Introduction to Algorithms

Dynamic Programming

Some Properties

- OBST is one special kind of advanced tree.
- It focus on how to reduce the cost of the search of the BST.
- It may not have the lowest height !
- It needs 3 tables to record probabilities, cost, and root.

Premise

- It has n keys (representation $k_1,k_2,\ldots,k_n$) in sorted order (so that $k_1<k_2<\ldots<k_n$), and we wish to build a binary search tree from these keys. For each k_i ,we have a probability p_i that a search will be for k_i.
- In contrast of, some searches may be for values not in k_i, and so we also have n+1 "dummy keys" $d_0,d_1,\ldots,d_n$ representating not in k_i.
- In particular, d_0 represents all values less than k_1, and d_n represents all values greater than k_n, and for $i=1,2,\ldots,n-1$, the dummy key d_i represents all values between k_i and k_{i+1}.

∗ **The dummy keys are leaves (external nodes), and the data keys mean internal nodes.**

Formula & Prove

- The case of search are two situations, one is success, and the other, without saying, is failure.

- We can get the first statement :

$$(i=1\sim n)\underset{\substack{\uparrow\\ \textit{Success}}}{\sum} p_i + (i=0\sim n)\underset{\substack{\uparrow\\ \textit{Failure}}}{\sum} q_i = 1$$

- Because we have probabilities of searches for each key and each dummy key, we can determine the expected cost of a search in a given binary search tree T.
- Assume that the actual cost of a search is the number of nodes examined, i.e., the depth of the node found by the search in T, plus1. The expected cost of a search in T is:
- E[search cost in T]

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

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

Where depth_T denotes a node's depth in the tree T.

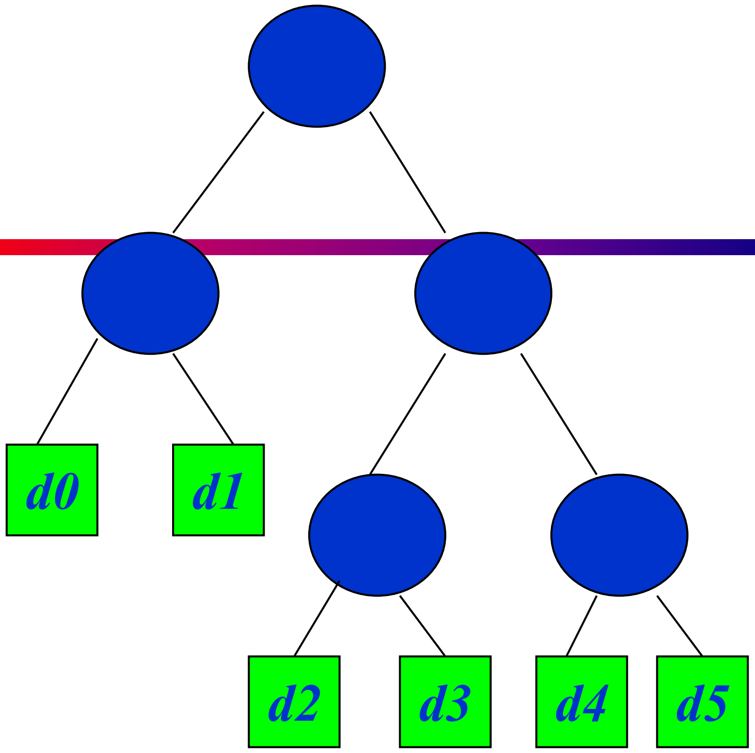


Figure (a)

i	0	1	2	3	4	5
pi		0.15	0.10	0.05	0.10	0.20
qi	0.05	0.10	0.05	0.05	0.05	0.10

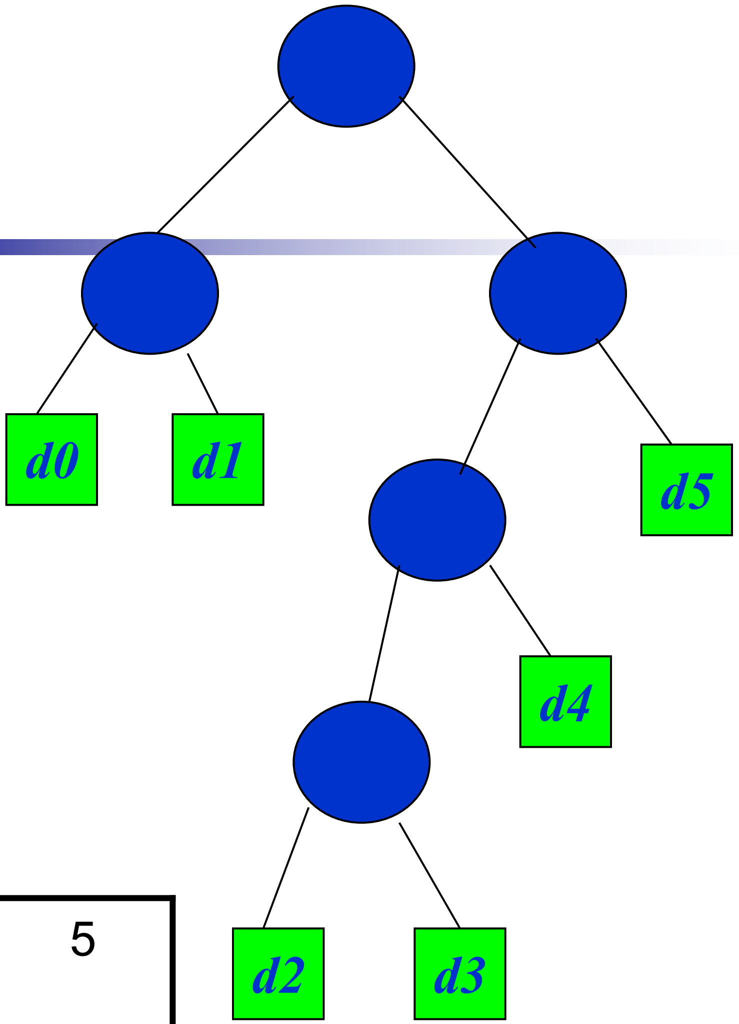


Figure (b)

- By Figure (a), we can calculate the expected search cost node by node:

Cost= Probability * (Depth+1)

Node#	Depth	probability	cost
k1	1	0.15	0.30
k2	0	0.10	0.10
k3	2	0.05	0.15
k4	1	0.10	0.20
K5	2	0.20	0.60
d0	2	0.05	0.15
d1	3	0.10	0.30
d2	3	0.05	0.20
d3	3	0.05	0.20
d4	3	0.05	0.20
d5	3	0.10	0.40

- And the total cost = (0.30 + 0.10 + 0.15 + 0.20 + 0.60 + 0.15 + 0.30 + 0.20 + 0.20 + 0.20 + 0.40) = 2.80
- So Figure (a) costs 2.80 ,on another, the Figure (b) costs 2.75, and that tree is really optimal.
- We can see that the height of (b) is more than (a) , and the key k5 has the greatest search probability of any key, yet the root of the OBST shown is k2.(The lowest expected cost of any BST with k5 at the root is 2.85)

Step1:The structure of an OBST

- To characterize the optimal substructure of OBST, we start with an observation about subtrees. Consider any subtree of a BST. It must contain keys in a contiguous range ki,…,kj, for some $1 \leqq i \leqq j \leqq n$.
- In addition, a subtree that contains keys ki,…,kj must also have as its leaves the dummy keys d_{i-1} ,…,d_j.

Optimal Substrucutre

- We need to use the optimal substructure to show that we can construct an optimal solution to the problem from optimal solutions to subproblems.
- Given keys k_i ,…, k_j, one of these keys, k_r (I ≦r ≦j), will be the root of an optimal subtree. The left subtree will contain the keys (k_i ,…, k_{r-1}) and the dummy keys(d_{i-1} ,…, d_{r-1}), and the right subtree will contain the keys (k_{r+1} ,…, k_j) and the dummy keys(d_r ,…, d_j).
- As long as we examine all candidate roots k_r, where i ≦r ≦j, and we determine all optimal binary search trees containing k_i ,…, k_{r-1} and those containing k_{r+1} ,…, k_j , we are guaranteed that we will find an OBST.

- There is one detail worth nothing about “empty” subtrees. Suppose that in a subtree with keys $k_i,\ldots,k_j$, we select k_i as the root. Its left subtree contains the keys $k_i,\ldots, k_{i-1}$. It is natural to interpret this sequence as containing no keys. It is easy to know that subtrees also contain dummy keys. The sequence has no actual keys but does contain the single dummy key d_{i-1}.
- Symmetrically, if we select k_j as the root, then k_j‘s right subtree contains the keys, $k_{j+1} \ldots,k_j$; this right subtree contains no actual keys, but it does contain the dummy key d_j.

Step2: A recursive solution

- We are ready to define the value of an optimal solution recursively. We pick our subproblem domain as finding an OBST containing the keys $k_i,\ldots,k_j$, where $i \geqq 1$, $j \leqq n$, and $j \geqq i-1$. (It is when $j=i-1$ that ther are no actual keys; we have just the dummy key d_{i-1}.)
- Let us define e[i,j] as the expected cost of searching an OBST containing the keys $k_i,\ldots, k_j$. Ultimately, we wish to compute e[1,n].

- The easy case occurs when $j=i-1$. Then we have just the dummy key d_{i-1}. The expected search cost is $e[i,i-1]= q_{i-1}$.
- When $j \geqq 1$, we need to select a root k_r from among $k_i,\ldots,k_j$ and then make an OBST with keys $k_i,\ldots,k_{r-1}$ its left subtree and an OBST with keys $k_{r+1},\ldots,k_j$ its right subtree. By the time, what happens to the expected search cost of a subtree when it becomes a subtree of a node? The answer is that the depth of each node in the subtree increases by 1.

- By the second statement, the excepted search cost of this subtree increases by the sum of all the probabilities in the subtree. For a subtree with keys $k_i,\ldots,k_j$ let us denote this sum of probabilities as

$\underline{w(i,j) = (l=i\sim j) \sum p_l} \; \underline{+ (l=i-1\sim j) \sum q_l}$

Thus, if k_r is the root of an optimal subtree containing keys $k_i,\ldots,k_j$, we have

$e[i,j]= p_r + (e[i,r-1]+w(i,r-1))+(e[r+1,j]+w(r+1,j))$

Note that $w(i, j) = w(i,r-1)+ p_r +w(r+1,j)$

- We rewrite e[i,j] as

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

  The recursive equation as above assumes that we know which node k_r to use as the root. We choose the root that gives the lowest expected search cost, giving us our final recursive formulation:

e[i,j]=

case1: if $i \leqq j, i \leqq r \leqq j$

$$e[i,j]=\min\{e[i,r-1]+e[r+1,j]+w(i,j)\}$$

case2: if j=i-1; e[i,j]= q_{i-1}

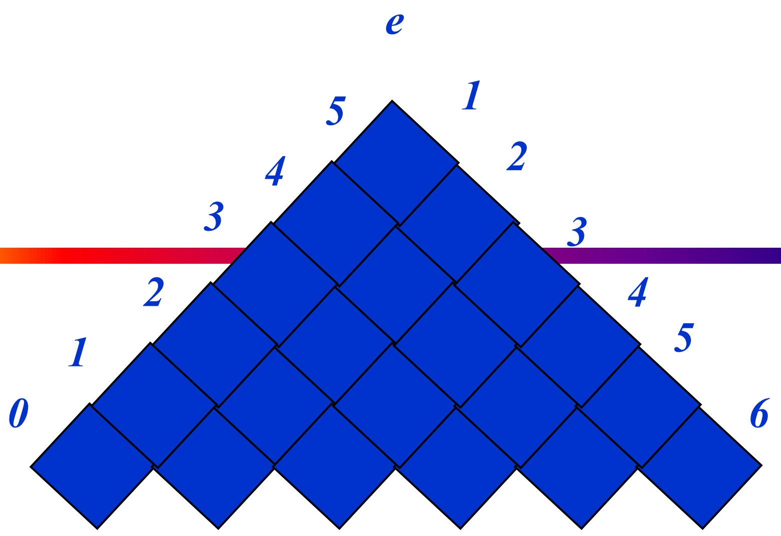
- The e[i,j] values give the expected search costs in OBST. To help us keep track of the structure of OBST, we define root[i,j], for $1 \leqq i \leqq j \leqq n$, to be the index r for which k_r is the root of an OBST containing keys $k_i,\ldots,k_j$.

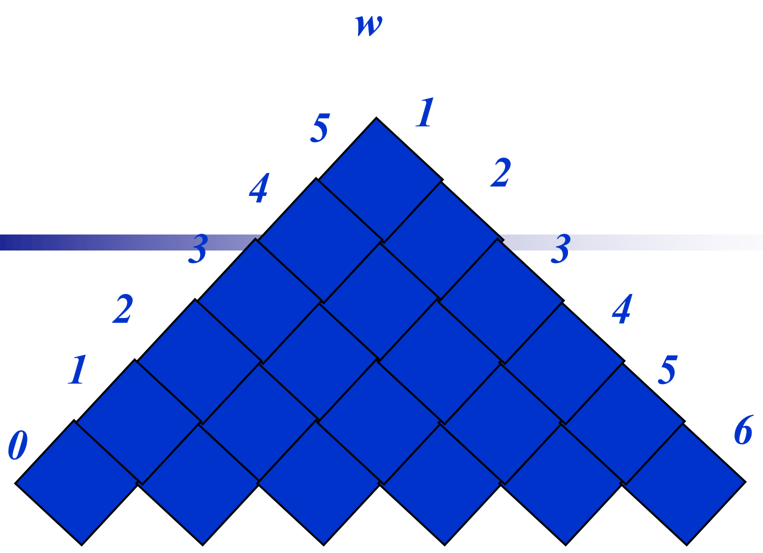
Step3: Computing the expected search cost of an OBST

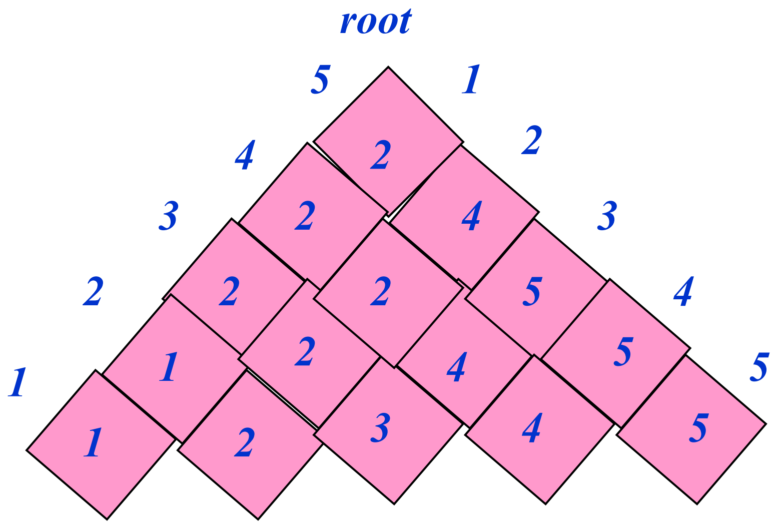
- We store the e[i,j] values in a table e[1..n+1, 0..n].
- The first index needs to run to n+1rather than n because in order to have a subtree containing only the dummy key d_n, we will need to compute and store e[n+1,n].
- The second index needs to start from 0 because in order to have a subtree containing only the dummy key d_0, we will need to compute and store e[1,0].
- We will use only the entries e[i,j] for which $j \geqq i-1$. we also use a table root[i,j], for recording the root of the subtree containing keys $k_i, \ldots, k_j$. This table uses only the entries for which $1 \leqq i \leqq j \leqq n$.

- We will need one other table for efficiency. Rather than computing the value of w(i,j) from scratch every time we are computing e[i,j] ------ we store these values in a table w[1..n+1,0..n]. For the base case, we compute w[I,i-1] = q_{i-1} for 1≦I ≦n.
- For j≧i, we compute :

  w[i,j]=w[I,j-1]+p_i + q_i







root
1
2
3
4
5
1
2
3
4
5
2
2
4
2
2
5
1
2
4
5
1
2
3
4
5

The tables e[i,j], w[i,j], and root [i,j]computed by Optimal-BST