context

picture: from math to algorithms to programming

courses before DSaA:
computational thinking, networks and graphs, logic and sets

possible courses after DSaA:
automata and complexity, algorithm engineering, bachelor project

---

$P$ and $NP$

a more precise model
for instance size of the input is number of bits needed to encode it

complexity class $P$
contains all problems that are decidable in polynomial time

complexity class $NP$
contains all problems that are decidable non-deterministically in polynomial time

undecidable problems

clearly $P \subseteq NP$, but $P = NP$? is an open question

problems in $NP$

intuitively:

every instance has finitely many solutions

correctness of a possible solution can be checked in polynomial time
**NP-completeness**

A problem is **NP-complete** if

- it is in **NP** and
- every problem in **NP** can be reduced to it in polynomial time

It is believed that for many problems in **NP** there is no polynomial algorithm

**bounded tiling**

Given an $n \times n$ square

Given 1 $\times$ 1 tiles where the sides are colored

Given a first row of $n$ tiles in the square that are correctly tiled

Can the complete $n \times n$ square be correctly tiled?

Bounded tiling is **NP-complete**

There are finitely many solutions, and the correctness check of a candidate-solution can be done in polynomial time

**satisfiability of first-order propositional logic**

A formula is **satisfiable** if there is an assignment from the variables to true or false that makes the formula true

The satisfiability problem (for prop1) is in **NP**

Cook’s Theorem: the satisfiability problem is **NP-complete** (proof via reduction of bounded tiling to satisfiability)

**more NP-complete problems**

Determine whether a directed graph has an Hamiltonian cycle (a cycle containing all vertices)

However: determine whether a directed graph has an Euler tour is in $O(E)$

Traveling salesman

Decision problem for knapsack01

Optimization problem for knapsack 01 (our problem) is pseudo-polynomial

Fractional knapsack is in $O(n \cdot \log n)$
overview

* context of the course
* greedy
  * activity selection
  * fractional knapsack
  * Huffman codes
  * single-source shortest path
* material

**greedy**

we look for an optimal solution to a problem

exhaustive search is too much work

we take in each step a *locally optimal choice*

if the problem has the greedy choice property

then *locally optimal choices lead to a globally optimal solution*

**making change**

Hieronymus Bosch, Allegorie op de Gulzigheid, 1495

remember making change

for the euro-setting: making change admits a greedy choice

for the artificial setting \{1, 3, 4\} we need dynamic programming
overview

- context of the course
- greedy
- activity selection
- fractional knapsack
- Huffman codes
- single-source shortest path
- material

activity selection: definition

given:
set $S$ of activities $a_i$ each with start time $s_i$ and finish time $f_i$
$s_i < f_i$

definition compatible:
two activities $a_i$ and $a_k$ are compatible if $f_i \leq s_k$ or $f_k \leq s_i$

question:
give a maximum-size set of mutually compatible activities

activity selection: example

set $S$ of activities:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$f_i$</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

$a_1$ and $a_3$ are not compatible

$\{a_1, a_6\}$ is a set of mutually compatible activities

$\{a_1, a_4, a_5\}$ is a maximum-size set of mutually compatible activities

$\{a_1, a_2, a_5\}$ is a maximum-size set of mutually compatible activities

we say $\{a_1, a_4, a_5\}$ is a solution for $S$

set $S$ is considered ordered on increasing finish time

algorithm for activity selection?

problem: $S_{ij}$ set of activities that start after $a_i$ and finish before $a_j$

wanted: $A_{ij}$ maximum-size subset of $S_{ij}$ of mutually compatible activities

suppose $a_k$ is in $A_{ij}$
then (informally) $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$

dynamic programming: take best of this, by checking all possible $a_k$

optimal substructure: $A_{ik}$ and $A_{kj}$ are solutions of problems $S_{ik}$ and $S_{kj}$
simplification: greedy approach

take $a_1$: activity with smallest finish time $f_1$
then solve $S_{1j}$
greedy choice for activity with smallest finish time

greedy approach: correctness

let $S$ be an activity-selection problem and
let $a_1$ be an activity in $S$ with smallest finish time
we show: there is a solution for $S$ containing $a_1$

let $A$ be a solution
if $a_1 \in A$ then done
if $a_1 \notin A$ then consider $a_k \in A$ with smallest finish time
because $f_1 \leq f_k$ the set $A - \{a_k\} \cup \{a_1\}$ is also a solution

iterative greedy algorithm

input: array $s$ of start times and array $f$ of finish times
monotonically increasing in finish time

$k$ gives the most recent addition to $A$
time complexity in $\Theta n$ with $n$ amount of activities

Algorithm activitySelector($s, f$):

1. $n := s.length$
2. $A := \{a_1\}$
3. $k := 1$
4. for $m = 2$ to $n$ do
   5. if $s[m] \geq f[k]$ then
      6. $A := A \cup \{a_m\}$
      7. $k := m$

activity selection: variations

every task has benefit
every task has deadline
minimalize number of resources
overview

- context of the course
- greedy
- activity selection
- fractional knapsack
- Huffman codes
- single-source shortest path
- material

fractional knapsack: example

- 50 kilo jewels, benefit 1 million euro, 0,02 million per kilo
- 1 kilo chewing gum, benefit 20 euro, not much
- 5 kilo diamonds, benefit 5 million euro, 1 million euro per kilo
- 10 kilo gold, benefit 500000 euro, 0,05 million euro per kilo
- backpack for 20 kilo

We take as much as we can from the best benefit per weight:
- 5 kilo diamonds, total weight 5
- 10 kilo gold, total weight 15
- 5 kilo jewels, total weight 20

fractional knapsack: problem

given:
a set $S$ with $n$ items
every item $i$ has weight $w_i$ and benefit $b_i$
maximum total weight $W$
goal:
take fractions $x_i$ of all items $i$ such that
$\sum_{i \in S} b_i \cdot \frac{x_i}{w_i}$ maximal
under constraint $\sum_{i \in S} x_i \leq W$

take in each step as much as possible from the item $i$ with $b_i/w_i$ maximal

that is: we take in each step a greedy choice
fractional knapsack: algorithm

input: set $S$ with items with benefit and weight, and total weight $W$

Algorithm fractionalKnapsack($S$, $W$):
    for each item $i \in S$ do
        $x_i := 0$
        $v_i := b_i / w_i$
    $w := 0$
    while $w < W$ do
        remove from $S$ an item $i$ with highest value index
        $a := \min\{w_i, W - w\}$
        $x_i := a$
        $w := w + a$

fractional knapsack: correctness

let $A$ be a solution for fractional knapsack problem $S$, $W$
assume $A$ not greedy
so there are $i$ and $k$ with $\frac{b_i}{w_i} > \frac{b_k}{w_k}$ so $i$ is better
and $x_i < w_i$ $A$ does not take all of $i$
and $x_k > 0$ $A$ takes some of $k$
let $a = \min\{x_k, w_i - x_i\}$ the mistake part (all of $x_k$ or what remains of $x_i$)
take $x'_k = x_k - a$ and $x'_i = x_i + a$

$$\frac{b_i x_i}{w_i} + \frac{b_k x_k}{w_k} \leq \frac{b_i (x_i + a)}{w_i} + \frac{b_k (x_k - a)}{w_k} = \frac{b_i x_i}{w_i} + \frac{b_k x_k}{w_k} + a\left(\frac{b_i}{w_i} - \frac{b_k}{w_k}\right)$$

so $A$ was not optimal

greedy choice also for knapsack01?

for largest benefit does not work; take $W = 4$:

<table>
<thead>
<tr>
<th>s</th>
<th>w</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$s_3$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

for lowest weight or highest benefit per weight does not work; take $W = 2$:

<table>
<thead>
<tr>
<th>s</th>
<th>b</th>
<th>w</th>
<th>b/w</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$s_2$</td>
<td>3</td>
<td>2</td>
<td>3/2</td>
</tr>
</tbody>
</table>

fractional knapsack: time complexity

we represent the set $S$ as a priority queue
with highest priority for highers benefit per weight value
we implement the priority queue as a heap
then: analysis yields that fractional knapsack is in $O(n \log n)$
there exists a linear time algorithm for fractional knapsack
not treated in this course
overview

- context of the course
- greedy
- activity selection
- fractional knapsack
- Huffman codes
- single-source shortest path
- material

Huffman encoding: intuition

encoding of characters
character that occurs more frequently has a shorter encoding

binary coding of characters

we use words over 0 and 1 to encode characters

fixed-length code:
example: \( a = 000, b = 001 \)
ASCII, Unicode

variable length code:
example: \( a = 0, b = 101, c = 100 \)
Huffman codes

prefix codes

definition:
no codeword is the prefix of another codeword

assumption:
we work with prefix codes

example prefix code:
\( a = 0, b = 101, c = 100 \)

example non prefix code:
\( a = 0, b = 00 \)
representing codes
as a binary tree where left is 0, right is 1, leaves are codes
optimal code is represented by a full binary tree
(every node has 0 or 2 children)

<table>
<thead>
<tr>
<th>00</th>
<th>010</th>
<th>011</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

example:

```
00
010
011
10
11
```

```
a
b
c
d
e
```

Huffman codes

assume a set of characters
every character $c$ has a an attribute for frequency $c.freq$
we build a tree $T$ for an optimal coding

```
Algorithm HuffmanCode(C):
    n := |C|
    Q := C
    for i = 1 to n - 1 do
        new node z
        z.left := x := removeMin(Q)
        z.right := y := removeMin(Q)
        z.freq := x.freq + y.freq
        insert(Q, z)
    return removeMin(Q)
```

Huffman codes: example

```
a b c d r
5 2 1 1 2
```

Huffman's algorithm

greedy
non-deterministic

inventor: David Huffman in 1952 during his PhD
consider an alphabet of $n$ characters; each with frequency

init takes $\log n$

for-loop takes $n \cdot \log n$

using a min-priority queue implemented using a binary min-heap

in $O(n \log n)$
extra material

- wiki about character encoding
- wiki about David Huffman
- splay tree, which has amortized complexity in $O(\log n)$ for insertion