TECHNISCHE
UNIVERSITÄT
DARMSTADT

- What is a Problem?
  - **Problem:** binary relation between a set *I* of instances and a set S of possible solutions.

    Example Maxsum problem:

    Input: Sequence $a_1,...,a_n$ of integer numbers. Let $f(i,j) := a_i+a_{i+1}+...+a_j$,
    for $1 \leq i \leq j \leq n$.
    Searched: The maximum $f(i,j)$.

    The set of instances consists of all sequences with finite length that consist of integer numbers. The set of solutions is the set of integer numbers. With the help of the definition of f and by the demand for a maximum f(i,j), these integer-number sequences are related to maximum sums.

Complexity Theory

- What is a problem (2)?

  We distinguish between the abstract *problem* and the *description* of the problem and its instances.

  If you want to communicate a problem, you must encode it.

  Encoding of a problem and its instances, i.e. their descriptions:

  – An alphabet is a set of symbols. They are not God-given, but we have to agree on them. E.g.:
    - {A,..,Z,a,..,z,0,…,9} suffice for most of every-day correspondence
    - Sound for linguistic communication
    - {0,1} is especially well suited in order to describe problems for computers

Complexity Theory

– In order to describe a problem, we need an alphabet. Moreover, we have to define rules that describe the meaning of symbol/character combinations; so called encoding schemes.

- **Integer numbers** have a **binary** discription (i.e. **bits**). We write

$$n = \pm \sum_{i=0}^{k} x_i \cdot 2^i,\ x_i \in \{0,1\} \quad and\ k = \lfloor \log_2(|n|) \rfloor$$

therefore, the **coding length <k>** of an integer number is given by

$$\langle k \rangle = \lfloor \log_2(|n|) \rfloor + 1 + 1 = \lfloor \log_2 |n| \rfloor + 2$$

- **Rational numbers:** Let r be a rational number. Then, there is an integer number p and a natural number q such that r = p/q.

<r> = <p> + <q>

Complexity Theory

- **Vectors**

  for $x = (x_1,\ldots,x_n) \in \mathbb{Q}^n$ it is

$$\langle x \rangle = \sum_{i=1}^{n} \langle x_i \rangle$$

- **Matrices**

  for $A \in \mathbb{Q}^{m\mathrm{x}n}$ it is

$$\langle A \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle$$

Input length: The number of bits, consumed in order to completely describe an instance *I* is called input length $\langle I \rangle$.

Complexity Theory

- Solving a problem, i.e. assigning correct solutions to arbitrary instances of the problem, can be more or less difficult.
  - **E.g.: In a most complicated case a problem may be undecidable**:
    - **given**: Coding of an algorithm (= program) for a Random Access Machine (RAM, more or less a computer with unlimited memory capacities), as well as a w $\in \Sigma$
      Question: Does the program stop after finite time on input w?

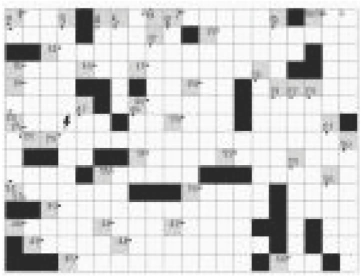
“not decidable” means: there is no algorithm which might be able to give the correct answer to all instances of the problem.

In the following, all problems will be decidable. The only question will be how many resources in form of time or space are consumed.

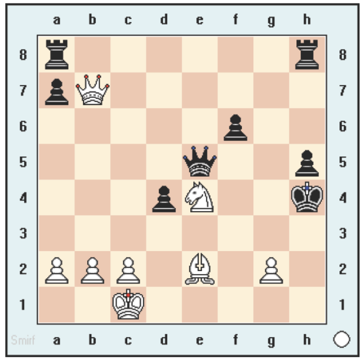
Every day life problems

- What is more difficult?

– Mental arithmetic

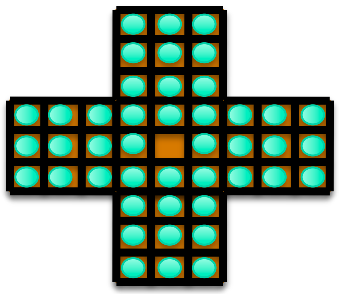


– Crossword puzzle



– Chess

– Socoban



– Solitair

Algorithms and compute model

TECHNISCHE UNIVERSITÄT DARMSTADT

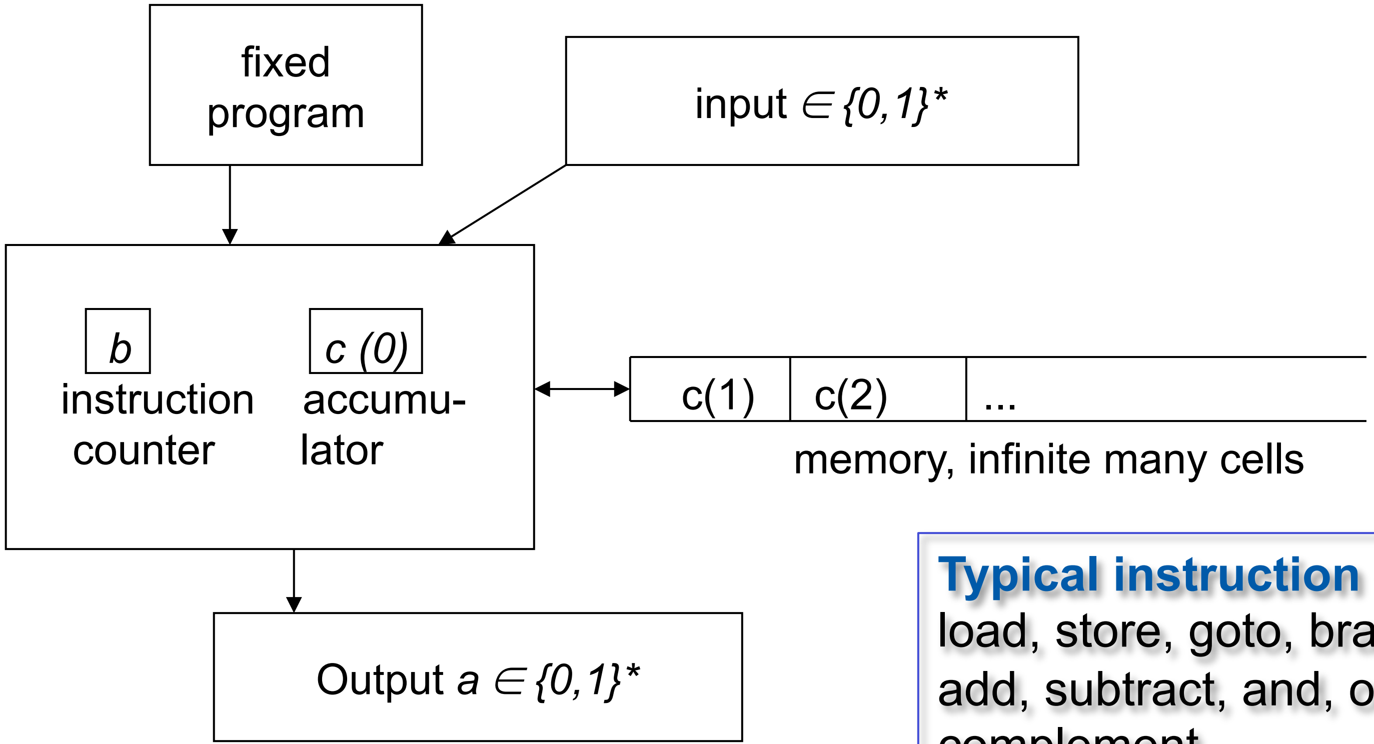
- **What is an algorithm?**
  - An **algorithmus** is an instruction sequence for solving a problem step by step. We say, an algorithm A solves a problem Π, if A finds a correct solution for arbitrary instances $I \in \Pi$ of the problem within a finite number of steps. The instruction sequence must have constant encoding length.
  - A step is an *elementary operation*. (??) An elementary operation of pie baking („oven on 180°") differs from an elementary operation of car („tire pressure to 2.0 bar").

Obviously, the definition of an elementary operation depends on the machine that executes our algorithm A!

Algorithms and compute model

→ Compute model, efficiency measure.

register machine (Random Access Machine RAM)



Typical instruction set:
load, store, goto, branch on zero, add, subtract, and, or, bit-complement

Algorithms and compute model

Additional distinction: unit-cost vs. log-cost model

Unit-cost model: *every instruction of the RAM costs one time unit*

Typical instruction set:
+, -, *, /, compare, delete, write and read of rational numbers, control flow with the help of if … else branches, loops
Mostly, we will use this cost-model.

Log-cost model: *each instruction costs Θ(k) time, where k is the number of bits of the operands.*

Typical instruction set:
load, store, goto, branch on zero, add, subtract, bitwise and, bitwise or, bit-complement
This model is more realistic and is relevant e.g. in optimization, e.g. when you analyze the so called ellipsoid method.

Algorithms and compute model

Efficiency measures (algorithm A): worst-case, average-case, best-case

$T_A(x)$ = number of instruction, executed by A on input x.
$S_A(x)$ = largest address in memory, used by A on input x.

- **Worst Case runtime:** $T_A^{wc}(n) := max\ \{T_A(x) \mid <x> = n\}$
- **Average Case runtime:** $T_A^{ac}(n) := \sum_{\{x \mid <x> = n\}} p_x\, T_A(x),$
  demands knowledge on probabilities or equal distribution is assumed
- **Best Case runtime:** $T_A^{bc}(n) := min\ \{T_A(x) \mid <x> = n\}$

  (In our examples so far, it was $T_A^{bc}(n) \approx T_A^{wc}(n)$ *.)*
- **Space requirement**: $S_A^{wc}(n) := max\ \{S_A(x) \mid <x> = n\}$

Algorithms and compute model

TECHNISCHE UNIVERSITÄT DARMSTADT

- Definition: (worst-case) complexity of an algorithmus
  – Let A be a deterministic (RAM-)algorithm, that stops on any input.
  – The **runtime (time complexity)** of A is a function f: N→N,
    - f(n) being the maximum number of step of A, **running on inputs of length n**.
    - Linear-time-algorithmus: $f(n) \leq c\ n$ for a constant c
    - Polynomial-time-algorithmus: $f(n) \leq c\ n^k$ for constants c and k (and *n* sufficiently large)

- Definition: Complexity of a problem
  – The time- (space-) complexity of a problem p is the runtime of the fastest (least space consuming) algorithm that solves Problem
  – A problem p is "solveble in polynomial time", if there is an algorithm A, a polynomial Π and an $n_0 \in \mathbb{N}$, such thatz for all $n > n_0$ **is valid : $f(n) <= p(n)$**

Algorithms and compute model

Example: Adding 1 in binary system

Input: binary representation $x_{n-1} \dots x_0$ of x

Output: binary representation of $x + 1$

Algorithm:

if $x_{n-1} \dots x_0 = (1 \dots 1)$,

return $y_n \dots y_0 = (1\ 0 \dots 0)$,

else *search for critical position*, i.e. the smallest i with $x_i = 0$.

return $(x_{n-1} \dots x_{i+1}\ 1\ 0 \dots 0)$.

Runtime: # manipulated bits

worst case : $n + 1$ (at input $1 \dots 1$)

best case : 1 (e.g. at input $1 \dots 10$)

Algorithms and compute model

Average Case:

1 Operations with probability ½
2 Operations with probability ¼
...
n Operations with probability $(1/2)^n$

$$1*1/2 + 2*1/4+3*1/8+... = \sum_{i=0}^{n-1} \frac{1}{2^{i+1}}(i+1) \leq \sum_{i=0}^{\infty} \frac{1}{2^{i+1}}(i+1) = 2$$

- Average case near best case!

- As we,,. Examples exist, where average case is far away from worst case and best case

Algorithms and compute model

- **Example, showing the dependencies of runtime and input length:**

  - Def. Fibonacci-numbers: $F_0=0$, $F_1=1$, $F_n=F_{n-1}+F_{n-2}$
  - Very slow algorithm:

    ```
    fib(n)
      if n <= 1 return F_n
      else      return fib(n-1)+fib(n-2)
    ```

    Runtime: O(2^n). However, n is index in Fibonacci-sequence, and ist encoding length is logarithmic in n. Thus, let k = <n>. Then the runtime in k is O((2^(2^k))
  - Slow algorithm:

    ```
    f0=0; f1=1
    for i = 2 to n do
       tmp=f1;
       f1=f1+f0;
       f0=tmp;
    if n=0, return f0 , else return f1
    ```

Algorithms and compute model

– Def. Fibonacci-numbers: $F_0=0$, $F_1=1$, $F_n=F_{n-1}+F_{n-2}$

fast algorithm:

inspect $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ as well as $F = \begin{pmatrix} f_n & f_{n-1} \\ f_{n-1} & f_{n-2} \end{pmatrix}$

$$A^1 = \begin{pmatrix} f_2 & f_1 \\ f_1 & f_0 \end{pmatrix}, A^2 = \begin{pmatrix} f_3 & f_2 \\ f_2 & f_1 \end{pmatrix}, \text{and}$$

$$F \cdot A = \begin{pmatrix} f_n & f_{n-1} \\ f_{n-1} & f_{n-2} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_n + f_{n-1} & f_n \\ f_{n-1} + f_{n-2} & f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}$$

Obviiously, $A^n{}_{1,2}$ is the n-th Fib-number. However: so what?

Algorithms and compute model

TECHNISCHE UNIVERSITÄT DARMSTADT

- Computation of A^n

- Let us inspect the binary representation of n:

$$n = \sum_{i=0}^{k} x_i \cdot 2^i, x_i \in \{0,1\} \text{ and } k = \lfloor \log_2(|n|) \rfloor \text{ and therefore}$$

$$A^n = A^{\sum_{i=0}^{k} x_i \cdot 2^i} = \prod_{i=0}^{k} A^{x_i \cdot 2^i} = \prod_{\left\{\substack{x_i = 1 \\ 0 \le i \le k}\right\}} A^{2^i}$$

- E.g. m = 13 = 1101_2. Build A, A^4, A^8 and build $A*A^4*A^8 = A^{1+4+8}$

It is $$A^{(2^i)} = (A^{\overbrace{2\cdots 2}^{O(k)-times}}) = \overbrace{(\ldots(A^2)^2\ldots)^2}^{O(k)-times:\, squaring}$$

- Effort for exponentiation: O(k)
  Effort for A^n : O(k)

In the Unit-cost Model!!