算法设计与分析

第3章 动态规划
(4)

谢晓芹
哈尔滨工程大学计算机科学与技术学院

学习要点

- 理解动态规划算法的概念。
- 掌握动态规划算法的基本要素

  （1）最优子结构性质

  （2）重叠子问题性质
- 掌握设计动态规划算法的步骤。

  （1）找出最优解的性质，并刻划其结构特征。

  （2）递归地定义最优值。

  （3）以自底向上的方式计算出最优值。

  （4）根据计算最优值时得到的信息，构造最优解。

学习要点

- 通过应用范例学习动态规划算法设计策略

（1）矩阵连乘问题

（2）最长公共子序列

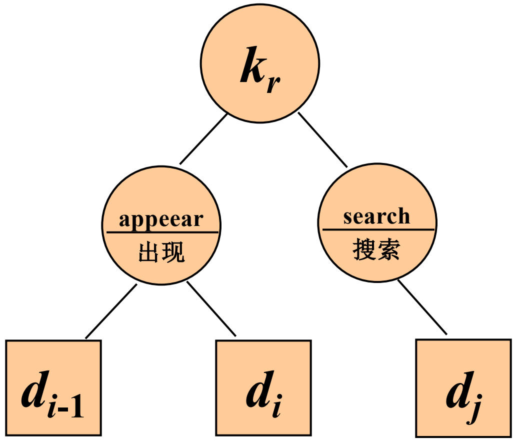
（3）0/1背包问题

（4）凸多边形最优三角剖分

（5）最优二叉搜索树

最优二叉搜索树-引言

- 设计程序将英文文档翻译成中文
  - 例如自动翻译机



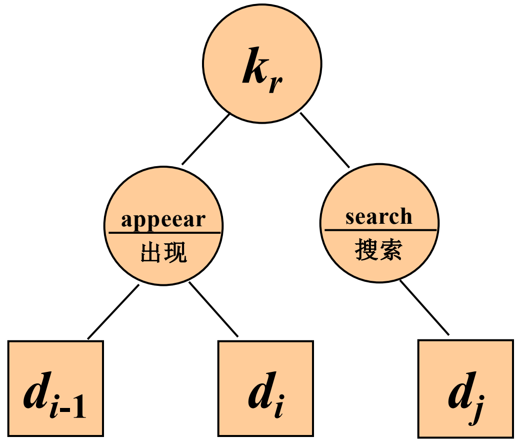
最优二叉搜索树-引言

- 自动翻译机
  - 使用线性表操作每次运行需要O(n)搜索时间

生词表	
字段1	字段2
aggregate analysis	综合分析;总体分析
amortized	分期,分摊
arbitrary	任意的, 武断的, 独裁的, 专断的
auxiliary	辅助的, 补助的
binomial	二项的, 二项式的
bog	沼泽, 陷于泥沼
contemporary	当代的, 同时代的
convention	归约, 常规
crucial	至关紧要的
disjoint	不相交的,不相接的,分离的
distinct	清楚的, 明显的, 截然不同的, 独特的

最优二叉搜索树-引言

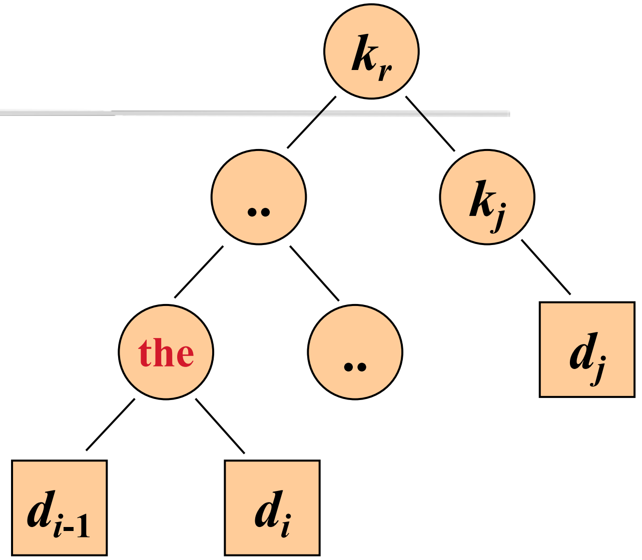
- 建立一棵二叉搜索树
  - n 个英文词作为 keys
  - 中文翻译作为从属数据（ satellite data ）
- 查找操作:
  - 对于任何一个单词的搜索，使用二分搜索法的时间为O(lg n)
  - 对于文章中出现的每个单词，都需要搜索该二叉树，如何使得总的搜索次数最少？



生词表

字段1	字段2
aggregate analysis	综合分析;总体分析
amortized	分期,分摊
arbitrary	任意的, 武断的, 独裁的, 专断的
auxiliary	辅助的, 补助的
binomial	二项的, 二项式的
…	…

最优二叉搜索树-引言



- 问题1：单词出现的频率不同

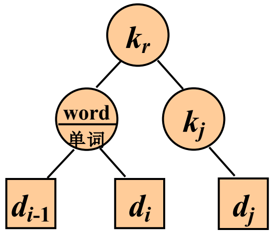
怎么办？

- 被访问的节点数 = 1+被搜索 key 的节点深度。(当在二叉树中搜索 key 时)
- 高频词出现在远离树根的节点，少用词在接近根的节点？
  - 减慢翻译速度

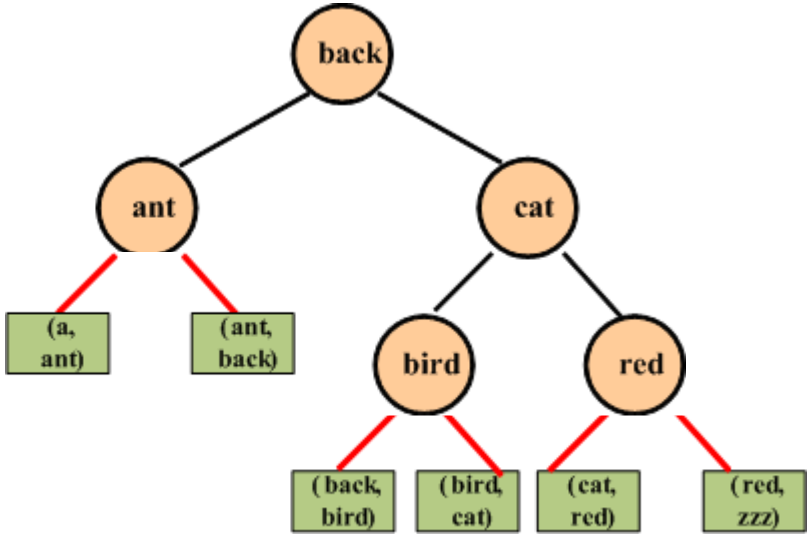
- 结论：希望高频出现的单词接近树根

最优二叉搜索树-引言

- 问题2：英语单词没有对应的汉语译文
  - 英语单词不出现在二叉树中

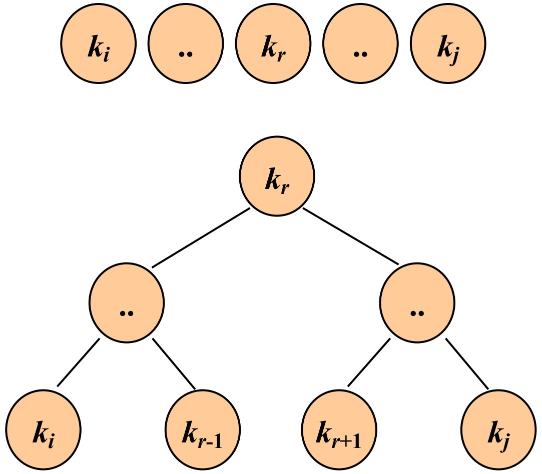


- 给出一个范围，或者近似的单词
  - 例如 {back,ant,cat,bird,red}
- 虚拟键d_i



最优二叉搜索树-引言

- 设已知每个单词出现的概率，如何组织一棵**二叉搜索树**，使得在**所有搜索**中，被访问的节点的总数**最少**？ 就能实现自动翻译器速度最快



最优二叉搜索树

- 二叉搜索树
  - 若它的左子树不空，则左子树上所有节点的值均小于它的根节点的值；
  - 若它的右子树不空，则右子树上所有节点的值均大于它的根节点的值；
  - 它的左、右子树也分别为二叉搜索树
  - 二叉搜索树叶节点是形如（ x_i, x_{i+1} ）的开区间，表示为虚拟键d_i

怎么建？

最优二叉搜索树

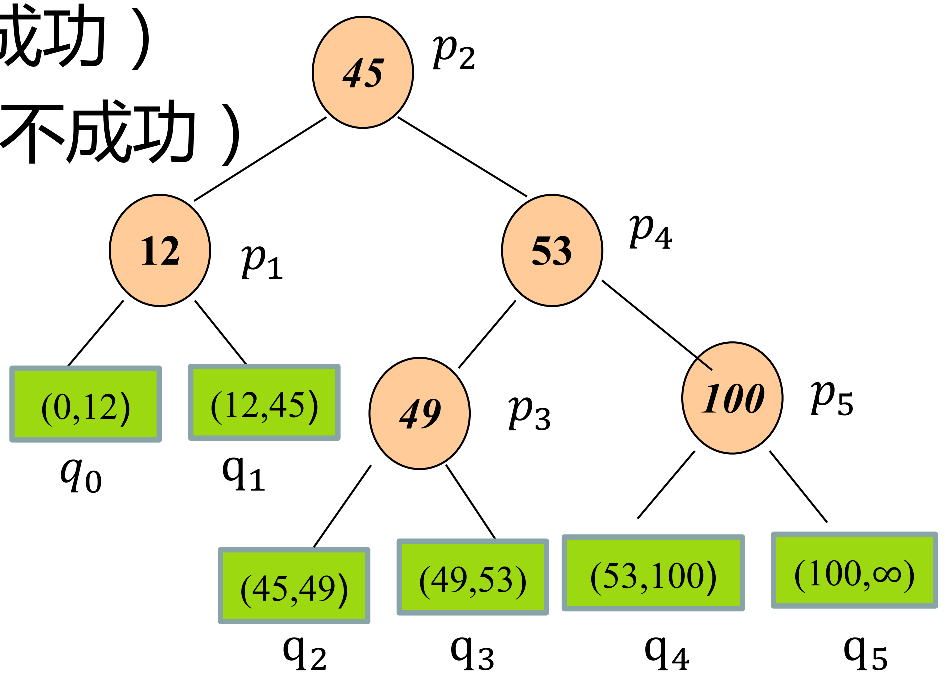
- S={x_1, x_2, …, x_n}是一个**有序**集, 用**二叉树**的结点来存储S中的元素，得到有序集S的二叉搜索树.

如果集合A上有序关系R，则称A为有序集

- 假设要翻译单词x，首先要在字典序列S={x_1, x_2, …, x_n}中寻找x
- 在二叉搜索树中找x，搜索结果有两种：
  - 在内部结点找到x= x_i, 概率为p_i　　（查找成功）
  - 在叶结点确定x∈(x_i, x_{i+1}), 概率为q_i（查找不成功）
  - 每次搜索要么成功，要么不成功, 有

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

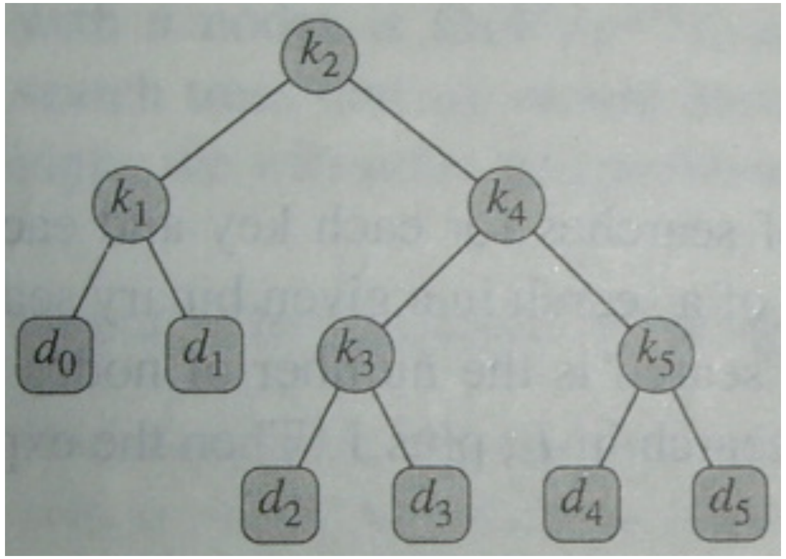
- (q_0,p_1,q_1,…,p_n,q_n)称为S的存取概率分布
  - 例如: S={45,12,53,49,100}

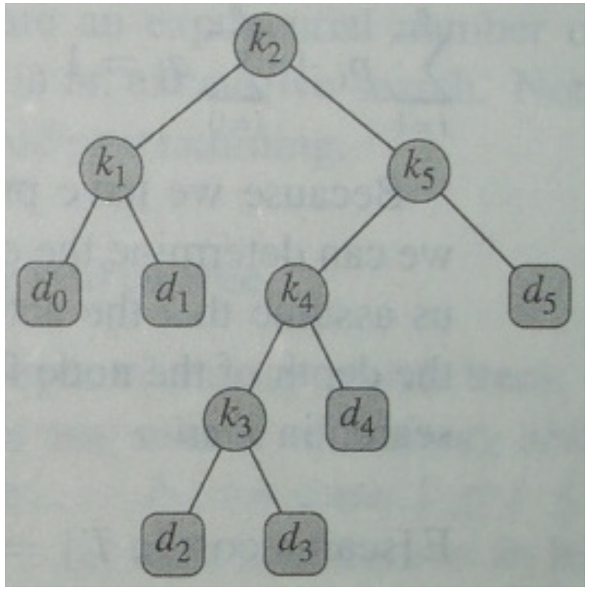


最优二叉搜索树

- 例，S =(k_1,k_2, k_3,k_4,k_5)的存取概率分布

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10





最优二叉搜索树

- 最优二叉搜索树问题: 对于有序集S及其存取概率分布(q_0,p_1, q_1,…, p_n, q_n), 在**所有**表示有序集S的**二叉搜索树**中找出一棵最优二叉搜索树BST（具有最小平均路长）
  - 输入： S=(k_1,k_2,…,k_n), (q_0,p_1, q_1,…, p_n, q_n);
  - 输出：最优二叉搜索树BST（具有最小平均路长）；

最优二叉搜索树

- 分析1：给定有序集$S = <k_1, k_2, ..., k_n>$ ，$(k_1 < k_2 < \cdots < k_n)$, **如何建立** BST?
  - 对每个key k_i , 搜索概率为p_i

    →对应一个**内结点**
  - 建立n+1个 “虚拟键” $d_0, d_1, ..., d_n$, 搜索概率为q_i

    →每个d_i对应建立一个**叶子结点**

    （对应不在S中的值）
    - d_0表示该值 $< k_1$；
    - d_n 表示该值$> k_n$
    - $1 \le i \le n-1$时, $k_i < d_i < k_{i+1}$.

快速排序

最优二叉搜索树

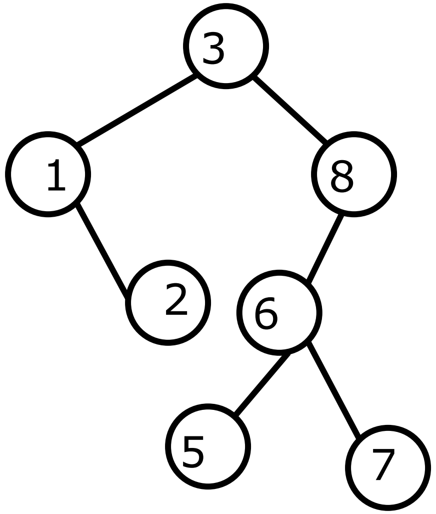
- 根节点？

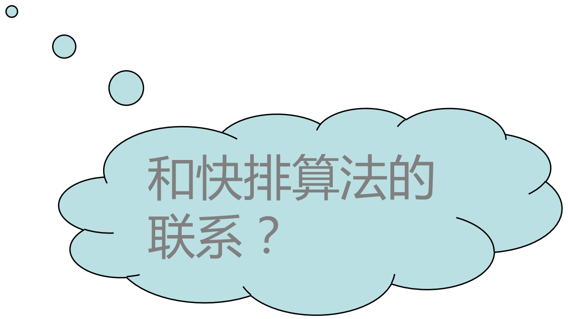
- 怎么插结点？

- eg：S={3, 1, 8, 2, 6, 7, 5}

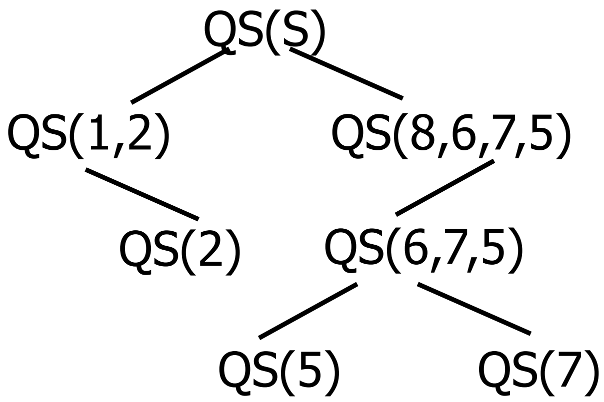
最优二叉搜索树

- 建立二叉搜索树：快排
  - 例如：无序S={3,1,8,2,6,7,5}





插入节点的关键：
递归调用快排QuickSort(S)

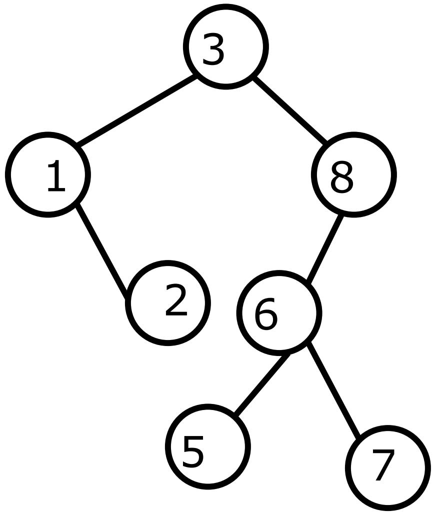


➢在随机的情况下，二叉搜索树的平均查找长度和 logn 是等数量级的

最优二叉搜索树

- 建立二叉搜索树：BSTSort()方法

```
BSTSort(A){
    T← ϕ;
    for i=1 to n
        do TreeInsert(T,S[i]);
    InOrder-TreeWalk(S);
}
```



- 分析：
  - BSTSort和快排：同样比较，不同顺序
  - 建立BST的时间：和快排运行时间近似
    - 最坏 $\Theta(n^2)$, 平均Θ(nlgn)
  - 随机化： RandomizedBSTSort()

最优二叉搜索树

- 分析2：如何求二叉搜索树的平均路长?
  - 在给定 BST内的一次搜索的期望代价=需比较的结点个数平均值
- 设一次搜索的实际代价为检查比较的结点个数，则二叉搜索树T的一次搜索的期望代价为
  - k_i结点， 深度为$depth_T(\mathrm{k}_i)$
  - 搜索成功的代价$depth_T(\mathrm{k}_i)+1$,不成功的代价$depth_T(\mathrm{d}_i)+1$

层数：查找结点个数

$$\mathrm{E}[\text{search cost in } T]=\sum_{i=1}^{n}(\mathrm{depth}_T(k_i)+1)\cdot p_i+\sum_{i=0}^{n}(\mathrm{depth}_T(d_i)+1)\cdot q_i$$

二叉搜索树的平均路长

$$=1+\sum_{i=1}^{n}\mathrm{depth}_T(k_i)\cdot p_i+\sum_{i=0}^{n}\mathrm{depth}_T(d_i)\cdot q_i\ ,$$

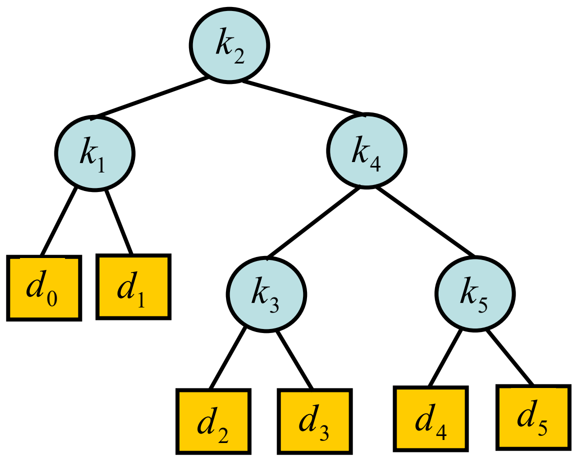
层数*搜索概率

其中depth_T () 表示一个节点在树T中的深度

快速排序T(n)=O(nlogn)， 二叉搜索树的平均查找长度和log n是等数量级的

最优二叉搜索树

- 可以逐个结点(node by node)计算期望的搜索代价:
- 例1



E[search cost in T]

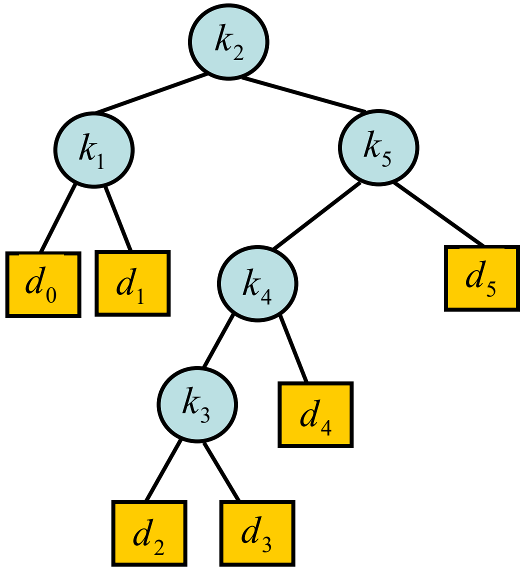
$$= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i)+1)\cdot q_i$$

$$= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i)\cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i)\cdot q_i$$

node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

最优二叉搜索树

- 例2



E[search cost in T]

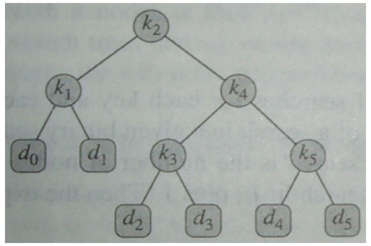
$$= \sum_{i=1}^{n} (\text{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i)+1)\cdot q_i$$

$$= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i)\cdot p_i + \sum_{i=0}^{n} \text{depth}_T(d_i)\cdot q_i$$

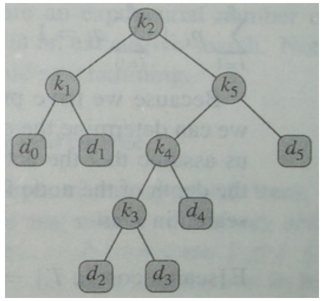
node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	3	0.05	0.20
k_4	2	0.10	0.30
k_5	1	0.20	0.40
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	4	0.05	0.25
d_3	4	0.05	0.25
d_4	3	0.05	0.20
d_5	2	0.10	0.30
Total			**2.75**

最优二叉搜索树

- 分析3：如何求最小路长的BST？
  - 关键：最优BST的特征？
  - 图 (b) 给出了一棵最优 BST，其期望代价为2.75
    - 不一定要求树的高度最小
    - 不一定将概率最大的 key 放在树根（例如：具有最大概率的k5在根节点时得到BST的最低代价为2.85)



(a) cost: 2.80



(b) cost: 2.75

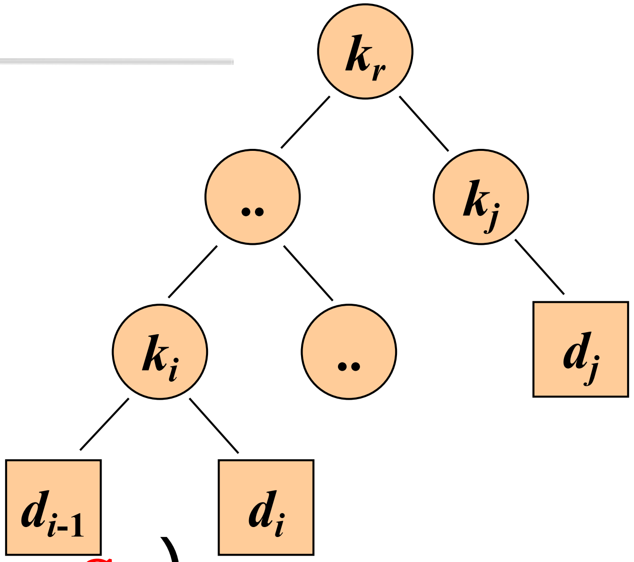
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

最优二叉搜索树

- 穷举检查所有的可能性的算法效率很低
  - 矩阵链乘法; LCS
- 为了构造 a BST，对具有 n 个节点的任何二叉树，将其节点分别标记为k_1, k_2, …, k_n然后附加 虚拟键作为叶子。则有 $\Omega(\boldsymbol{4^n/n^{3/2}})$ 种二叉树。
  - 穷举搜索法的时间复杂度为**指数级**
- 动态规划算法？

步骤1:最优子结构性质

- 分析: 考虑T的一棵**子树**
  - 结点（ $k_i,\ldots,k_j$ ）, $1\le i\le j\le n$
  - 叶子结点（虚拟键） $d_{i-1},\ldots,d_j$

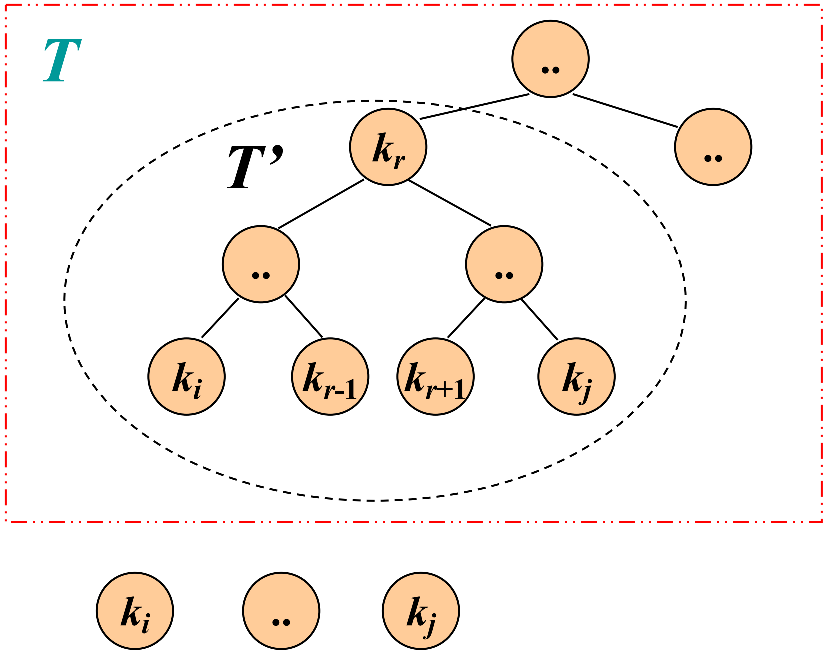


- 原问题: $S=(k_1,k_2,\ldots,k_n)$,存取概率$(q_0,p_1,q_1,\ldots,p_n,q_n)$
  - 最优解：二叉搜索树T, 内结点$k_1,\ldots,k_n$, 叶子结点$d_0,d_1,\ldots,d_n$
- 子问题：有序集$S=(k_i,\ldots,k_j)$以及存取概率分布$(q_{i-1},p_i,q_i,\ldots,p_j,q_j)$的一棵最优二叉搜索树
  - 最优解：二叉搜索树T'，内结点$k_i,\ldots,k_j$, 叶子结点$d_{i-1},d_i,\ldots,d_j$

是否具有最优子结构?

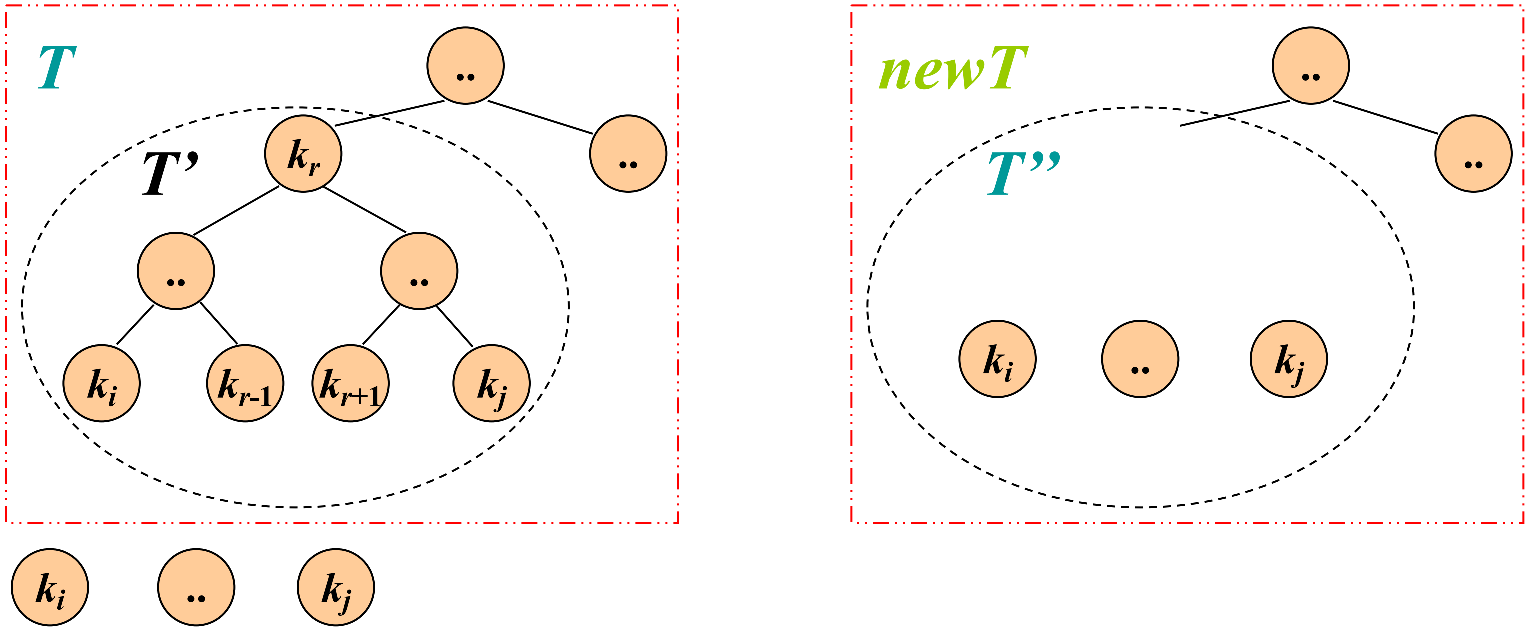
步骤1:最优子结构性质

- 最优子结构
  - 设原问题：$S=(k_1,\ldots k_n)$, ($q_0,p_1,q_1,\ldots,p_n,q_n$)的解（最优BST）为 T
  - 设 T' 为最优BST T的一个子树， T' 包含keys $k_i, \ldots, k_j$, 那么 T' 是子问题〔关于 $k_i, \ldots, k_j$ 和 $d_{i-1}, \ldots, d_j$〕的最优BST



步骤1:最优子结构性质

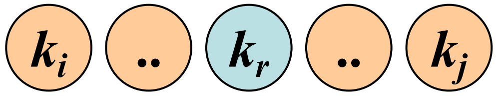
- 剪贴思想（Cut-and-paste）.
  - 假设T′ 不是最优。
  - 则设有一棵子树 *T″* ，其期望代价比 T′ 更少
  - cut *T′* out of *T* and paste in *T″* ，获得一棵期望代价比T还少的BST，和已知T是最优BST产生矛盾。



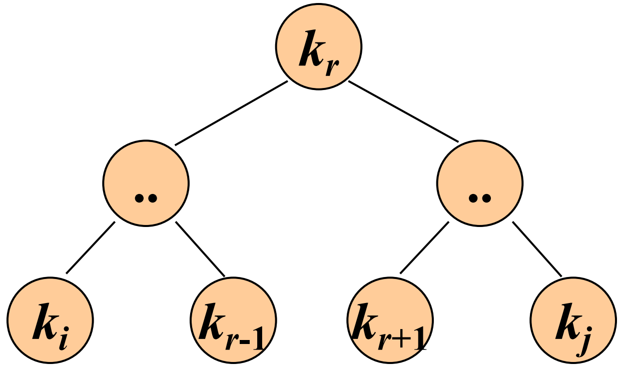
步骤1:最优子结构性质

- 使用最优子结构, 可以根据子问题的最优解来构造原问题的一个最优解.
- 如何求子问题的最优BST？
- 关键：确定BST的树根

Why?



- 分析：
  - 给定keys $k_i,\ldots,k_j$, 假设k_r ($i \le r \le j$)是最优子树的根
  - k_r左子树包括 k_i , ..., k_{r-1}和 d_{i-1} , ..., d_{r-1}
  - k_r右子树包括k_{r+1}, ..., k_j和d_r , ..., d_j.



矩阵链乘法的分割点k

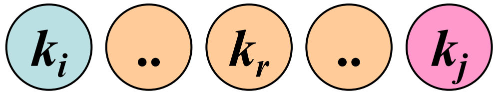
步骤1:最优子结构性质

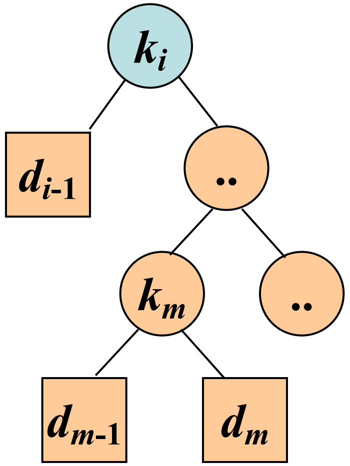
- 求解子问题最优BST方法
  - 检查所有可能的根节点k_r , i≤r≤ j, 检查包含$k_i, \dots, k_{r-1}$ 的所有最优BST和包含$k_{r+1}, \dots, k_j$的最优BST, 就可以找到一棵包含$k_i, \dots, k_j$ 的最优BST

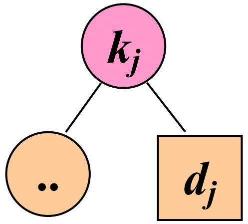
- 原问题求解方法
  - 检查所有的 k_r ，找到相应的子问题的最优BST，进而将找到原问题的最优BST

步骤1:最优子结构性质

- k_r在不同位置时的特殊情况分析
  - “empty”子树
  - 设有一棵子树包含keys（ $k_i, \ldots, k_j$ ）
    - 选择 k_i 作为根结点：
      - k_i 左子树不包含keys，但包含一个虚拟键 d_{i-1}.
    - 同样，选择 k_j 作为根结点：
      - k_j 右子树不包含keys，但包含一个虚拟键d_j.







步骤2: 递归计算最优值

- 求解策略：
  - 先求树最优平均路长，再求具体BST
  - 根据子问题的最优解来构造原问题的一个最优解

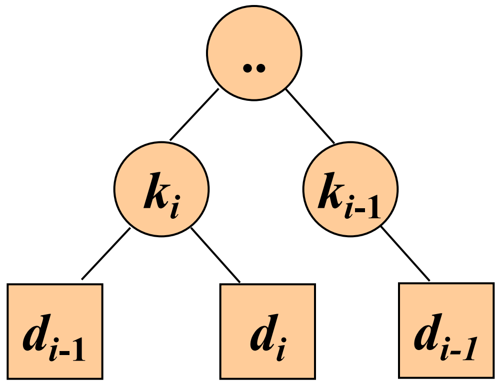
- 定义**e[i, j]**, 表示一棵包含keys = k_i, ..., k_j的最优BST的平均路长（期望代价）
- 原问题：计算e[1, n]

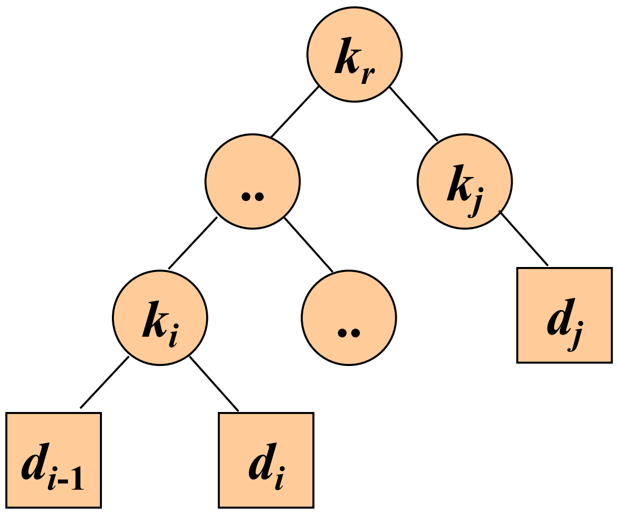
步骤2: 递归计算最优值

- 子问题：寻找包含有序集$\{k_i, ..., k_j\}$ 的最优BST, 其中 i≥1, j≤n, and j≥i-1

平均路长为e[i,j]

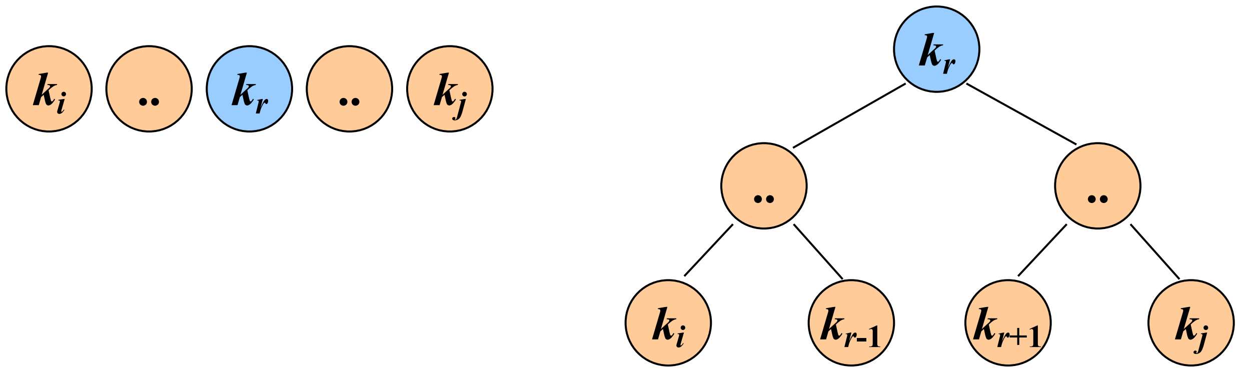
- 边界条件：当 j=i-1，没有实际key值
  - 只有虚拟键: d_{i-1}
  - e[i, i-1] = q_{i-1}
  - e[1,0]= q_0
  - e[n+1,n]= q_n
- When j≥i ?





步骤2: 递归计算最优值

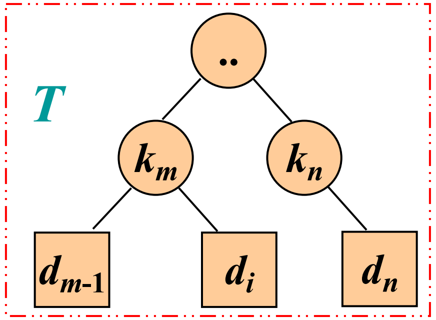
- 当j≥i,
  - 从k_i, …, k_j中选择 k_r 作为根结点,
  - 造一棵最优BST(包含k_i, …, k_{r-1})作为左子树
  - 造一棵最优BST (包含k_{r+1}, …, k_j) 作为右子树.



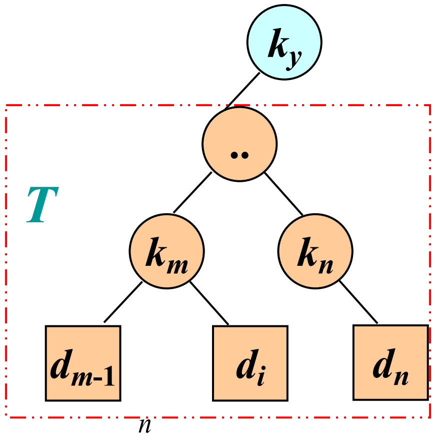
$$e[i,j] = p_r + e[i, r\text{-}1] + e[r+1, j] \ ?$$

步骤2: 递归计算最优值

- 子树的代价和原树的代价之间有何关系？
- 或：当一棵树成为一个结点的子树时,期望搜索代价如何变化?
  - 子树中每个节点的深度增加 1 ，该子树的期望搜索的 cost 的增量为该子树所有节点的搜索概率之和，增量为：



$$w[i,j]=\sum_{l=1}^{j} p_l + \sum_{l=i-1}^{j} q_l$$



$$\mathrm{E}_T = e[m,n]$$

$$\begin{aligned}\mathrm{E}_T &= \sum_{x=m}^{n}(\mathrm{depth}(k_x)+1\underline{+1)}\cdot p_i + \sum_{x=m\text{-}1}^{n}(\mathrm{depth}(d_x)+1\underline{+1)}\cdot q_x \\ &= \sum_{x=m}^{n}(\mathrm{depth}(k_x)+1)\cdot p_i + \sum_{x=m-1}^{n}(\mathrm{depth}(d_x)+1)\cdot q_x + \underline{\sum_{x=m}^{n} p_i + \sum_{x=m\text{-}1}^{n} q_x} \\ &= e[m,n]+w[m,n]\end{aligned}$$

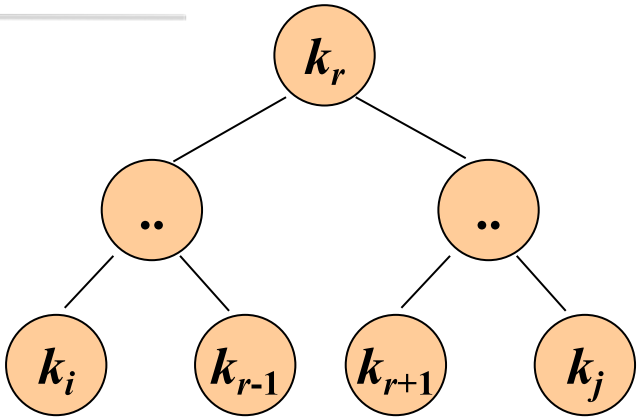
步骤2: 递归计算最优值

- 若 k_r 是最优子树 (包含keys $k_i, \ldots, k_j$)的根, 有
  - e[i, j] = p_r+(e[i, r-1]+w[i, r-1]) + (e[r+1, j]+w[r+1, j]) ?
  - 注意： w[i, j] = w[i, r-1] + p_r + w[r+1, j]

$$\left(w[i,r-1] = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l \quad , \quad w[r+1,j] = \sum_{l=r+1}^{j} p_l + \sum_{l=r}^{j} q_l \right)$$

  - 重写 e[i, j] 为：

    e[i, j] = e[i, r-1] + e[r+1, j] + w[i, j]

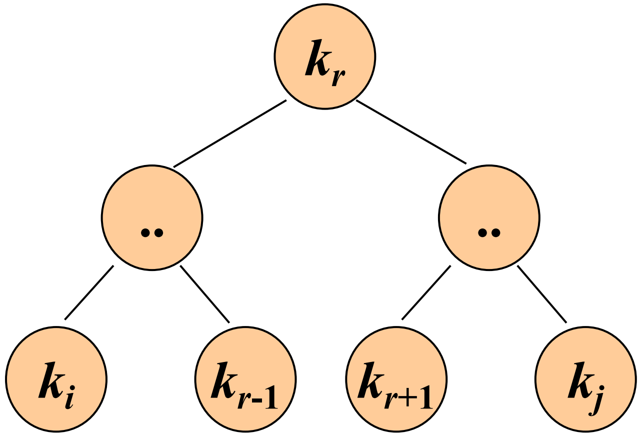


- 问题：该递归公式假设知道采用哪个结点 k_r 作为根

步骤2: 递归计算最优值

- 选择 k_r 作为具有最小代价的BST的根结点，有如下递归公式：

$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j. \end{cases}$$

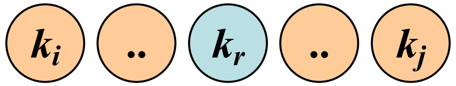


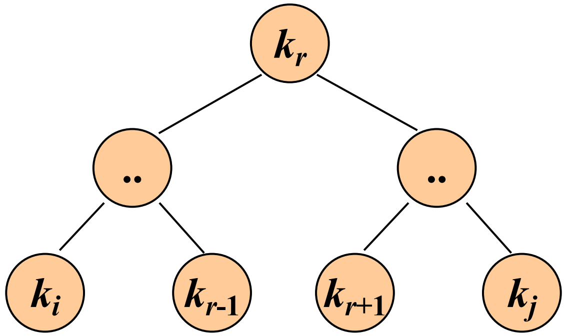
  - e[i, j] 是最优BST的期望代价（平均搜索路长）.
  - 为了记录最优BST的结构, 定义root[i, j] , for 1≤i≤j≤n , 保存索引r ， k_r 是包含 键值 k_i, ..., k_j 的最优BST树的根

步骤3:计算期望搜索代价

- 最优BST和矩阵连乘问题相似
  - 直接的递归实现效率不高

$$A_i .. A_r .. A_j$$





$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j} \{ e[i, r-1] + e[r+1, j] + w[i,j] \} & \text{if } i \le j. \end{cases}$$

  - 将 e[i, j]保存在表 e[1.. n+1, 0.. n] .
    - 第一个索引到n+1, 表示子树只包含d_n 的情况, 需要计算e[n+1, n]
    - 第二个索引从0开始, 表示子树只包含d_0 的情况, 需要计算e[1, 0] .
  - 使用表格root[i, j] , 记录包含键k_i, ..., k_j的子树的根结点， 1≤i≤j≤n

步骤3:计算期望搜索代价

- e[i, j] = e[i, r-1] + e[r+1, j] + w[i, j]
  - 无需每次计算 e[i, j] 时都计算 w[i, j]（耗时Θ(j-i) 次加法）：把值存在 table w[1.. n+1, 0.. n].
  - 边界情况, 计算w[i, i-1] = q_{i-1} for 1≤i≤n.
  - For j≥i,

$$w[i,j]=\sum_{l=i}^{j}p_l+\sum_{l=i-1}^{j}q_l=w[i,j-1]+p_j+q_j$$

- 因此, w[i, j] 计算要Θ(n^2) 复杂度
- 输入: 概率p_1, ..., p_n and q_0, ..., q_n 和大小 n
- 输出:表e和表root

步骤3:计算期望搜索代价

$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j. \end{cases} \qquad w[i,j]=\sum_{l=i}^{j}p_l+\sum_{l=i-1}^{j}q_l = w[i,j-1]+p_j+q_j$$

```
OPTIMAL-BST(p, q, n)
for i←1 to n+1
    do e[i, i-1]←q_{i-1}
       w[i, i-1]←q_{i-1}
for l←1 to n
    do for i←1 to n-l+1   // ?1
          do j←i+l-1      // ?2
             e[i, j]←∞
             w[i, j]←w[i, j-1]+p_j+q_j
             for r←i to j
                 do t←e[i, r-1]+e[r+1, j]+w[i, j]
                    if t<e[i, j]
                       then e[i, j]←t
                            root[i, j]←r
return e and root
```

二叉树的键的个数
(决定子问题的规模)

子问题的起始(i)
和终止(j)位置

?1

$k_1, k_2, \ldots, k_n$

$e[i, j]$:

***l* 个元素的 Opti-BST 的 cost**

$i = 1,\ j = l,$

$i = 2,\ j = l+1,$

…

$i = x,\ j = n,$

$n-x+1=l \Rightarrow x=n-l+1$

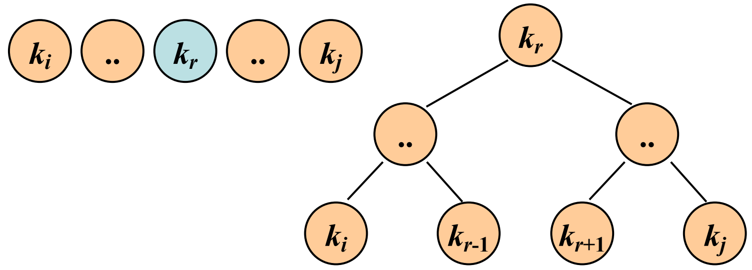
?2

$j-i+1 = i+l-1-i+1 = l$

步骤3:计算期望搜索代价

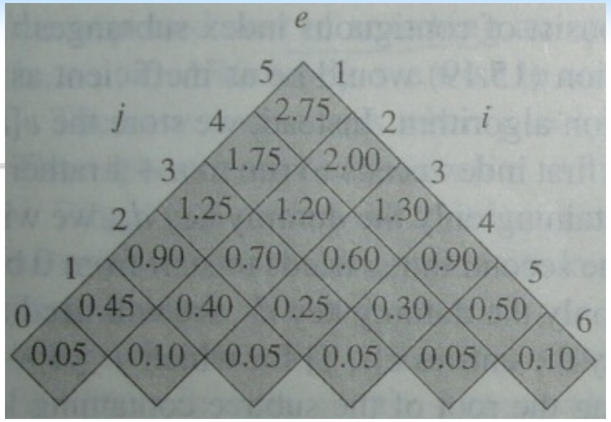
```
OPTIMAL-BST(p, q, n)
1  for i←1 to n+1
2      do e[i, i-1]←q_{i-1}
3         w[i, i-1]←q_{i-1}
4  for l←1 to n          // 求l 个元素的Opti-BST
5      do for i←1 to n-l+1
6             do j←i+l-1
7                e[i, j]←∞
8                w[i, j]←w[i, j-1]+p_j+q_j
9                for r←i to j
10                   do t←e[i, r-1]+e[r+1, j]+w[i, j]
11                      if t<e[i, j]
12                         then e[i, j]←t
13                              root[i, j]←r
14 return e and root
```

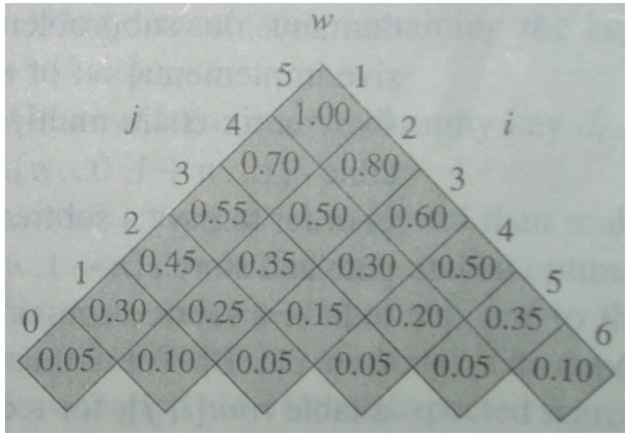
最内层循环, 9–13行, 对包含 k_i , ..., k_j 的最优 BST，尝试每一个 k_r 作为树根

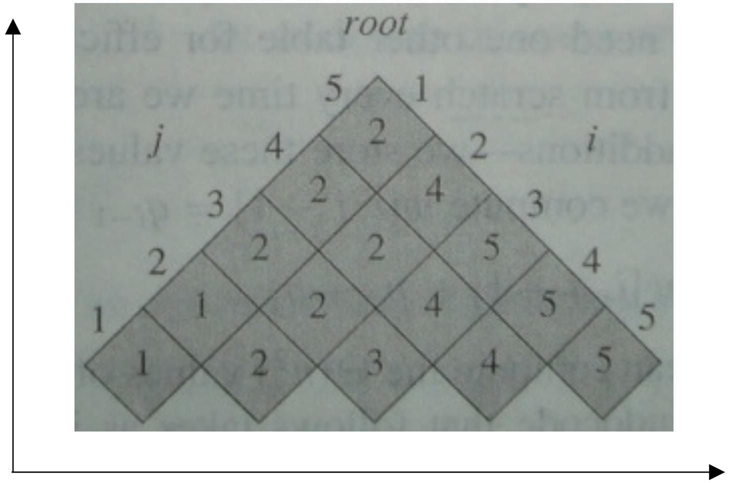


步骤3:计算期望搜索代价

```
OPTIMAL-BST(p, q, n)
for i←1 to n+1
    do e[i, i-1]←q_{i-1}
       w[i, i-1]←q_{i-1}
for l←1 to n
    do for i←1 to n-l+1
           do j←i+l-1
              e[i, j]←∞
              w[i, j]←w[i, j-1]+p_j+q_j
              for r←i to j
                  do t←e[i, r-1]+e[r+1, j]+w[i, j]
                     if t<e[i, j]
                        then e[i, j]←t
                             root[i, j]←r
return e and root
```







步骤3:计算期望搜索代价

```
OPTIMAL-BST(p, q, n)
for i←1 to n+1
    do e[i, i-1]←q_{i-1}
       w[i, i-1]←q_{i-1}
for l←1 to n
    do for i←1 to n-l+1    // n-l+1 times
           do j←i+l-1
              e[i, j]←∞
              w[i, j]←w[i, j-1]+p_j+q_j
              for r←i to j    // j-i+1=i+l-1-i+1=l
                  do t←e[i, r-1]+e[r+1, j]+w[i, j]
                     if t<e[i, j]
                        then e[i, j]←t
                             root[i, j]←r
return e and root
```

运行时间:$\Theta(n^3)$

分析:

$O(n^3)$: 三层嵌套循环，每层循环最多执行n次

$\Omega(n^3)$:

$$\sum_{l=1}^{n}(n-l+1)l=\sum_{l=1}^{n}(n+1)l-\sum_{l=1}^{n}l^2$$

$$=\frac{(n+1)(n+1)n}{2}-\frac{n(n+1)(2n+1)}{6}$$

$$=\frac{n(n+1)(n+2)}{6}=\Omega(n^3)$$

步骤4:构造最优解

- root(i,j)保存最优T(i,j)的根节点元素
- root(1,n)为所求BST的根节点元素
  - 左子树T(1,root(1,n)-1)的根为root(1,root(1,n)-1)
  - 右子树T(root(1,n)+1,n)的根为root(root(1,n)+1,n)

分治法与动态规划法的比较

	分治法	动态规划法
相同点	最优子结构，通过合并子问题的解得到原问题的解.	
不同点	把原问题划分为独立的子问题	子问题不独立，不同子问题共享相同的子子问题
	递归地解这些子问题	填表形式自底向上求解
	做很多不必要的工作, 重复求解公共子子问题	只求解公共子子问题一次，并用一张表来保存所有的答案,避免大量的重复计算.

动态规划法小结

- 动态规划法包含以下四步(CRCC).
  - 1.找出最优解的性质，并刻划其结构特征(**C**haracterize the structure of an optimal solution).
  - 2.递归地定义最优值(**R**ecursively define the value of an optimal solution).
  - 3.以自底向上的方式计算出最优值(**C**ompute the value of an optimal solution in a **Bottom-up** fashion).
  - 4.根据计算最优值时得到的信息，构造最优解(**C**onstruct an optimal solution from computed information).

动态规划法小结

- 理解动态规划算法的概念
- 难点： 掌握动态规划算法的基本要素

（1）最优子结构性质

（2）重叠子问题性质

- 重点：掌握设计动态规划算法的步骤

（1）找出最优解的性质，并刻划其结构特征。

（2）递归地定义最优值。

（3）以自底向上的方式计算出最优值。

（4）根据计算最优值时得到的信息，构造最优解。

动态规划法小结

- 算法的共性
  - 表格化
  - 自底向上
  - 基于递归式

动态规划法小结

- 动态规划算法常用于优化问题 (optimization problems).
- 优化问题
  - 有很多可能的解（ solutions ）
  - 每个解有一个值（value）
  - 需要找到具有最优值（最大或最小）的解
- 称问题的某个解为一个最优解，而不是单纯地称为最优解，因为可能有多个解能得出问题的最优值

动态规划的关键

- 动态规划的关键
  - 找出一个问题所包含的子问题及其表现形式

- 子问题的模式
  - 子问题是原问题的前缀
  - 子问题是原问题的中缀
  - 子问题是原问题的子树

动态规划的关键

- 子问题模式1
  - 原问题：$x_1,x_2,\ldots,x_n$
  - 子问题是： $x_1,x_2,\ldots,x_i$
  - 子问题个数：线性的
- 子问题模式2
  - 原问题：$x_1,x_2,\ldots,x_m$,和$y_1,y_2,\ldots,y_n$
  - 子问题是：$x_1,x_2,\ldots,x_i$和$x_1,x_2,\ldots,x_j$
  - 子问题个数：O(mn)

动态规划的关键

- 子问题模式3
  - 原问题：$x_1,x_2,\ldots,x_n$
  - 子问题是： $x_i,x_2,\ldots,x_j$
  - 子问题个数：$O(n^2)$
- 子问题模式4
  - 原问题：树
  - 子问题是：其子树
  - 若树有n个结点，它有多少个子问题呢？

思考题

- 最优二叉搜索树问题和矩阵连乘问题有什么不同？
- 提示：
  - 问题构成
  - 边界/边界值
  - 递归条件
  - 切分点范围

扩展题-子集和问题

- 子集和问题：给定整数集，问是否存在该整数集的一个子集，使得该子集元素的和为0.
  - 例如：整数集[-7,-4,-2, 6, 8]，存在子集[-4,-2, 6]，该子集和为0.
- 分析：
  - 枚举法：求出所有输入集的子集，逐一验证其和是否为0.
  - 子集个数：每个元素只有出现和不出现两种情况，故子集总数为2^n
  - 枚举方法：通过<1,1,1,1,1>来选择相应的子集