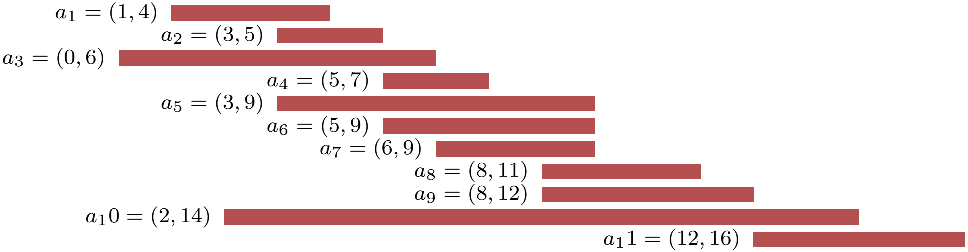
21. Greedy Algorithms

Activity Selection, Fractional Knapsack Problem, Huffman Coding
Cormen et al, Kap. 16.1, 16.3

Activity Selection

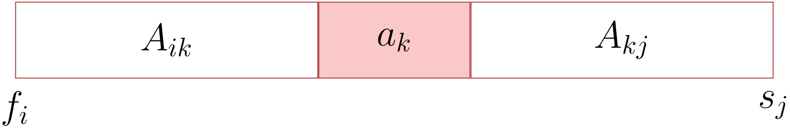
Coordination of activities that use a common resource exclusively. Activities $S = \{a_1, a_2, \dots, a_n\}$ with start- and finishing times $0 \leq s_i \leq f_i < \infty$, increasingly sorted by finishing times.



Activity Selection Problem: Find a maximal subset of compatible (non-intersecting) activities.

Dynamic Programming Approach?

Let $S_{ij} = \{a_k : f_i \leq s_k \wedge f_k \leq s_j\}$. Let A_{ij} be a maximal subset of compatible activities from S_{ij}. Moreover, let $a_k \in A_{ij}$ and $A_{ik} = S_{ik} \cap A_{ij}$, $A_{ki} = S_{kj} \cap A_{ij}$, thus $A_{ij} = A_{ik} + \{a_k\} + A_{kj}$.



Straightforward: A_{ik} and A_{kj} must be maximal, otherwise $A_{ij} = A_{ik} + \{a_k\} + A_{kj}$ would not be maximal.

Dynamic Programming Approach?

Let $c_{ij} = |A_{ij}|$. Then the following recursion holds $c_{ij} = c_{ik} + c_{kj} + 1$, therefore

$$c_{ij} = \begin{cases} 0 & \text{falls } S_{ij} = \emptyset, \\ \max_{a_k \in S_{ij}} \{c_{ik} + c_{kj} + 1\} & \text{falls } S_{ij} \neq \emptyset. \end{cases}$$

Could now try dynamic programming.

Greedy

Intuition: choose the activity that provides the earliest end time (a_1). That leaves maximal space for other activities.

Remaining problem: activities that start after a_1 ends. (There are no activites that can end before a_1 starts.)

Greedy

Theorem

Given: Subproblem S_k, a_m an activity from S_k with earliest end time. Then a_m is contained in a maximal subset of compatible activities from S_k.

Let A_k be a maximal subset with compatible activities from S_K and a_j be an activity from A_k with earliest end time. If $a_j = a_m \Rightarrow$ done. If $a_j \neq a_m$. Then consider $A'_k = A_k - \{a_j\} \cup \{a_m\}$. A'_k conists of compatible activities and is also maximal because $|A'_k| = |A_k|$.

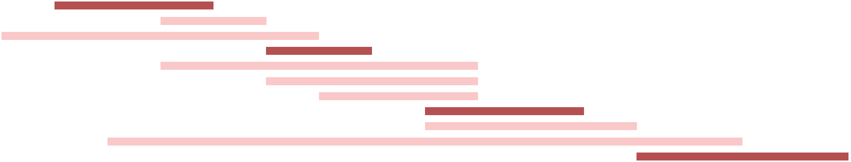
■

Algorithm RecursiveActivitySelect(s, f, k, n)

Input : Sequence of start and end points (s_i, f_i), $1 \leq i \leq n$, $s_i < f_i$, $f_i \leq f_{i+1}$ for all i. $1 \leq k \leq n$

Output : Set of all compatible activitivies.

```
$m \leftarrow k + 1$
while $m \leq n$ and $s_m \leq f_k$ do
    $m \leftarrow m + 1$
if $m \leq n$ then
    return $\{a_m\} \cup$ RecursiveActivitySelect$(s, f, m, n)$
else
    return $\emptyset$
```



Algorithm IterativeActivitySelect(s, f, n)

Input : Sequence of start and end points (s_i, f_i), $1 \leq i \leq n$, $s_i < f_i$, $f_i \leq f_{i+1}$ for all i.

Output : Maximal set of compatible activities.

```
$A \leftarrow \{a_1\}$
$k \leftarrow 1$
for $m \leftarrow 2$ to $n$ do
    if $s_m \geq f_k$ then
        $A \leftarrow A \cup \{a_m\}$
        $k \leftarrow m$
return $A$
```

Runtime of both algorithms: $\Theta(n)$

The Fractional Knapsack Problem

set of $n \in \mathbb{N}$ items $\{1, \ldots, n\}$ Each item i has value $v_i \in \mathbb{N}$ and weight $w_i \in \mathbb{N}$. The maximum weight is given as $W \in \mathbb{N}$. Input is denoted as $E = (v_i, w_i)_{i=1,\ldots,n}$.

Wanted: Fractions $0 \leq q_i \leq 1$ ($1 \leq i \leq n$) that maximise the sum $\sum_{i=1}^{n} q_i \cdot v_i$ under $\sum_{i=1}^{n} q_i \cdot w_i \leq W$.

Greedy heuristics

Sort the items decreasingly by value per weight v_i/w_i.

Assumption $v_i/w_i \geq v_{i+1}/w_{i+1}$

Let $j = \max\{0 \leq k \leq n : \sum_{i=1}^{k} w_i \leq W\}$. Set

- $q_i = 1$ for all $1 \leq i \leq j$.
- $q_{j+1} = \frac{W - \sum_{i=1}^{j} w_i}{w_{j+1}}$.
- $q_i = 0$ for all $i > j + 1$.

That is fast: $\Theta(n \log n)$ for sorting and $\Theta(n)$ for the computation of the q_i.

Correctness

Assumption: optimal solution (r_i) ($1 \leq i \leq n$).

The knapsack is full: $\sum_i r_i \cdot w_i = \sum_i q_i \cdot w_i = W$.

Consider k: smallest i with $r_i \neq q_i$ Definition of greedy: $q_k > r_k$. Let $x = q_k - r_k > 0$.

Construct a new solution (r'_i): $r'_i = r_i \forall i < k$. $r'_k = q_k$. Remove weight $\sum_{i=k+1}^{n} \delta_i = x \cdot w_k$ from items $k+1$ to n. This works because $\sum_{i=k}^{n} r_i \cdot w_i = \sum_{i=k}^{n} q_i \cdot w_i$.

Correctness

$$\begin{aligned}\sum_{i=k}^{n} r_i' v_i &= r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} (r_i w_i - \delta_i) \frac{v_i}{w_i} \\ &\geq r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} r_i w_i \frac{v_i}{w_i} - \delta_i \frac{v_k}{w_k} \\ &= r_k v_k + x w_k \frac{v_k}{w_k} - x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} r_i w_i \frac{v_i}{w_i} = \sum_{i=k}^{n} r_i v_i .\end{aligned}$$

Thus (r_i') is also optimal. Iterative application of this idea generates the solution (q_i).

Huffman-Codes

Goal: memory-efficient saving of a sequence of characters using a binary code with code words..

Example

File consisting of 100.000 characters from the alphabet $\{a, \ldots, f\}$.

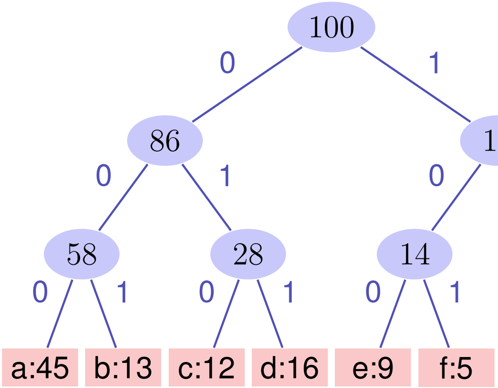
	a	b	c	d	e	f
Frequency (Thousands)	45	13	12	16	9	5
Code word with fix length	000	001	010	011	100	101
Code word variable length	0	101	100	111	1101	1100

File size (code with fix length): 300.000 bits.
File size (code with variable length): 224.000 bits.

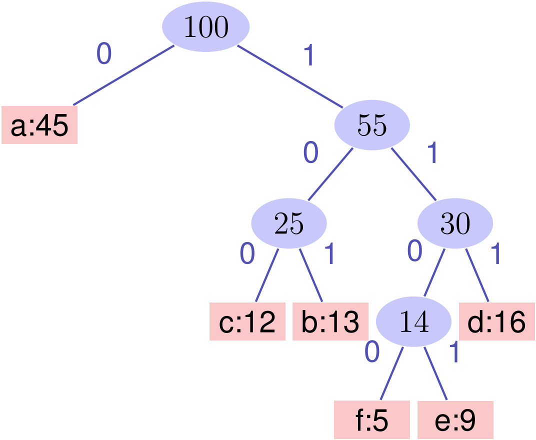
Huffman-Codes

- Consider prefix-codes: no code word can start with a different codeword.
- Prefix codes can, compared with other codes, achieve the optimal *data compression* (without proof here).
- Encoding: concatenation of the code words without stop character (difference to morsing).
  $affe \to 0 \cdot 1100 \cdot 1100 \cdot 1101 \to 0110011001101$
- Decoding simple because prefixcode
  $0110011001101 \to 0 \cdot 1100 \cdot 1100 \cdot 1101 \to affe$

Code trees



Code words with fixed length



Code words with variable length

Properties of the Code Trees

- An optimal coding of a file is alway represented by a complete binary tree: every inner node has two children.
- Let C be the set of all code words, $f(c)$ the frequency of a codeword c and $d_T(c)$ the depth of a code word in tree T. Define the cost of a tree as

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c).$$

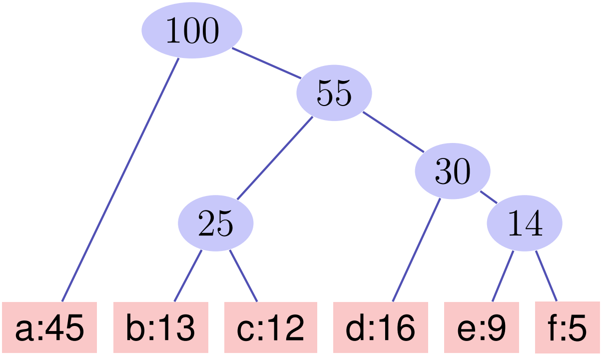
(cost = number bits of the encoded file)

In the following a code tree is called optimal when it minimizes the costs.

Algorithm Idea

Tree construction bottom up

- Start with the set C of code words
- Replace iteriatively the two nodes with smallest frequency by a new parent node.



Algorithm Huffman(C)

```
Input :    code words $c \in C$
Output :   Root of an optimal code tree

$n \leftarrow |C|$
$Q \leftarrow C$
for $i = 1$ to $n - 1$ do
    allocate a new node $z$
    $z$.left ← ExtractMin($Q$)                 // extract word with minimal frequency.
    $z$.right ← ExtractMin($Q$)
    $z$.freq ← $z$.left.freq + $z$.right.freq
    Insert($Q, z$)
return ExtractMin($Q$)
```

Analyse

Use a heap: build Heap in $\mathcal{O}(n)$. Extract-Min in $O(\log n)$ for n Elements. Yields a runtime of $O(n \log n)$.

The greedy approach is correct

Theorem

Let x, y *be two symbols with smallest frequencies in* C *and let* $T'(C')$ *be an optimal code tree to the alphabet* $C' = C - \{x, y\} + \{z\}$ *with a new symbol* z *with* $f(z) = f(x) + f(y)$*. Then the tree* $T(C)$ *that is constructed from* $T'(C')$ *by replacing the node* z *by an inner node with children* x *and* y *is an optimal code tree for the alphabet* C*.*

Proof

It holds that $f(x) \cdot d_T(x) + f(y) \cdot d_T(y) = (f(x) + f(y)) \cdot (d_{T'}(z) + 1) = f(z) \cdot d_{T'}(x) + f(x) + f(y)$. Thus $B(T') = B(T) - f(x) - f(y)$.

Assumption: T is not optimal. Then there is an optimal tree T'' with $B(T'') < B(T)$. We assume that x and y are brothers in T''. Let T''' be the tree where the inner node with children x and y is replaced by z. Then it holds that $B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$. Contradiction to the optimality of T'.

The assumption that x and y are brothers in T'' can be justified because a swap of elements with smallest frequency to the lowest level of the tree can at most decrease the value of B.