liufeng_king的专栏

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 10:37 3339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个顶点中存储的元素，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

$$a_i \ge 0, 0 \le i \le n; b_j \ge 0, 1 \le j \le n; \sum_{i=0}^{n} a_i + \sum_{j=1}^{n} b_j = 1$$

集合{$a_0,b_1,a_1,\ldots\ldots b_n,a_n$}称为集合S的存取概率分布。

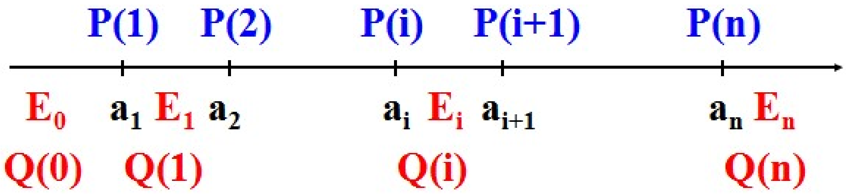
最优二叉搜索树：在一个表示S的二叉树T中，设存储元素x_i的结点深度为c_i；叶结点（x_j，x_{j+1}）的结点深度为d_j。

$$p = \sum_{i=1}^{n} b_i(1+c_i) + \sum_{j=0}^{n} a_j d_j$$

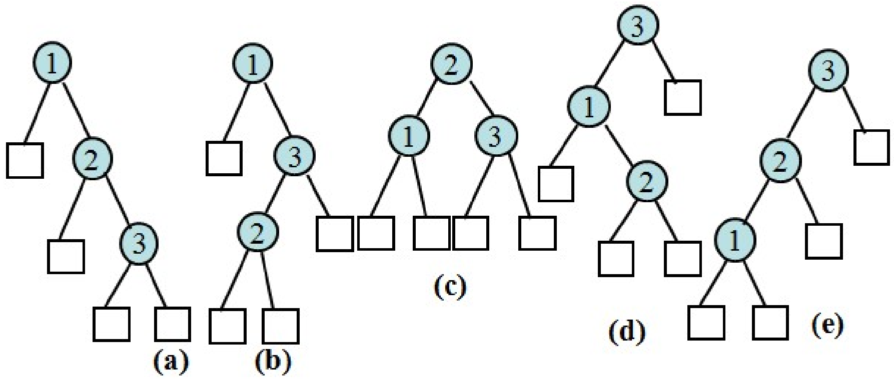
注：在检索过程中，每进行一次比较，就进入下面一层，对于成功的检索，比较的次数就是所在的层数加***1***。对于不成功的检索，被检索的关键码属于那个外部结点代表的可能关键码集合，比较次数就等于此外部结点的层数。对于图的内结点而言，第***0***层需要比较操作次数为***1***，第***1***层需要比较***2***次，第***2***层需要***3***次。

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($a_0,b_1,a_1,\ldots\ldots b_n,a_n$)，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Pi是对ai检索的概率。设qi是对满足ai<X<ai+1,0<=i<=n的标识符X检索的概率，（假定a0=-—∞且an+1=+ ∞)。



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P1=0.5, P2=0.1, P3=0.05,q0=0.15, q1=0.1, q2=0.05, q3=0.05，求每棵树的平均比较次数（成本）。

$P_{a(n)}$=1 × p1 + 2 × p2+3 × p3 + 1×q0 +2×q1+ 3×(q2 + q3) =1 × 0.5+ 2 × 0.1+3 ×0.05 + 1×0.05 +2×0.1+ 3×(0.05 + 0.05) =1.5

$P_{b(n)}$=1 × p1 + 2 × p3+3 × p2 + 1×q0 +2×q3 + 3×(q1 + q2) =1 × 0.5+ 2 × 0.05 + 3 ×0.1 + 1×0.15 +2×0.05+ 3×(0.1 + 0.05) =1.6

$P_{c(n)}$=1 × p2 + 2 × (p1 + p3) + 2×(q0 +q1 +q2 + q3) =1 × 0.1+ 2 × (0.5 + 0.05) + 2×(0.15 + 0.1 + 0.05 + 0.05) =1.9

$P_{d(n)}$=1 × p3 + 2 × p1+3 × p2 + 1 × q3+2 × q0 +3 × (q1+ q2) =1 × 0.05 + 2 × 0.5 + 3 × 0.1 + 1×0.05 + 2 × 0.15 + 3 × (0.1 + 0.05) =2.15

$P_{e(n)}$=1 × p3 + 2 × p2+3 × p1 + 1 × q3+2 × q2 +3 × (q0 + q1) =1 × 0.05 + 2 × 0.1+ 3 × 0.5 + 1×0.05 + 2 × 0.15 + 3 × (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, …, k-1 和a_0, a_1, …, a_{k-1} 都将位于左子树 L 上，其余结点 (k+1, …, n 和 a_k, a_{k+1}, …, 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) + …… 。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 , ⋯ , x_j和叶顶点(x_{i-1} , x_i) , ⋯ , (x_j , x_{j+1})的子树可以看作是有序集{ x_i , ⋯ , 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：

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

。{x_i , ⋯ , x_j}的存储概率分布为{a_{i-1}, b_i, …, b_j, a_j }，其中，a_h，b_k分别是下面的条件概率：

$$\bar{b}_k = b_k / w_{ij},\quad i\le k\le j;\qquad \bar{a}_h = a_h / w_{ij},\quad i-1\le h\le j$$

。

设T_{ij}是有序集$\{x_i, \cdots, x_j\}$关于存储概率分布为$\{a_{i-1}, b_i, \ldots, b_j, a_j\}$的一棵最优二叉搜索树，其平均路长为$p_{ij}$，$T_{ij}$的根顶点存储的元素$x_m$，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：

左子树的搜索概率

右子树的搜索概率

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

由于T_l是关于集合$\{x_i, \cdots, x_{m-1}\}$的一棵二叉搜索树，故$P_l >= P_{i,m-1}$。若$Pl > P_{i,m-1}$，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：

$$w_{ij} p_{ij} = w_{ij} + \min_{i \le k \le j} \{ w_{i,k-1} p_{i,k-1} + w_{k+1,j} p_{k+1,j} \}, \quad i \le j$$

初始时：

$$p_{i,i-1} = 0, \quad 1 \le i \le n$$

记 $w_{i,j}\ p_{i,j}$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：

$$m(i,j) = w_{ij} + \min_{i \le k \le j} \{ m(i,k-1) + m(k+1,j) \}, \quad i \le j$$

$$m(i,i-1) = 0, \quad i = 1,2,\cdots,n$$

4、求解过程：

1)没有内部节点时，构造T[1][0],T[2][1],T[3][2]……，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]…, 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…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] , …， T[n][n]，
1 T[1][2], T[2][3], …，T[n-1][n]，
2 T[1][3], T[2][4], …，T[n-2][n]，
……
r T[1][r+1], T[2][r+2], …，T[i][i+r]，…，T[n-r][n]
……
n-1 T[1][n]

具体代码如下：

[cpp]

```cpp
//3d11-1 最优二叉搜索树 动态规划
#include "stdafx.h"
#include <iostream>
using namespace std;

const int N = 3;

void OptimalBinarySearchTree(double a[],double b[],int n,double **m,int **s,double **w);
void Traceback(int n,int i,int j,int **s,int f,char ch);
```

```
 
int main()
{
    double a[] = {0.15,0.1,0.05,0.05};
    double b[] = {0.00,0.5,0.1,0.05};

    cout<<"有序集的概率分布为: "<<endl;
    for(int i=0; i<N+1; i++)
    {
        cout<<"a"<<i<<"="<<a[i]<<",b"<<i<<"="<<b[i]<<endl;
    }

    double **m = new double *[N+2];
    int **s = new int *[N+2];
    double **w =new double *[N+2];

    for(int i=0;i<N+2;i++)
    {
        m[i] = new double[N+2];
        s[i] = new int[N+2];
        w[i] = new double[N+2];
    }

    OptimalBinarySearchTree(a,b,N,m,s,w);
    cout<<"二叉搜索树最小平均路长为: "<<m[1][N]<<endl;
    cout<<"构造的最优二叉树为:"<<endl;
    Traceback(N,1,N,s,0,'0');

    for(int i=0;i<N+2;i++)
    {
        delete m[i];
        delete s[i];
        delete w[i];
    }
    delete[] m;
    delete[] s;
    delete[] w;
    return 0;
}

void OptimalBinarySearchTree(double a[],double b[],int n,double **m,int **s,double **w)
{
    //初始化构造无内部节点的情况
    for(int i=0; i<=n; i++)
    {
        w[i+1][i] = a[i];
        m[i+1][i] = 0;
    }

    for(int r=0; r<n; r++)//r代表起止下标的差
    {
        for(int i=1; i<=n-r; i++)//i为起始元素下标
        {
            int j = i+r;//j为终止元素下标

            //构造T[i][j] 填写w[i][j],m[i][j],s[i][j]
            //首选i作为根，其左子树为空，右子树为节点
            w[i][j]=w[i][j-1]+a[j]+b[j];
            m[i][j]=m[i+1][j];
            s[i][j]=i;

            //不选i作为根，设k为其根，则k=i+1，……j
            //左子树为节点：i,i+1……k-1,右子树为节点：k+1,k+2,……j
            for(int k=i+1; k<=j; k++)
            {
                double t = m[i][k-1]+m[k+1][j];

                if(t<m[i][j])
                {
                    m[i][j]=t;
                    s[i][j]=k;//根节点元素
                }
            }
            m[i][j]+=w[i][j];
        }
    }
}

void Traceback(int n,int i,int j,int **s,int f,char ch)
```

```cpp
{
    int k=s[i][j];
    if(k>0)
    {
        if(f==0)
        {
            //根
            cout<<"Root: "<<k<<" (i:j):("<<i<<","<<j<<")"<<endl;
        }
        else
        {
            //子树
            cout<<ch<<" of "<<f<<":"<<k<<" (i:j):("<<i<<","<<j<<")"<<endl;
        }

        int t = k-1;
        if(t>=i && t<=n)
        {
            //回溯左子树
            Traceback(n,i,t,s,k,'L');
        }
        t=k+1;
        if(t<=j)
        {
            //回溯右子树
            Traceback(n,t,j,s,k,'R');
        }
    }
}
```

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(n^2)。算法的主要计算量在于计算 $w_{ij}+\min_{i\le k\le j}\{m(i,k-1)+m(k+1,j)\}$ 。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：$m(i,j)=\min_{s(i,j-1)\le k\le s(i+1,j)}\{m(i,k-1)+w(i,k-1)+m(k+1,j)+w(k+1,j)\},i<j$ 而此状态转移方程的时间复杂度为O(n^2)。由此，对算法改进后的代码如下：

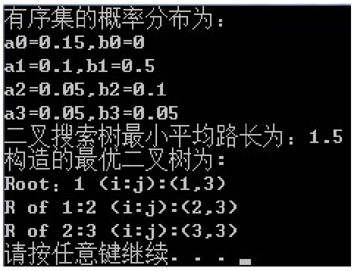
[cpp]

```cpp
//3d11-1 最优二叉搜索树 动态规划加速原理 四边形不等式
#include "stdafx.h"
#include <iostream>
using namespace std;

const int N = 3;

void OptimalBinarySearchTree(double a[],double b[],int n,double **m,int **s,double **w);
void Traceback(int n,int i,int j,int **s,int f,char ch);

int main()
{
    double a[] = {0.15,0.1,0.05,0.05};
    double b[] = {0.00,0.5,0.1,0.05};

    cout<<"有序集的概率分布为: "<<endl;
    for(int i=0; i<N+1; i++)
    {
        cout<<"a"<<i<<"="<<a[i]<<",b"<<i<<"="<<b[i]<<endl;
    }

    double **m = new double *[N+2];
```

```
    int **s = new int *[N+2];
    double **w =new double *[N+2];

    for(int i=0;i<N+2;i++)
    {
        m[i] = new double[N+2];
        s[i] = new int[N+2];
        w[i] = new double[N+2];
    }

    OptimalBinarySearchTree(a,b,N,m,s,w);
    cout<<"二叉搜索树最小平均路长为: "<<m[1][N]<<endl;
    cout<<"构造的最优二叉树为:"<<endl;
    Traceback(N,1,N,s,0,'0');

    for(int i=0;i<N+2;i++)
    {
        delete m[i];
        delete s[i];
        delete w[i];
    }
    delete[] m;
    delete[] s;
    delete[] w;
    return 0;
}

void OptimalBinarySearchTree(double a[],double b[],int n,double **m,int **s,double **w)
{
    //初始化构造无内部节点的情况
    for(int i=0; i<=n; i++)
    {
        w[i+1][i] = a[i];
        m[i+1][i] = 0;
        s[i+1][i] = 0;
    }

    for(int r=0; r<n; r++)//r代表起止下标的差
    {
        for(int i=1; i<=n-r; i++)//i为起始元素下标
        {
            int j = i+r;//j为终止元素下标
            int i1 = s[i][j-1]>i?s[i][j-1]:i;
            int j1 = s[i+1][j]>i?s[i+1][j]:j;

            //构造T[i][j] 填写w[i][j],m[i][j],s[i][j]
            //首选i作为根，其左子树为空，右子树为节点
            w[i][j]=w[i][j-1]+a[j]+b[j];
            m[i][j]=m[i][i1-1]+m[i1+1][j];
            s[i][j]=i1;

            //不选i作为根，设k为其根，则k=i+1，……j
            //左子树为节点：i,i+1……k-1,右子树为节点：k+1,k+2,……j
            for(int k=i1+1; k<=j1; k++)
            {
                double t = m[i][k-1]+m[k+1][j];

                if(t<m[i][j])
                {
                    m[i][j]=t;
                    s[i][j]=k;//根节点元素
                }
            }
            m[i][j]+=w[i][j];
        }
    }
}

void Traceback(int n,int i,int j,int **s,int f,char ch)
{
    int k=s[i][j];
    if(k>0)
    {
        if(f==0)
        {
            //根
            cout<<"Root: "<<k<<" (i:j):("<<i<<","<<j<<")"<<endl;
        }
        else
```

```
        {
            //子树
            cout<<ch<<" of "<<f<<":"<<k<<" (i:j):("<<i<<","<<j<<")"<<endl;
        }

        int t = k-1;
        if(t>=i && t<=n)
        {
            //回溯左子树
            Traceback(n,i,t,s,k,'L');
        }
        t=k+1;
        if(t<=j)
        {
            //回溯右子树
            Traceback(n,t,j,s,k,'R');
        }
    }
}
```

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

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 × p1 + 2 × p3+3 × p2 + 1×q0 + 3×(q2 + q3) =1 × 0.5+ 2 × 0.05 + 3 ×0.1 + 1×0.15 +2×0.05+ 3×(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