ALGORITHM DESIGN
DYNAMIC PROGRAMMING

University of Waterloo

List of Slides

1
2 Dynamic Programming Approach
3 Fibonacci Sequence (cont.)
4 Fibonacci Sequence (cont.)
5 Bottom-Up vs. Top-Down
6 Optimal Binary Search Tree
7 Optimal BST: Examples
8 Optimal BST (cont.)
9 Optimal BST (cont.)
10 All-points Shortest Path
11 Principle of Optimality
12 Summary

Dynamic Programming Approach

IDEA: Use recursive (e.g., divide-and-conquer) approach but be careful **not to recompute** the same values

Example: Fibonacci numbers

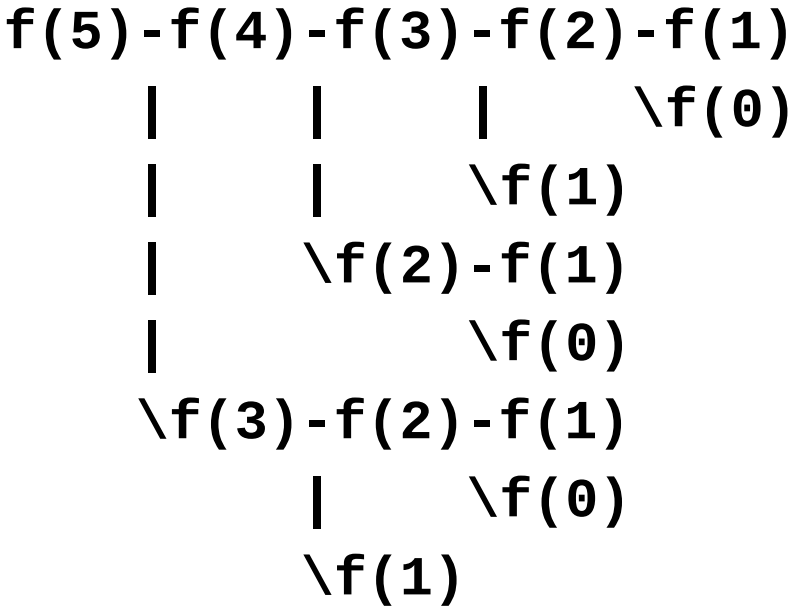
Fibonacci(n) =
  if $n < 2$
    then
      return n
    else
      return Fibonacci$(n-1)$+Fibonacci$(n-2)$

Running time: $\Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$

Fibonacci Sequence (cont.)

Why is it so bad?

Trace of **Fibonacci**(k) calls for $n = 5$:



Fibonacci Sequence (cont.)

IDEA: compute "bottom-up":

new-Fibonacci(n) =
    if $n < 2$ **then** return n
    $a := 1$ // Fib(0)
    $b := 1$ // Fib(1)
    for $i := 3$ **to** n
        $a, b := b, a + b$ // Fib(n),Fib(n+1)
    return b

$\Rightarrow$ runs in $O(n)$

Bottom-Up vs. Top-Down

GENERAL IDEA:

Remember call-return pairs for recursive functions

keep (necessary) results in a table

$\Rightarrow$ bottom-up technique

construct results from small to large

most common

$\Rightarrow$ top-down technique

remember calls made

short-circuit calls by known results

Optimal Binary Search Tree

Problem: given

— values $w_1, \ldots, w_N$ (in increasing order) and

— fixed probabilities $p_1, \ldots, p_N$

Goal: construct "optimal" search tree (that minimizes expected search time w.r.t. the given probabilities):

$$\min \sum_{i=1}^{N} p_i(1 + d_i)$$

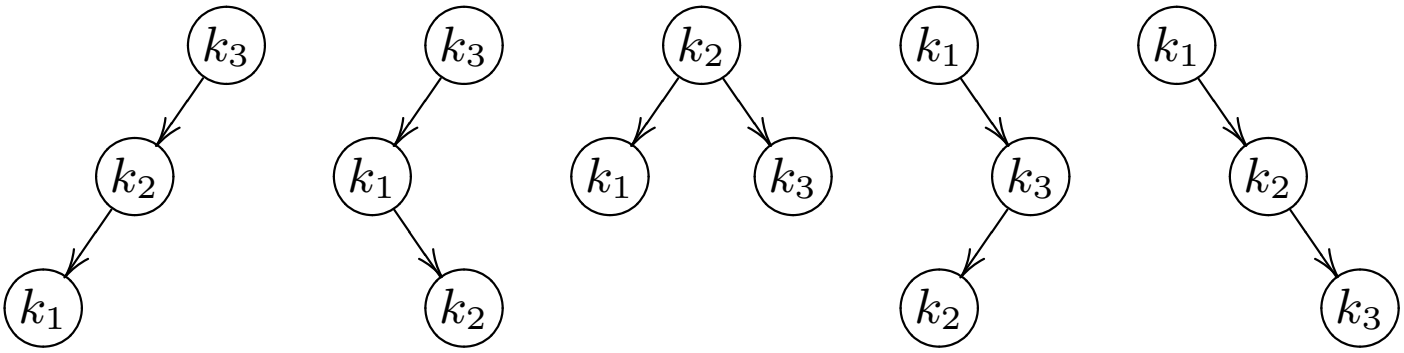
where d_i is the depth of the value w_i.

Optimal BST: Examples

Consider a BST storing $k_1 < k_2 < k_3$

$\Rightarrow$ frequency of access of the key $k_1(k_2, k_3)$
is p (q, r), respectively.

All BST's for k_1, k_2, k_3:

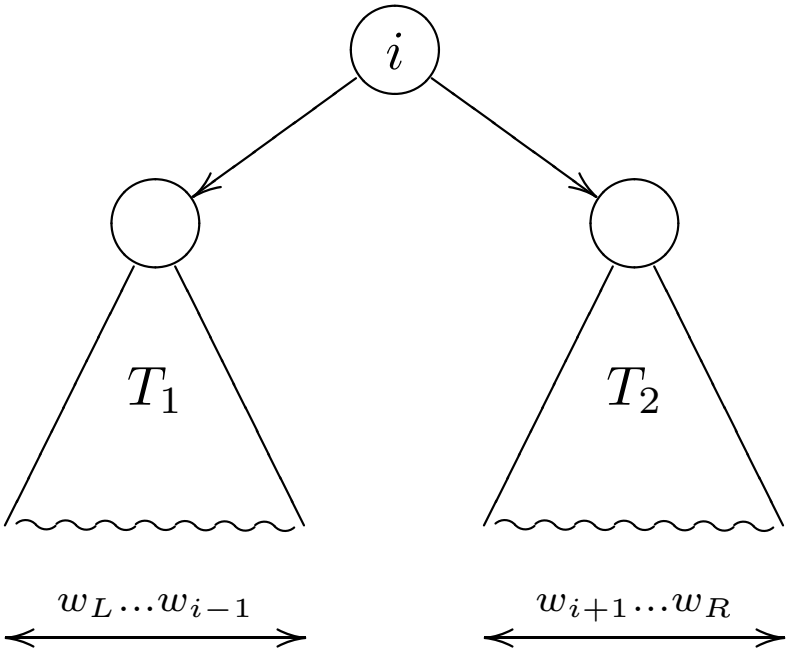


$3p + 2q + r$ $2p + 3q + r$ $2p + q + 2r$ $p + 3q + 2r$ $p + 2q + 3r$

. . . with access costs w.r.t. p, q, r.

Optimal BST (cont.)

Recursive definition of cost of tree for words $w_L, \ldots, w_R$:



$$C_{L,R} = \min_{L \le i \le R} \left(C_{L,i-1} + C_{i+1,R} + \sum_{j=L}^{R} p_j \right)$$

with the base case $C_{L,L} = 0$.

Optimal BST (cont.)

Recursive (top-down) solution:

$\Rightarrow$ recomputes the cost of "small" trees many times

$$T(n) = O(n) + 2\sum_{i=1}^{n-1} T(i)$$

Dynamic programming (bottom-up) solution:

$\Rightarrow$ computes the costs for optimal "small" trees
and stores the resulting $C_{i,j}$ in an $O(n^2)$-sized table

$\Rightarrow$ time complexity $\Theta(n^3)$ (can be reduced to $O(n^2)$)

All-points Shortest Path

Input: A directed graph $G = (V, E)$ with positive integer edge weights $d_{i,j}$.

Output: For each ordered pair $(u, v) \in V^2$, the length of the shortest path from u to v.

Let $D_k(i, j)$ be the length of the shortest path from v_i to v_j whose intermediate nodes come from $\{v_1, \ldots, v_k\}$.

$$D_k(i,j) = \begin{cases} d_{i,j} & k = 0 \\ \min\{D_{k-1}(i,j), & \\ \qquad D_{k-1}(i,k) + D_{k-1}(k,j)\} & k > 0 \end{cases}$$

Time complexity $\Theta(n^3)$.

Principle of Optimality

The *principle of optimality* holds for an optimization problem if for every optimal sequence of decisions, every subsequence is also optimal.

Example:
If p is a shortest path from u to v and p contains w, then:

- The initial portion of p from u to w is a shortest path from u to w; and
- The final portion of p from w to v is a shortest path from w to v.

However:
If p is a longest simple path from u to v that contains w, then it is not necessarily the case that the initial portion of p from u to w is the longest simple path from u to w.

Summary

- Very powerful method

  $\Rightarrow$ avoids unnecessary reevaluation

- Two main principles:

  1. memoing of call-return pairs (e.g., in a table)
     $\Rightarrow$ make sure to “evaluate in the right order”
  2. the optimality principle