Algorithms Theory

14 – Dynamic Programming (3)

Optimal binary search trees

Prof. Dr. S. Albers

Method of dynamic programming

Recursive approach: Solve a problem by solving several smaller analogous subproblems of the same type. Then combine these solutions to generate a solution to the original problem.

Drawback: Repeated computation of solutions

Dynamic-programming method: Once a subproblem has been solved, store its solution in a table so that it can be retrieved later by simple table lookup.

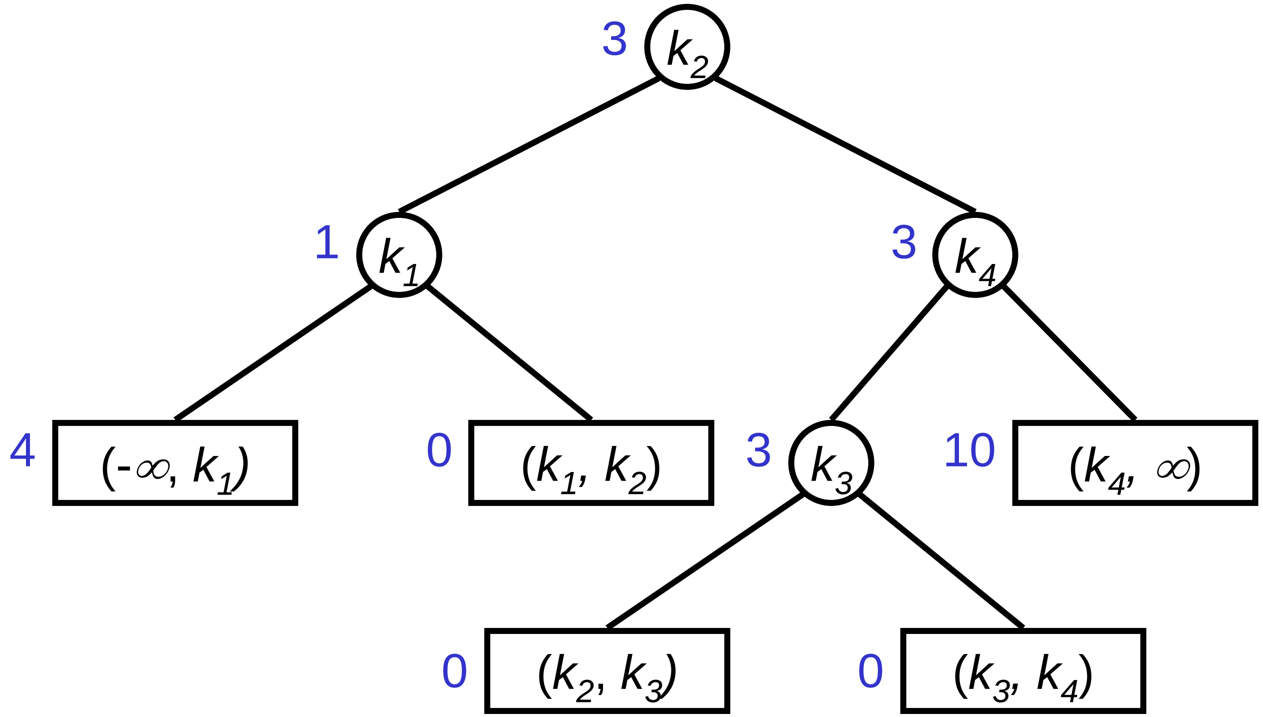
Optimal substructure

Dynamic programming is typically applied to
optimization problems.

An optimal solution to the original problem contains
optimal solutions to smaller subproblems.

Construction of optimal binary search trees

$(-\infty,k_1)$	k_1	(k_1,k_2)	k_2	(k_2,k_3)	k_3	(k_3,k_4)	k_4	(k_4,∞)
4	1	0	3	0	3	0	3	10



weighted path length:

$3 \cdot 1 + 2 \cdot (1 + 3) + 3 \cdot 3 + 2 \cdot (4 + 10)$

Construction of optimal binary search trees

Given: set S of keys

$$S = \{k_1,....k_n\} \quad -\infty = k_0 < k_1 < ... < k_n < k_{n+1} = \infty$$

a_i: (absolute) frequency of requests to key k_i

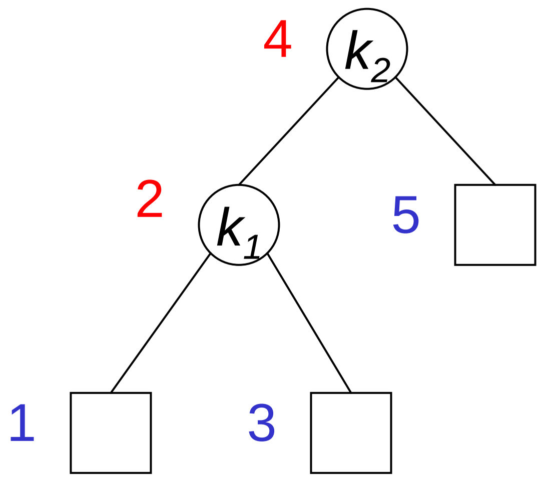
b_j: (absolute) frequency of requests to $x \in (k_j , k_{j+1})$

Weighted path length *P(T)* of a binary search tree *T* for *S*:

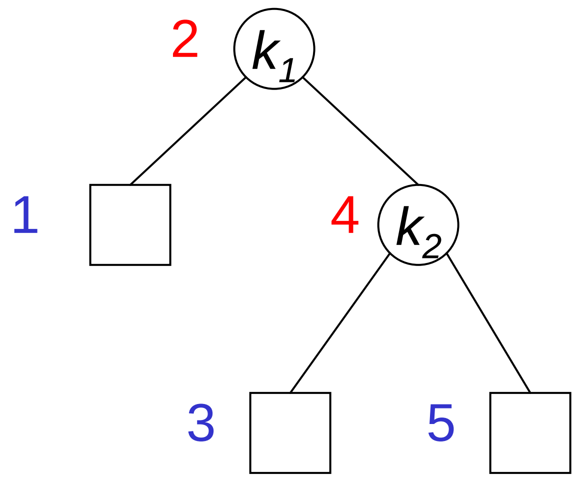
$$P(T) = \sum_{i=1}^{n} (depth(k_i) + 1)a_i + \sum_{j=0}^{n} depth((k_j, k_{j+1}))b_j$$

Goal: Binary search tree with minimum weighted path length *P* for *S* .

Construction of optimal binary search trees

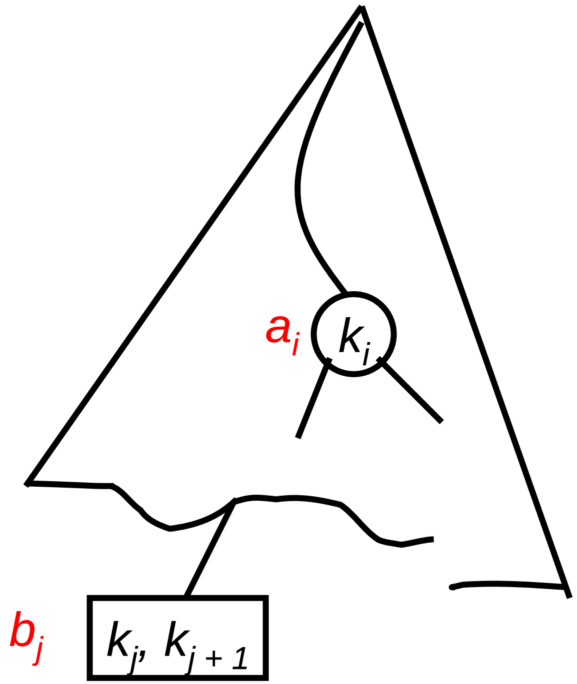


$P(T_1) = 21$



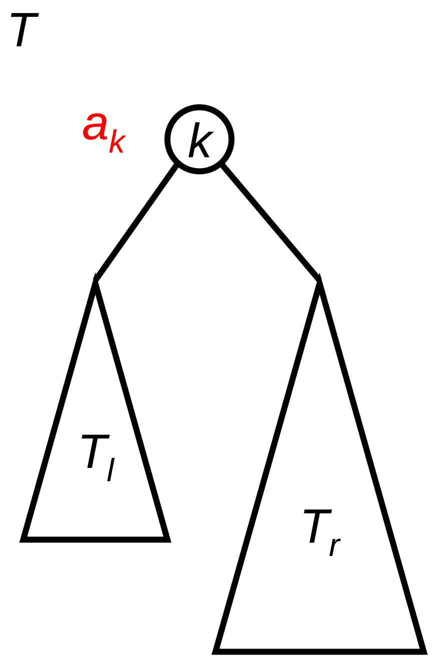
$P(T_2) = 27$

Construction of optimal binary search trees



An optimal binary search tree is a binary search tree with minimum weighted path length.

Construction of optimal binary search trees



$$P(T) = P(T_l) + W(T_l) + P(T_r) + W(T_r) + a_{root}$$

$$= P(T_l) + P(T_r) + W(T) \text{ mit}$$

$W(T)$:= total weight of all nodes in T

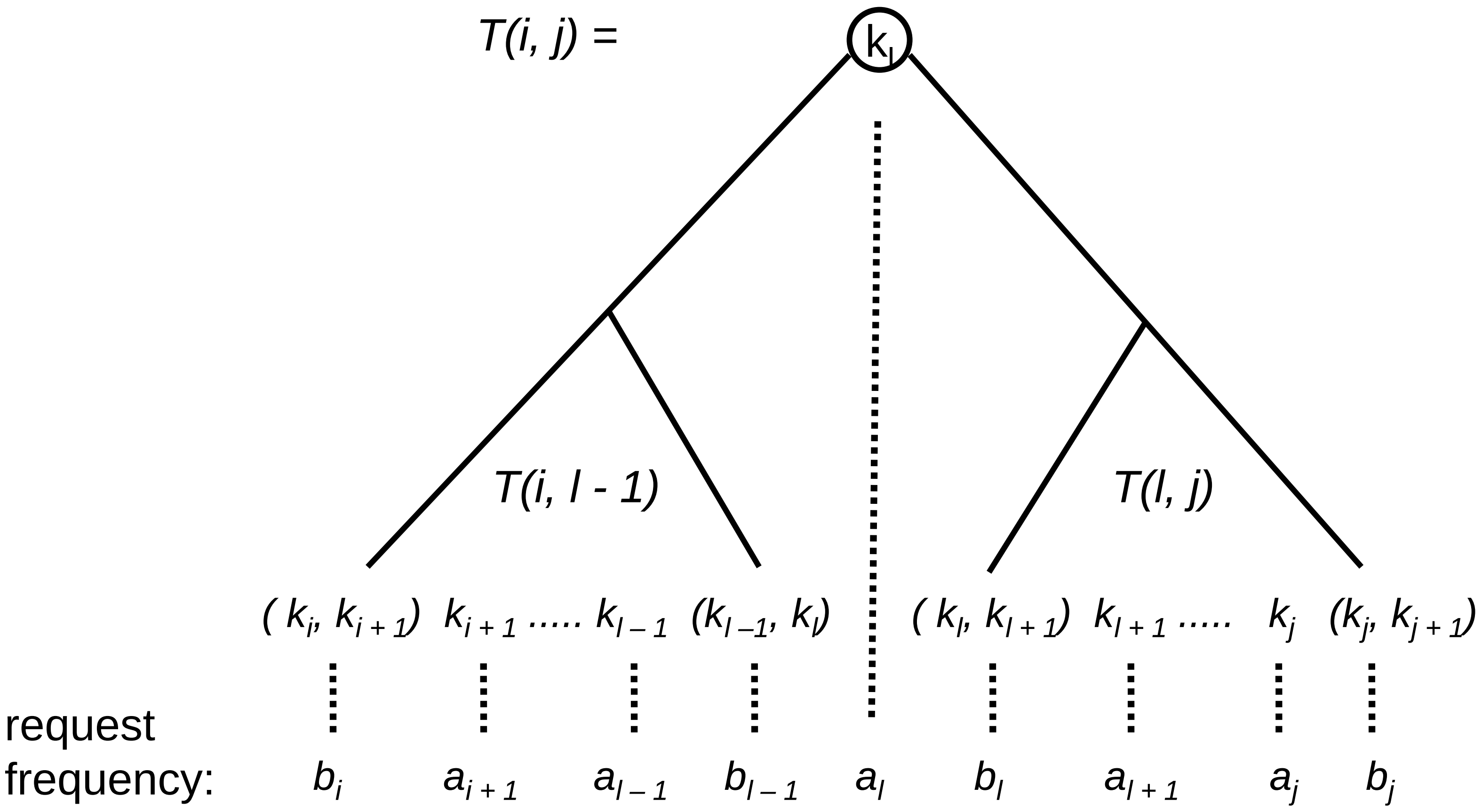
If T is a tree with minimum weighted path length for S, then subtrees T_l and T_r are trees with minimum weighted path length for subsets of S.

Construction of optimal binary search trees

Let

$T(i, j)$: optimal binary search tree for $(k_i, k_{i+1})\ k_{i+1} \ldots k_j\ (k_j, k_{j+1})$,
$W(i, j)$: weight of $T(i, j)$, i.e. $W(i, j) = b_i + a_{i+1} + \ldots + a_j + b_j$,
$P(i, j)$: weighted path length of $T(i, j)$.

Construction of optimal binary search trees



Construction of optimal binary search trees

$W(i, i) = b_i$, for $0 \leq i \leq n$

$W(i, j) = W(i, j-1) + a_j + b_j$, for $0 \leq i < j \leq n$

$P(i, i) = 0$, for $0 \leq i \leq n$

$$P(i, j) = W(i, j) + \min_{i < l \leq j} \{ P(i, l-1) + P(l, j) \}, \text{ for } 0 \leq i < j \leq n \qquad (*)$$

$r(i, j)$ = the index l for which the minimum is achieved in (*)

Construction of optimal binary search trees

Base cases

Case 1: $h = j - i = 0$

$T(i, i) = (k_i, k_{i+1})$

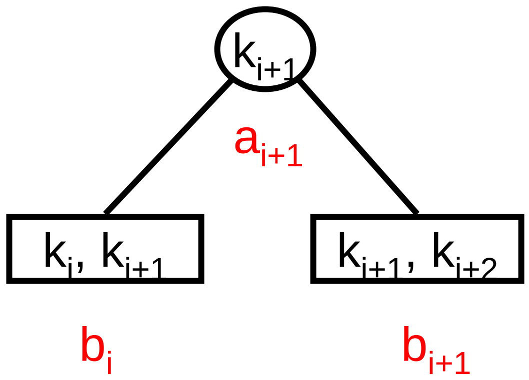
$W(i, i) = b_i$

$P(i, i) = 0$, $r(i, i)$ not defined

Construction of optimal binary search trees

Case 2: $h = j - i = 1$

$T(i, i+1)$



$W(i, i+1) = b_i + a_{i+1} + b_{i+1} = W(i, i) + a_{i+1} + W(i+1, i+1)$

$P(i, i+1) = W(i, i+1)$

$r(i, i+1) = i + 1$

Computating the minimum weighted path length using dynamic programming

Case 3: $h = j - i > 1$

for $h = 2$ **to** n **do**

  for $i = 0$ **to** $(n - h)$ **do**

    { $j = i + h$;

     determine (greatest) l, $i < l \leq j$, s.t. $P(i, l - 1) + P(l, j)$ is minimal

     $P(i, j) = P(i, l - 1) + P(l, j) + W(i, j)$;

     $r(i, j) = l$;

    }

Construction of optimal binary search trees

Define:

$$\left.\begin{array}{l} P(i,j) := \text{min. weighted path length for} \\ W(i,j) := \text{sum of} \end{array}\right\} b_i a_{i+1} b_{i+1} \ldots a_j b_j$$

Then:

$$W(i,j) = \begin{cases} b_i & \text{if } i = j \\ W(i, j-1) + a_j + W(j,j) & \text{otherwise} \end{cases}$$

$$P(i,j) = \begin{cases} 0 & \text{if } i = j \\ W(i,j) + \min\limits_{i<l\le j} \{P(i,l-1) + P(l,j)\} & \text{otherwise} \end{cases}$$

→ Computing the solution $P(0,n)$ takes $O(n^3)$ time.
and requires $O(n^2)$ space.

Construction of optimal binary search trees

Theorem

An optimal binary search tree for n keys and $n + 1$ intervals with known request frequencies can be constructed in $O(n^3)$ time.