

13 – Dynamic Programming (3)

Optimal Binary Search Trees

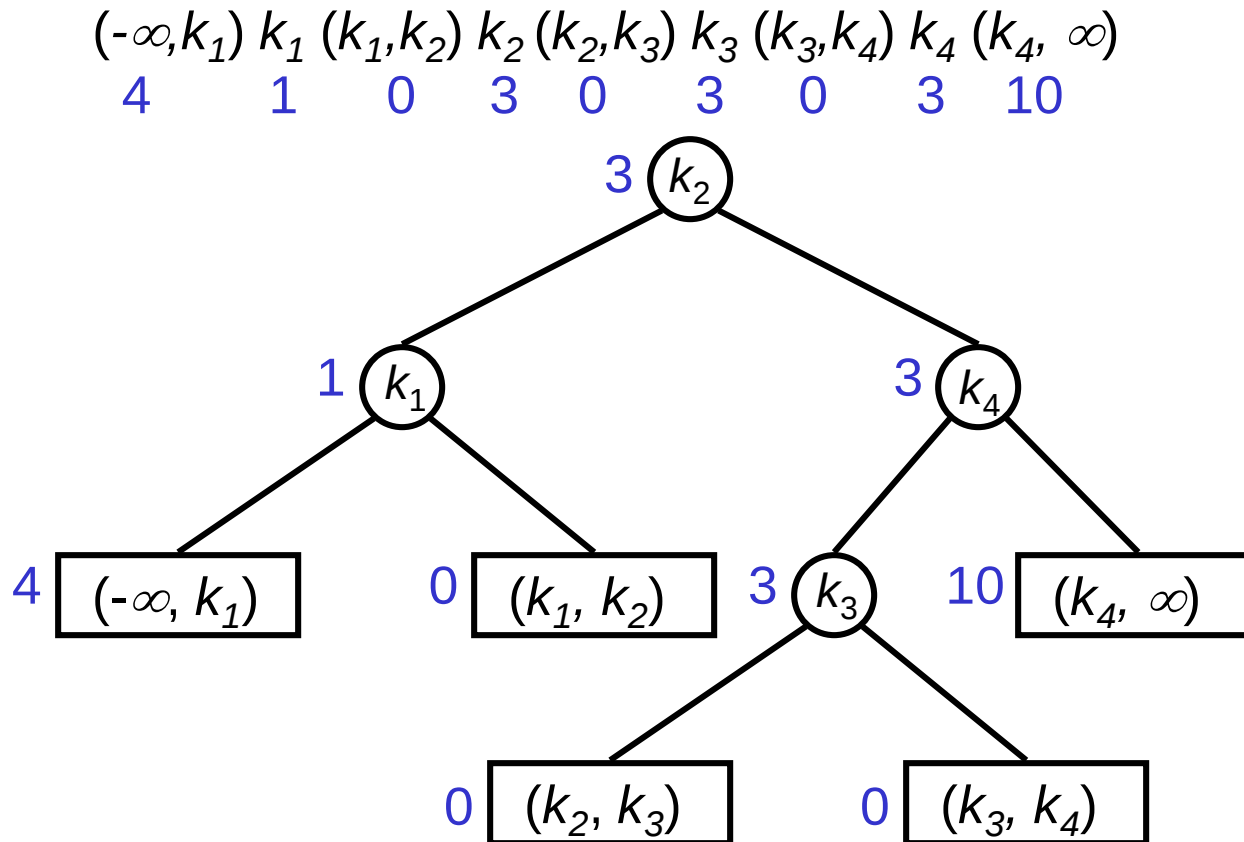
Subset Sums & Knapsacks

Average-case analysis

Average-case analysis of algorithms and data structures: Input is generated according to a **known probability distribution**. This distribution can be learned over time.

Optimal binary search tree: **Request probabilities / frequencies** for the keys are known in advance. Construct a binary search tree minimizing the expected / average search time.

Optimal binary search trees



Weighted path length:

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

Optimal binary search trees

Problem: Set S of keys

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

a_i : (absolute) frequency of requests to key k_i

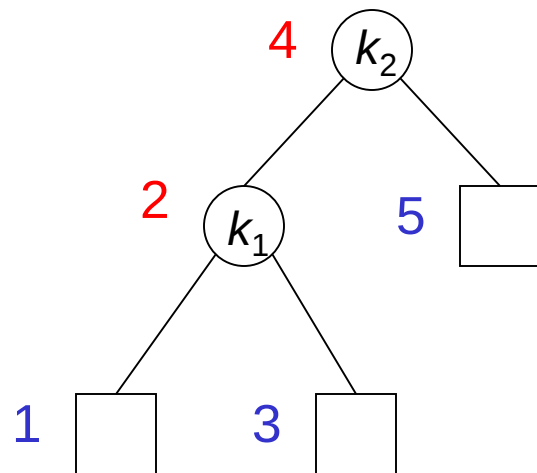
b_j : (absolute) frequency of requests to $x \in (k_j, k_{j+1})$

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

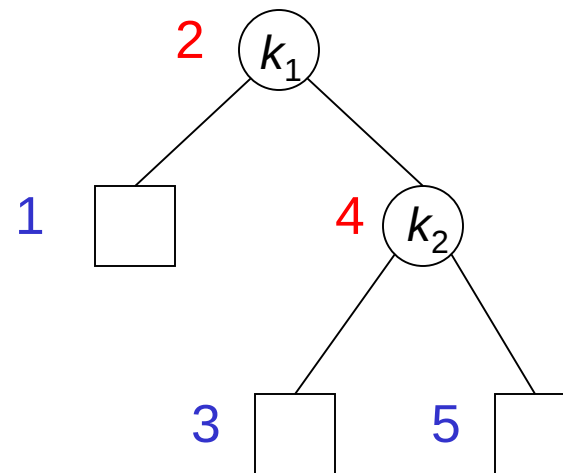
$$P(T) = \sum_{i=1}^n (\text{depth}(k_i) + 1) a_i + \sum_{j=0}^n \text{depth}((k_j, k_{j+1})) b_j$$

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

Example

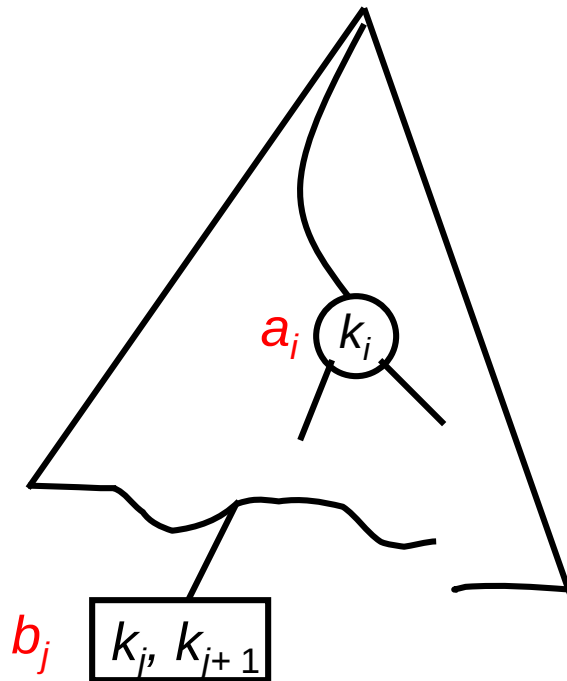


$$P(T_1) = 21$$



$$P(T_2) = 27$$

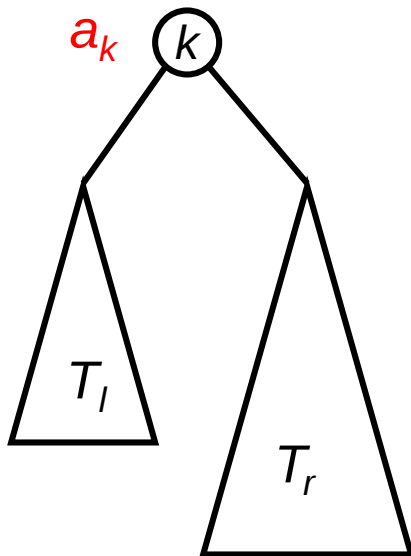
Construction of optimal binary search trees



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

Dynamic programming approach

T



$$P(T) = P(T_l) + W(T_l) + P(T_r) + W(T_r) + a_k$$

$$= P(T_l) + P(T_r) + W(T)$$

$$W(T) := \text{total weight of all nodes in } T$$

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 .

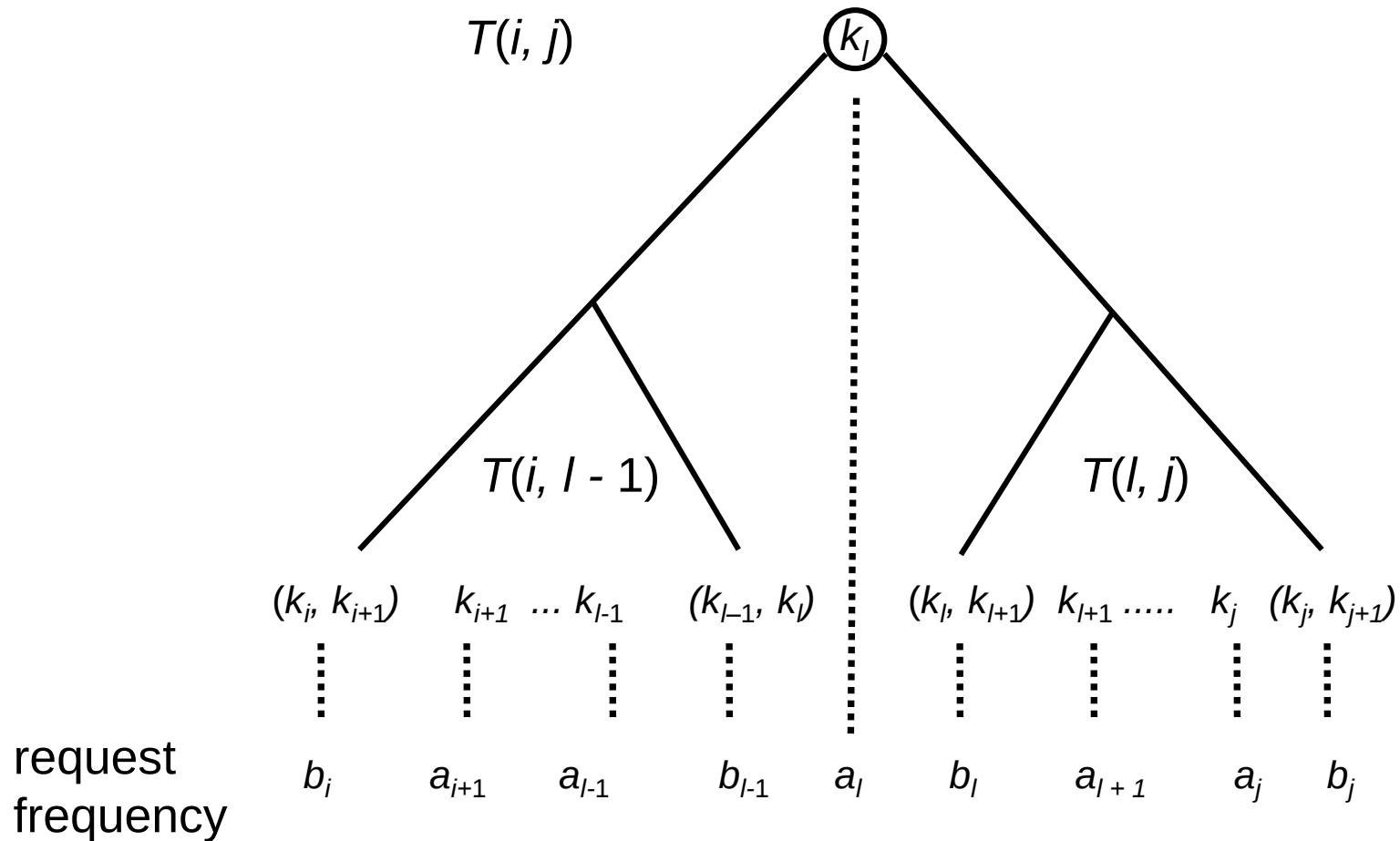
Definition of subproblems

$T(i, j)$: optimal binary search tree for $(k_i, k_{i+1}) k_{i+1} \dots k_j (k_j, k_{j+1})$

$W(i, j)$: weight of $T(i, j)$, i.e. $W(i, j) = b_i + a_{i+1} + \dots + a_j + b_j$

$P(i, j)$: weighted path length of $T(i, j)$

Subproblems



Recurrences

$$W(i, i) = b_i \quad \text{for } 0 \leq i \leq n$$

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

$$P(i, i) = 0 \quad \text{for } 0 \leq i \leq n$$

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

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

Bottum-up approach

Base cases

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

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

$$W(i, i) = b_i$$

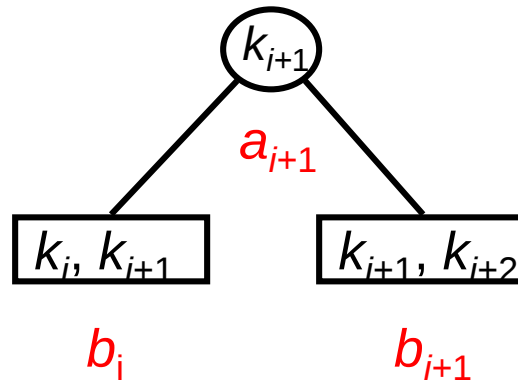
$$P(i, i) = 0$$

$r(i, i)$ not defined

Bottum-up approach

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

$T(i, i+1)$



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

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

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

Bottum-up approach

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

```
1  for  $s := 2$  to  $n$  do
2    for  $i := 0$  to  $n - s$  do
3       $j := i + s$ ;
4      Determine (greatest)  $l$ ,  $i < l \leq j$ , s.t.  $P(i, l - 1) + P(l, j)$  is minimal;
5       $W(i, j) := W(i, j-1) + a_j + b_j$ ;
6       $P(i, j) := P(i, l - 1) + P(l, j) + W(i, j)$ ;
7       $r(i, j) := l$ ;
8    endfor;
9  endfor;
```

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

Improvement

Lemma: For all i, j such that $0 \leq i < j \leq n$, $r(i, j-1) \leq r(i, j) \leq r(i+1, j)$.

Given this lemma, for fixed s , the total time of the inner for-loop is:

$$\begin{aligned}
 & O(n - s + \sum_{i=0}^{n-s} (r(i+1, i+s) - r(i, i+s-1) + 1)) \\
 &= O(n - s + r(1, s) - r(0, s-1) + 1 \\
 &\quad + r(2, s+1) - r(1, s) + 1 \\
 &\quad + r(3, s+2) - r(2, s+1) + 1 \\
 &\quad \dots \\
 &\quad + r(n-s+1, n) - r(n-s, n-1) + 1) \\
 &= O(n - s + r(n-s+1, n) - r(0, s-1)) \\
 &= O(n)
 \end{aligned}$$

Improvement

Proof of the Lemma: Induction on $s = j - i$.

For $s = 1$, the statement is vacuous. So consider $s = 2$.

In $T(i, i+1)$ key k_{i+1} is in the root, so that $r(i, i+1) = i+1$. In $T(i+1, i+2)$ key k_{i+2} is in the root, so that $r(i+1, i+2) = i+2$. In $T(i, i+2)$ key k_{i+1} or k_{i+2} can reside in the root, which implies $r(i, i+2) \in \{i+1, i+2\}$ and the desired inequality holds.

We study $s > 2$ and prove $r(i, j-1) \leq r(i, j)$. The second inequality $r(i, j) \leq r(i+1, j)$ can be shown analogously. Consider an optimal tree $T(i, j-1)$. Replace leaf (k_{j-1}, k_j) by a node containing k_j along with the leaves (k_{j-1}, k_j) and (k_j, k_{j+1}) . Let T be the resulting tree. There holds

$$P(T) = P(i, j-1) + b_{j-1} + (d+1)(a_j + b_j),$$

where d denotes the depth of k_j in T .

Proof of the lemma

Suppose that $P(T) > P(i, j)$ since otherwise we are done.

Consider an optimal tree $T(i, j)$ and let d' be the depth of k_j .

Claim 1: There holds $d > d'$.

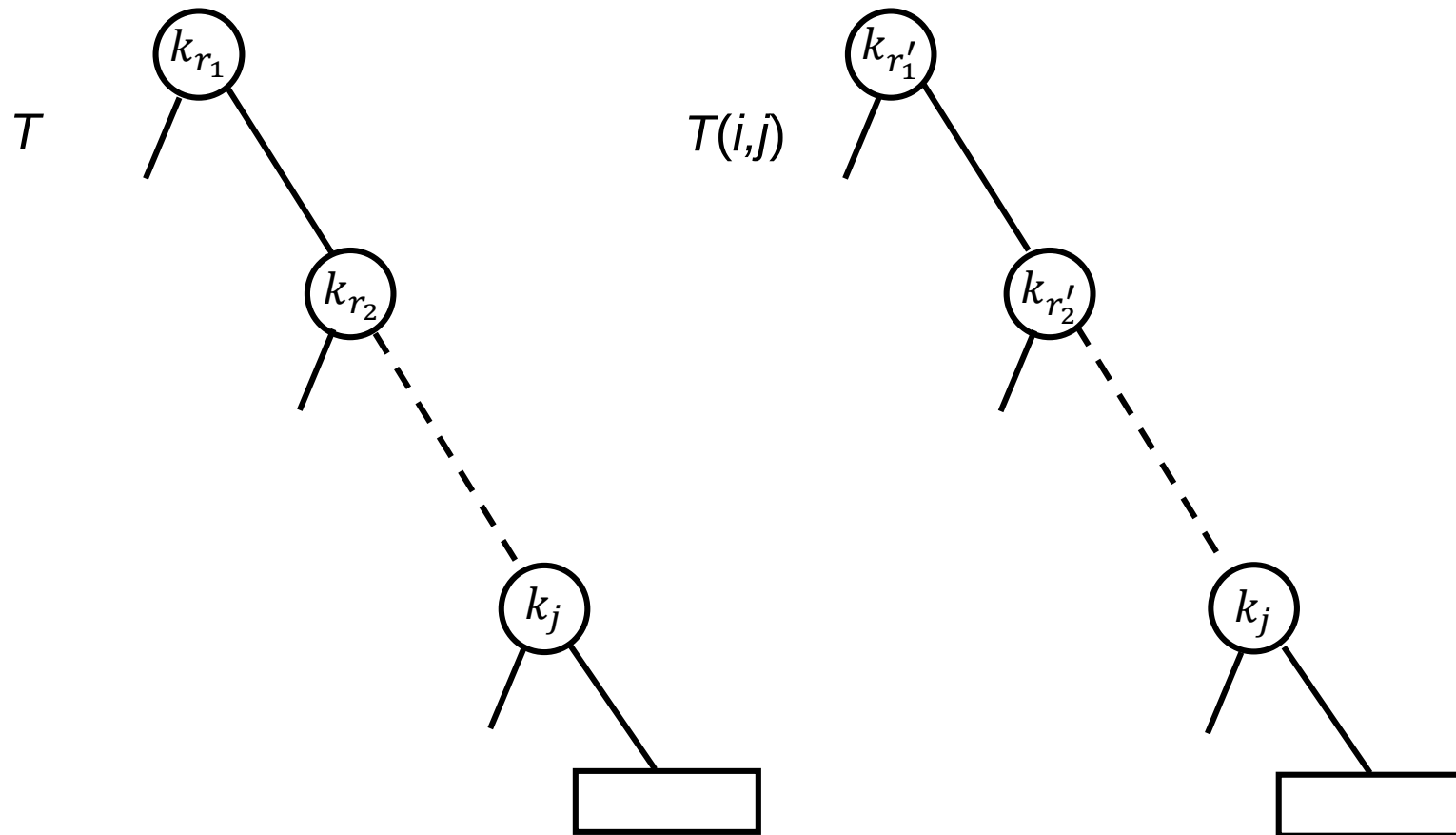
For the proof of the claim take $T(i, j)$ and replace key k_j along with leaves (k_{j-1}, k_j) and (k_j, k_{j+1}) by leaf (k_{j-1}, k_j) . The resulting tree has a weighted path length of

$$\begin{aligned} P(i, j) - b_{j-1} - (d'+1)(a_j + b_j) &< P(T) - b_{j-1} - (d'+1)(a_j + b_j) \\ &= P(i, j-1) + (d-d')(a_j + b_j). \end{aligned}$$

Hence, if $d \leq d'$, the resulting tree has a weighted path length strictly smaller than $P(i, j-1)$, contradicting the optimality of $T(i, j-1)$.

In T and $T(i, j)$ consider the paths from the root to k_j ; cf. the figure on the next page.

Proof of the lemma



Proof of the lemma

Suppose that $r_1 > r'_1$; otherwise there is nothing to show.

Claim 2: There exists an l such that $r_l = r'_l$.

By assumption $r'_1 < r_1$. The induction hypothesis implies $r'_2 \leq r_2$. If $r'_2 < r_2$, then $r'_3 \leq r_3$. In general if $r'_{l-1} < r_{l-1}$, the induction hypothesis implies $r'_l \leq r_l$. Since $d' < d$ and k_j is the largest key in the trees, there must exist an l such that $r_l = r'_l$.

In T we now replace the right subtree of k_{r_l} by the corresponding subtree in $T(i, j)$. The new tree has a minimum weighted path length: Otherwise in $T(i, j-1)$ the nodes containing $k_{r_1}, \dots, k_{r_l}$ and their left subtrees could be replaced by the corresponding structure in $T(i, j)$, contradicting the optimality of $T(i, j-1)$.

In the new, optimal tree key k_{r_1} is in the root.

Main result

Theorem:

An optimal binary search tree for n keys and $n + 1$ intervals with known request frequencies can be constructed in $O(n^2)$ time.

Subset Sums and Knapsacks

Subset Sum Problem: n items $\{1, \dots, n\}$

Item i has a non-negative weight w_i . Bound $W \geq 0$

Goal: Find $S \subseteq \{1, \dots, n\}$ that maximizes $\sum_{i \in S} w_i$ subject to the restriction $\sum_{i \in S} w_i \leq W$.

Knapsack Problem: n items $\{1, \dots, n\}$

Item i has a non-negative weight w_i and a value v_i . Bound $W \geq 0$

Goal: Find $S \subseteq \{1, \dots, n\}$ that maximizes $\sum_{i \in S} v_i$ subject to the restriction $\sum_{i \in S} w_i \leq W$.

Subset Sums: Dynamic programming

Assume that $W \in \mathbb{N}$ and $w_i \in \mathbb{N}$ for $i=1, \dots, n$.

For $1 \leq i \leq n$ and each integer w with $0 \leq w \leq W$

$$\text{OPT}(i, w) = \max_S \sum_{j \in S} w_j \quad \text{where } S \subseteq \{1, \dots, n\} \text{ and } \sum_{i \in S} w_i \leq W.$$

O = optimal solution

- $n \notin O$: $\text{OPT}(n, W) = \text{OPT}(n-1, W)$
- $n \in O$: $\text{OPT}(n, W) = w_n + \text{OPT}(n-1, W - w_n)$

If $w < w_i$, then $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. Otherwise

$$\text{OPT}(i, w) = \max\{ \text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i) \}.$$

Dynamic programming algorithm

Array $M[0..n, 0..W]$ optimal solutions $\text{OPT}(i, w)$, $0 \leq i \leq n$ $0 \leq w \leq W$.

Algorithm SubsetSum(n, W)

```
1  for  $w := 0$  to  $W$  do
2       $M[0, w] := 0$ ;
3  endfor;
4  for  $i := 1$  to  $n$  do
5      for  $w := 0$  to  $W$  do
6          if  $w < w_i$ 
7              then  $\text{OPT}(i, w) = \text{OPT}(i-1, w)$ ;
8              else  $\text{OPT}(i, w) := \max\{ M[i-1, w], w_i + M[i-1, w - w_i] \}$ ;
9          endif;
10     endfor;
11 endfor;
```

Pseudopolynomial running time: $O(nW)$

Knapsack: Dynamic programming

Assume that $W \in \mathbb{N}$ and $w_i \in \mathbb{N}$ for $i=1, \dots, n$.

For $1 \leq i \leq n$ and each integer w with $0 \leq w \leq W$

$$\text{OPT}(i, w) = \max_S \sum_{j \in S} v_j \quad \text{where } S \subseteq \{1, \dots, n\} \text{ and } \sum_{j \in S} w_j \leq W.$$

O = optimal solution

- $n \notin O$: $\text{OPT}(n, W) = \text{OPT}(n-1, W)$
- $n \in O$: $\text{OPT}(n, W) = v_n + \text{OPT}(n-1, W - w_n)$

If $w < w_i$, then $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. Otherwise

$$\text{OPT}(i, w) = \max\{ \text{OPT}(i-1, w), v_i + \text{OPT}(i-1, w - w_i) \}.$$

Pseudopolynomial running time: $O(nW)$