r=0; r<n; r++)//r代表起止下标的差
60.     {
61.         for(int i=1; i<=n-r; i++)//i为起始元素下标
62.         {
63.             int j = i+r;//j为终止元素下标
64.
65.             //构造T[i][j] 填写w[i][j],m[i][j],s[i][j]
66.             //首选i作为根, 其左子树为空, 右子树为节点
67.             w[i][j]=w[i][j-1]+a[j]+b[j];
68.             m[i][j]=m[i+1][j];
69.             s[i][j]=i;
70.
71.             //不选i作为根, 设k为其根, 则k=i+1, ...,j
72.             //左子树为节点: i,i+1.....k-1,右子树为节点: k+1,k+2,.....j
73.             for(int k=i+1; k<=j; k++)
74.             {
75.                 double t = m[i][k-1]+m[k+1][j];
76.
77.                 if(t<m[i][j])
78.                 {
79.                     m[i][j]=t;
80.                     s[i][j]=k;//根节点元素
81.                 }
82.             }
83.             m[i][j]+=w[i][j];
84.         }
85.     }
86. }
87.
88. void Traceback(int n,int i,int j,int **s,int f,char ch)

```

```

89.  {
90.     int k=s[i][j];
91.     if(k>0)
92.     {
93.         if(f==0)
94.         {
95.             //根
96.             cout<<"Root: "<<k<<" (i:j):("<<i<<","<<j<<")<<endl;
97.         }
98.         else
99.         {
100.            //子树
101.            cout<<ch<<" of "<<f<<": "<<k<<" (i:j):("<<i<<","<<j<<")<<endl;
102.        }
103.
104.        int t = k-1;
105.        if(t>=i && t<=n)
106.        {
107.            //回溯左子树
108.            Traceback(n,i,t,s,k,'L');
109.        }
110.        t=k+1;
111.        if(t<=j)
112.        {
113.            //回溯右子树
114.            Traceback(n,t,j,s,k,'R');
115.        }
116.    }
117. }

```

4、构造最优解：

算法OptimalBinarySearchTree中用s[i][j]保存最优子树T(i,j)的根节点中的元素。当s[i][n]=k时，x_k为所求二叉搜索树根节点元素。其左子树为T(1,k-1)。因此，i=s[1][k-1]表示T(1,k-1)的根节点元素为x_i。依次类推，容易由s记录的信息在O(n)时间内构造出所求的最优二叉搜索树。

5、复杂度分析与优化：

算法中用到3个数组m,s和w，故所需空间复杂度为O(n²)。算法的主要计算量在于计算 $w_{ij} + \min_{i \leq k \leq j} \{m(i, k-1) + m(k+1, j)\}$ 。对于固定的r，它需要的计算时间O(j-

i+1)=O(r+1)。因此算法所耗费的总时间为： $\sum_{r=0}^{n-1} \sum_{i=1}^{n-r} O(r+1) = O(n^3)$ 。事实上，由《动态规划加速原理之四边形不等式》可以得到：

$$m(i, j) = \min_{s(i, j-1) \leq k \leq s(i+1, j)} \{m(i, k-1) + w(i, k-1) + m(k+1, j) + w(k+1, j)\}, i < j$$
 而此状态转移方程的时间复杂度为O(n²)。由此，对算法改进后的代码如下：

```

[cpp]
01. //3d11-1 最优二叉搜索树 动态规划加速原理 四边形不等式
02. #include "stdafx.h"
03. #include <iostream>
04. using namespace std;
05.
06. const int N = 3;
07.
08. void OptimalBinarySearchTree(double a[], double b[], int n, double **m, int **s, double **w);
09. void Traceback(int n, int i, int j, int **s, int f, char ch);
10.
11. int main()
12. {
13.     double a[] = {0.15, 0.1, 0.05, 0.05};
14.     double b[] = {0.00, 0.5, 0.1, 0.05};
15.
16.     cout<<"有序集的概率分布为: "<<endl;
17.     for(int i=0; i<N+1; i++)
18.     {
19.         cout<<"a"<<i<<"="<<a[i]<<","<<b"<<i<<"="<<b[i]<<endl;
20.     }
21.
22.     double **m = new double *[N+2];

```

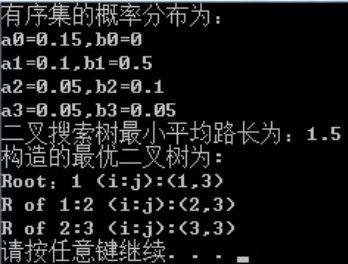
```

23.     int **s = new int *[N+2];
24.     double **w = new double *[N+2];
25.
26.     for(int i=0; i<N+2; i++)
27.     {
28.         m[i] = new double[N+2];
29.         s[i] = new int[N+2];
30.         w[i] = new double[N+2];
31.     }
32.
33.     OptimalBinarySearchTree(a,b,N,m,s,w);
34.     cout<<"二叉搜索树最小平均路长为: "<<m[1][N]<<endl;
35.     cout<<"构造的最优二叉树为:"<<endl;
36.     Traceback(N,1,N,s,0,'0');
37.
38.     for(int i=0; i<N+2; i++)
39.     {
40.         delete m[i];
41.         delete s[i];
42.         delete w[i];
43.     }
44.     delete[] m;
45.     delete[] s;
46.     delete[] w;
47.     return 0;
48. }
49.
50. void OptimalBinarySearchTree(double a[],double b[],int n,double **m,int **s,double **w)
51. {
52.     //初始化构造无内部节点的情况
53.     for(int i=0; i<=n; i++)
54.     {
55.         w[i+1][i] = a[i];
56.         m[i+1][i] = 0;
57.         s[i+1][i] = 0;
58.     }
59.
60.     for(int r=0; r<n; r++)//r代表起止下标的差
61.     {
62.         for(int i=1; i<=n-r; i++)//i为起始元素下标
63.         {
64.             int j = i+r;//j为终止元素下标
65.             int i1 = s[i][j-1]>i?s[i][j-1]:i;
66.             int j1 = s[i+1][j]>i?s[i+1][j]:j;
67.
68.             //构造T[i][j] 填写w[i][j],m[i][j],s[i][j]
69.             //首选i作为根, 其左子树为空, 右子树为节点
70.             w[i][j]=w[i][j-1]+a[j]+b[j];
71.             m[i][j]=m[i][i1-1]+m[i1+1][j];
72.             s[i][j]=i1;
73.
74.             //不选i作为根, 设k为其根, 则k=i+1, ...,j
75.             //左子树为节点: i,i+1,...,k-1,右子树为节点: k+1,k+2,...,j
76.             for(int k=i+1; k<=j1; k++)
77.             {
78.                 double t = m[i][k-1]+m[k+1][j];
79.
80.                 if(t<m[i][j])
81.                 {
82.                     m[i][j]=t;
83.                     s[i][j]=k;//根节点元素
84.                 }
85.             }
86.             m[i][j]+=w[i][j];
87.         }
88.     }
89. }
90.
91. void Traceback(int n,int i,int j,int **s,int f,char ch)
92. {
93.     int k=s[i][j];
94.     if(k>0)
95.     {
96.         if(f==0)
97.         {
98.             //根
99.             cout<<"Root: "<<k<<" (i:j):("<i<<","<j<<")<<endl;
100.        }
101.        else

```

```
102.         {
103.             //子树
104.             cout<<ch<<" of "<<f<<":"<<k<<" (i:j):("<i<<","<j<<")"<<endl;
105.         }
106.
107.         int t = k-1;
108.         if(t>i && t<=n)
109.         {
110.             //回溯左子树
111.             Traceback(n,i,t,s,k,'L');
112.         }
113.         t=k+1;
114.         if(t<=j)
115.         {
116.             //回溯右子树
117.             Traceback(n,t,j,s,k,'R');
118.         }
119.     }
120. }
```

运行结果如图：



上一篇 0019算法笔记——【动态规划】0-1背包问题
下一篇 0021算法笔记——【贪心算法】贪心算法与活动安排问题

顶 4 踩 0


主题推荐 动态规划 算法 搜索 namespace 结构

猜你在找


- | | |
|--|---------------------------|
| 0009算法笔记动态规划动态规划与斐波那契数列问题最 | 动态规划入门之硬币问题 |
| 最少硬币找零问题-动态规划 | KMP算法原理与实现精简 |
| 手把手实现红黑树 | 拓扑排序 |
| Machine Learning——LMS 算法 | 数据结构学习笔记 —— 队列的应用举例离散事件模拟 |
| T0J 1959 P0J 1562 Oil Deposits BFS DFS入门题 C语 | 使用UMeng微博授权失败报错 |

查看评论

2楼 share_help 2014-06-20 23:15发表

 请问为什么上述的动态最优二叉搜索树的结果不是你给出的那样呢？

1楼 mark_blue 2013-10-07 20:35发表


$$Pb(n)=1 \times p_1 + 2 \times p_3 + 3 \times p_2 + 1 \times q_0 + 3 \times (q_2 + q_3) = 1 \times 0.5 + 2 \times 0.05 + 3 \times 0.1 + 1 \times 0.15 + 2 \times 0.05 + 3 \times (0.05 + 0.05) = 1.6$$

查找树 b图中为什么外只算了3个外接点的查找概率？

Re: [share_help](#) 2014-06-20 23:18发表



请问为什么你上述的动态最优二叉搜索树的算法在运行到一半就会被强制性的关闭，这要如何解决？

Re: [风仲达](#) 2013-10-15 10:01发表



回复Revivedsun：谢谢指出，已改正。

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