BITS, PILANI – K. K. BIRLA GOA CAMPUS

Design & Analysis of Algorithms

(CS F364)

Lecture No. 9

All-Pairs Shortest Path

G = (V, E) be a graph

G has **no negative weight cycles**

Vertices are labeled 1 to n

If (i, j) is an edge its weight is denoted by $\boldsymbol{w_{ij}}$

Optimal Substructure Property

Easy to Prove

Recursive formulation - 1

Let $l_{ij}^{(m)}$ be the minimum weight of any path from vertex ***i*** to vertex ***j*** that contains atmost ***m*** edges.

Then,

$$l_{ij}^{(0)} = \begin{cases} 0, if\ i = j \\ \infty, if\ i \neq j \end{cases}$$

$$l_{ij}^{(m)} = min(l_{ij}^{(m-1)}, \min_{1 \le k \le n}\{l_{ik}^{(m-1)} + w_{kj}\})$$

minimum over all **predecessors** ***k*** of ***j***

Recursive formulation - 2

Let $d_{ij}^{(k)}$ be the weight of the shortest path from vertex *i* to vertex *j*,

for which all the intermediate vertices are in the set *{1,2,….,k}*

Then

$d_{ij}^{(k)} = w_{ij}$ *if k = 0*

$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$ if $k \geq 1$

We have tacitly used the fact that an optimal path through *k* does not visit *k* twice.

Optimal Binary Search Tree

We start with a problem regarding binary search trees in an environment in which the probabilities of accessing elements and gaps between elements is known.

Goal:

We want to find the binary search tree that minimizes the expected number of nodes probed on a search.

Optimal Binary Search Tree

Formally

We have ***n*** keys (represented as $k_1,k_2,\ldots,k_n$) 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"** $d_0,d_1,\ldots,d_n$
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}.
For each d_i ,we have a probability q_i that a search will be for d_i
The dummy keys are leaves (external nodes), and the data keys mean internal nodes.

Optimal Binary Search Tree

- Every search is either successful or unsuccessful and so we have

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

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.***

Let us 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***.

Optimal Binary Search Tree

- Then the expected cost of a search in ***T*** is

E[search cost in ***T***]

$= \sum_{i=1}^{n}(depth_T(k_i) + 1)p_i + \sum_{i=0}^{n}(depth_T(d_i) + 1)q_i$

$= 1 + \sum_{i=1}^{n} depth_T(k_i)p_i + \sum_{i=0}^{n} depth_T(d_i)q_i$ **(1)**

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

Example

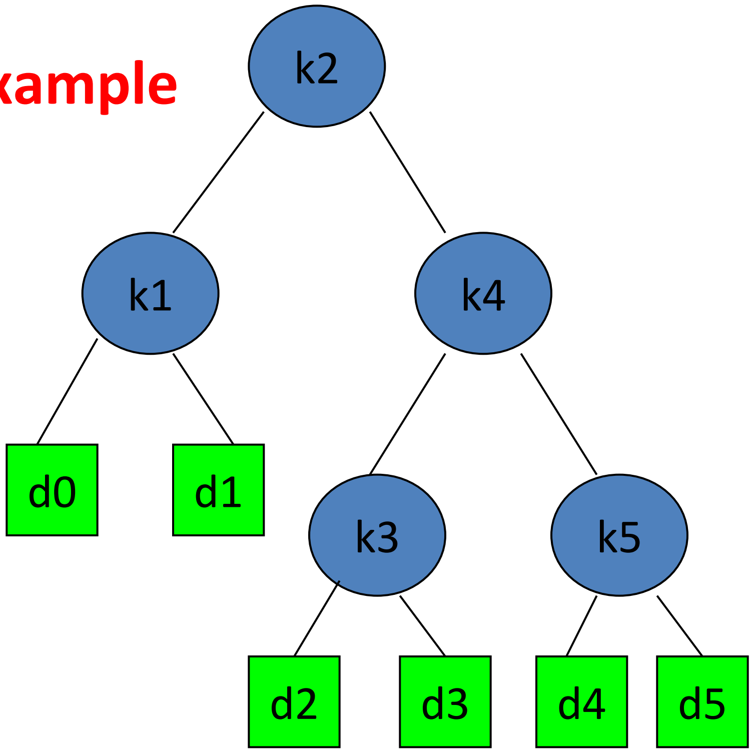

Figure (a)

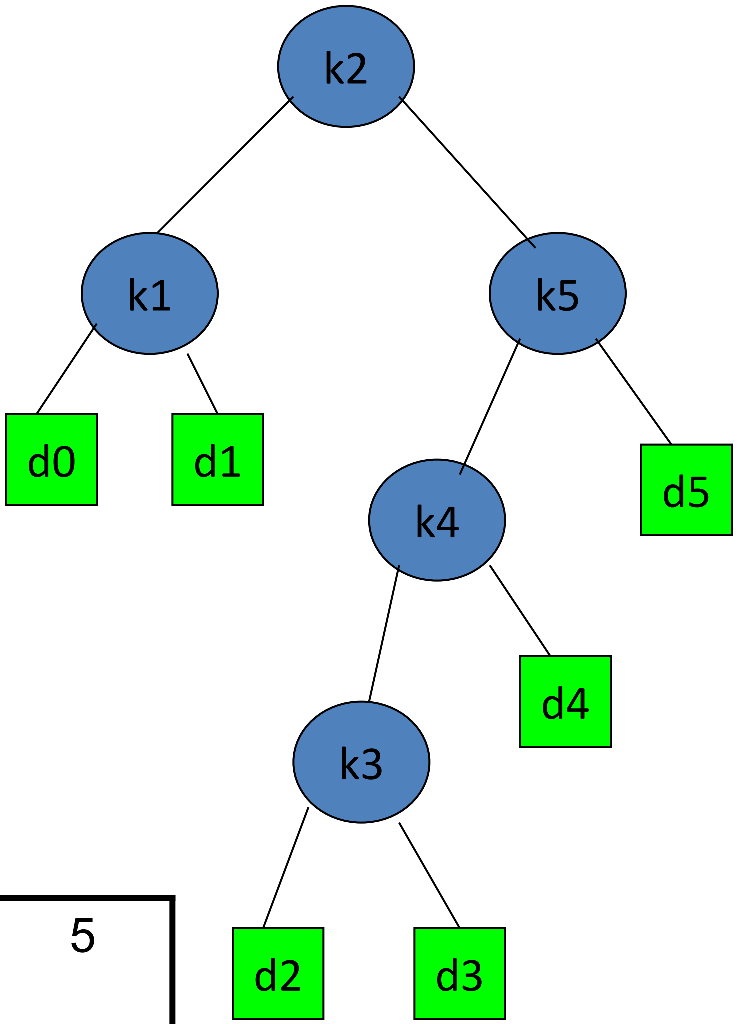

Figure (b)

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

Optimal Binary Search Tree

Observe:

Figure (a) costs 2.80 ,on another,

the Figure (b) costs 2.75,

and that tree is really optimal.

We can see the height of (b) is more than (a)

So optimal BST is not necessarily a tree whose height is smallest.

Optimal Binary Search Tree

Brute Force Approach:

Exhaustive checking of all possibilities.

Question

What is the number of binary search trees on ***n*** keys?

Answer

$$t(n) = \sum_{i=1}^{n} t(i-1)\, t(n-i).$$

The number of BST is atleast **$\Omega(2^n)$**

Dynamic Programming Solution

Step1 : **Optimal Substructure Property**

Exercise

Step2 : **Recursive Formulation**

We pick our subproblem domain as finding an Optimal BST containing the ***keys*** $k_i,\ldots,k_j$, where $i \geq 1$, $j \leq n$, and $j \geq i-1$. (It is when $j = i-1$ that there 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 Optimal BST containing the keys $k_i,\ldots, k_j$

Ultimately, we wish to compute $e[1,n]$.

Optimal Binary Search Tree

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 $j \geq 1$, we need to select a root k_r from among $k_i ,\ldots,k_j$

and then make an Optimal BST with keys $k_i,\ldots,k_{r-1}$ as its left subtree

and an Optimal BST with keys $k_{r+1},\ldots,k_j$ as its right subtree.

Optimal Binary Search Tree

- What happens to the expected search cost of a subtree when it becomes a subtree of a node?

The depth of each node in the subtree increases by 1.

So, by equation (1) the excepted search cost of this subtree increases by the sum of all the probabilities in the subtree.

For a subtree with keys $\boldsymbol{k_i},\ldots,\boldsymbol{k_j}$ let us denote this sum of probabilities as

$$\boldsymbol{w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_i}$$

Optimal Binary Search Tree

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

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

Noting that $\boldsymbol{w(i, j) = w(i,r-1)+ p_r +w(r+1,j)}$

We rewrite $\boldsymbol{e[i, j]}$ as

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

The recursive equation as above assumes that we know which node $\boldsymbol{k_r}$ to use as the root.

We choose the root that gives the lowest expected search cost

Optimal Binary Search Tree

- Final recursive formulation:

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

The ***e[i, j]*** values give the expected search costs in Optimal BST.

To help us keep track of the structure of Optimal BST, we define ***root[i, j]***, for ***1≤ i ≤ j ≤ n***, to be the index ***r*** for which $\boldsymbol{k_r}$ is

the root of an Optimal BST containing keys $\boldsymbol{k_i,\ldots,k_j}$