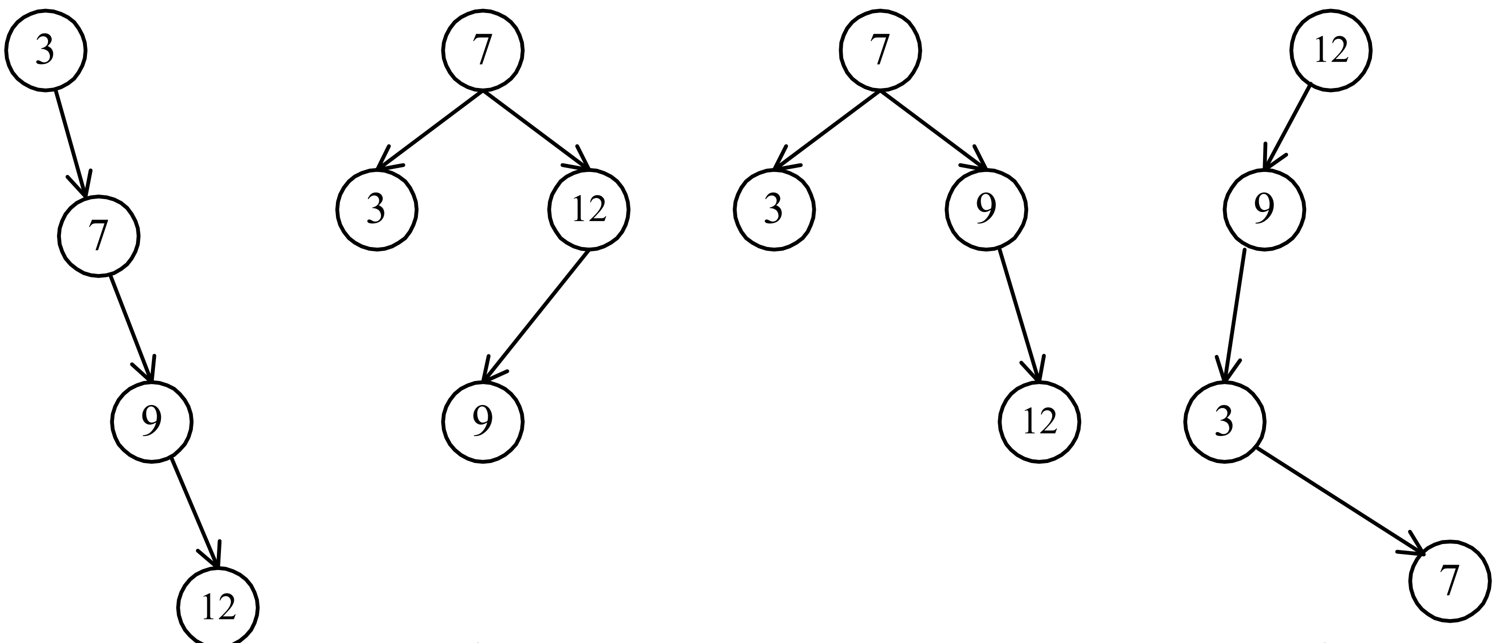
CSE408
Optimal binary search tree and Knapsack problem

Lecture # 23

Optimal binary search trees

- e.g. binary search trees for 3, 7, 9, 12;



(a) (b) (c) (d)

Optimal binary search trees

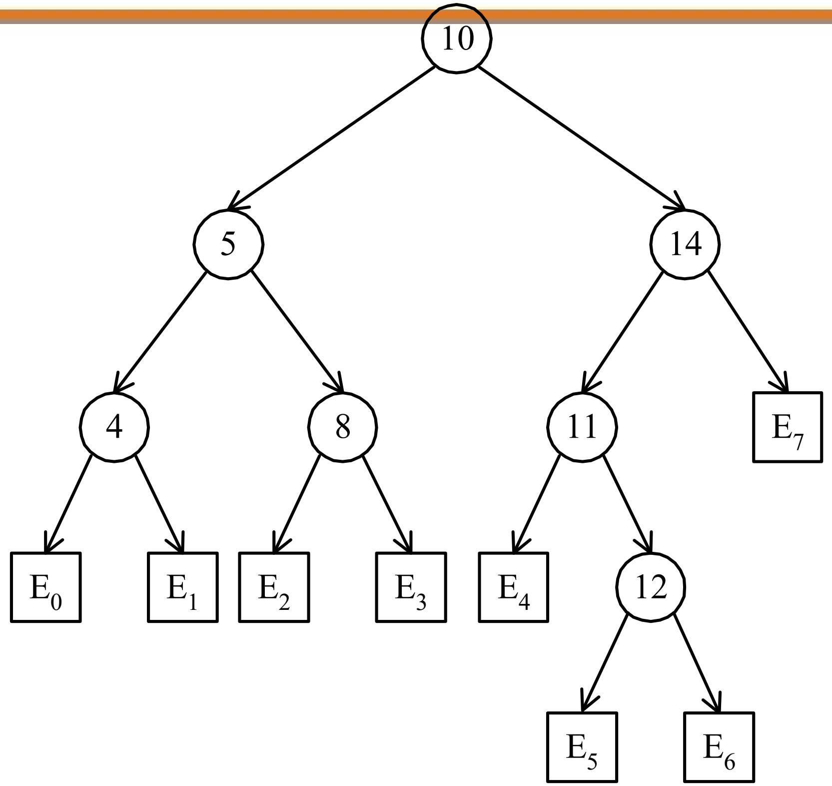
- n identifiers : $a_1 < a_2 < a_3 < \ldots < a_n$

P_i, $1 \le i \le n$: the probability that a_i is searched.

Q_i, $0 \le i \le n$: the probability that x is searched

where $a_i < x < a_{i+1}$ ($a_0 = -\infty$, $a_{n+1} = \infty$).

$$\sum_{i=1}^{n} P_i + \sum_{i=1}^{n} Q_i = 1$$



- Identifiers : 4, 5, 8, 10, 11, 12, 14
- Internal node : successful search, P_i
- External node : unsuccessful search, Q_i

- The expected cost of a binary tree:

$$\sum_{n=1}^{n} P_i * \text{level}(a_i) + \sum_{n=0}^{n} Q_i * (\text{level}(E_i) - 1)$$

- The level of the root : 1

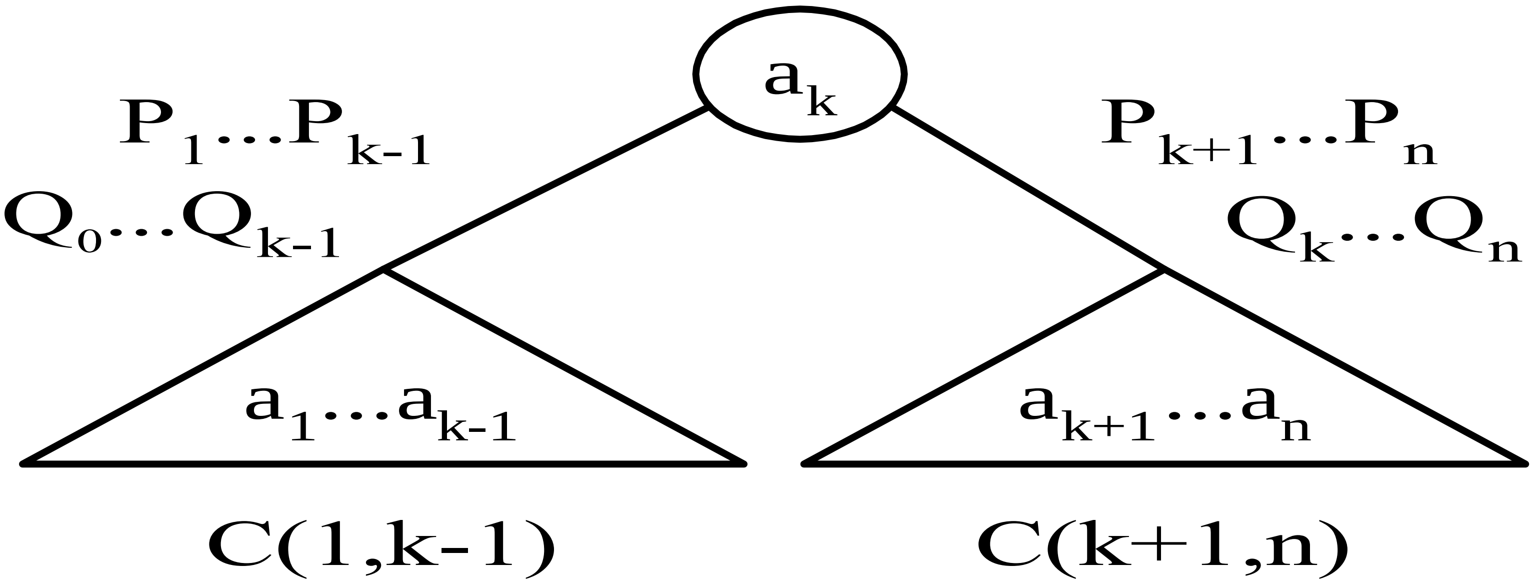
The dynamic programming approach

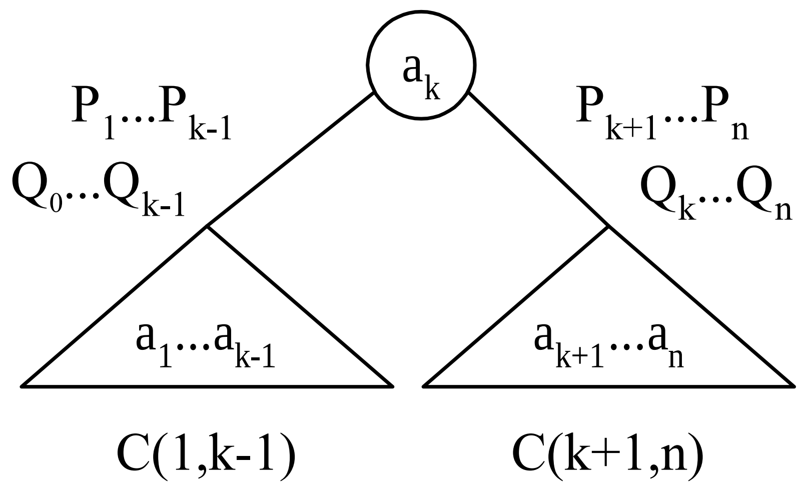
- Let C(i, j) denote the cost of an optimal binary search tree containing $a_i, \ldots, a_j$.
- The cost of the optimal binary search tree with a_k as its root :

$$C(1,n)=\min_{1\le k\le n}\left\{P_k+\left[Q_0+\sum_{i=1}^{k-1}\left(P_i+Q_i\right)+C(1,k-1)\right]+\left[Q_k+\sum_{i=k+1}^{n}\left(P_i+Q_i\right)+C(k+1,n)\right]\right\}$$

$$C(1,n)=\min_{1\le k\le n}\left\{P_k+\left[Q_0+\sum_{i=1}^{k-1}\left(P_i+Q_i\right)+C(1,k-1)\right]+\left[Q_k+\sum_{i=k+1}^{n}\left(P_i+Q_i\right)+C(k+1,n)\right]\right\}$$

General formula





Computation relationships of sub trees

- e.g. n=4

$$O\left(\sum_{1 \le m \le n} (m(n-m))\right) = O(n^3)$$

- Time complexity : $O(n^3)$

  when j-i=m, there are (n-m) C(i, j)'s to compute.

  Each C(i, j) with j-i=m can be computed in O(m) time.

$$O\left(\sum_{1 \le m \le n} (m(n-m))\right) = O(n^3)$$

Knapsack problem

There are two versions of the problem:

1. "0-1 knapsack problem"
   - Items are indivisible; you either take an item or not. Some special instances can be solved with *dynamic programming*

2. "Fractional knapsack problem"
   - Items are divisible: you can take any fraction of an item

0-1 Knapsack problem

- Given a knapsack with maximum capacity W, and a set S consisting of n items
- Each item i has some weight w_i and benefit value b_i (all w_i and W are integer values)
- Problem: How to pack the knapsack to achieve maximum total value of packed items?

0-1 Knapsack problem

- Problem, in other words, is to find

$$\max \sum_{i \in T} b_i \text{ subject to } \sum_{i \in T} w_i \leq W$$

- The problem is called a "*0-1*" problem, because each item must be entirely accepted or rejected.

0-1 Knapsack problem: brute-force approach

Let's first solve this problem with a straightforward algorithm

- Since there are *n* items, there are 2^n possible combinations of items.
- We go through all combinations and find the one with maximum value and with total weight less or equal to *W*
- Running time will be $O(2^n)$

0-1 Knapsack problem: dynamic programming pproach

- We can do better with an algorithm based on dynamic programming

- We need to carefully identify the subproblems

Defining a Subproblem

- Given a knapsack with maximum capacity W, and a set S consisting of n items
- Each item i has some weight w_i and benefit value b_i (all w_i and W are integer values)
- <u>Problem</u>: How to pack the knapsack to achieve maximum total value of packed items?

Defining a Sub problem

- We can do better with an algorithm based on dynamic programming

- We need to carefully identify the subproblems

Let's try this:
If items are labeled *1..n*, then a subproblem would be to find an optimal solution for
$S_k = \{items\ labeled\ 1, 2, ..\ k\}$

Thank You !!!